Program Acceleration Using Nearest Distance Associative Search

Mohsen Imani, Daniel Peroni, and Tajana Rosing
Computer Science and Engineering, UC San Diego, La Jolla, CA 92093, USA
{moimani, dperoni, tajana}@ucsd.edu

***Abstract*—Data generated by current computing systems is rapidly increasing as they become more interconnected as part of the *Internet of Things* (IoT). The growing amount of generated data, such as multimedia, needs to be accelerated using efficient massive parallel processors. Associative memories, in tandem with processing elements, in the form of look-up tables, can reduce energy consumption by eliminating redundant computations. In this paper, we propose a resistive associative unit, called RAU, which approximately performs basic computations with significantly higher efficiency compared to traditional processing units. RAU stores high frequency patterns corresponding to each operation and then retrieves the nearest distance row to the input data as an approximate output. In order to avoid using a large and energy intensive RAU, our design adaptively detects inputs with lower frequency and assigns them to precise cores to process. For each application, our design is able to adjust the ratio of data processed between RAU and precise cores to ensure computational accuracy. We consider the application of RAU on an AMD Southern Island GPU, a recent GPGPU architecture. Our experimental evaluation shows that GPGPU enhanced with RAU can achieve 61% average energy savings, and 2.2× speedup over eight diverse OpenCL applications, while ensuring acceptable quality of computation. The energy-delay product improvement of enhanced GPGPU is 5.7× and 2.8× higher compared to conventional and state-of-the-art approximate GPGPU, respectively.**

I. Introduction

In order for embedded devices to become *Internet of things* (IoT) computing nodes, they must be capable of processing raw sensing data streams, which traditionally have run on servers. This is, in part, motivated by the need to immediately access information regardless of connectivity [1]–[6]. Unfortunately, existing device architectures cannot sustain the computational loads required by IoT, because the algorithms necessary to process IoT data consume too much power and would not nearly have enough performance to meet real-time needs for feedback. Therefore, we need to build systems capable of responding to our needs with acceptable quality of response, while also being capable of significantly faster and much more energy efficient performance [7]–[10]. Many of the algorithms which processes sensor data are, at their heart, statistical and thus, do not require exact answers [11]. Similarly, in audio and video processing we have long exploited the fact that humans do not perceive all colors and sounds equally well.

Associative memory in the form of a look-up table (LUT) is a promising solution to improve the energy efficiency of parallel processors [12]–[16]. Associative memory prestores a set of frequent patterns and retrieves them at runtime in order to reduce the redundant computations. In hardware, associative memory is implemented using ternary content addressable memory (TCAM). However, CMOS-based TCAMs consume significant energy during searches, limiting their application as associative memories. The high density and zero leakage power of non-volatile memories (NVMs) represents a promising opportunity for building efficient memory and computing units [17], [18]. The main idea of the approximation is, rather than accurately computing on existing processing units, TCAM returns precomputed results, not only for perfect matches of operands, but also for inexact matches. Voltage overscaling has been introduced to apply approximation and further reduce the associative memory energy consumption [19]–[26]. This technique has low energy improvement due to its inability to tune the level of output accuracy. However, these techniques cannot properly trade energy and accuracy because the Hamming distance metric it uses does not consider the exact impact of each bit indices on approximation. In addition, associative memory in exact or approximate mode cannot provide any performance improvement.

In contrast to prior work which applied associative memory for computational reuse, we designed a resistive associative unit, called RAU, which can accelerate computation as a stand-alone memory-based computing unit. RAU stores high frequency patterns corresponding to each operation and searches for a nearest distance data to the input operand at execution time. In order to avoid using large RAU and to dynamically tune the computation accuracy, RAU detects inputs with lower frequency and assigns them to precise cores to process. For each application, our design adjusts the ratio of data processed between RAU and precise cores in order to ensure desired computation accuracy. We apply RAU to an AMD Southern Island GPU, where RAU is integrated beside each floating point unit in the GPGPU architecture in m pipeline stages. Our experimental evaluation shows that GPGPU using RAU can achieve 61% average energy savings and 2.2× speedup over eight general OpenCL and machine learning applications, while ensuring the quality of computation is in acceptable range.

II. Related Work

Associative memory has been applied to a wide range of domains including associative computing, memorization, networking, and recently approximate computing [19], [27]. Associative memory uses TCAM blocks to store frequent patterns and reuse them at runtime. The conventional CMOS-based TCAM consists of two SRAM blocks and consumes substantial energy for each search operation [28]. Non-volatile memories (NVMs) represent a new opportunity to design efficient TCAMs with high density and low leakage power [29]. Resistive Random Access Memory (ReRAM) and Magnetic RAMs (MRAMs) are two common types of non-volatile memories used in memory design [30]. Although resistive CAMs are faster, denser, and more energy efficient when compared to MRAM CAMs, their limited write endurance ($\sim 10^7$ resistive vs. $\sim 10^{15}$ Magnetic) reduces their application in the computing domain [31]. In this work, we use ReRAM-based TCAM and address the endurance issue by applying offline profiling and limiting the number of updates to once during the application change.

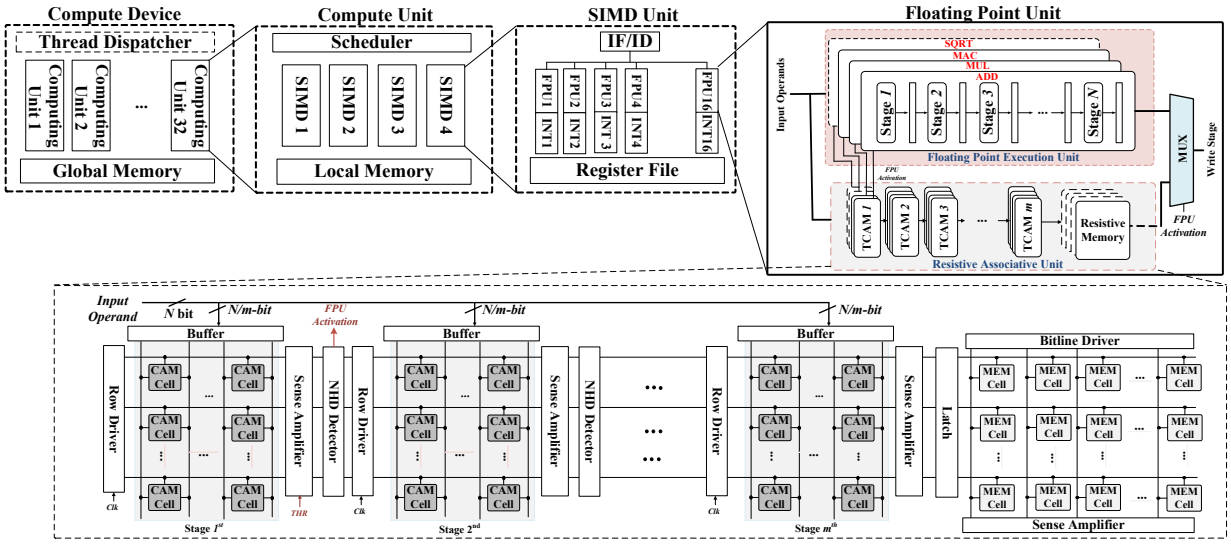

Fig. 1. (a) Integration of RAU in AMD Southern Island GPU architecture and (b) the detail structure of multistage RAU.

Previous works used associative memories in parallel with processor architectures for computational reuse, memoization and enabling approximate computing [27]. In memoization, a single cycle associative memory beside each processing cores maintains the context of error free execution [27]. For computational reuse, associative memories exploit data locality to reduce redundant computation. However, TCAM in associative memory architecture consume large amounts of power during searches. To improve the search energy consumption, work in [32] implements voltage overscaling on a resistive TCAM block. However, resistive TCAM rows under voltage overscaling inexactly match to input data within a few number of Hamming distances. The lack of approximate configurability limits the application of approximate hardware to a small range of error tolerant applications. Work in [19] proposed configurable associative memory which can relax computation using voltage overscaling of TCAM rows/bitlines to trade energy and accuracy for each application. However, these techniques: **(i)** cannot improve computation performance, since the performance is bounded by GPU pipeline stages, and **(ii)** cannot optimally trade energy and accuracy since the hamming distance metric does not consider the exact impact of each bit index on computation.

Work in [33], [34] used resistive CAM in order to accelerate CPU computation in application level. However, this work uses inaccurate Hamming distance as similarity metric which results in significantly low accuracy for GPU usage. In contrast, we propose a resistive associative unit which can improve GPGPU computation by providing large energy savings and speed up. Instead of using Hamming distance metric to find the closet row, RAU returns a row with the closest distance to the input data at each search operation. Our design adaptively identifies the data with lower frequency and runs that portion of data on exact hardware in order to guarantee computation accuracy.

III. GPGPU Acceleration

A. Data Locality and Approximation

There is significant data locality in GPGPU workloads. For example, in the multimedia domain, many images or video frames have pixel similarities. We can take advantage of this data locality by storing frequently occurring computations to memory. When the GPU receives inputs the memory banks are checked for a matching pattern, and if a set matching the inputs is located, the associated output is returned. The Floating Point Unit (FPU) does not need to run when a hit occurs, saving energy. The application of memory is not limited to exact matches. A well designed implementation can return not only exact matches, but close matches as well. If the allowable deviation from the stored values is increased, the number of matches to values stored in memory also increases. However, this comes at the cost of output accuracy. When no deviation limit is set, the output for the values closest matching the inputs is returned.

B. Resistive Associative Unit and its Integration

We propose a resistive associative unit (RAU) which exploits data locality to approximately process the data without requiring a processing core. RAU consists of a ternary content addressable memory (TCAM) to store high frequency patterns and memory for their corresponding outputs. In contrast to conventional associative memory, RAU has the computing capability to search for a TCAM row which has the nearest distance to the input data. When an input operand hits on TCAM, the corresponding row of resistive memory is activated to read the approximate result of computation. The distance metric used to find nearest row has a major impact on accuracy. Hamming distance can be used, however, this metric does not provide sufficient computation accuracy because it does not consider the impact of different bit indices on computation. In GPU architecture, the most significant bits (MSBs) have much higher impact on computation than the least significant bits (LSBs). In contrast, our distance metric considers the impact of bit indices when finding the nearest value.

The RAU can be used in many different domains as an efficient approximate computing unit. In this paper, we focus on parallel processing in a particular GPU architecture. Fig. 1 shows the AMD Southern Island GPU architecture, which consists of 32 computing units, each with four SIMD. Each SIMD consists of 16 lanes, and has both integer and floating point units. In GPU architecture, the FPUs are the slowest and consume the most energy during computation. Fig. 1 shows the structure of the proposed RAU consisting of m serial stages working in pipeline search. In RAU

processing, the input data is split into m equal N/m-bit sized parts, where N is input word size. RAU starts the search operation from data in most significant block (1^{st} Stage) and based on hit(s) of the stage, our design activates the rows of the second RAU stage selectively. Instead of searching whole rows of the second stage, our design searches for a nearest row in the selected rows. This selective row activation and serial search continues until RAU reaches a single active row on the last stage. This multistage search significantly improves the energy consumption of RAU blocks by reducing the number of active rows. The clock frequency of RAU is different from FPU pipeline stages, because RAU stage ($<$ 2048-rows) can compute faster than FPU stages. The RAU clock frequency is set based on the size of each RAU stage.

1) Stand-alone RAU Computing: For inputs that are outside associative memory's scope, conventional TCAMs do not return a value. In contrast, our TCAM in RAU works approximately and returns a row which has the closest distance to the input operand, allowing RAU to be used as stand-alone approximate computing unit. For example, in GPU architecture, the RAU can replace an FPU to approximately model the FPU's basic functionalities such as addition, multiplication, and square root. The RAU computation accuracy depends on the number of pre-stored patterns. Although increasing the size of RAU improves computation accuracy, the additional rows increase search energy and delay, because a larger TCAM requires an input buffer to distribute data amongst all rows simultaneously. This buffer makes up a large portion of the total TCAM energy and search delay.

2) Hybrid RAU Computing: Our design also supports hybrid computation, where workloads are partially processed on memory with the remainder run on precise cores. A small stand-alone RAU searching for the nearest data may suffer low computation accuracy for some applications. Therefore, instead of significantly increasing the RAU size, we assign the less frequent data to FPUs to process. In this way, we can leave the RAU block small to support high frequency patterns. In many applications, the output accuracy is sensitive to some key, yet infrequent, values. For example, in *Sobel filter*, the data stored on the edges usually has lower frequency, but higher impact on computation accuracy. To achieve large energy savings using smaller RAU, our design adaptively identifies sensitive and far inputs, i.e., inputs with large distance from stored memory values, and processes them on FPUs. To ensure computation accuracy, the ratio of data split between RAU and FPU can be set by changing the maximum distance the first RAU stage accepts during its search operation. To find the proper distance threshold (*THR*) for each application, our framework, shown in Fig. 2, starts running computation on precise and approximate GPGPU. In the case of having higher quality than required level ($QoS > Q_R$), our framework increases the *THR*, means increasing the portion of data running on RAU. This process continues until quality, *QoS*, drops below the required value, Q_R. In this work we defined Q_R to be 90% and 98% for general and machine learning applications respectively, verified by prior work [19], [35]. In that case, our design selects the *THR* value in previous iteration as the maximum threshold value which ensures quality of service.

IV. RAU Hardware Design

A. RAU Memory Cell

In this work, we use VTEAM memristor model [36] for our memory design simulation with R_{ON} and R_{OFF} of $10k\Omega$ and $10M\Omega$ respectively. We design a crossbar content addressable memory for the RAU block ($1T-nR$). Crossbar memory is an access-free transistor memory architecture, which can achieve high density, significantly reduced energy, and scalability. The area of crossbar resistive memory is $4F^2/n$, where F is the minimum feature size and the n is the number of resistive layers in 3 dimensional space. Therefore, in our case, the RAU can be implemented at the top of the CMOS-based FPUs with negligible area overhead. Fig. 3 shows the structure and functionality of crossbar TCAM, where the values are stored on memory cells based on the memristor states. Crossbar cells in RAU store the low resistance (R_{ON}) and high resistance (R_{OFF}), representing zero and one values respectively, which is inverse of traditional crossbar cells. This provides inverse functionality of TCAM and allows us to design a CAM which can search for nearest distance data.

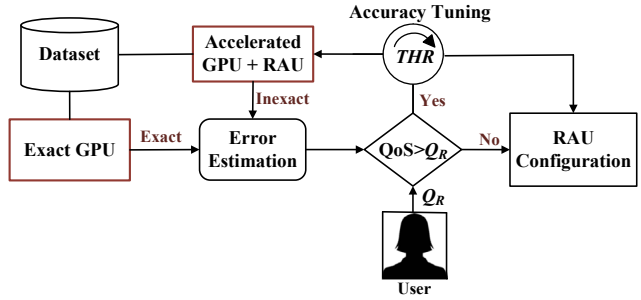

Fig. 2. Framework to support tunable approximate computing (Q_R is the requested quality by the user)

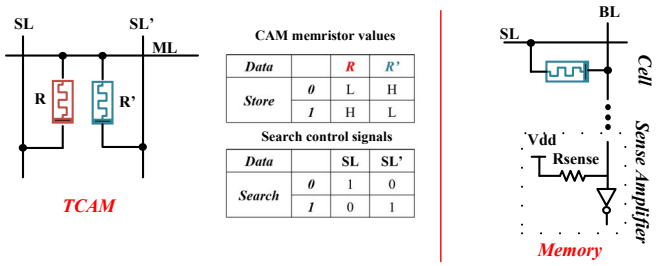

Fig. 3. The TCAM and memory cells in proposed RAU design.

B. Nearest Distance CAM

Conventional TCAMs do not have the ability to search for a nearest distance row to an input key. To better understand the functionality of RAU, first consider the search operation in conventional TCAM. The search operation has two phases: precharging and evaluation. Before each search operation, a row driver precharges all TCAM rows (matched line). Then, in evaluation mode, input data is distributed among all rows using vertical bitline. A traditional TCAM cell discharges *ML* in case of a mismatch, but in our proposed RAU, the memristor values are programmed inversely. This means a TCAM cell with a similar stored value to the input discharges the *ML*, while the mismatch cells maintain *ML* charge. Fig. 4 shows the timing characteristic of a proposed 8 bit TCAM row, when an input data matches with stored value with different numbers of bit matches ($i = 1, ..., 5$ bits). During the search operation, the matched cells start discharging the *ML*. The rate of *ML* discharging depends on the number of matched cells. To detect a nearest Hamming distance row, we need to detect a row which has the fastest *ML* discharging voltage. In RAU, the sense amplifier finds the nearest

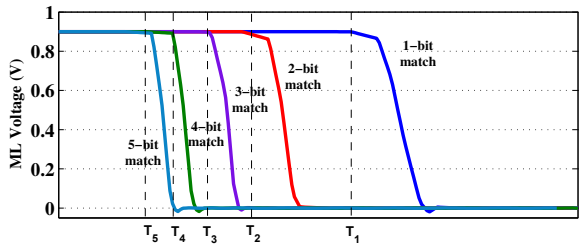

Fig. 4. ML discharging current in a CAM row with different number of matches.

rows by tracking the *ML* discharging voltage in all TCAM rows.

C. Peripheral Circuitry

Fig. 5a shows the peripheral circuitry to support nearest distance search. For each TCAM stage, peripheral circuitry consists of three parts: a sense amplifier, a nearest Hamming distance (NHD) detector, and a row driver. A sense amplifier strengths the *MLs* voltage to discharge the ML voltage in a row which has more bits matching with input data. An NHD detector is an resistive circuitry which samples output sense amplifier voltages in all rows and finds the time that first row(s) start discharging the *ML*. The value of R_{Sense} in the NHD detector is set based on the number of TCAM rows. When an NHD detects the first matched row(s), it signals the sense amplifier to stop the search operation on all TCAM rows. Meanwhile, the C_{Dr} capacitor on selected nearest row(s) starts charging. Finally, the activated capacitor(s) will selectively precharge row(s) of the next TCAM stage in the next clock cycle (*Clk*).

Fig. 5b shows the structure of sense amplifier circuitry in the first and other RAU stages, i.e., 2^{nd} to m^{th}, to support nearest distance search. The dashed lines in the figure are extra circuitry of the first stage RAU. The goal of first stage sense amplifier is to stop hits on the first TCAM stage, when the input data does not closely match with stored RAU values. The *THR* signal dynamically sets a maximum far matching (e.g., h bits) that our design can accept in nearest distance search. If an input data has larger distance than h-bit, the *THR* signal activates M_T transistor and stops the search operation on the remaining RAU stages. *THR* decides whether the computation needs to process on RAU or FPUs. In case of *CG*=1, the search continues on next RAU stages, otherwise the computation starts running on FPU. The time of *THR* activation, which determines the level of approximation, can be dynamically set based on a maximum acceptable distance on nearest search.

Fig. 6 shows the HSPICE waveform verifying the correct functionality of the proposed design. The *ML* precharges in the i^{th} TCAM stage on the positive clock edge (T_1). The search at the i^{th} TCAM stage starts by distributing input data among all TCAM rows (T_2). After that the ML_i discharges as the nearest distance row to input data (T_3). When the sense amplifier detects the discharging current of ML_i, it enables the *EnL* signal (T_4). Finally, the *EnL* signal activates the ML_{i+1} of the next *ML* by precharging it to Vdd (T_5).

V. Experimental Results

A. Experimental Setup

We implemented RAUs on the AMD Southern Island GPU architecture, Radeon HD 7970 device, a recent GPU architecture. We adopt the image processing from AMD APP SDK v2.5 in OpenCL to make it suitable for streaming applications. We use

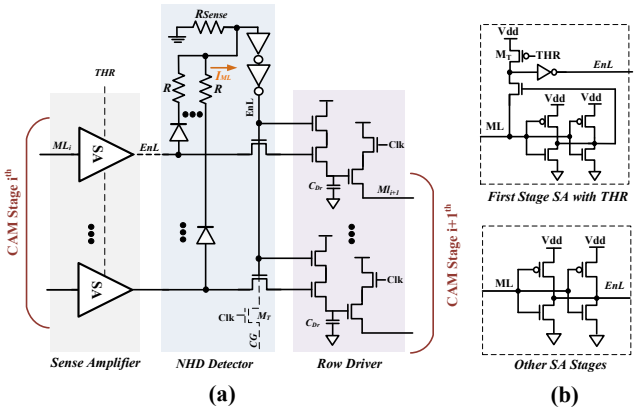

Fig. 5. (a) RAU peripheral circuitry to support nearest distance search. (b) Sense amplifier of different RAU stages.

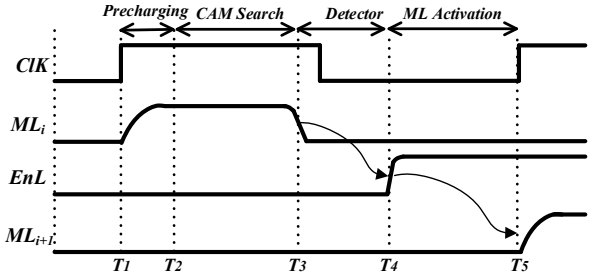

Fig. 6. HSPICE waveforms to show the functionality of proposed RAU peripheral circuitry.

Multi2sim, a cycle accurate CPU-GPU simulator and modified the kernel code to do profiling and runtime simulation [37].

B. RAU Configurations

The number of RAU stages greatly impacts computation energy and speedup, as well as accuracy. A short RAU block can provide following advantages/disadvantages: **(i)** Using short RAU pipeline results in fast precharging and search operation on TCAM block, which results in search speedup. **(ii)** RAU stages work in pipeline stages where the rows of the next TCAM stage activates based on the hits of the current stage. This selective row activation reduces the number of active rows and improves energy savings on next RAU block. **(iii)** The disadvantage of using short RAU is fewer opportunities for our hardware to adaptively balance the ratio of the running application on RAU and FPU. The first RAU stage does not contain enough bits to change *THR* and decide a portion of running data on RAU and FPU. We use Energy-Delay Product (EDP) as a metric to find the best block size over eight tested applications. Based on our results, the minimum EDP occurs at middle block sizes, i.e., 4 bits and 8 bits, depending on application type. The average EDP for eight applications shows using 4 bits block is the best configuration which can provide maximum 2.3× EDP improvement compared to traditional GPGPU architecture.

C. Stand-alone RAU Computing

Table I shows shows energy improvement, performance speedup of GPGPU when the FPUs are replaced with RAU of different sizes. In each RAU size, the energy improvement and speedup are relative to conventional GPGPU architecture using FPUs. Table I also shows quality loss (QL) of running eight diverse GPGPU applications on RAU with different sizes. Increasing the RAU size

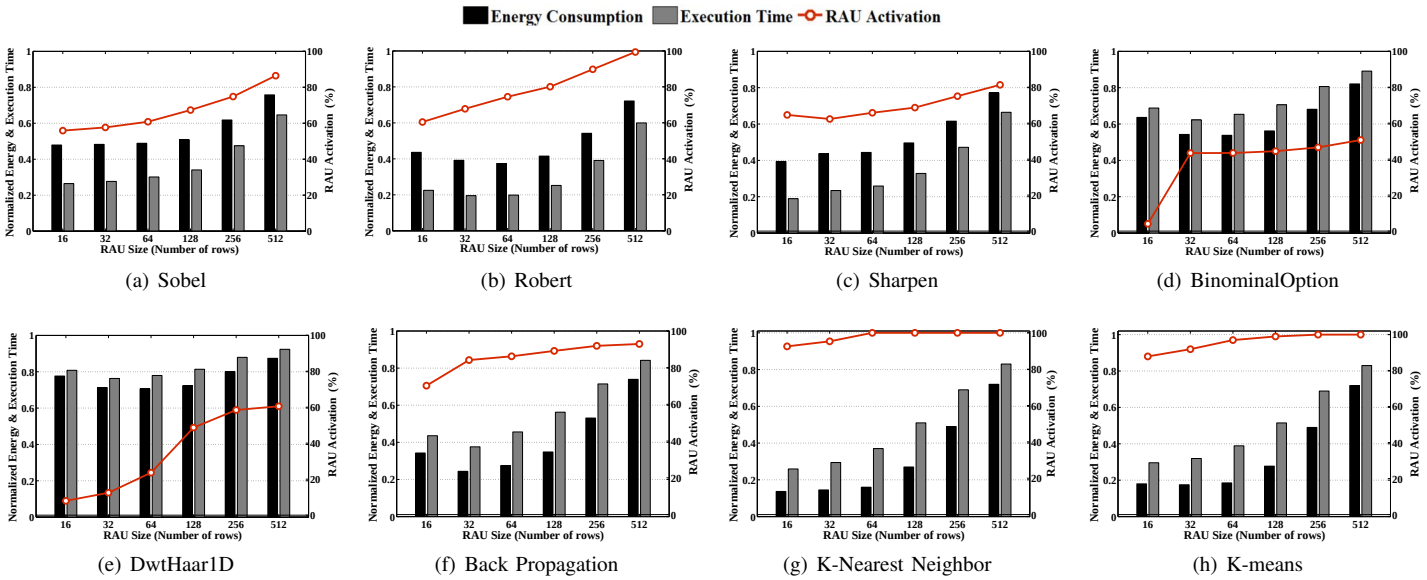

Fig. 7. Energy consumption and performance speedup of running applications on tunable enhanced GPGPU architecture, normalized to conventional GPGPU. The RAU activation shows the ratio of hits in RAU to total hits in FPU. For each application and each RAU size this activation is set in order to guarantees the quality of service.

TABLE I
ENERGY IMPROVEMENT, PERFORMANCE SPEEDUP AND QUALITY LOSS (QL) OF RAU-BASED COMPUTATION IN DIFFERENT SIZES.

RAU Size (# rows)	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
Energy Improv.	14×	9.6×	6.2×	3.7×	2.0×	1.3×	1.1×
Speedup	7.5×	5.7×	4.0×	2.9×	2.1×	1.8×	1.4×
Quality Loss (QL)							
RAU Size (# rows)	**16**	**32**	**64**	**128**	**256**	**512**	**1024**
Sobel	64%	59%	59%	37%	24%	16%	11%
Robert	25%	27%	25%	24%	16%	8%	7%
Sharpen	98%	90%	75%	36%	32%	27%	24%
Binom	76%	63%	47%	34%	26%	18%	14%
DwtHaar	24%	17%	11%	9%	7%	5%	2%
Backprop	28%	17%	11%	5%	2%	1%	1%
KNN	11%	5%	2%	1%	0%	0%	0%
K-means	13%	10%	4%	1%	0%	0%	0%

from 16-rows improves applications quality by storing more highly frequent patterns. However, this accuracy improvement saturates in large RAU, because the additional rows contain lower frequency patterns, resulting in diminishing error reductions. Although RAU runs much faster and energy efficiently in small RAU sizes, the computation accuracy is not acceptable for those cases. Therefore, we require a large RAU block to provide acceptable accuracy, which results in lower energy and performance improvement. For instance, using 1024 row RAU can only provide required accuracy for Sobel application, with minor energy/performance improvement compare to conventional GPGPU architecture. Looking at machine learning algorithms, RAU can provide higher quality of service running them on stand-alone mode. The result shows that k-means algorithm can provide acceptable quality of clustering (for ML algorithm defined as QL$<2\%$) when working with 128-rows RAU or larger. Similarly, other learning algorithms can provide acceptable quality when using small RAU sizes. Our evaluations show that running machine learning algorithms on stand-alone RAU (256-rows) provides 2.0× and 2.1× energy improvement and speedup compare to conventional GPU using FPUs.

D. Hybrid FPU-RAU Computing

In order to keep the RAU small, but maintain acceptable accuracy, we assign a portion of input data to GPU FPUs to process. To support this functionality, our design compares inputs with minimum distance and, in the case of significant distance from the input key (>threshold), assigns them to precise FPUs to process. Fig. 7 shows the RAU activation running each application which ensures the quality of service (QL$<10\%$ for general application and QL$<2\%$ for machine learning). RAU activation defines as the portion of time that input data processes by RAU. As we expected, large RAU size increases computation accuracy. When using small RAU, the major part of application's computation need to be executed on precise FPUs. Fig. 7 shows the normalized energy consumption and execution time of enhanced GPGPU in different RAU sizes running five general GPGPU and three machine learning applications. In each RAU size, both energy and execution time have been normalized to energy and runtime of GPGPU without any RAU blocks. Our evaluation shows that in small RAU, increasing the RAU size positively impacts energy improvement and speed up of associative memory, because increasing the number of rows improves computation accuracy which allows us to assign fewer computations to FPUs.

However, in most applications, RAU larger than 32 rows degrades computation energy and performance, because beyond 32 rows the highly frequent patterns have already been stored in RAU and adding additional low frequency patterns cannot improve hit rate or accuracy significantly. Considering the average EDP improvement over all applications shows that using 32 rows RAU results in best EDP compared to other RAU sizes.

Table II shows the EDP improvement of different applications running on proposed enhanced GPGPU with 32 rows RAU block. We compare the efficiency of proposed design with state-of-the-art approximate GPGPU using approximate memristive associative memory (A^2M^2) [32] and resistive configurable associative memory (*ReCAM*) [19]. We use the best A^2M^2 and *ReCAM* configurations which result in maximum energy savings. In A^2M^2, we used 8-row TCAM under different voltage overscaling levels

TABLE II
EDP IMPROVEMENT AND QUALITY LOSS IN ENHANCED GPGPU WITH, A^2M^2, *ReCAM* AND PROPOSED RAU.

GPGPU		Sobel	Robert	Sharpen	Binomial	DwtHaar	Back Prop	KNN	K-means
A^2M^2 **[32]**	Hamming Distance	1-bit	2-bit	2-bit	1-bit	2-bit	0-bit	2-bit	1-bit
	EDP improvement	1.56×	1.69×	1.51×	1.19×	1.33×	1×	1.72×	1.65×
	QL (%)	8.3	9.2	8.4	6.1	8.9	NA	1.8	0.9
ReCAM **[19]**	Relaxed rows	20	24	20	12	28	16	24	24
	EDP improvement	2.43×	2.22×	1.92×	1.20×	2.15×	1.87×	2.01×	2.43×
	QL (%)	7.4	8.4	9.2	7.4	9.7	1.9	1.6	1.2
RAU	RAU activation	57%	67%	63%	51%	46%	89%	100%	99%
	EDP improvement	3.61×	5.12×	4.27×	2.95×	2.58×	5.14×	10.73×	11.41×
	QL (%)	6.1	8.9	6.4	9.1	7.4	0	0.9	0

to enable inexact matching with different Hamming distances. In ReCAM, the search computation in 32-row TCAM relaxes in selective rows (in 4-row granularity) to provide maximum energy savings. However, the performance of GPU enhanced by these two designs has been bounded by FPU pipeline stage. Therefore, they cannot provide any computation speedup. Our evaluation shows that accepting 10% and 2% quality loss for general and machine learning applications, the enhanced GPGPU with A^2M^2 and *ReCAM* provide 31% and 49% energy savings (1.45× and 2.02× EDP improvement) for eight tested applications. While GPGPU enhanced with the proposed RAU can provide 61% energy savings and 2.2× speed up (5.72× EDP improvement) compared to traditional GPGPU. At the same level of accuracy, EDP improvement of proposed design is 2.8× higher than state-of-the-art approximate GPGPU design [19].

We consider the overhead of RAU on the GPGPU architecture. As we pointed in Section III, in hybrid mode the RAU does not change the FPU clock frequency, thus our design would not not work slower than conventional GPGPU (with no RAU). In terms of area overhead, our evaluation shows that using 64-row RAU next to each FPU (which provides maximum efficiency) can result in 1.9% area overhead. This area overhead is negligible considering the energy and performance efficiency that our design can provide.

VI. CONCLUSION

In this paper, we propose a resistive associative unit, which can approximately perform memory-based computation instead of beside traditional processor cores. RAU models basic computations by storing their high frequency input patterns and searching for nearest distance input at runtime. RAU can tune the level of computation accuracy by adaptively finding low frequency data and assigning them to precise cores to process. Our evaluation shows that integrating RAU beside GPGPU floating point units results in 61% lower energy consumption and 2.2× speedup when compared to GPGPU, while also ensuring acceptable quality of service. In term of energy-delay product, proposed design achieves 5.7× and 2.8× improvement compare to traditional GPGPU and state-of-the-art approximate GPGPU, respectively.

VII. ACKNOWLEDGMENT

This work was supported by NSF grants #1730158 and #1527034.

REFERENCES

[1] J. Gubbi *et al.*, "Internet of things (iot): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
[2] D. Miorandi *et al.*, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.
[3] B. Yao *et al.*, "Multifractal analysis of image profiles for the characterization and detection of defects in additive manufacturing," *Journal of Manufacturing Science and Engineering*, 2017.
[4] S. Ghoreishi *et al.*, "Adaptive uncertainty propagation for coupled multidisciplinary systems," *AIAA Journal*, pp. 1–11, 2017.
[5] M. Imani *et al.*, "Maximum-likelihood adaptive filter for partially observed boolean dynamical systems," *ITSP*, vol. 65, no. 2, pp. 359–371, 2017.
[6] M. Imani *et al.*, "Low-power sparse hyperdimensional encoder for language recognition," *IEEE Design & Test*, vol. 34, no. 6, pp. 94–101, 2017.
[7] Z. Vasicek *et al.*, "Evolutionary approach to approximate digital circuits design," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 432–444, 2015.
[8] S. Venkataramani *et al.*, "Approximate computing and the quest for computing efficiency," in *Proceedings of the 52nd Annual Design Automation Conference*, p. 120, ACM, 2015.
[9] M. Imani *et al.*, "Voicehd: Hyperdimensional computing for efficient speech recognition," 2017.
[10] M. Imani *et al.*, "Nngine: Ultra-efficient nearest neighbor accelerator based on in-memory computing," 2017.
[11] V. Gupta *et al.*, "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.
[12] T. Kohonen, *Associative memory: A system-theoretical approach*, vol. 17. Springer Science & Business Media, 2012.
[13] M. Imani *et al.*, "Exploring hyperdimensional associative memory," in *HPCA*, pp. 445–456, IEEE, 2017.
[14] Y. Kim *et al.*, "Orchard: Visual object recognition accelerator based on approximate in-memory processing," in *ICCAD*, 2017.
[15] M. S. Riazi *et al.*, "Camsure: Secure content-addressable memory for approximate search,"
[16] M. Imani *et al.*, "Efficient query processing in crossbar memory," in *ISLPED*, pp. 1–6, IEEE, 2017.
[17] M. N. Bojnordi *et al.*, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*, pp. 1–13, IEEE, 2016.
[18] S. M. Seyedzadeh *et al.*, "Pres: Pseudo-random encoding scheme to increase the bit flip reduction in the memory," in *DAC*, pp. 1–6, IEEE, 2015.
[19] M. Imani *et al.*, "Resistive configurable associative memory for approximate computing," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1327–1332, IEEE, 2016.
[20] M. Imani *et al.*, "Approximate computing using multiple-access single-charge associative memory," *TETC*, 2016.
[21] M. Imani *et al.*, "Multi-stage tunable approximate search in resistive associative memory," *TMSCS*, 2017.
[22] M. Imani *et al.*, "Masc: Ultra-low energy multiple-access single-charge tcam for approximate computing," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 373–378, IEEE, 2016.
[23] M. Imani *et al.*, "Processing acceleration with resistive memory-based computation," in *MEMSYS*, pp. 208–210, ACM, 2016.
[24] M. Imani *et al.*, "Acam: Approximate computing based on adaptive associative memory with online learning.," in *ISLPED*, pp. 162–167, 2016.
[25] J. Sim *et al.*, "Enabling efficient system design using vertical nanowire transistor current mode logic,"
[26] M. Imani *et al.*, "Remam: low energy resistive multi-stage associative memory for energy efficient computing," in *ISQED*, pp. 101–106, IEEE, 2016.
[27] J.-M. Arnau *et al.*, "Eliminating redundant fragment shader executions on a mobile gpu via hardware memoization," in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 529–540, IEEE, 2014.
[28] A. Goel *et al.*, "Small subset queries and bloom filters using ternary associative memories, with applications," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 1, pp. 143–154, 2010.
[29] Y. Kim *et al.*, "Cause: critical application usage-aware memory system using non-volatile memory for mobile devices," in *ICCAD*, pp. 690–696, IEEE, 2015.
[30] M. V. Beigi *et al.*, "Tesla: Using microfluidics to thermally stabilize 3d stacked stt-ram caches," in *Computer Design (ICCD), 2016 IEEE 34th International Conference on*, pp. 344–347, IEEE, 2016.
[31] C. J. Xue *et al.*, "Emerging non-volatile memories: opportunities and challenges," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pp. 325–334, 2011.
[32] A. Rahimi *et al.*, "Approximate associative memristive memory for energy-efficient gpus," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1497–1502, IEEE, 2015.
[33] M. Imani *et al.*, "Resistive cam acceleration for tunable approximate computing," *IEEE Transactions on Emerging Topics in Computing*, 2017.
[34] M. Imani *et al.*, "Nvalt: Non-volatile approximate lookup table for gpu acceleration," *Embedded Systems Letters*, 2017.
[35] H. Esmaeilzadeh *et al.*, "Neural acceleration for general-purpose approximate programs," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 449–460, IEEE Computer Society, 2012.
[36] S. Kvatinsky *et al.*, "Vteam: A general model for voltage-controlled memristors," *TCAS II*, vol. 62, no. 8, pp. 786–790, 2015.
[37] R. Ubal *et al.*, "Multi2sim: a simulation framework for cpu-gpu computing," in *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pp. 335–344, ACM, 2012.