An Alternative Analytical Approach to Associative Processing

Soroosh Khoram, Yue Zha, and Jing Li

Abstract—Associative Processing (AP) is a promising alternative to the Von Neumann model as it addresses the memory wall problem through its inherent in-memory computations. However, because of the countless design parameter choices, comparisons between implementations of two so radically different models are challenging for simulation-based methods. To tackle these challenges, we develop an alternative analytical approach based on a new concept called architecturally-determined complexity. Using this method, we asymptotically evaluate the runtime/storage/energy bounds of the two models, i.e., AP and Von Neumann. We further apply the method to gain more insights into the performance bottlenecks of traditional AP and develop a new machine model named Two Dimensional AP to address these limitations. Finally, we experimentally validate our analytical method and confirm that the simulation results match our theoretical projections.

Index Terms—Associative processors, analysis of algorithms and problem complexity, modeling techniques, models of computation

1 Introduction

Associative Processing (AP) is a promising computational paradigm that aims to tackle the Von Neumann bottleneck by virtue of a radically different machine model that natively supports in-memory computations. However, practical implementations of AP using CMOS technology incurred prohibitive costs. These cost constraints have been recently alleviated with the advances in the Non-Volatile Memory (NVM) technologies, reigniting interest in associative processors [4], [5], [10].

Despite the advantages of AP, comparing implementations of such an unconventional model with conventional Von Neumann architectures remains challenging due to the lack of a standard evaluation procedure. Existing approaches typically require exhaustive simulations at different levels, that is, system, micro-architecture, logic gate, and circuit. This process is tedious and time consuming because of the numerous simulation parameters that can be chosen from the large design space.

In this paper, we address this challenge by taking an alternative analytic approach. In particular, we make three key contributions. The first contribution is to develop an analytical method that serves as a theoretical basis for evaluating different machine models. This method focuses on the evaluation of the high-level machine model, ignoring implementation details. With this approach, we gain further insights into the performance bottlenecks of the traditional AP machine model. Specifically, we show that the runtime of this model (here also referred to as 1D AP) is bounded by the word-size as it only supports one-dimensional inter-word parallelism.

Another contribution of this paper is to develop a new machine model, namely *Two Dimensional AP* (2D AP), based on our theoretical analysis of the traditional AP. The 2D AP improves performance by exploiting both *intra-word* and *inter-word* parallelisms. We further apply the analytical method to better understand the design trade-offs of 2D AP, which would be costly to obtain using simulations.

The third contribution is to validate our analytical method through full system simulation using an in-house cycle-accurate AP simulator (for both 1D AP and 2D AP) integrated into the gem5 system simulator. We show that ignoring constants and low-level implementation details in our theoretical projections does not lead to a different conclusion from the experimental results, ascertaining the accuracy of our analysis.

- *The authors are with the Department of Electrical and Computer Engineering, University of Wisconsin - Madison, Madison, WI 53706. E-mail: {khoram, yzha3}@wisc.edu, jli@ece.wisc.edu.*

Manuscript received 26 July 2017; revised 8 Dec. 2017; accepted 29 Dec. 2017. Date of publication 3 Jan. 2018; date of current version 30 Mar. 2018.
(Corresponding author: Soroosh Khoram.)
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/LCA.2018.2789424

2 Background

AP is a general-purpose parallel machine model that is inherently different from the traditional Von Neumann model. In the Von Neumann model, the processor (CPU/GPU/FPGA/CGRA) reads the input data from the memory and writes the computation results back afterwards. Data movement occurs through the system bus and memory hierarchies, all of which impose significant latency and energy penalties. Conversely, AP operates concurrently on every word where data is stored, eliminating the need for data movement.

The main building blocks of the AP model are depicted in Fig. 1a. It consists of a Content Addressable Memory (CAM), a set of special-purpose registers (key, mask, and tags) and a reduction tree. The CAM comprises an array of bit cells organized in bit columns and word rows. Key is used to store a value that can be written into or compared against the CAM words. Mask defines the active fields for these compare and write operations, enabling bit selectivity. Tags are used to store compare results for words and are set to logic "1" upon matches. The combination of key and mask can be used to write to all tagged words in parallel. Finally, the reduction tree post-processes the compare results stored in the tags.

Traditional AP performs computations in a *bit-serial, word-parallel* manner. Fig. 1 illustrates this using an example of adding two N-element vectors with m-bit elements, A and B, to produce a result vector, R, and a single-bit carry, C. Each single-bit addition required for an m-bit addition is carried out in a series of lookup table (LUT) passes based on Table 1 (1D AP). In each pass, one entry of the table (a 3-bit input pattern, A, B, Ci in the table) is compared against the contents of the corresponding bit columns of all words (1–N) of the CAM, tagging the matches; the results (two-bit output pattern, R and Co) are then written into all tagged rows. The carry-out, Co, overwrites the carry-in Ci, and thus both are stored in column C in Fig. 1. Furthermore, assuming R to be initially zero, some patterns result in *no changes* (Table 1, 1D AP). Therefore, only 5 patterns (marked as *pass* in Table 1) are looked up. AP repeats these steps for all m bits of R for full additions.

In traditional AP, the runtime is independent of the dataset size. That is due to the constant-time lookups of AP, which exploit its abundant internal bandwidth for *in situ* computations. As long as the data fits into one or multiple APs, lookup time remains constant even as the dataset size grows.

3 Analytical Method

We develop a mathematical model that, without loss of generality, can be applied to quantitatively evaluate the processing capabilities of a diverse set of architectures (CPU, GPU, NP, and AP) by abstracting the details of technology/micro-architecture implementations and the workloads into several high-level architectural and algorithmic parameters. We further apply the model to derive a new method for architecture evaluation based on a concept called *architecturally-determined complexity*, which we define as the theoretical bounds for the computation resources an architecture requires to perform a task. Here, by architecture we are referring to the high-level machine model. Specifically, the proposed method asymptotically, evaluates the runtime, storage space, and energy of performing computations on a specific architecture, ignoring constants and low-level factors. In this sense, it is different from the commonly used application-driven complexity, i.e., the Big $\mathcal{O}$ notation, which analyzes the asymptotic behavior of time and space

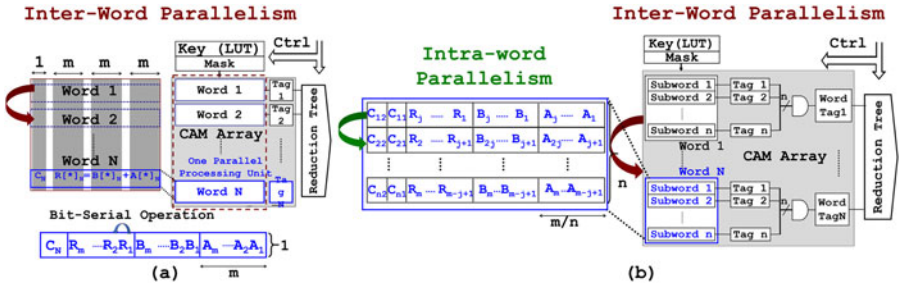

Fig. 1. The machine model of a) 1D AP b) 2D AP. 1D AP performs operations bit-serially while 2D AP has intra-word parallelism.

complexity of running algorithms specifically on Von Neumann architectures. In this section, we present the details of the model and elaborate our method to find the architecturally-determined complexity for both AP and CPU, and provide insights on the architectural limitations of traditional AP.

3.1 Runtime Analysis

Our runtime model comprises two components: 1) computation cycles that are spent on useful computations, and 2) hazard penalties that are spent on resolving cash misses, branch mis-prediction, etc. Each of these components further consists of two terms: 1) a hardware dependent term, $C_{\{.\}}$, representing the hardware cycles spent on processing one input data word, and 2) a workload-dependent term, $N_{\{.\}}$, representing the frequency of $C_{\{.\}}$. Thus, in a vector addition for example, the key terms C_{comp} and N_{comp} are the number of cycles spent on addition and the number of single additions (length of the vector), respectively. In addition, we consider the number of active cores, N_{core}, for the computation component. Calculating the runtime requires all these values including hardware-dependent terms ($C_{\{.\}}$). However, since $C_{\{.\}}$'s are generally constant ($O(1)$ complexity), they can be ignored for the asymptotic analysis. We further argue that hazard frequencies are, in the worst case, the same complexity as computation frequencies, and thus can also be ignored. We next present the evaluations for CPU and AP, focusing on the remaining terms in our asymptotic analysis.

$$C_{cycles} = C_{comp}\frac{N_{comp}}{N_{core}} + \sum_{i \in H} C_i N_i$$
$$H = \{cache\ miss, branch\ miss - prediction, \ldots\}. \quad (1)$$

The single-core CPU has only one active core ($N_{core} = 1$). This simplifies the model to that of previous works [3], which can be used to derive the required terms. Since the complexity of N_{comp} is determined by the input data size, the architecturally-determined runtime complexity derived from this model is the same as the Big $\mathcal{O}$ complexity.

AP, conversely, can reduce the complexity by an order compared to CPU. This is due to the tremendous parallelism of AP, allowing for very large values of N_{core} with the same complexity as the dataset size. We will explore the advantages and limitations of AP with an example in Section 3.4.

3.2 Energy Analysis

Similar to the runtime analysis, the energy consumption can be estimated by partitioning it into computation and hazard terms as shown in Equation (2). Different from the runtime analysis, N_{core} does not appear in the equation as the energy is summed over all cores. Similar to before, hardware-dependent terms, $E_{\{.\}}$, are computed through benchmarking and asymptotic analysis is applied to the data-dependent term, N_{comp}.

$$E_{nergy} = E_{comp}N_{comp} + \sum_{i \in H} E_i N_i. \quad (2)$$

3.3 Storage Analysis

In the storage complexity analysis, we evaluate how the storage space requirements grow with the input problem size. In the case of CPU, this analysis is similar to the conventional asymptotic storage studies. The storage complexity of AP can be estimated based on the space required to store the input/output data and the intermediate variables.

3.4 Key Insights on Traditional AP

We apply the proposed method to derive the architecturally-determined complexity of runtime, energy, and storage space of traditional AP, using a case study on vector addition. Since AP performs all additions in parallel (Section 2), assuming two vectors of size $\mathcal{N}$ with m-bit words, we have $O(\mathcal{N})$ values of N_{comp} and N_{core}. This results in constant time operation with respect to $\mathcal{N}$. N_{comp} on AP further contains a word size-dependent component since the addition is decomposed into a set of *compares* and *writes* as defined in AP's machine model. In this case, AP requires $5m$ compare and write passes resulting in $\mathcal{O}(m)$ addition runtime (since $N_{comp} = O(m\mathcal{N})$). The energy term, E_{comp}, can be similarly evaluated by decomposing addition into compares and writes. A compare consumes energy in words upon a *mismatch*, while a write consumes bit update energy in *matching* words. In the addition example, we can estimate that each compare results in mismatches in $\frac{7}{8}$ of all words. The remaining $\frac{1}{8}$ (matching words) will have 2 bit writes (R and C) in the subsequent write operation. Thus, the AP has an average energy consumption of $(\frac{7}{8}+\frac{2}{8})5m$ per word ($\mathcal{O}(m\mathcal{N})$ overall). We note that this means, vector addition on AP has $\mathcal{O}(m^2\mathcal{N})$ Energy-Delay Product (EDP). Finally, this addition needs to store the two operands (A and B), one output (R) and one carry bit (C) which results in a space requirement of $3m+1$ per word, with $\mathcal{O}(m)$ complexity. We see that the AP performance and energy efficiency are bounded by the word size (m). This is because traditional AP limits the parallelism to the *inter-word* dimension, which is why we also call it *1D AP*. We address this limitation by introducing a new machine model.

4 Two Dimensional AP

To address the limitations of 1D AP, we propose *2D AP*, which exploits *two* dimensions of parallelism, inter-word and intra-word. Here, we first describe the 2D AP model. Then, we explain how it performs computations using the same vector addition example. We will show that the 2D AP reduces the runtime and EDP complexity compared to 1D AP.

Fig. 1b depicts the abstract machine model of the proposed 2D AP. In 2D AP, a word entry is broken down into n subword entries, each of which is associated with its own tag so that all subwords can be simultaneously compared against the key. Because of this, 2D AP is capable of performing computations on each word multiple bits at a time. Furthermore, all subword tags of each word are logically *AND*ed together to create a word tag. This tag is used when we need to compare words against a wide key that spans across multiple subwords.

We will show how this extra dimension of parallelism enables faster computations on 2D AP using the addition example. As Fig. 1b shows, the m-bit operands, A and B, are divided into n subwords. Each subword is responsible for calculating $\frac{m}{n}$ bits of the result R, which it performs in parallel with other subwords. However, the carry chain can limit this parallel computation. To solve this issue,

TABLE 1
Addition Lookup Table (LUT) for 1D AP and 2D AP

Ci	*B*	*A*	*Co*	*R*	*1D AP*	*2D AP*
0	0	0	0	0	no change	no change
0	0	1	0	1	pass	no change
0	1	0	0	1	pass	no change
0	1	1	1	0	pass	pass
1	0	0	0	1	pass	pass
1	0	1	1	0	no change	no change
1	1	0	1	0	no change	no change
1	1	1	1	1	pass	no change

the 2D AP first *speculatively* calculates the carry-in for all subwords, then it finds the valid carry-in for each subword, and finally performs subword additions in parallel. This is done in three steps:

1. *Speculative Carry Calculation*. The carry-out for each subword is calculated for both possible values of carry-in (0 and 1). This is done by applying a simplified version of the LUT (Table 1, 2D AP). Since only the carry-out for each subword needs to be calculated, this LUT has fewer required passes, resulting in faster operations. This step requires $\frac{4m}{n}$ cycles (2 LUT passes for each speculative carry). After this step is finished, for each subword, two speculative carry bits are available in each subword, but only one of them is valid.
2. *Valid Carry Selection*. The valid carry-in for the first subword (initially 0) is chosen. Based on that, the carry-in of the next subword is chosen and so on. This step is performed sequentially. However, its runtime depends on the subword count n, instead of the word size m. This step requires $2n$ cycles.
3. *Result Generation*. Finally, the real carry-in for each subword has been calculated in step 2. Thus, the same LUT for the operation as in 1D AP (as shown in Table 1) is again applied to all subwords using real carry-ins to calculate the final result in parallel. This step requires $\frac{5m}{n}$ cycles (5 LUT passes per bit).

By adding the runtimes for these three steps, we get a total runtime of $\frac{9m}{n} + 2n$. Optimizing this result for n gives us $n = 3\sqrt{\frac{m}{2}}$ and a total runtime of $6\sqrt{2m}$ with $\mathcal{O}(\sqrt{m})$ complexity. Table 2 shows that the 2D AP has a similarly lower complexity for other arithmetic operations as well. This reduced complexity does incur extra space for the speculative carry bits, but the overhead increases linearly with $n(\mathcal{O}(\sqrt{m}))$ and thus does not affect the space complexity. This implementation also consumes some extra energy for the first two steps, which equals to $2m + 3n$ per word. Nonetheless, this increase in energy does not change the energy complexity. In fact, since the 2D AP has an EDP complexity of $\mathcal{O}(m\sqrt{m})$ as opposed to the $\mathcal{O}(m^2)$ complexity of the 1D AP, it is more energy efficient.

5 Insights Based on the Analysis

In this section, we present the insights from our analysis on the 1D AP, 2D AP, and CPU models. Specifically, we compare representative arithmetic operations and algorithm benchmarks for these architectures. Our analysis indicates that applications with poor data locality, simple computations, and high data-level parallelism can greatly benefit from the AP architecture.

Arithmetic operation analysis. The results presented in Table 2 show that by following this method, we can improve performance and energy efficiency complexity for various operations. We note that the value of n is ultimately a design parameter that needs to be determined by architects. However, our analysis shows that the theoretical optimal n varies across different arithmetic operations and choosing the optimal value for one could lead to sub-optimal results for others. Nevertheless, such a sub-optimal choice of n leads to the same improvements in complexity as the optimal value. Thus, based on our complexity analysis, 2D AP outperforms 1D AP in all operations and, as the word size m increases, it exhibits further performance and efficiency improvements over the 1D AP.

Algorithm analysis. We further apply the method to derive the architecturally-determined complexity for four benchmarks running on 1D AP, 2D AP, and CPU. We choose BitCount [6], blowfish, qsort, and kNN [2] in our evaluation and summarize the results in Table 2. From these results, we can see that *1)* The CPU performance has a higher complexity with respect to the total input dataset size $\mathcal{N}$ resulting from the constant-time lookups of AP. Of note also is the lower AP complexity of qsort which has irregular memory accesses, implying that the AP is suitable for applications with poor data locality. That is because AP significantly reduces the latency of accessing the different memory locations using associative lookups and in-memory processing. *2)* With the exception of the average case of qsort, the CPU energy efficiency drops faster than both APs as the dataset size increases. *3)* Compared to 1D AP, 2D AP achieves lower runtime and EDP complexities due to its lower arithmetic operation complexities. Consequently, 2D AP achieves the highest performance and efficiency for these algorithms. These trends are likely to hold in similar data-intensive algorithms that perform highly-parallel arithmetic operations.

6 Evaluation and Validation

In this section, we validate our analysis using full system simulations and experimentally verify the scaling trends of runtime, storage, and EDP of key primitive operations and algorithm benchmarks on CPU, 1D AP, and 2D AP.

Experimental Setup. For our full system simulations, we use heavily modified gem5 [1] and DRAMSim2 [9] simulators to obtain cycle-accurate performance results for both AP architectures. We extract the physical parameters of the AP array from 180nm 1D1R cross-point cells [8], scaled to the 22nm technology node. The

TABLE 2
Comparison of CPU, 1D AP, and 2D AP for Arithmetic Operations and Algorithms, Using the Analytical Method

		Arithmetic Operations		Algorithms			
		Add	Mult	BitCount	qsort (average/worst case)	blowfish	kNN
Runtime	CPU			$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N}\log\mathcal{N})$ / $\mathcal{O}(\mathcal{N}^2)$	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$
	1DAP	$5m \sim \mathcal{O}(m)$	$4m^2 \sim \mathcal{O}(m^2)$	$\mathcal{O}(m^2)$	$\mathcal{O}(m\mathcal{N})$	$\mathcal{O}(m)$	$\mathcal{O}(m^2)$
	2DAP	$6\sqrt{2m} \sim \mathcal{O}(\sqrt{m})$	$6m\sqrt{2m} \sim \mathcal{O}(m\sqrt{m})$	$\mathcal{O}(m\sqrt{m})$	$\mathcal{O}(\sqrt{m}\mathcal{N})$	$\mathcal{O}(\sqrt{m})$	$\mathcal{O}(m\sqrt{m})$
Space	CPU			$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$
	1DAP	$3m + 1 \sim \mathcal{O}(m)$	$4m \sim \mathcal{O}(m)$	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$
	2DAP	$3m + 9\sqrt{\frac{m}{2}} \sim \mathcal{O}(m)$	$4m + 9\sqrt{\frac{m}{2}} \sim \mathcal{O}(m)$	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$	$\mathcal{O}(\mathcal{N})$
EDP	CPU			$\mathcal{O}(\mathcal{N}^2)$	$\mathcal{O}(\mathcal{N}^2\log^2\mathcal{N})$ / $\mathcal{O}(\mathcal{N}^4)$	$\mathcal{O}(\mathcal{N}^2)$	$\mathcal{O}(\mathcal{N}^2)$
	1DAP	$28m^2 \sim \mathcal{O}(m^2)$	$24m^4 \sim \mathcal{O}(m^4)$	$\mathcal{O}(m^3\mathcal{N})$	$\mathcal{O}(m^2\mathcal{N}^3)$	$\mathcal{O}(m^2\mathcal{N})$	$\mathcal{O}(m^3\mathcal{N})$
	2DAP	$46m\sqrt{2m} + 54m \sim \mathcal{O}(m\sqrt{m})$	$76m^3\sqrt{m} + 12m^3 \sim \mathcal{O}(m^3\sqrt{m})$	$\mathcal{O}(m^2\sqrt{m}\mathcal{N})$	$\mathcal{O}(m\mathcal{N}^3)$	$\mathcal{O}(m\mathcal{N})$	$\mathcal{O}(m^2\sqrt{m}\mathcal{N})$

$\mathcal{N}$: dataset size, m: word size.

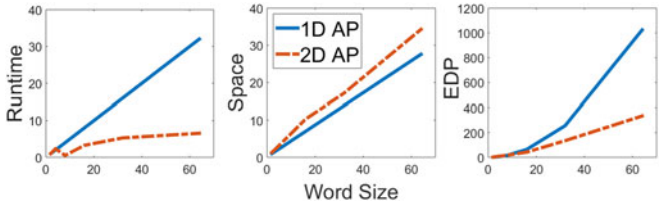

Fig. 2. The runtime, space, and EDP for addition on 1D AP and 2D AP. Results have been normalized to the first 1D AP result.

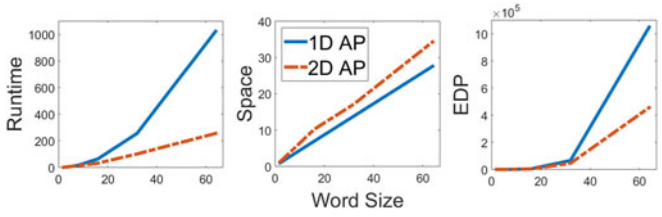

Fig. 3. The runtime, space, and EDP for multiplication on 1D AP and 2D AP. Results have been normalized to the first 1D AP result.

TABLE 3
Turning Points of $\mathcal{N}$ when 1D/2D AP Outperform CPU

	BitCount	qsort	blowfish	kNN
1D AP	9.5 kB	3.3 MB	5.3 kB	25 MB
2D AP	3.6 kB	1.2 MB	4.1 kB	12 MB

latency and energy numbers are obtained through Cadence and HSPICE simulations with the 22nm PTM low-power model [7]. We also use an 8-core CPU, used in previous works [4], as a baseline.

Validation on arithmetic operations. First, we evaluate runtime/storage/EDP of running fixed-point additions and multiplications on both APs for various word sizes (m). In Figs. 2 and 3, we confirm that the experimental results follow the same scaling trends as our theoretical projections, summarized in Table 2. Furthermore, we show that the performance and EDP gaps between 2D AP and 1D AP increase when increasing word size. These improvements come at a small storage cost which increases at a slower, $\mathcal{O}(\sqrt{m})$ rate, as was predicted.

Validation on benchmark algorithms. We validate the proposed model and complexity analysis using simulation. We extract the model constants for the CPU based on a previous work [3], which calculates the CPU cycles for running the algorithm and handling cache miss hazards. We perform cycle-accurate simulations to extract the AP constants including C_{comp} and E_{comp} as well as C_i and E_i to account for the bandwidth latency and energy consumption of the memory lanes. Using these constants and the proposed model, we calculate the turning point when APs outperform CPU, as shown in Table 3. We then confirm these results through experiments shown in Figs. 4, 5, and 6. In Figs. 4 and 5, we compare performance, storage space, and EDP of running BitCount and KNN on CPU, 1D AP, and 2D AP, for different dataset sizes ($\mathcal{N}$). As expected, with the increase in $\mathcal{N}$, the CPU runtime of BitCount and KNN increases linearly, while the runtimes of both APs remain constant. This results in the AP performances surpassing CPU. The value of $\mathcal{N}$ corresponding to this point is consistent with our prediction (Table 3). Moreover, both APs show lower EDPs than CPU and the gap between the CPU and APs increases for larger datasets. Comparing the two APs, 2D AP has lower runtime and EDP due to its faster and more efficient arithmetic operations. Finally, in Fig. 6, as the word size of BitCount increases, the runtime and EDP of 2D AP increase more slowly than 1D AP. Overall, these experiments confirm that 2D AP achieves better performance and energy efficiency than CPU and 1D AP.

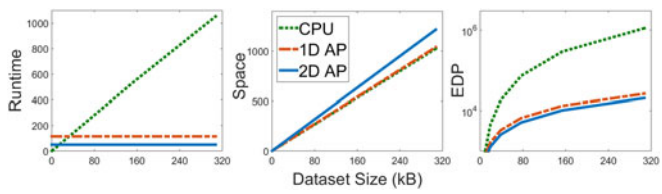

Fig. 4. The runtime, space, and EDP for CPU, 1D AP, and 2D AP running BitCount, with respect to dataset size. Results have been normalized to the first CPU result.

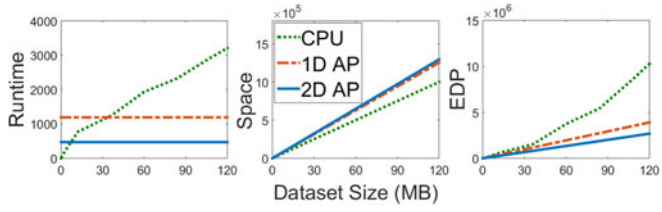

Fig. 5. The runtime, space, and EDP for CPU, 1D AP, and 2D AP running kNN, with respect to dataset size. Results have been normalized to the first CPU result.

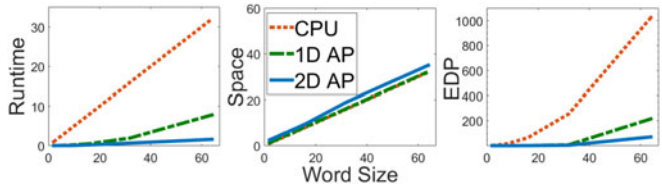

Fig. 6. The runtime, space, and EDP for CPU, 1D AP and 2D AP running BitCount, with respect to word size. Results have been normalized to the first CPU result.

7 Conclusion

In this paper, we developed an analytical method based on the concept of architecturally-defined complexity, which effectively addressed the limitations of simulation-based approaches for comparing radically different architectures. Using this method, we gained better insights into the performance bottlenecks of the traditional AP. We then proposed a new machine model called 2D AP, which alleviated these limitations. Finally, we experimentally validated the proposed method using simulation and confirmed that the results match our theoretical projections.

References

[1] N. Binkert, et al., "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2024716.2024718

[2] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *CoRR*, vol. abs/1607.02565, 2016, http://arxiv.org/abs/1607.02565

[3] S. Eyerman, K. Hoste, and L. Eeckhout, "Mechanistic-empirical processor performance modeling for constructing CPI stacks on real hardware," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2011, pp. 216–226.

[4] Q. Guo, X. Guo, Y. Bai, and E. Ipek, "A resistive TCAM accelerator for data-intensive computing," in *Proc. 44th Annu. IEEE/ACM Int. Symp. Microarchit.*, Dec. 2011, pp. 339–350.

[5] Q. Guo, X. Guo, R. Patel, E. Ipek, and E. G. Friedman, "AC-DIMM: Associative computing with STT-MRAM," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 189–200, Jun. 2013.

[6] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. 4th Annu. IEEE Int. Workshop Workload Characterization*, Dec. 2001, pp. 3–14.

[7] Nanoscale Integration and Modeling Group—Arizona State University, "Predictive technology model (PTM)." (2006). [Online]. Available: http://ptm.asu.edu/, Accessed on: 30-Jul.-2016.

[8] A. Kawahara, et al., "An 8Mb multi-layered cross-point reram macro with 443MB/s write throughput," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2012, pp. 432–434.

[9] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan. 2011.

[10] 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.