# CAP: Configurable Resistive Associative Processor for Near-Data Computing

Mohsen Imani, Tajana Rosing
CSE Department, University of California San Diego,
9500 Gilman Drive, La Jolla, CA 92093, CA, USA
E-mail: {moimani, tajana}@ucsd.edu

**Abstract**
Internet of Things is capable of generating huge amount of data, causing high overhead in terms of energy and performance if run on traditional CPUs and GPUs. This inefficiency comes from the limited cache size and memory bandwidth which result in large amount of data movement through memory hierarchy. In this paper, we propose a configurable associative processor, called CAP, which accelerates computation using multiple parallel memory-based cores capable of approximate or exact matching. CAP is integrated next to the main memory so it fetches the data directly from DRAM. To exploit data locality, CAMs adaptively split into highly and less frequent components and update at runtime. To further improve the CAP efficiency, we integrate a novel signature-based associative memory (SIGAM) beside each processing cores, to store highly frequent patterns and in runtime retrieve them in exact or approximate modes. Our experimental evaluations show that the CAP in approximate (exact) mode can achieve 9.4× and 5.3× (7.2× and 4.2×) energy improvement, and 4.1× and 1.3× speeds up compare to AMD GPU and ASIC CMOS-based designs while providing acceptable quality of service.

**Keywords**
Non-volatile memory, associative processor, approximate computing, content addressable memory.

## 1. Introduction

In 2015, the number of smart devices around the world exceeded 25 billion. This number is expected to double by 2020 [1]. The rate of data generation by the *Internet of things* (*IoT*) will quickly overtake the capabilities of current computing systems. The need for systems that can efficiently handle such large volumes of streaming data is undeniable [2]. Several IoT applications, e.g., machine learning, are statistic in heart and do not require highly accurate computation. Instead of doing all computation precisely, by accepting slight inaccuracy, we can get significant energy and performance improvements [3].

*IoT* increases the size of application's datasets exponentially. Running such large datasets on general purpose processors degrades both computation energy and performance. The inefficiency is due to limited cache size and memory bandwidth of processors which result in a large number of data movements through memory hierarchy. [4]. Near Data Computing (NDC) is one of the promising solution to overcome data movement [5], [6]. NDC locates processing units close to the main memory, such that the processor can directly access to DRAM data and avoid the memory/cache bandwidth bottleneck. However, adding extra dedicated CMOS-based processing units beside DRAM may result in additional energy and area overhead.

Memory-based computing can address the problem by performing the computation within memory [7]. Ternary content addressable memories (TCAMs) in a form of lookup table are a good candidate for memory-based computing. CAMs in CMOS technology consume a lot of search energy and occupy large area of the chip [8]. Low energy consumption and high density of non-volatile memories (NVMs) open new opportunities to have an efficient associative processor [9].

To the best of our knowledge, this paper is the first to propose configurable resistive associative processor, called CAP, which processes the data adaptively considering data locality and aware of running workload. CAP performs memory-based computation on crossbar CAM blocks beside the main memory to avoid large amount of data movement between the memory hierarchies. Our design consists of several cores where each core is designed based on a CAM. Each CAM preforms block-serial search operation (m-bit) and addresses endurance issue of resistive CAM by eliminating a number of writes. In contrast to all previous work which used naïve CAMs for computation, CAP building block is an adaptive CAM which splits TCAM to high and low frequency rows and updates them based on the running applications. To fully exploit data locality, we enable configurable approximation by utilizing a novel signature-based associative memory (SIGAM) beside the processing units. Our experimental evaluation shows that CAP can achieve 9.4× and 5.3× (7.2× and 4.2×) energy savings and 4.1× and 1.3× speeds up compare to AMD GPU and ASIC designs while providing acceptable quality of service.

## 2. Background and Related Work

Associative memory consists of two main blocks: a TCAM and a memory, where a set of input patterns and their corresponding outputs are respectively stored [10], [11], [12]. When a computation is issued, the operands are searched for in the TCAM. If there is a match, the corresponding result is returned from the memory [13]. In CMOS technology, TCAMs are designed using two SRAM cells, and so consume a lot of energy for each search operation. High density, CMOS compatibility and nearly-zero energy consumption make NVMs appropriate

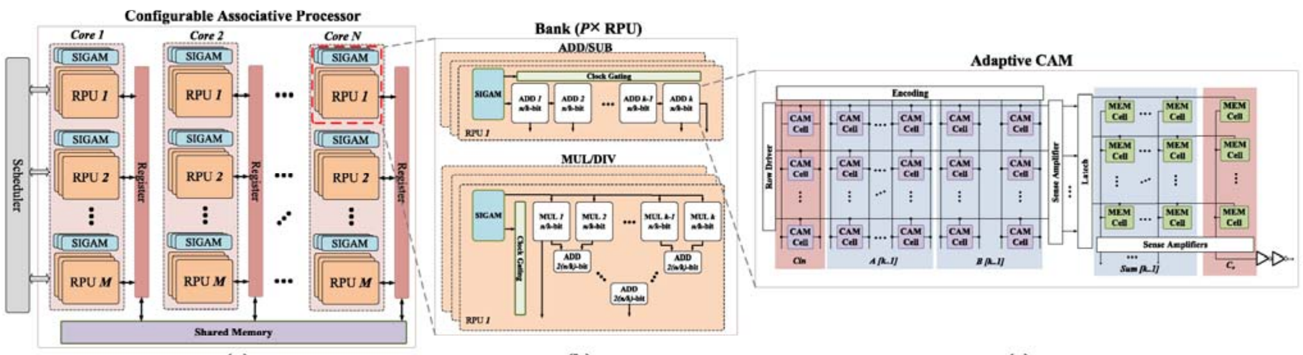

Figure 1. Overall structure of proposed CAP

candidates for the TCAM design. Resistive RAM (ReRAM) and Spin-transfer Torque RAM (STT-RAM) are two types of high speed and reliable NVMs based on memristive devices and magnetic tunneling junctions (MTJ), respectively [14]. The endurance of the ReRAM is $10^6$-$10^7$ write operations while for STT-RAMs it is more than $10^{15}$ [15-17]. TCAMs with MTJ device have higher endurance, and the ReRAM-based TCAMs provide higher search speed and area efficiency when utilizing 3D crossbar structure, which makes them more suitable for low energy associative memories. In this work, we use ReRAM but address the endurance problem by using a computation strategy that does not require frequent write operations.

Researchers have proposed many different energy-efficient techniques that exploit NVM-based TCAMs. Using NVM-based TCAMs next to parallel processors such as GPU can provide significant energy savings by reducing redundant computation [10], [11], [12], [18]. An approximate associative memory using TCAM with voltage overscaling was also proposed to relax FPU computation. This idea has been extended to a configurable approximate associative memory which tunes the level of approximation [18]. Although associative memories provide large energy savings, they still rely on the computing capability of the conventional general purpose processors. Thus, their designs cannot efficiently handle large data movement, which is the dominant component of energy consumption and performance bottleneck in workload parallelization.

The aim of associative processor is to perform the computation by looking up the preprocessed data without doing repeated computations [9], [19], [20]. Adder (ADD) is the main building block of computation. Other basic processing units such as subtractor (SUB), multiplier (MUL) and divider (DIV) can be implemented using adder. To add two n-bit data using associative processor (e.g. $A[n..0] + B[n..0] = Sum\ [n..0] + Carry\ [0]$), we require to store all $\boldsymbol{2^{2n}}$ input combinations on a lookup table. Meaning that comparing the input key with all stored patterns in row-serial manner, requires $O(2^n)$ cycles [21]. To improve this slow search operation, bit-serial search operation technique have been proposed in [9] and [22], where the computation can be performed by recalling 1-bit adder truth table (8-row) $\boldsymbol{n}$ times. In bit-serial technique, a simple $\boldsymbol{n}$-bit addition and multiplications require $O(\boldsymbol{n})$ and $O(\boldsymbol{n^2})$ to process. However, associative processors still have long search latency and high energy consumption as discussed below:

**Latency:** associative processors using bit-serial search have slow search operations, especially for MUL/DIV [19]. Using 1-bit adder block to design multiplication makes MUL a computational bottleneck. To address this problem, for multiplication we use ADD/MUL LUTs in tree structure to get high search speed. In addition, we use block-serial search operation which searches for input data in m-bit granularity, instead of bit-serial search.

**Energy:** Naïve computation of current associative processors does not consider data locality or type of workload in computation. In other words, they consume the same amount of energy for an operation independently of the input data. However, our associative processor: i) considers the data locality and application type in the processing units using a configurable CAM, ii) enables configurable processor approximation and iii) enables near data computing to avoid a large number of data movement.

## 3. Proposed CAP Design

Figure 1 shows the architecture of our proposed configurable associative processor (CAP), consisting of multiple parallel memory-based cores. CAP can be implemented beside the main memory to decrease the number of data movements through memory hierarchy. Instead of having multiple memory levels, CAP uses a shared cache among all processing cores. When application starts, a sequencer processor begins fetching instructions and data directly from DRAM. Then, scheduler assigns the current jobs to appropriate core(s). CAP adaptively processes the computation considering data locality and running workload. Table 1 shows the parameters of the proposed associative processor. Our design contains $\boldsymbol{N}$ subcores where each subcore has $\boldsymbol{M}$ banks. Each bank is a TCAM-based associative process representing ADD/SUB and MUL/DIV operations. ADD/SUB processing unit has $\boldsymbol{S}$ pipeline stages where each stage stores a lookup table corresponding to $\boldsymbol{k}$-bit basic operation ($\boldsymbol{S=n/k}$). In computation, the carry-out bit of the current stage is used as an input bit of the next stage associative memory (carry-in bit). After $\boldsymbol{S}$th stage, the stored data in the register shows the output of associative processor computation and the carry-out of the last stage represents the overflow bit.

Table 1.Design parameters of CAP

| Symbols | Description |
|---|---|
| $N$ | Number of subcores in a stage |
| $M$ | Number of banks of each subcore |
| $S$ | Number of stages of each subcore |
| $k$ | Granularity of block-serial search |
| $Size_{CAM}$ | CAM rows in each processing units |

Instead of doing bit-serial computation, CAP performs computation in block-serial way, where each block represents a look up table of $k$-bit addition. In order to get high performance MUL/DIV, we use multiplier architecture, shown in Figure 1b, which relies on the multiplication and adder look up tables in tree structure. The delay of block-serial adder is proportional to the number of available stages ($O(\boldsymbol{S})$ where $\boldsymbol{S}=(\boldsymbol{n}/\boldsymbol{k})$). For multiplier, the delay is proportional to delay of a k-bit multiplier, and ($log(\boldsymbol{S})$ times of) 2k-bit adder look up table. The ADD and MUL delay can be express as $T_{ADD}=\boldsymbol{S}*T_k$ and $T_{MUL}=Log(\boldsymbol{S})*T_{2k}+T_k$, where $T_m$ is the delay of an associative memory corresponding to m-bit ADD/MUL Lookup table (with $2^{2m+1}$ rows).

## 3.1. Block Serial Search

For $\boldsymbol{n}$-bit computation, the number of stages in proposed design, $\boldsymbol{S}$, depends on the $\boldsymbol{k}$ value in block-serial search. Lower $\boldsymbol{k}$ reduces the CAM size, but increases the number of required stages and depth of the pipeline. Our design searches for input data in $\boldsymbol{k}$-bit search granularity. The CAM needs a list of all possible input data corresponding to k-bit adder (containing carry-in bit), where each subcore has $\boldsymbol{S}=\boldsymbol{n}/\boldsymbol{k}$ stages. Table 2 shows the energy consumption and latency of doing 32-bit ADD and MUL on different bit-search granularities. As results show, large $\boldsymbol{k}$ increases the computation energy, but speeds up ADD computation by decreasing the number of stages. This indicates that based on the ratio of ADD/SUB to MUL/DID in an application, $\boldsymbol{k}$=1,2 or 3 bits block can provide higher energy-latency point. In Section 4.2.2, we discuss more about the impact of the block-serial search operation on applications performance and energy.

Table 2. The energy and latency of TCAM in different bit granularities

| Block size | | 1-bit | 2-bit | 3-bit | 4-bit |
|---|---|---|---|---|---|
| ADD/SUB | ***Energy(fJ)*** | 554 | 692 | 1378 | 3473 |
| | ***Latency(ns)*** | 8.4 | 7.3 | 7.9 | 9.3 |
| | ***# of stages*** | 32 | 16 | 8 | 4 |
| MUL/DIV | ***Energy(fJ)*** | 1636 | 1890 | 2858 | 5802 |
| | ***Latency(ns)*** | 6.4 | 7.8 | 9.6 | 12.7 |
| | ***# of stages*** | 6 | 5 | 4 | 3 |

## 3.2. Adaptive CAM

In NVM-based TCAMs, values are stored in cells based on the NVMs resistance state (Low or High). During the search mode, the input operands are compared to all pre-stored TCAM patterns and if the data is found, a charged Match Line (ML) activates the corresponding line of the resistive memory to retrieve the result of computation. We use an access-free transistor memory implemented purely by memristive devices to design both TCAM and resistive memory. Crossbar memory achieves significantly lower energy consumption and higher scalability, while occupying negligible area by implementing a 3D architecture. 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 3D.

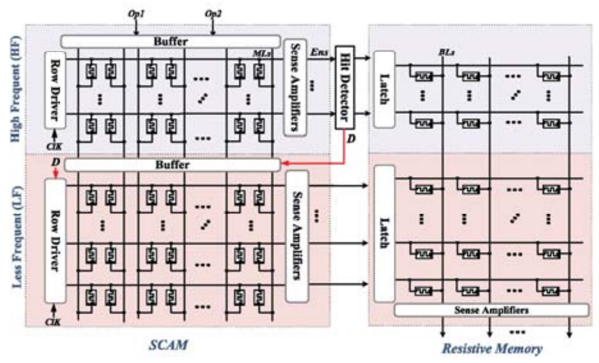

Figure 2. Structure of adaptive CAM consisting of low and highly frequent parts.

In associative processor the number of TCAM rows increases exponentially with the size of input data. Conventional TCAMs search for an input data among all TCAM rows in a single cycle. This inefficient search operation results in large energy and search latency per operation. To control the search energy, as Figure 2 shows, we split TCAM into two components; high frequency (*HF*) and low frequency (*LF*) rows. The search operation on TCAM stage starts at HF part. In case of a hit in *HF*, CAM stops search and saves energy. This hit can be detected using simple combination logic which can logically OR the outputs of sense amplifier in *HF* part. In case of a miss in *HF*, the CAM search continues in the rest of the CAM. Our evaluations show that storing a few rows (~20%) in *HF* part can provide very high hit rate (~60-80%) and energy savings. For each application, we use profiling to find the *HF* and *LF* patterns corresponding to ADD/SUB and MUL/DIV operations.

To evaluate the energy advantage of propose SCAM, let's consider the search delay of conventional CAM as $T_{CAM}$. Splitting the search operation into *HF* and *LF* parts increases the CAM latency to $T_{LF}+T_{HF}+T_{Detect}$. In this equation the $T_{LF}+T_{HF}<2*T_{CAM}$ because the search operation on small CAM can perform faster. If the input key misses on the *HF* part, it starts searching for an input data in the *LF* part in the second cycle. High percentage of hits in the *HF* part can accelerate the search operation in the SCAM. Assume we have $\beta$% hit rate on the *HF* part. It means that $\beta$% of the time we just consume the search energy/latency of the HF part. For the rest of ($1-\beta$)% of time, our design requires two cycles. The expected latency and energy savings of SCAM, assuming $T_{HF}+T_{Detect}=T_{LF}$ , can be calculated as:

$$T_{ACAM}=\beta(\%)\times T_{HF}+(1-\beta)(\%)\times(T_{LF}+T_{HF})+T_{Detect} \quad (1)$$

$$E_{ACAM}=\beta(\%)\times E_{HFP}+(1-\beta)(\%)\times(E_{LFP}+E_{HFP})+E_{Detect} \quad (2)$$

For example, if 20% of TCAM rows can provide $\beta=70\%$ hit rate, the proposed CAM can achieve ~2.3× energy savings compared to conventional TCAM. The ratio of *HF* to *LF* part of CAM should be set based on the application type. Figure 3 shows the search energy consumption of 4-bit CAM in different *HF/LF* ratios. In large size CAMs (>32rows), using small *HF* to *LF* ratio provides large energy savings, while this ratio needs to be higher in small size CAMs (e.g. 4-rows, 8-rows). CAP using large block sizes (k>1), exploits the data locality to provide higher hit rate, energy savings and computation acceleration.

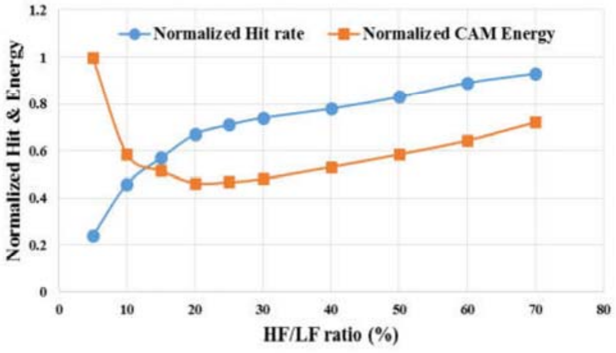

Figure 3. Average hit rate and energy consumption of 4-bit associative memory in different HF to LF ratios.

## 3.3. Signature-Based Associative Memory

To further improve computation energy while exploiting data locality, we propose small size signature-based associative memory (SIGAM) next to each processing unit which can store high frequency patterns. SIGAM saves energy by reducing the redundant computations. In SIGAM, a set of frequent pattern and their corresponding outputs store on the TCAM and resistive memory (MEM) respectively. As Figure 4 shows, before processing data on main CAP cores, the input operands compare to all pre-stored values in SIGAM. In case of a hit, our design clock gates CAP processing units and directly retrieves the preprocessed result of computation from *MEM* block. Therefore, the larger SIGAM, the higher average CAP clock gating time, thus the higher energy savings. Although, large *TCAM* improves hit rate, TCAM search energy can diminish the advantages of using associative memory.

Figure 5 shows the structure of proposed SIGAM consisting of signature TCAM (SIG-TCAM), main TCAM and resistive memory. SIG-TCAM first compares the signature of the input data with the signature of all pre-store values. Then, a hit in the SIG-TCAM selectively activates rows of the main TCAM stage. SIG-TCAM significantly reduces the search energy consumption of main TCAM by reducing the number of active rows in the main TCAM. Indeed, instead of searching large main TCAM structure for data matching, a pre-search performs on SIG-TCAM to reduce the number of active rows on the main block.

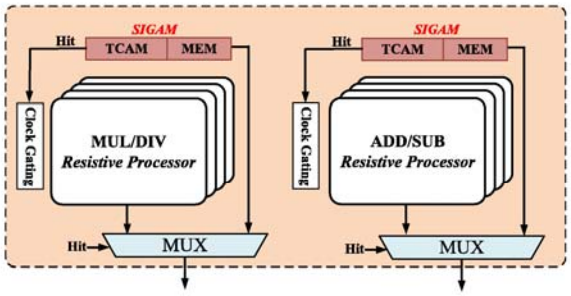

Figure 4. Approximation of proposed design SIGAM block

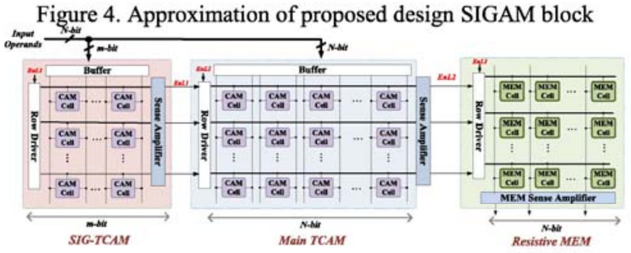

Figure 5. The architecture of proposed signature-based associative memory

All bits are not appropriate to use as signature bits. To select the proper bits, we extract the bit-level distribution of the pre-stored values on the main TCAM stage. Then, we select *m* bits which have higher potential to reduce the energy consumption of TCAM. These bits can be selected as bits that have bit distribution close to 50%, since each bit has potential to deactivate about half of main TCAM lines. However, considering the correlation of these bits is essential, since they may activate several common rows. Although, using more number of bits is likely to decrease the number of active rows at the main TCAM, large number of signature bits have negative impact on the total search energy, since total search energy is the combination of both SIG-TCAM and main TCAM:

$$E_{TCAM} = E_{SIG} + E_{Main} \quad (3)$$

The optimum number of signature bits depends on the application type. In our design, the value of the main TCAM and resistive memory can be determined using one-off profiling in design time. Therefore, we can find the optimal signature bits that have potential to provide lower search energy, by optimization. Therefore, we formularized the energy consumption of TCAM based on the SIG-TCAM and main TCAM energy as follow:

$$E_{SIG} = E_{Sense-Amp} + E_{Charging} \quad (4)$$

$$E_{Main} = \alpha \times (E_{Sense-Amp} + E_{Charging}) \quad (5)$$

$$\alpha = \min(Hit_{SIG}, 1 - Hit_{SIG}) \quad (6)$$

The energy consumption of both SIG and main TCAMs consist of sense amplifier and precharging energy. In SIG-TCAM, the sensing energy is fixed, while the precharging energy increases with the size of signature bits. The total main memory energy is the linearly related to the hit of the SIG TCAM (α value). Small α value reduces the number of the main TCAM active rows. This value depends on the signature bit selection. As the input data are not deterministic, the $\alpha$ value is the minimum number of SIG hit when the input data is zero or one.

We use Genetic optimization algorithm to find the optimum SIG TCAM size and bit indices which can provide maximum search energy. Our evaluation on four OpenCL applications shows that to store 32 most frequent patterns on a TCAM, using less than five signature bits is always enough to provide optimum energy saving. Table 3 shows the optimum number of signature bits and the average energy savings that 32-bit, 16-row SIGAM can achieve in different applications. Our evaluation shows that using less than four signature bits can improve the energy consumption of SIGAM ~70% in average. Figure 6 shows an example of search functionality of 9-bit input data ($101100100_2$) in SIGAM using 3-bit signature. Conventional technique naively searches for an input data in all TCAM rows which requires several switching activity. Instead, SIGAM architecture starts searching the input data on SIG TCAM (4rd, 5th and 7th bit indices of input data). So, the SIG hit on the two SIG TCAM rows activate the corresponding lines of main TCAM and resistive memory. In-advance row activation of the resistive memory allows our design to fast read of resistive memory without waiting for precharging.

Table 3. The search energy consumption of TCAM using SIGAM with optimum signature bits

| | *Robert* | *Sharpen* | *Matrix* | *DwHaar* |
|---|---|---|---|---|
| **# of SIG bits** | 4-bit | 2-bit | 3-bit | 4-bit |
| **SIG hit rate** | 84% | 73% | 78% | 86% |
| **Energy savings** | 73.5% | 61.7% | 68.6% | 72.2% |

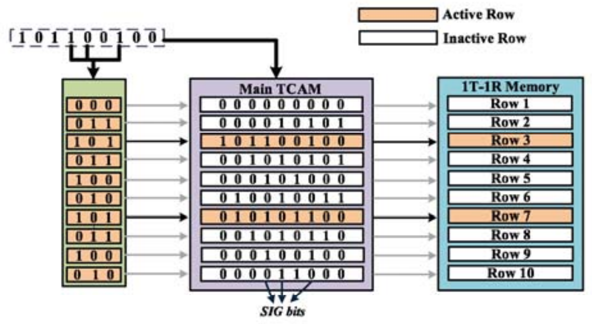

Figure 6. An example of search operation on the proposed SIGAM block

## 3.4. CAP Approximation Framework

Voltage overscaling (VoS) is one of the effective technique to improve TCAM search energy and hit rate at the cost of accepting inexact matching [18]. TCAM under VoS matches the input data with pre-store value with a few bits of hamming distances (depending on the voltage level). As the least significant bits have lower impact on computation result, we apply selective VoS on SIGAM block. SIGAM increases TCAM hit rate by allowing the input data to match with prestore TCAM values with 1-bit hamming distance in selective starting from LSB blocks. Higher hit rate increases the average time that the RePUs can be clock gated mode. Then, SIGAM returns a preprocessed result of computation stored on resistive memory. For each application, our framework increases the number of blocks on approximate mode till it can ensure acceptable quality of service.

We present a framework which is compatible with *OpenCL* as a standard for parallel programming of heterogeneous systems. We use Caltech 101 computer vision [23] dataset provides testing and training data for our image processing applications. For other general purpose applications, the input dataset has been generated randomly. Training is done on 10% of input patterns, while testing is done on all the data. For each application, input patterns are sorted on associative memory based on the frequency patterns obtained in raining mode (*k*-bit blocks). The high and low frequency patterns update *HF* and *LF* parts of CAM respectively. CAMs in all compute units are programmed concurrently with the same data based on the processing unit type.

To find the proper SIGAM configuration for each application, our framework compares the output of running an input data on exact and approximate processors. The framework continues putting more partial TCAM blocks (starting from LSBs at 8-bit granularity) into approximate mode as long as the application meets the quality requirements as shown in Figure 7. This requirement is defined as 30dB PSNR for image processing applications and 10% average relative error for other application.

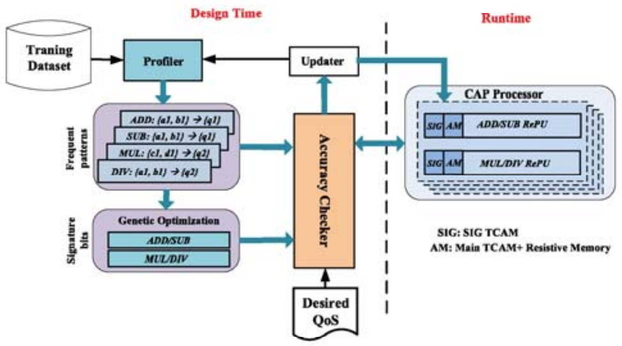

Figure 7. Framework to support CAP: design time profiling and runtime reuse.

## 4. Experimental Results

### 4.1. Experimental Setup

We compare the efficiency of proposed CAP with CMOS-based ASIC implementation of CAP in RTL and AMD Southern Island GPU Radeon HD 7970 device, a recent GPU architecture. For experimental simulation, we use System Verilog Description Language to design CAP. Then use Synopsys Design Compiler [24] to implement it in 45nm TSMC technology. For AMD GPU, we use Multi2sim, a cycle accurate CPU-GPU simulator [25] for simulation. Seven general applications: *Robert, Sharpen, Matrix Multiplication, DwHaar1D, Sobel, QuasiRandom, BlackScholes* are used to compare the energy and performance of proposed CAP with ASIC and GPU architecture. The image processing application has been adopted from AMD APP SDK v2.5 in OpenCL to make it suitable for streaming applications. The circuit of CAMs are designed using HSPICE on 45-nm technology.

### 4.2. CAP Configuration

#### 4.2.1. CAP Size

The performance and energy consumption of CAP depend on the number of processing units. As Figure 8 shows, the energy consumption of CAP grows linearly with the number of processing units. However, large number of processing units improves CAP performance by increasing parallelism. In our design two factors can limit the number of RPUs in CAP: **i)** power density considering chip thermal design. This power is 300W for recent parallel processors such as AMD GPUs. **ii)** Area occupation of the CAP. High density of crossbar non-volatile CAMs ($8F^2/n$) and resistive memories ($4F^2/n$) allows integrating associative memories in 3D structure, where $F$ is feature size and $n$ is the number of vertical layers. For n=1, this integration allows putting 3K RPU on a same silicon area as floating point units in AMD GPU which contains 32 computing units, where each has 4-SIMD and each SIMD contain 16 lanes (2048 streaming cores). The RPU density increase by extending the number of vertical layers. However, using more than 5-layers slows down the computation and exceeds 300W thermal design power. Therefore, a CAP processor can achieve 15K RPUs which is 6× larger than the number of processing units in AMD southern Island GPU, a most recent GPU architecture. Figure 8 shows the performance and the power consumption of the CAP compare to AMD GPU. As graph shows, the CAP performance increases linearly with the number of processing cores up to 6.1 TFLOP/s using 15K RPUs integrated in 5 vertical layers.

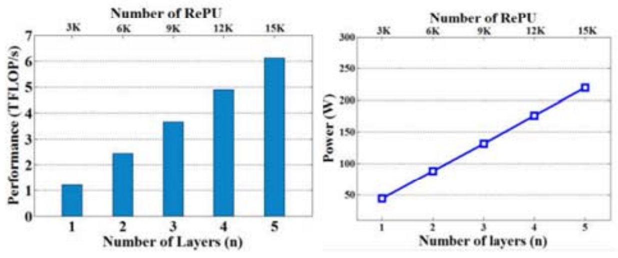

Figure 8. Performance and power consumption of CAP using different number of processing units

### 4.2.2. Block-Serial Search

As we explained in Section 3.1, in CAP the search operation performs in block serial approach. In ADD operation, the computation starts by looking up the $k$ bit data in the first associative memory stage, containing $2^{2k+1}$ rows. For looking up the second $k$ bits, we need to use a carry out bit of the first block as an input of the second stage. This means that the search operation on the next associative memory stage starts by reading the carry-out bit of the current stage. These dependencies slow down the serial search operation. To compensate such slow search, CAP needs to use large blocks because in block-serial, CAP does not wait for intermediate stages to read the carry bit.

Figure 9 shows the energy and performance improvements of CAP using different block size compared to ASIC and GPU architecture. CAP consumes significantly lower energy consumption compare to both designs. Although larger block decreases the number of stages linearly, the CAM size in each stage grows exponentially with the block size. Thus, CAP using larger block speeds up the search operation by reducing the number of intermediate access.

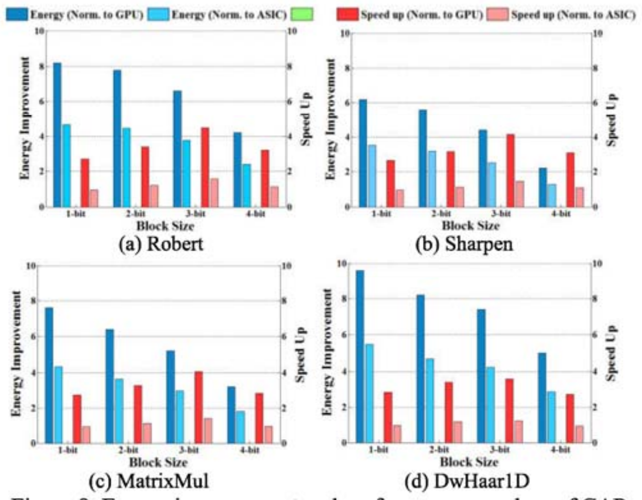

Figure 9. Energy improvement and performance speed up of CAP using different block size.

Increasing the block size from 1-bit, speed up the computation by reducing the number of intermediate accesses. However, CAM in 4-bit block size has $2^9$=512 rows, requiring slow and energy hungry input buffer to distribute input data among all CAM rows. we consider energy-delay product (EDP) as a figure of merit to find the best block size. Our evaluation shows that RPU using 3-bit block can achieve 4.4× and 23× EDP improvement in average compared to ASIC design. In this configuration, the CAP achieves 5.9× and 3.4× energy improvement and 4.1× and 1.3× speed up compare to ASIC and AMD GPU.

### 4.3. CAP Approximation

Our evaluation shows that for all applications using small size associative memory (8-row or 16-row) results maximum CAP energy saving. Figure 10 shows CAP normalized energy consumption using proposed SIGAM in different sizes. There is a tradeoff between the energy consumption of SIGAM and RPU depending on the size of the TCAM. A SIGAM with large number of rows increase the hit rate and thus the average time that the RPU can be clock gated. However, large SIGAM requires slow and power hungry buffer to distribute the input signals to all rows simultaneously. The delay and energy introduced by big buffer prevents searching the entire block in a single cycle and degrades the search energy efficiency. In addition, the ratio of SIGAM hit rate to search energy cannot further improve with increasing the TCAM size because of SIGAM hit rate saturation.

To further improve the CAP processor energy consumption, we relax the computation by implementing approximation on SIGAM block. Approximate SIGAM reduces the search energy consumption of the main TCAM and also improves the energy of RPU block. Table 4 lists the portion of SIGAM bitline on voltage overscaling using 8-bit relaxation granularity, providing less than 10% quality loss. Figure 10 shows the energy consumption of CAP processor applying under SIGAM approximation. Our evaluation shows that SIGAM can improve the CAP energy consumption by average 32% and 49% in exact and approximate modes providing less than 10% quality loss.

Table 4. Maximum portion of SIGAM in approximate mode while providing acceptable QoS

| CAM size | Robert | Sharpen | MatrixMul | DwHaar1D |
|---|---|---|---|---|
| *2-row* | 32-bit | 32-bit | 24-bit | 32-bit |
| *4-row* | 32-bit | 32-bit | 24-bit | 24-bit |
| *8-row* | 32-bit | 24-bit | 16-bit | 16-bit |
| *16-row* | 16-bit | 25-bit | 16-bit | 8-bit |
| *32-row* | 16-bit | 16-bit | 8-bit | 8-bit |

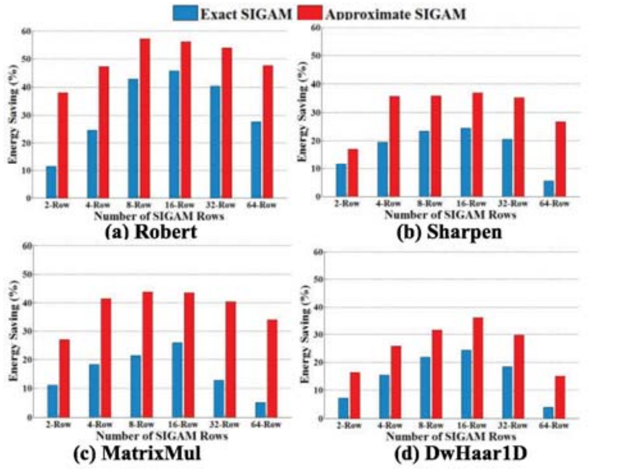

Figure 10. Normalized energy consumption of CAP using SIGAM in approximate and exact matching.

Table 5 shows the normalized energy saving of CAP in exact and approximate mode normalized to the ASIC design and AMD GPU. For each application, we use optimum SIGAM block size (3-bit) and number of rows (16-row) which result in maximum energy saving. Our evaluation shows that proposed CAP in approximate mode (exact mode) can achieve 5.3× and 9.4× (4.2× and 7.2×) lower energy consumption and 1.3× higher performance compared to ASIC and GPU architecture, while ensuing acceptable quality of service.

Table 5. Maximum energy improvement of CAP in Exact and approximate mode compared to ASIC and AMD GPU

| *Mode/* Normalized to | | *Robert* | *Sharpen* | *Matrix* | *DwHaar* | *Sobel* | *Quasi* | *Black* |
|---|---|---|---|---|---|---|---|---|
| Approx | ASIC | 8.6x | 4.0x | 5.3x | 6.1x | 4.7x | 5.1x | 3.7x |
| | GPU | 15.0x | 7.0x | 9.2x | 10.7x | 8.2x | 9.0x | 6.5x |
| Exact | ASIC | 6.9x | 3.3x | 4.0x | 5.6x | 3.3x | 3.7x | 2.3x |
| | GPU | 12.1x | 5.8x | 7.0x | 9.8x | 5.8x | 6.5x | 4.0x |

## 5. Conclusion

In this paper, we propose a configurable associative processor (CAP) to address the energy and performance limitations of current computing systems. The main contribution of this paper is three fold: i) in contrast to previous associative processors which naively perform computation, our design uses adaptive CAM which exploits data locality and reconfigures the architecture based on the running workloads ii) CAP changes the architecture from slow bit-serial search to block-serial search operation to speed up the search process iii) our design relaxes the CAP computation by applying configurable approximation starting from least significant bits. Our experimental evaluations show that the CAP processor in approximate (exact) mode can achieve 9.4× and 5.3× (7.2× and 4.2×) energy savings and 4.1× and 1.3× speeds up compare to AMD GPU and ASIC designs while meeting target quality requirement.

## 6. Acknowledgment

This work was supported by NSF grant #1527034 and Jacobs School of Engineering UCSD Powell Fellowship.

## 7. References

[1] J. Gantz, et al., "Extracting value from chaos," *IDC iview*, vol. 1142, pp. 1-12, 2011.

[2] J. Gubbi, et al., "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, pp. 1645-1660, 2013.

[3] M. Imani, et al., "Maximum-Likelihood Adaptive Filter for Partially Observed Boolean Dynamical Systems," IEEE Transactions on Signal Processing, vol. 65, pp. 359-371, 2017.

[4] A. M. Aly, et al., "M3: Stream processing on main-memory mapreduce," International Conference in Data Engineering (ICDE), pp. 1253-1256, 2012.

[5] R. Balasubramonian, et al., "Near-data processing: Insights from a MICRO-46 Workshop," Micro, IEEE, vol. 34, pp. 36-42, 2014.

[6] G. Loh, et al., "A processing in memory taxonomy and a case for studying fixed-function pim," in Workshop on Near-Data Processing (WoNDP), 2013.

[7] R. J. Gerrig, "The scope of memory-based processing," *Discourse Processes*, vol. 39, pp. 225-242, 2005.

[8] H. J. Mattausch, et al., "Associative memory for nearest-Hamming-distance search based on frequency mapping," *IEEE JSSC*, vol. 47, pp. 1448-1459, 2012.

[9] Q. Guo, et al., "Ac-dimm: associative computing with stt-mram," in *ACM SIGARCH Computer Architecture News*, pp. 189-200, 2013.

[10] M. Imani, et al., "MASC: Ultra-low energy multiple-access single-charge TCAM for approximate computing," in IEEE/ACM *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016, pp. 373-378.

[11] M. Imani, et al., "Approximate Computing using Multiple-Access Single-Charge Associative Memory," *IEEE Transactions on Emerging Topics in Computing (TETC),* 2016.

[12] M. Imani, et al., "ReMAM: Low energy Resistive Multi-stage Associative Memory for energy efficient computing," in *International Symposium on Quality Electronic Design (ISQED)*, pp. 101-106, 2016.

[13] M. Samragh, *et al.*, "LookNN: Neural Network with No Multiplication*," IEEE/ACM Design Automation and Test in Europe Conference (DATE),* 2017.

[14] M. Imani, et al., "ACAM: Approximate Computing Based on Adaptive Associative Memory with Online Learning," in *International Symposium on Low Power Electronics and Design (ISLPED),* 2016.

[15] Y. Kim, et al., "CAUSE: critical application usage-aware memory system using non-volatile memory for mobile devices," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 690-696, 2015.

[16] N. Khoshavi, et al., "Bit-Upset Vulnerability Factor for eDRAM Last Level Cache Immunity Analysis," in IEEE International Symposium on Quality Electronic Design (ISQED), pp. 6-11, 2016.

[17] N. Khoshavi, et al., "Read-Tuned STT-RAM and eDRAM Cache Hierarchies for Throughput and Energy Enhancement," arXiv preprint arXiv:1607.08086, 2016.

[18] M. Imani, et al., "Resistive Configurable Associative Memory for Approximate Computing." *IEEE/ACM Design Automation and Test in Europe Conference (DATE)*, 2016.

[19] J. L. Potter, "Associative Computing: A Programming Paradigm for Massively Parallel Computers" *Springer Science & Business Media*, 2012.

[20] M. Sharad, et al., "Ultra low power associative computing with spin neurons and resistive crossbar memory," *IEEE/ACM Design Automation Conference (DAC)*, 2013.

[21] C. C. Foster, "Content addressable parallel processors," *John Wiley & Sons, Inc.*, 1976.

[22] L. Yavits, et al., "Resistive Associative Processor," *Computer Architecture Letters*, 2014.

[23] Caltech library, online at:
"http://www.vision.caltech.edu/Image_Datasets/Caltech101/"

[24] Design Compiler, "Synopsys inc," ed, 2000.

[25] R. Ubal, et al., "Multi2Sim: a simulation framework for CPU-GPU computing," *IEEE Parallel Architecture and Compilation Techniques (PACT)*, pp. 335-344, 2012.