A Hybrid Approximate Computing Approach for Associative In-Memory Processors

Hasan Erdem Yantır, *Student Member, IEEE*, Ahmed M. Eltawil, *Senior Member, IEEE*, and Fadi J. Kurdahi, *Fellow, IEEE*

***Abstract*— The complexity of the computational problems is rising faster than the computational platforms' capabilities which are also becoming increasingly costly to operate due to their increased need for energy. This forces researchers to find alternative paradigms and methods for efficient computing. One promising paradigm is accelerating compute-intensive kernels using in-memory computing accelerators, where data movements are significantly reduced. Another increasingly popular method for improving energy efficiency is approximate computing. In this paper, we propose a methodology for efficient approximate in-memory computing. To maximize energy savings for a given approximation constraints, a hybrid approach is presented combining both voltage and precision scaling. This can be applied to an associative memory-based architecture that can be implemented today using CMOS memories (SRAM) but can be seamlessly scaled to emerging ReRAM-based memory technology later with minimal effort. For the evaluation of the proposed methodology, a diverse set of domains is covered, such as image processing, machine learning, machine vision, and digital signal processing. When compared to full-precision, unscaled implementations, average energy savings of 5.17× and 59.11×, and speedups of 2.1× and 3.24× in SRAM-based and ReRAM-based architectures, respectively, are reported.**

***Index Terms*— Memristor, in-memory computing, resistive associative processor (RAP), approximate computing, memristance scaling, approximate computing, approximate storage, architecture.**

I. Introduction

OVER the last decade, there has seen a sharp increase in the need for ultra-efficient computation platforms that necessitates robust, low-power processing cores. This need becomes more urgent due to the increased need for scaling computation to tackle key computations such as deep learning, artificial intelligence and their tremendous requirement of efficient big data processing. However, traditional computing architectures are approaching their limits in terms of scalability and power consumption [1] and cannot satisfy these requirements. On the other hand, research community speculates that CMOS scaling could end in around 2024 [2], making it unlikely that further area, performance, and energy improvements would be purely based on fabrication technology. To cope with these standing barriers facing the future of computing, one must look into other means of improving the efficiency of computation by increasing parallelism rather than depending on transistor feature reduction [3]. However, this approach becomes insufficient if processing elements cannot be fed by the memory at the desired processing rate, leading to a significantly degraded overall performance despite the advancement in the process technology and the parallelism. To address these processing requirements of the ever-increasing amount of information, new computing paradigms started to emerge that focus more on the memory bottleneck problem together with the emerging semiconductor technologies. Broadly, these architectures aim to accelerate operations directly in-memory or near-memory to eliminate the data movement costs.

Theoretically, the barrier of memory bottleneck can be overcome by processing the data directly inside the memory without moving it back and forth between memory and processor. The research on in-memory computing paradigms aims at this target [4]. The paradigm proposes replacing the logic with memory structures, virtually eliminating the need for memory load/store operations during computation. *Associative processing* (AP) is a very promising computational paradigm for in-memory computing which had been proposed in 1970s [5], [6] and has been recently attracting attention due to the advance in the semiconductor and scaling technology. *Associative processors* (AP) process the data inside the memory in a fashion similar to Single Instruction Multiple Data (SIMD) processors. Each row of the AP behaves as a simple processor and an operation is done on all memory rows (words) as parallel. Therefore, the execution time of an operation on the AP does not depend on the vector size as long as they fit into the memory [7]. This opportunity largely overcomes the memory-wall problem of traditional computing architectures (i.e., Von Neumann architectures) since the memory and the processor are combined so there is no memory bottleneck. Many parallel systems are uniquely suited to this approach due to the vector-based nature of their processing pipelines. In an analogy, APs can be considered as a next step on the path of the CPU (central processing unit) to GPU (graphical processing unit) transformation. When compared with CPUs, GPUs have simpler processing cores, however, they are throughput optimized on the contrary of CPUs which

Manuscript received January 7, 2018; revised May 11, 2018; accepted June 21, 2018. Date of publication July 3, 2018; date of current version December 11, 2018. This paper was recommended by Guest Editor V. De. *(Corresponding author: Hasan Erdem Yantır.)*

The authors are with the Center for Embedded and Cyber-physical Systems, University of California at Irvine, Irvine, CA 92697 USA (e-mail: hyantir@uci.edu; aeltawil@uci.edu; kurdahi@uci.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JETCAS.2018.2852701

are latency optimized. When compared GPUs, APs have much simpler cores (i.e., a single memory row), so its throughput becomes more than GPUs since more cores can be placed in a chip. Another distinct advantage of APs is that the architecture can almost seamlessly evolve from today's SRAM implementation to tomorrow's emerging storage technologies, such as Resistive RAM (ReRAM)[1] by simply transplanting the memory cells to achieve higher scalability and performance while maintaining the same top level system architecture and the corresponding investment in software.

Approximate computing is another computing paradigm that relaxes the correctness constraints of a system for the sake of energy and performance improvement [8], [9]. Approximate computing is applied to the applications that can tailor to some error at the output. These applications generally consist of signal processing and multimedia applications. These applications commonly have a SIMD like computation pattern in which APs can be a suitable candidate. More than that, APs supports the approximate computing inherently, so they are a very promising employment platform for the approximate computing [10]. In this study, we investigate the combination of two approximate computing methods in APs to propose a hybrid approach for approximate in-memory computing. The contribution of the study to the research community can be summarized as follows:

- The hybrid approximate computing technique for APs is introduced to decrease the energy consumption and increase the performance. The approach combines the two different approximate computing approaches from system and circuit levels.
- A circuit level approximate computing based on cell scaling for SRAM-based APs is introduced as an alternative approach to memristance scaling in ReRAM-based APs.
- A design-time method is proposed to find the optimal approximation degree in APs for both ReRAM and SRAM based APs.
- Lastly, the study proposes an architectural extension to the APs to tune the approximation degree of APs dynamically during the runtime of the applications.

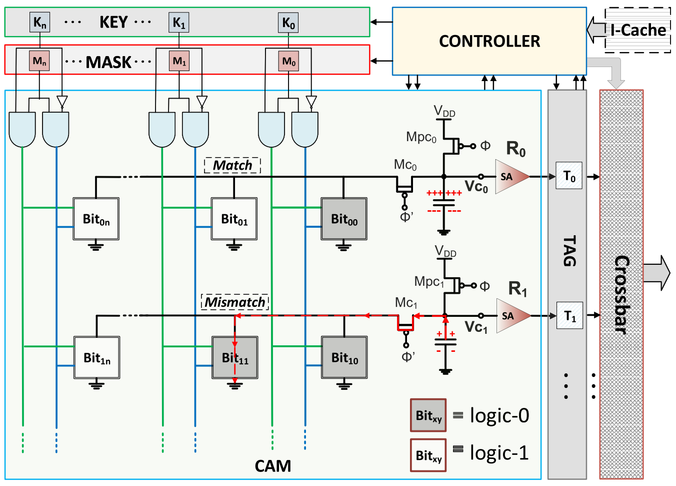
The rest of the paper is organized as follows: In the following section, background and previous studies on associative computing and approximate computing are presented. Section III introduces the hybrid approximation in APs in details. The section also proposes a design-time method for finding the optimal degree of approximation. Experimentation and evaluation results are discussed in Section IV together with the exploited experimentation framework. The final section concludes the work.

II. Background

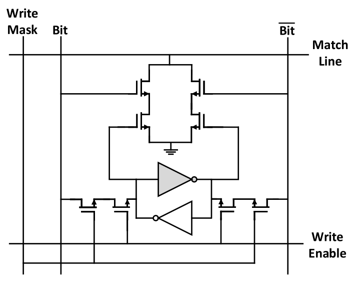
A. Associative Processors

Presented in Figure 1a, the architecture of an associative processor (AP) originated in the seventies and eighties [5] and was elaborated upon in recent literature [117]. AP comprises

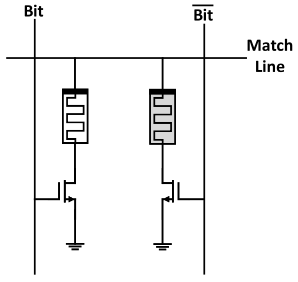
[1]In this paper, we use the terms ReRAM and memristor interchangeably.



(a)



(b)



(c)

Fig. 1. Associative processor architecture and cell candidates. (a) Architecture. (b) SRAM-based CAM Cell. (c) ReRAM-based CAM Cell.

a content addressable memory (CAM), controller, instruction cache, interconnection circuit, and specific registers. The controller generates the required mask and key values for each corresponding instruction. The interconnection matrix (crossbar) is a basic circuit switched matrix that allows rows of the AP to communicate in parallel with other APs (in a pipelined fashion) or within the same AP either while shuffling the data between AP rows. This communication can be either bitwise or wordwise. The key register is used to present the value that is written or compared against. The mask register indicates which bit or bits are activated during comparison or write. The rows matched by the compare operation are marked in the tag field where the rows tagged with logic-1 means that the corresponding CAM row has been matched with the given key and mask value. For example, if we send key 100 and mask 101 to the CAM, the tag bits of the corresponding rows whose first and third bits are logic-0 and logic-1 respectively becomes logic-1.

Utilizing consecutive compare and write cycles,it is possible to perform parallel operations on the AP. Referring to Figure 1a, the controller successively fetches a sequence of (key, mask) pairs from the I-cache and writes those values to the CAM, triggering a corresponding sequence of parallel operation along all the CAM rows. In other words, instead of reading and sending the data over functions (e.g., reading two values from the memory and sending it to CPU for addition), the functions (i.e., look-up tables) are sent over the data

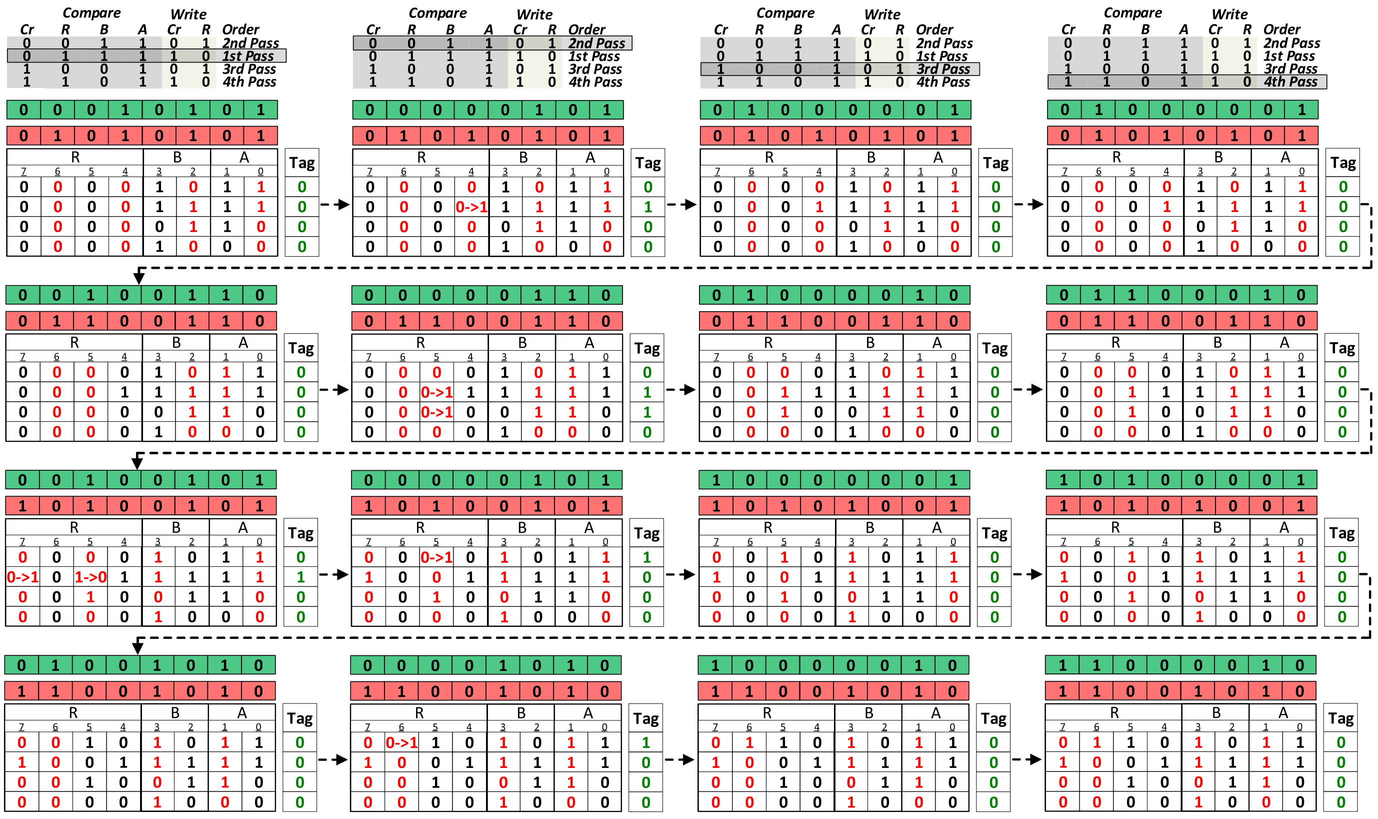


Fig. 2. Vector multiplication operation. The sequence of compare and write operations are shown for a complete unsigned vector multiplication.

(e.g., LUT for the addition are applied on data residing in the memory). As an example, Figure 2 illustrates the complete flow for the unsigned multiplication of two 1×4 2-bit vectors, A and B, i.e. $R[i] \leftarrow B[i] \times A[i], i = 0\ldots3$ where the tables in the first row corresponds to the LUT for the unsigned multiplication. In this operation, B is added to R if the corresponding A bit is 1. The other tables show the progress in the CAM contents together with the key/mask registers (in green and red respectively) and the tag status. The pseudo-code in Algorithm 1 shows the controller flow for the unsigned multiplication operation where the outer loop scans each bit of A and the inner loop scans bits of B. The multiplication is done in a bit-serial, word-parallel manner. At the beginning of each cycle, mask value are set to the current bit locations and the key value to the corresponding LUT entry. Then compare operation is performed by the key value in the columns specified by the mask value. Lastly, the write operations are performed on the matched rows of the CAM. To perform the operation, *Multiplication(R, B, A)* instruction is executed where the parameters corresponds to the column numbers of the corresponding vectors (e.g., A is columns between 1-0). Initially, A contains (i.e., bits 1-0) the values of [3; 3; 2; 0] and B (i.e., bits 3-2) contains the values of [2; 3; 1; 2] as shown in the upper-left (initial) CAM. The result columns (R) are zero as initial. In each CAM of the figure, the key value from the compare column of the LUT is applied sequentially to each bit position and is compared simultaneously against all vector elements. The highlighted entry in the LUT shows the applied key on the corresponding bit. The red entries in the CAMs correspond to the masked columns on which the LUT entry is applied. In the result columns (R), there are two active columns at a time where one is the carry bit (Cr) and the other one is the result bit (R.k). After each comparison, the matching rows are tagged with 1. This will enable the write operation which writes the corresponding LUT entry (shown in the write column of the LUT) only to the unmasked cells in the rows that match the key. The arrows of the figure specify the flow of the operation. Each row of the figure shows a partial multiplication operation between the single bits of A and B, and the partial addition of the result (e.g., the second row shows the multiplication between the second bits of A and the first bits of B together with its addition with the previous result came from the first row). For each bit of A, position of the carry (Cr) is changed in accordance with the current partial addition. The process is repeated for all the passes in the prescribed order shown in Figure 2. Finally, the value stored in R becomes [6; 9; 2; 0] which is equal to $B\times A$ (i.e., $[2\times3; 3\times3; 1\times2; 2\times0]$). Thus multiplying two vectors that are 2-bit wide takes 16 steps ($4m$ compares and $6m$ writes where m is bit-width), independently of the vector size. The study in [11] also depicts the simpler addition operation on the AP.

Irrespective of the technology or architecture, a CAM employed in APs must have the features of locating a row that has a given data (compare) and writing to the specified columns of the given rows (write). The compare operation is basically done by a pre-charge capacitor assigned to each row. First, the capacitor is pre-charged, and then a searched key is applied to the masked columns of the CAM array. If there is a mismatch, the capacitor leaks the charge due to a low-resistive path formed as a result of a mismatch in at least one cell. If there is a match, the charge leaks very slowly due to the high resistance path. At the end, the remaining voltage across the capacitor is sensed by a sense amplifier. If it is more than a reference value (V_{ref}), the sense amplifier (SA) outputs logic-1 and logic-0 if it is less than. Even though these features are same for a CAM, the cell implementations can be different. The two popular implementation of CAM cells is ReRAM-based or SRAM-based implementations. In ReRAM-based cell (Figure 1c), the data is stored in two memristors with its reverse [12]. To store logic-1, the right and left memristors are set to high and low memristances respectively or vice versa for logic-0. This cell structure complies with the requirements of the associative computing and exists as a commercial product [13], [14]. Even though there are endurance issues about the ReRAM-cells, there are some studies addressing the issue. For example, [15] mentions about some techniques about the endurance improvement by adding a serial resistance to the memristor. In SRAM-based cell (Figure 1b), a coupled inverter stores the data with its inverse. The AP using memristor-based CAM cells is named as *ReRAM-based Associative Processor (RAP)* and SRAM-based one as *SRAM-based Associative Processors (SAP)*. Alternatively, RAP can be named as the MAP (memristive associative processor) also [10].

Algorithm 1 Unsigned Multiplication ($\mathbf{R}=\mathbf{B}\times\mathbf{A}$) in the AP

```
function Multiplication(R, B, A)
  for i = B.LSB to B.MSB
    Set Cr
    for j = A.LSB to A.MSB
      Set R.k
      for pass = 1 to 4
        SetMask(Cr, R, B.i, A.j);
        SetKey(LUT_Mult(pass), Cr, R.k, B.i,
        A.j);
        Compare(Mask, Key);
        Write(LUT_Mult(pass), Cr, R.k);
      end
    end
  end
end
```

B. Approximate Computing

Approximate computing is a computing paradigm where the accuracy of the applications are sacrificed for the sake of performance, speed up, and area improvements [8]. In fact, all applications in the computer systems work as approximate (or not exact) since the computing capability is limited by the precision (bit width) and the data representation. However, approximate computing targets to use less precision than what the computer system supports. Approximate computing recently became more popular because of the dark silicon area. While high accuracy is crucial for some applications, many tasks are suited for this approach since the perfect output is not a must. For example, the applications such as signal processing, machine learning, multimedia, computer vision, etc. are intrinsically error resilient. As an example, the approximate computing can be employed in image processing algorithm and the results may not harm the taste of vision since human perception system is limited. To illustrate, Figure 3 shows two outputs from the Sobel filter [16] which is an edge detection algorithm. Figure 3a shows the result of the filtered image when all pixels are represented in their full range (i.e., 8 bits integer). On the other hand, Figure 3b shows the result of the same algorithm where the pixels of the same input image is approximated by trimming their last two least significant bits (i.e., each computation is performed on 6-bit). The figure shows that the approximately processed image exhibits a peak signal-to-noise ratio (PSNR) of 33.17 dB with an average error of 6.20%. It can be visually inferred that those two figures are almost indistinguishable. However, the approximate system allows the utilization of approximate adders and multipliers which performs the operations on fewer bits, therefore provides energy reduction and performance improvement.

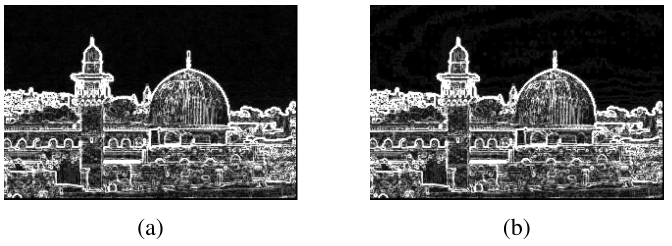


Fig. 3. Full (a) and approximate (b) outputs of the sobel filter algorithm on a image where PSNR is 33.17 dB for the approximate image.

Approximate computing is a broad term and there are many approximate computing studies in the literature [8]. These studies are generally classified according to the level of the approximation; logic-circuit level and algorithm-architecture levels [17]. In the circuit level, the most common method is designing functionally equivalent but approximate circuits for the arithmetic functions. These circuits can consume less power and provides speedup and area savings [18]. Voltage over-scaling in the circuits is also an example of approximate computing where the components are forced to run on the boundary conditions when error resiliency is sacrificed. The scaling can be done on voltage or clocks (timing) [19]. In this level, approximate circuits can be synthesized by design tools that are able to replace the functional units with its corresponding approximate one [20]. These approximate circuits are employed the error resilient parts of the whole computational system. Algorithm-architecture level approximation aims to militate in favor of the more significant components of a system over less important ones while still satisfying the target metric (e.g., performance, energy budget, area) [21]. As an example case, a computer program can ignore some of the iteration in the less important loops or skip some memory accesses in a less important part of the execution.

Approximate in-memory computing is another research area in which the computation itself done in the memory as approximate. The study in [10] proposes the approximate in-memory computing in RAPs. The proposed two methods are bit-trimming where some least significant bits are truncated and the memristance scaling where the memristors that stores the least significant bits are scaled to keep the values approximately. In [22], an architectural RCAM extension to the GPUs is proposed which keeps the results of the frequently executed operations. This block saves the GPU from re-executions. The cache-like extension and the approximate matching in the RCAM provide the speedup and energy savings with the expense of an acceptable decrease in quality. The same idea is improved in [23] and [24] by enabling the different degree of approximation for the rows and bit columns and adding single-charge multiple-access RCAMs respectively. In [24], the resistive CAM accelerator approach is proposed for tunable CPU approximation. In [25], an approximate in-memory hamming distance calculation is proposed based on a memristive associative memory. The study in [26] performs the two important subtasks of a visual recognition system as approximate inside the memory to get an advantage in energy and performance. In [27], the approximate storage of images in STT-RAM by writing the data approximately and then managing the memory access adaptively with respect to write mode (approximate vs precise). In all these studies except [10], the computation is done inside the memory partially and the remaining parts are done outside. This study aims to do all computation inside the memory by using associative processing on the CAM structures.

III. Methodology

A. Motivation

We identify two ways to perform approximate computing in APs; *bit trimming (BT)* and *cell voltage scaling (CS)* [10]. In bit trimming, one or more least significant bits of a number are disregarded and the operations start from the relatively more significant bits since the operations on the AP are done bit by bit from the least significant bit (LSB) through the most significant bit (MSB). In this way, the accuracy of a value in the AP is approximated by ignoring its less significant bits. This situation provides both energy saving and the speedup since the operations require fewer cycles (both compare and write). The second method introduced previously is cell scaling. Although cell scaling can be used to denote scaling of many parameters (e.g., voltage, value range, area, etc.), in here, it corresponds to voltage scaling. This method can be applied to the ReRAM-based APs by scaling the R_{on} and R_{off} values of the ReRAMs in the scaled RCAM columns. In this case, the ReRAMs in the scaled columns are switched within a narrower resistance range rather than the full range. This method necessities low write voltages (V_{WR}) together with shorter write pulses. In turn, it provides both energy savings and speedup. However, a compare operation on the columns including these scaled columns results in less robust matching and likely errors at the output of the sense amplifier. Similar to RAP cells, the SAP cells can be approximated by lowering their supply voltages, thus providing energy savings. The decrease in the supply voltage results in a decrease in the write energy and static energy. On the other hand, the cells whose supply voltage is less than the nominal voltage can introduce errors during the operation, just like the RAP case.

Figure 4a shows the accuracy lost in an 1K, 16-bit FFT algorithm simulated on the environment described in Section IV-A. The figure shows how bit-trimming and cell scaling affect the accuracy in both RAPs and SAPs. The x-axis of the figure shows the number of trimmed or scaled bits and the y-axis shows the resulting accuracy in percentage where left-y axis shows the result for BT and right-y axis shows the result for cell scaling. The figure proves that the trimming single bit introduces more error to the system than the scaling same bit. When compared with the cell scaling (both ReRAM and SRAM scalings), the bit trimming leads to coarser grain approximation since bit trimming discards the column totally. On the other hand, the cell scaling still contributes to the operation somehow correctly or incorrectly in a probabilistic manner, so it becomes a finer grain approximation than the bit trimming. For instance, even though trimming 4-bit leads to 15% error in the result, scaling the same bits result in 1% and 1.3% error for memristance and SAP-cell scalings respectively. From the other point of view, given a quality expectation, the number of scaled bits can be more than the number of trimmed bits. Figure 4b shows the similar figure for energy reduction in left-y axis and speedup in right-y axis. The figure depicts that bit trimming provides more energy reduction and speedup when compared with cell scaling. As an example case, as trimming the LSB bit in SAP results in 15% energy savings and 12% speedup, scaling same bit provides 1% energy reduction and 2% speedup.

From Figure 4a, it can be inferred that bit-trimming provides more speedup and energy at the expense of more decrease in the accuracy. On the other hand, cell scaling exists as a more robust approximation technique than bit trimming. In approximate computing systems, the main goal is getting the optimum performance with the attainable highest acceptable quality. For this purpose, bit-trimming in APs can be more reasonable when applied to the LSBs which have less effect on the computational accuracy than the MSBs. On the other hand, cell-scaling can be done to tune the degree of approximation (in a finer grain manner) in the remaining bits after applying bit-trimming on some LSBs. Even though cell scaling provides a finer grain approximation with much lower improvements in performance and energy, this method can be very useful for such cases where the bit-trimming methods decrease the overall quality lower than the expectations. As a combination of these two techniques, the proposed hybrid approximate computing approach targets to find the optimum degree of approximation in APs with respect to a given quality metric by efficiently exploiting both methods.

Figure 5 shows the proposed hybrid approximation technique on SAPs (Figure 5a) and RAPs (Figure 5b) respectively. In both APs, the bit-trimming is done by disregarding the LSBs from the computation. In this way, the computation in the APs starts from relatively more significant bits. In RAPs, the trimmed bits do not cause an additional overhead since ReRAMs are non-volatile storage. Specifically, in SAPs, the supply voltage of the trimmed columns can also be cut off in order not to cause excessive static energy consumption. After trimming some LSBs, the cell scaling is applied on the number of columns which has relatively higher significance. However, cell scaling introduces less error to the system than bit-trimming, so the system can still get more advantage from

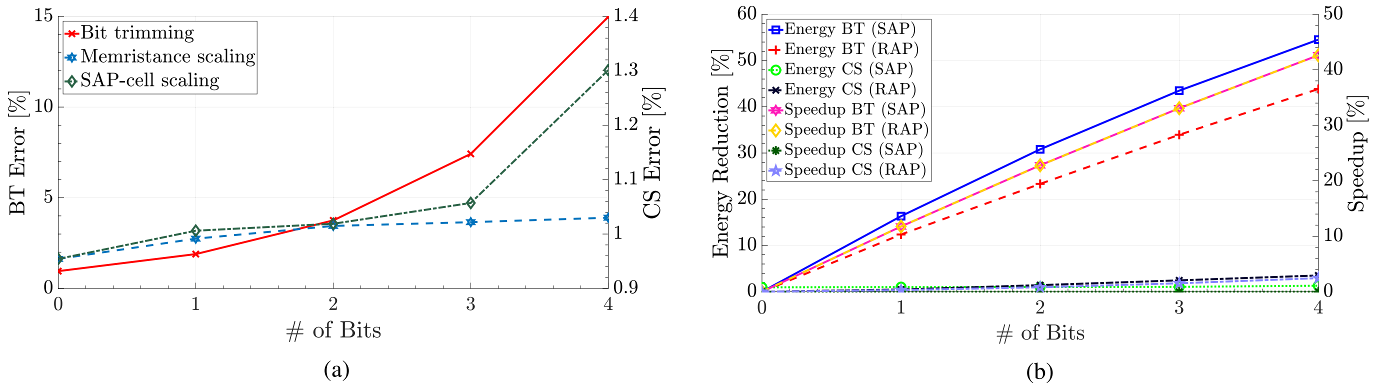


Fig. 4. The effect of bit trimming and cell scaling on SAPs and RAPs. (a) Error vs approximation methods. (b) Energy reduction and speedup vs approximation methods.

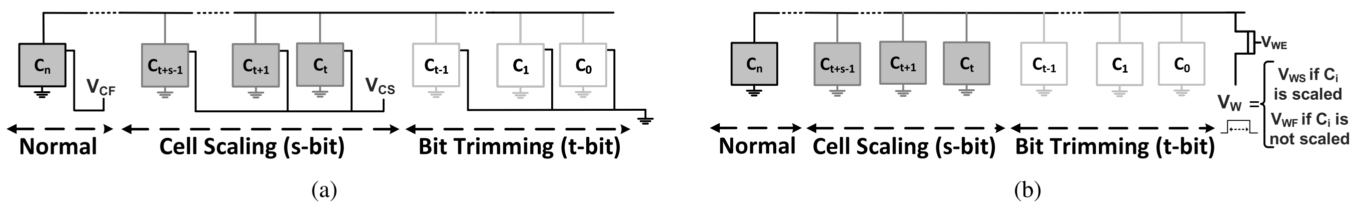


Fig. 5. Approximate computing in associative processors (APs). (a) Approximate computing in SAP. (b) Approximate computing in RAP.

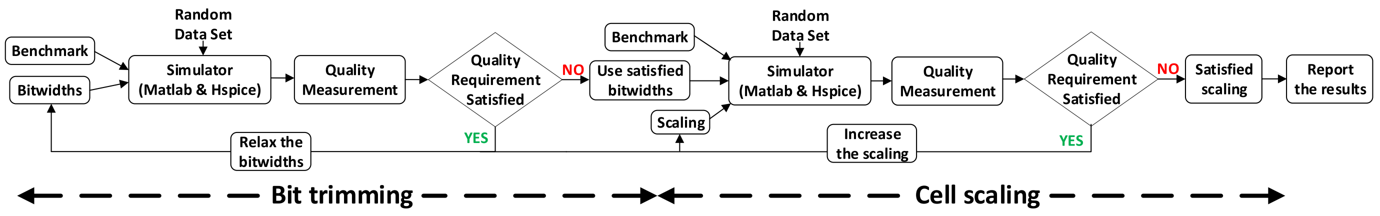


Fig. 6. The design flow for approximate AP systems.

the approximate computing still by complying with the quality metrics specific to the application. In SAPs, the scaling is done by applying a scaled voltage (V_{CS}) to the scaled cells that is less than the full cell voltage (V_{CF}), i.e., $V_{CS} < V_{CF}$. In RAPs, the scaling is performed by changing the write voltage where V_{WF} is the write voltage for the full cells and V_{WS} is the write voltage for the scaled cells. For example, if cell-0 (C_0) is scaled, the V_{WS} voltage is applied to write to the corresponding memristor. Apart from the voltage levels, the pulse widths are different for each cases since memristance scaling decreases the write time.

B. Design Flow

Deciding on the number of trimmed bits and scaled columns is a design parameter and depends on the application and its requirements. The design flow shown in Figure 6 demonstrates a process to select the optimum configuration. The main item in the flow is an in-house simulator developed specifically for both APs. As input, the simulator accepts the benchmark (instructions), system/circuit parameters (bit width, variations, scaling, transistor/memristor models, etc.), and input data. The simulator consists of two simulation engines; system and circuit level simulators. The system level simulator performs the simulation on Matlab while circuit level one drives the HSPICE. Both engines run as cycle accurate. The simulator outputs the energy consumption, accuracy (quality), and timing results of the simulated application.

The design flow consists of two phases; configuring the bit trimming, and configuring the cell scaling. Since bit trimming can be considered as a coarser grain approximation when compared with the scaling, the flow firstly targets to find the optimum bit width approximation in the first phase, and then apply optimum scaling on top of bit trimming approximation in the second phase. First, both approximation parameters (bit widths and scaling) are set to the most relaxed case so that the system performs the operation on the full range without any scaling. After the simulation, the quality is assessed for at least n runs on different random input sets (we use $n = 10$ in our experiments). If the quality is satisfied, the bit width constraint is relaxed (more bits is trimmed) and a new simulation is done with the new constraints. This loop continues until finding a bit trimming case which does not satisfy the quality requirements. After finding this case, the parameters set to the optimum bit widths (the preceding one) and the second phase (cell scaling) starts. During the cell scaling, the same flow is performed by relaxing the scaling parameters (i.e., the number of the scaled columns). The key point in this flow is setting the quality

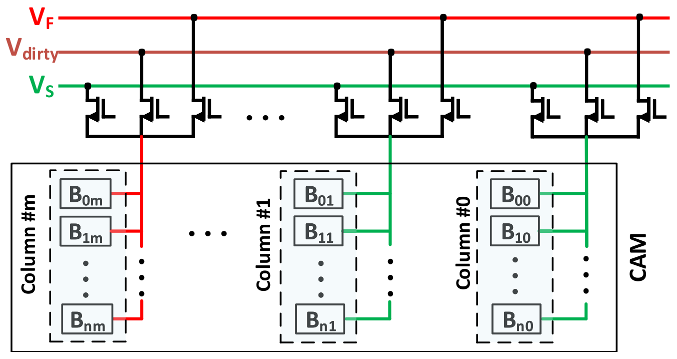


Fig. 7. Dynamic cell scaling in APs.

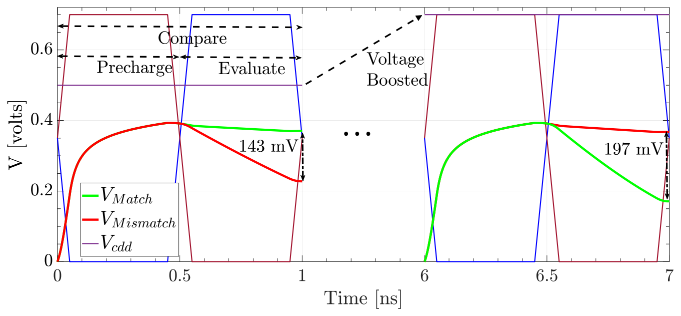


Fig. 8. Dynamic approximate computing in SAPs.

requirement which is a configurable parameter depends on the application itself. After the optimum scaling parameters found within the given degree of bit trimming, the results are reported in the last stage of the flow. The results show a map to indicate which bits are trimmed and which ones are scaled.

C. Dynamic Approximation

Both the cell scaling and the bit trimming can be done on LSBs and can be defined in the design time by using the explained design flow. Moreover, since the starting point of the operations can be modified on the fly by the controller, approximation of the bit-trimming can be done dynamically. For bit-trimming, the controller simply shifts the starting bit position of the operands. For example, to trim a single bit from the operand A in Figure 2, the controller runs the instruction of *Multiplication([7, 5], [3, 2], [1, 1])* instead of *Multiplication([7, 5], [3, 2], [1, 0])*, that is, the operands can be changed dynamically by the controller. Additionally, the degree of scaling can be controlled by a circuit during runtime by using adaptive supply voltages. For example, the method in [28] proposes a fast technique to change the voltage supplies. Figure 7 shows the general methodology of this technique where each column can be supported by a different voltage source. The controller unit of the APs decides on the cell supply voltage of the SAP cells and the write voltages of the RAP cells for writing logic-0 and logic-1. The applied voltage pulses also differ in RAP cells. Even though there are more than two voltage levels with different polarities for the memristor scaling, only one voltage control circuit is required per column. Therefore, the additional overhead is minimal and largely amortized over the thousands of rows comprising the AP. In the figure, it is shown that while column-0 and column-1 are supplied at the scaled voltage V_S (V_{CS} in SAPs and V_{WS} in RAPs), the last column (i.e., the most significant one) is supplied by full voltage V_F (V_{CF} in SAPs and V_{WF} in RAPs). Accordingly, the proposed hybrid approximation method in the APs is also dynamically tunable for both the bit trimming and the cell scaling. Therefore, at any time the system can increase the accuracy by shifting the starting bit of the operations to the right (i.e. trimming fewer LSBs) at the expense of increased operation delay and energy. This can be partially counterbalanced by changing the degree of cell scaling. Depending on the characteristics, resiliency, tolerance, and environmental conditions of the application, the system accuracy can be set by tuning these two approximation parameters dynamically.

Figure 8 shows an example waveform of dynamic voltage scaling by using the short-stop technique in [28]. The figure shows the two compare cycles on four bits that belongs to the multiplication operation in Figure 2. In the first cycle (0 ns - 1 ns), three of the columns are supplied with the full voltage (V_{CF}) and the least significant one does with scaled voltage (V_{CS}). As a result, the noise margin between the match and mismatch operation is resulted as 143 mV. At some point, the system (or controller specifically) decides to change the degree of approximation by increasing the accuracy of the scaled bit in the expense of more energy consumption. During this change, the supply voltage of these columns are boosted from 0.5 V to 0.7 V. In [29], the time required to boost voltage takes less than 5 ns. In the second compare operation on the same bit (between 6 ns - 7 ns), the system results in a more robust operation with a noise margin of 197 mV. At any time, this voltage can be lowered for the sake of less power consumption. The same voltage boosting technique can be applied to RAP systems but to change the write voltage to set the memristance range.

TABLE I
THE EVALUATED BENCHMARKS, THEIR PLATFORMS, AND QUALITY METRICS FROM [30]

Benchmark	Domain	Bit Size	Quality Metric
Sobel Filter (sf)	Image Processing	p8 (12)	Image Diff.
FIR (fir)	Signal Processing	d8 & c8 (17)	Avg. Relative Error
FFT (fft)	Signal Processing	d16 & e16 (30)	Avg. Relative Error
Image Binarization (ib)	Image Processing	p8 (9)	Image Diff.
Convolution (conv)	Machine Learning	d24 & e16 (43)	Avg. Relative Error
FastWalsh (fwht)	Signal Processing	p17 (35)	Image Diff.
Mean Filter (mf)	Machine Vision	p8 & c16 (28)	Image Diff.
CNN	Deep Learning	d8 & k8 (18)	Avg. Relative Error

*** d:data, p:pixel, e:twiddle factor, c:coefficient, k:kernel, (): intermediate bit width**

IV. EXPERIMENTATION

A. Experimental Setup

For the evaluation of the hybrid approximate computing in the APs, energy reduction, speedup, and energy×speedup results for 7 benchmarks from AX-BENCH (approximate computing benchmark) tool [30] presented in Table I. The table also presents the number of input bits of the benchmarks.

TABLE II
CELL SCALING IN AP FOR BOTH SAP AND RAP (a) SAP CELL SCALING CASES (b) RAP CELL SCALING CASES

(a)

Case	$\mathbf{V_{cdd}}$	$\mathbf{P_{static}}$	$\mathbf{E_{write}}$	$\mathbf{P_e}(avg)$
Full	0.7 V	0.52 µW	0.24 fJ	0
Approximate	0.5 V	4.66 nW	0.06 fJ	0.021

(b)

Case	$\mathbf{R_{on}}$ (Ω)	$\mathbf{R_{off}}$ (Ω)	$\mathbf{T_{write}}$	$\mathbf{E_{write}}$	$\mathbf{P_e}(avg)$
Full	100	100 k	2 ns	21.7 pJ	0
Normal	1.7 k	79 k	1 ns	349.6 fJ	0
Approximate	3.2 k	40 k	0.5 ns	121.8 fJ	0.027

In some cases, the benchmark accepts more than one input with different bit widths. For this reason, the type of the input is specified with a letter (e.g., d for data and e for twiddle factor in FFT). Even though the AP accepts these bit sizes as input, during the computation the bit widths of the intermediate results can be more (e.g., multiplication of two m-bit numbers results in a 2m-bit number). The maximum bit widths of the intermediate results are specified within the parentheses. In the system level and circuit level simulations, the cycle accurate simulator in [10] is used. For the transistor in the circuit level simulation, Predictive Technology Models (PTM) from [31] are used to simulate high-density APs with sub 20nm feature sizes [32]. In the RAPs, we adopt a fabricated nano-second switching time memristor which has a size of 50 nm [33]. In the simulations, its corresponding SPICE model in [34] is used. For the sense amplifier, a low-power, sub-ns amplifier design in [35] is employed in the circuit. For both APs, the pre-charge voltage is set to 0.5v. Since the AP operation outputs more mismatches than the matches statistically, the V_{ref} is selected to make the mismatch error rate as low as possible [36]. In all simulations and comparisons with other architectures, the maximum quality degradation is set as less than 10% (based on the metric in [30]).

Approximate computing forces a system to work on the boundary conditions. Under these conditions, the factors deemed to safe under normal conditions endanger the correctness of the system. For example, a scaled SRAM-cell can continue normal operation without any error in the circuit simulations. However, if the variations are taken into account, the system becomes more vulnerable to the errors. In a similar manner, a scaled ReRAM cell can decrease the voltage level across the capacitor under the V_{ref} level and it is misinterpreted as logic-0 even though it would be logic-1 (i.e., there is a match). Therefore, an approximate system must be evaluated by regarding the variation sources which does not pose a threat to the non-approximate one. For this reason, to make the results as reliable as possible, the variations arising from the voltage sources and the memristive devices are inserted into the system. For memristance distributions, the distribution is taken from an experimental data [15] on the fabricated memristors by fitting it into the normal distribution. For voltage sources (pre-charge voltage V_{pc}, SRAM-cell supply voltage V_{cdd}, write voltages V_{wr}, and threshold voltage V_{ref}), the normal distribution with a 3% sigma is assumed even though sigma of variations are reported as 2.5% in other studies [37], [38]. Since memristor variations is also a result of variations in the write voltage, another variation onto the ReRAM-cell write voltage is not included. During the simulations, these variations are all modeled as different distributions and the random circuit parameters are generated from these distributions. The circuit simulator (HSPICE) is run several hundred thousands of cycles to obtain the metrics under these variations. The method presented in [36] is used to make both RAP and SAP as resilient as against these variations where V_{ref} of the sense amplifier is set to make the error probabilities as few as possible.

Table II shows the cases and their corresponding parameters and results for both SRAM and ReRAM cell scalings. The last column in the table shows the probability of average error ($P_e(avg)$) which the corresponding case causes after a compare operation. For the scaling in the SAP (Table IIa), the supply voltage of the scaled columns is decreased to 0.5V from the nominal voltage of 0.7V. This decrease leads to a decrease in both static energy and the write energy. On the other hand, the reliability of the cells decreases as well. During a compare operation, in the case of a mismatch, these cells increases the resistivity through the leaking path, and in some cases (e.g., more than one scaled columns are included in a compare operation), the compare operation may result incorrectly. The cell scaling in RAP is done by shrinking the write time and voltage. Table IIb shows the three different cases (*full*, *normal*, *approximate*) for the ReRAM scaling. In order to find these cases, the design space exploration is performed on the memristor model in [34]. The important concern in here is that the switching between R_{on} and R_{off} must be symmetric, that is, the state of the memristor can be switched in both directions within the same period. On the other hand, the voltage levels and polarities can be different depending on the memristor model as usual. In the table, the first row corresponds to the full case where the ReRAM is switched within its binary range (100 kΩ-100 Ω). The normal case shows a not-aggressive scaled case that does not pose a threat on its own still under the variability conditions. The last case presents the approximate case where the ReRAMs are scaled aggressively. This case results in the least energy and the most speedup with the expense of errors. During the experimentations, the full case is used for the most accurate case where there is no approximation. For the approximate computing in RAPs, the normal case is used for the most significant digits and the approximate case is used for the least significant digits.

B. Evaluation

In the evaluation of the proposed hybrid approximate computing methodology, firstly, the proposed architecture is compared with previous work which proposes the bit-trimming and the cell-scaling individually for APs [10]. Table III shows the comparison in terms of energy reduction and speedup for SAPs (Table IIIa) and RAPs (Table IIIb). In addition to

TABLE III

COMPARISON OF APPROXIMATION METHODS ON SAP (a) AND RAP (b) WITH DIFFERENT BENCHMARKS WITH 10% MAXIMUM QUALITY DEGRADATION

(a)

	Hybrid				Bit Trimming				Cell Scaling			
	Configuration	Speedup	Energy	E x S	Configuration	Speedup	Energy	E x S	Configuration	Speedup	Energy	E x S
SF	10s2t	1.24	3.41	4.24	2t	1.24	1.46	1.82	12s	1.00	2.54	2.54
IB	5s3t	1.60	2.40	3.85	3t	1.60	1.83	2.93	8s	1.00	1.45	1.45
CONV	4s16t	2.74	4.61	12.64	16t	2.74	4.04	11.08	26s	1.00	2.10	2.10
FWHT	14s2t	1.13	2.72	3.08	2t	1.13	1.22	1.38	14s	1.00	2.05	2.05
FFT	17s10t	2.07	6.47	13.37	10t	2.07	2.57	5.31	24s	1.00	2.40	2.40
FIR	4s5t	1.95	4.72	9.22	5t	1.95	2.40	4.69	16s	1.00	2.41	2.41
MF	10s14t	3.29	10.15	33.34	14t	3.29	5.09	16.72	24s	1.00	3.06	3.06
CNN	6s7t	2.74	6.88	18.83	7t	2.74	4.06	11.12	16s	1.00	4.48	4.48
Average		2.10	5.17	12.32	-	2.10	2.83	6.88	-	1.00	2.56	2.56

(b)

	Hybrid				Bit Trimming				Cell Scaling			
	Configuration	Speedup	Energy	E x S	Configuration	Speedup	Energy	E x S	Configuration	Speedup	Energy	E x S
SF	10s2t	2.04	48.16	98.11	2t	1.75	37.92	66.47	12s	1.64	37.61	61.63
IB	5s3t	2.67	55.54	148.11	3t	2.29	41.02	93.77	8s	1.67	41.68	69.46
CONV	4s16t	3.95	52.48	207.34	16t	3.89	51.19	199.31	26s	1.64	22.93	37.71
FWHT	8s2t	1.71	41.27	70.49	2t	1.59	35.11	55.70	12s	1.55	36.92	57.06
FFT	9s16t	3.14	39.03	122.39	12t	2.93	35.61	104.42	20s	1.63	21.44	34.92
FIR	4s5t	2.91	49.52	144.05	5t	2.79	45.39	126.53	12s	1.67	26.65	44.41
MF	8s12t	5.24	97.46	510.62	14t	4.69	81.58	382.91	23s	1.71	35.40	60.58
CNN	6s7t	4.26	89.45	381.35	7t	3.91	73.31	286.81	15s	1.72	37.47	64.36
Average		3.24	59.11	210.31	-	2.98	50.14	164.49	-	1.65	32.51	53.77

*** t: trimmed, s: scaled**

the convenient AxBenchs in Table I, the figure also includes the result for the convolutional neural network (CNN). For CNN implementation on the AP, only parts of the neural network are implemented since the whole CNN simulation (as cycle-accurate) would take months. The table also includes the configuration information for the approximation technique obtained from the design flow (see Figure 6). In the table, if bit is trimmed, the actual bit-width becomes less than the one presented in Table I. The *t, s* stands for the trimming and scaling respectively and the number before these abbreviations indicate the number of trimmed and scaled bits of the corresponding input, respectively. The number of trimmed or scaled bits are shown over the total number of bits of the intermediate results presented in Table I. In energy reduction, when the hybrid methodology is applied to the system, it outperforms in both SAP and RAP architectures when compared with the other methods. For example, the hybrid approximation provides an advantage in energy reduction between 14.1% to 151% for the SAPs and 2.5% to 35.4% for the RAPs over the bit trimming. As presented in the configuration of each approximation method, this savings mainly comes from less error contribution of the cell scaling method which in turn allows to scale more bits. Over the cell scaling, the advantage of the hybrid approximation in energy reduction ranges from 32.5% to 231.7% for SAPs and from 11.8% to 175.3% for the RAPs. The main reason of the higher energy savings in the SAP architectures is that the SAP architecture provides more scaling than the RAP architectures since its error probability is less. Additionally, the cell scaling in the SAPs contributes to both compare and write cycles as opposed to RAPs in which only write energy reduced. Furthermore, scaling in RCAM cells increases the compare energy of RAPs since the residual charge across the capacitor goes to a lower level than the normal case after a compare operation due to the more leaky memristors. This was a side-effect of memristance scaling, but in overall the scaling provides an advantage. It is worth to mention in here that hybrid approach is done on top of the AP architecture where bit trimming is applied. In other words, after applying bit trimming, the cell-scaling is performed to fill the remaining quality allowance to obtain more energy reduction and performance within a 10% quality degradation limit. For the speedup in the SAPs, the hybrid approximation has no advantage over the bit trimming method since the scaling in the SRAM cells does not provide a speedup, on the other hand, there is an advantage gain of average 9.3% in the RAP architectures of the hybrid approximation. If the application allows further approximation through the cell scaling after bit trimming, the hybrid method can get more benefit from it. As an example, in FFT, after trimming four bits from the data, the four more bits from the remaining number can be scaled to maximize the energy reduction and the performance. However, an error introduced at the beginning of the in-memory computations propagates through all computations until the end. For this reason, if the benchmark is complex and requires relatively more operations, the benefit from the approximate computing becomes limited. For instance, a separable convolution operation consists of 2 pipelined FFT operations and requires data to pass through many butterfly stages. For this reason, this application allows limited approximation and its energy reduction and speedup are less than the other benchmarks'.

Table III proves that the hybrid approximation technique has advantage over both previous approximation techniques applicable to in-memory associative processors. On the other hand, from the broader perspective, the indication of how

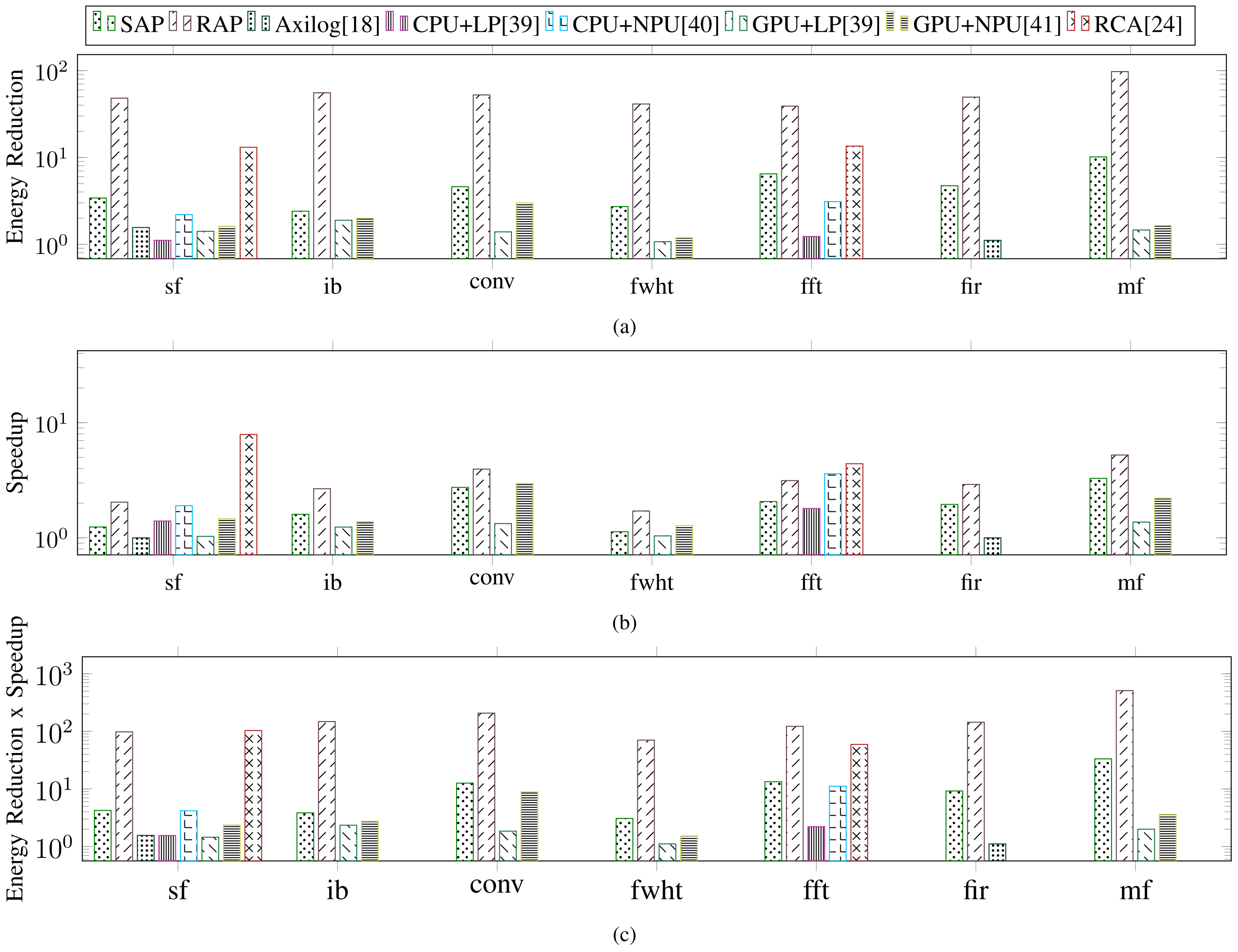


Fig. 9. Comparison of approximation methods on SAP, RAP, ASIC, CPU, and GPU platforms with different benchmarks with 10% maximum quality degradation. (a) Energy Reduction. (b) Speedup. (c) Energy reduction × Speedup.

APs performs in approximate computing is another topic. Figure 9 demonstrates the performance results of the different architectures and approximation techniques. In the results, hybrid approximation on the RAP and SAP architectures are compared against loop perforation (LP) [39], neural processing unit (NPU) [40], [41], resistive CAM accelerator [24], and Axilog HDL [18] a design tool for approximate ASIC designs. In the comparison, the evaluated results from [24], [30] are compared against hybrid approximation results. In reporting results, the relative energy reduction and speedup are reported which are the ratio of non approximate case over approximate case. Figure shows how hybrid approximation has a benefit over the reported other approximations. When compared with SAP, approximation in ReRAM-based AP provides better results in terms of energy reduction and speedup. This is because that a scaling in the write operation of the ReRAM cells returns a huge advantage in both factors. Even though all these have different architectures and technologies and seem as non-comparable on the same basis, the results shows shows how well approximate computing achieves on these architectures with different methodologies. In other words, the aim in here is to prove that associative in-memory computing with hybrid approximation has inherent advantages that make it better suited for approximate computing, achieving higher relative improvement in all three figures of merit (performance, energy and energy performance product).

V. Conclusion

In this study, we combined the two approximate computing methodologies in APs to form a hybrid approximate in-memory computing architecture. These two techniques are cell scaling and bit trimming. The proposed methods are naturally supported by the APs as dynamic and tunable. The reliability concerns coming with the approximation are also inspected. The suitability, practicality, and reliability of this methodology are investigated through the SPICE-based circuit simulations. The competency of the proposed method is proven by benchmark results. It is also proven that APs supports approximate computing inherently and provides a better employment place than other architectures for approximate computing since it facilitates dynamically (run time) tunable approximation.

References

[1] International Roadmap for Devices and Systems. (2016). *Beyond CMOS (Emerging Research Devices), White Paper*. [Online]. Available: http://irds.ieee.org/images/files/pdf/2016_BC.pdf

[2] "More Moore, White Paper," Int. Roadmap Devices Syst., Tech. Rep., 2016. [Online]. Available: https://irds.ieee.org/images/files/pdf/2016_MM.pdf

[3] S. Borkar, "Exascale computing—A fact or a fiction?" in *Proc. IEEE 27th Int. Symp. Parallel Distrib. Process. (IPDPS)*, Washington, DC, USA, 2013, p. 3, doi: 10.1109/IPDPS.2013.121.

[4] W.-H. Chen *et al.*, "A 65 nm 1 Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2018, pp. 494–496.

[5] C. C. Foster, *Content Addressable Parallel Processors*. New York, NY, USA: Wiley, 1976.

[6] I. D. Scherson and S. Ilgen, "A reconfigurable fully parallel associative processor," *J. Parallel Distrib. Comput.*, vol. 6, no. 1, pp. 69–89, Feb. 1989.

[7] J. L. Potter, *Associative Computing: A Programming Paradigm for Massively Parallel Computers*. New York, NY, USA: Perseus, 1991.

[8] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62-1–62-33, Mar. 2016.

[9] A. Agrawal *et al.*, "Approximate computing: Challenges and opportunities," in *Proc. IEEE Int. Conf. Rebooting Comput. (ICRC)*, Oct. 2016, pp. 1–8.

[10] H. E. Yantir, A. M. Eltawil, and F. J. Kurdahi, "Approximate memristive in-memory computing," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, Sep. 2017, Art. no. 129, doi: 10.1145/3126526.

[11] L. Yavits, S. Kvatinsky, A. Morad, and R. Ginosar, "Resistive associative processor," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 148–151, Jul. 2015.

[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 J. Solid-State Circuits*, vol. 49, no. 4, pp. 896–907, Apr. 2014.

[13] Fujitsu Semiconductor Limited. (Dec. 2016). *4m (512 k 8) Bit SPI Memory Reram Datasheet*. [Online]. Available: http://www.fujitsu.com/global/documents/products/devices/semiconductor/memory/reram/MB85AS4MT-DS501-00045-1v0-E.pdf

[14] C Inc. (2016). *1t1r Product: Picoe—Embedded Resistive Ram*. [Online]. Available: http://www.crossbar-inc.com/products/ip-cores/

[15] K. M. Kim *et al.*, "Voltage divider effect for the improvement of variability and endurance of TaO$_x$ memristor," *Sci. Rep.*, vol. 6, Feb. 2016, Art. no. 20085, doi: 10.1038/srep20085.

[16] I. Sobel and G. Feldman, "A 3 × 3 isotropic gradient operator for image processing," presented at the Stanford Artif. Intell. Project (SAIL), 1968.

[17] K. Roy, "Approximate computing for energy-efficient error-resilient multimedia systems," in *Proc. IEEE 16th Int. Symp. Design Diagnostics Electron. Circuits Syst. (DDECS)*, Apr. 2013, pp. 5–6.

[18] A. Yazdanbakhsh *et al.*, "Axilog: Language support for approximate hardware design," in *Proc. DATE*, Mar. 2015, pp. 812–817.

[19] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *Proc. Design, Automat., Test Conf. Exhib. (DATE)*, Mar. 2011, pp. 1–6.

[20] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "SALSA: Systematic logic synthesis of approximate circuits," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, Jun. 2012, pp. 796–801.

[21] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard, "Quality of service profiling," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng. (ICSE)*, New York, NY, USA, vol. 1, May 2010, pp. 25–34, doi: 10.1145/1806799.1806808.

[22] A. Rahimi, A. Ghofrani, K.-T. Cheng, L. Benini, and R. K. Gupta, "Approximate associative memristive memory for energy-efficient GPUs," in *Proc. Design, Automat. Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 1497–1502.

[23] M. Imani, A. Rahimi, and T. S. Rosing, "Resistive configurable associative memory for approximate computing," in *Proc. IEEE Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2016, pp. 1327–1332.

[24] M. Imani, D. Peroni, A. Rahimi, and T. Rosing, "Resistive CAM acceleration for tunable approximate computing," *IEEE Trans. Emerg. Topics Comput.*, preprint, doi: 10.1109/TETC.2016.2642057.

[25] M. M. A. Taha, W. Woods, and C. Teuscher, "Approximate in-memory Hamming distance calculation with a memristive associative memory," in *Proc. IEEE/ACM Int. Symp. Nanoscale Archit. (NANOARCH)*, Jul. 2016, pp. 159–164.

[26] Y. Kim, M. Imani, and T. Rosing, "ORCHARD: Visual object recognition accelerator based on approximate in-memory processing," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 25–32.

[27] H. Zhao, L. Xue, P. Chi, and J. Zhao, "Approximate image storage with multi-level cell STT-MRAM main memory," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 268–275.

[28] N. Pinckney, M. Fojtik, B. Giridhar, D. Sylvester, and D. Blaauw, "Shortstop: An on-chip fast supply boosting technique," in *Proc. Symp. VLSI Circuits*, Jun. 2013, pp. C290–C291.

[29] M. Imani, A. Rahimi, P. Mercati, and T. S. Rosing, "Multi-stage tunable approximate search in resistive associative memory," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 1, pp. 17–29, Jan./Mar. 2018.

[30] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "AxBench: A multiplatform benchmark suite for approximate computing," *IEEE Design Test*, vol. 34, no. 2, pp. 60–68, Apr. 2017.

[31] Arizona State University. (2012). *Predictive Technology Model*. [Online]. Available: http://ptm.asu.edu/

[32] S. Sinha, G. Yeric, V. Chandra, B. Cline, and Y. Cao, "Exploring sub-20nm FinFET design with predictive technology models," in *Proc. 49th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2012, pp. 283–288.

[33] F. Miao *et al.*, "Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor," *Adv. Mater.*, vol. 23, no. 47, pp. 5633–5640, Nov. 2011.

[34] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Generalized memristive device SPICE model and its application in circuit design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 8, pp. 1201–1214, Aug. 2013.

[35] D. Schinkel, E. Mensink, E. Klumperink, E. van Tuijl, and B. Nauta, "A double-tail latch-type voltage sense amplifier with 18ps setup+hold time," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, San Francisco, CA, USA, Feb. 2007, pp. 314–605.

[36] H. E. Yantır, M. E. Fouda, A. M. Eltawil, and F. J. Kurdahi, "Process variations-aware resistive associative processor design," in *Proc. IEEE 34th Int. Conf. Comput. Design (ICCD)*, Oct. 2016, pp. 49–55.

[37] U. R. Karpuzcu, N. S. Kim, and J. Torrellas, "Coping with parametric variation at near-threshold voltages," *IEEE Micro*, vol. 33, no. 4, pp. 6–14, Jul./Aug. 2013.

[38] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas, "VARIUS-NTV: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages," in *Proc. 42nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Washington, DC, USA, 2012, pp. 1–11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2354410.2355180

[39] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng. (Rinard)*, New York, NY, USA, 2011, pp. 124–134, doi: 10.1145/2025113.2025133.

[40] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *Proc. 45th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Washington, DC, USA, 2012, pp. 449–460, doi: 10.1109/MICRO.2012.48.

[41] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh, "Neural acceleration for GPU throughput processors," in *Proc. 48th Int. Symp. Microarchitecture (MICRO)*, New York, NY, USA, 2015, pp. 482–493, doi: 10.1145/2830772.2830810.

Hasan Erdem Yantır (S'13) received the joint B.Sc. degree from the Department of Computer Engineering and the Department of Electrical and Electronics Engineering, Yeditepe University, Istanbul, Turkey, in 2011, the M.Sc. degree from the Department of Computer Engineering, Bogazici University, Istanbul. He is currently pursuing the Ph.D. degree with the University of California at Irvine. He was an Engineer with Ericsson from 2011 to 2012 and Ford Motor Company from 2012 to 2013. His research interests include digital IC design, in-memory computing, computer architectures, and reconfigurable computing.

Ahmed M. Eltawil (S'97–M'03–SM'14) received the B.Sc. and M.Sc. degrees (Hons.) from Cairo University, Giza, Egypt, in 1997 and 1999, respectively, and the Ph.D. degree from the University of California at Los Angeles, Los Angeles, CA, USA, in 2003. He has been with the Department of Electrical Engineering and Computer Science, University of California at Irvine, Irvine, CA, USA, since 2005, where he is currently the Founder and the Director of the Wireless Systems and Circuits Laboratory. In 2015, he founded Lextrum Inc., to develop and commercialize full duplex solutions for 5G communications systems. He is also a Professor with the University of California. In addition to his department affiliation, he is also affiliated to a number of research centers across the University of California. His current research interests include low-power digital circuit and signal processing architectures with an emphasis on mobile computing and communication systems. He was a recipient of several awards and distinguished grants, including the NSF CAREER Grant in 2010, supporting his research in low-power systems. He has been on the technical program committees and steering committees for numerous workshops, symposia, and conferences in the areas of low-power computing and wireless communication system design.

Fadi J. Kurdahi (S'85–M'87–SM'03–F'05) received the B.E. degree in electrical engineering from the American University of Beirut in 1981 and the Ph.D. degree from the University of Southern California in 1987. Since 1987, he has been a Faculty Member with the Department of Electrical Engineering and Computer Science, University of California at Irvine, where he conducts research in the areas of computer-aided design, high level synthesis, and design methodology of large scale systems. He currently serves as the Director of the Center for Embedded and Cyber-physical Systems, comprised of world-class researchers in the general area of Embedded and Cyber-Physical Systems. He is a fellow of AAAS. He received the Best Paper Award of the IEEE TRANSACTIONS ON VLSI in 2002, the Best Paper Award at ISQED in 2006, and four other distinguished paper awards at DAC, EuroDAC, ASP-DAC, and ISQED. He also received the Distinguished Alumnus Award from Alma Mater, American University of Beirut, in 2008. He was the program chair or the general chair on program committees of several workshops, symposia, and conferences in the area of CAD, VLSI, and system design. He served on numerous editorial boards.