Reviriego, P., Liu, S., Ertl, O., Niknia, F. & Lombardi, F. (2022, 1 julio). Computing the Similarity Estimate Using Approximate Memory. IEEE Transactions on Emerging Topics in Computing, 10(3), 1593-1604.

DOI: 10.1109/tetc.2021.3109559

Computing the Similarity Estimate Using Approximate Memory

Pedro Reviriego[1], Shanshan Liu[2], Otmar Ertl[3], Farzad Niknia[2] and Fabrizio Lombardi[2]

Abstract—In many computing applications there is a need to compute the similarity of sets of elements. When the sets have many elements or comparison involves many sets, computing the similarity requires significant computational effort and storage capacity. As in most cases, a reasonably accurate estimate is sufficient, many algorithms for similarity estimation have been proposed during the last decades. Those algorithms compute signatures for the sets and use them to estimate similarity. However, as the number of sets that need to be compared grows, even these similarity estimation algorithms require significant memory with its associated power dissipation. This paper for the first time considers the use of approximate memories for similarity estimation. A theoretical analysis and simulation results are provided; initially it is shown that similarity sketches can tolerate large bit error rates and thus, they can benefit from using approximate memories without substantially compromising the accuracy of the similarity estimate. An understanding of the effect of errors in the stored signatures on the similarity estimate is pursued. A scheme to mitigate the impact of errors is presented; the proposed scheme tolerates even larger bit error rates and does not need additional memory. For example, bit error rates of up to 10^{-4} have less than a 1% impact on the accuracy of the estimate when the memory is unprotected, and larger bit errors rates can be tolerated if the memory is parity protected. These findings can be used for voltage supply scaling and increasing the refresh time in SRAMs and DRAMs. Based on those initial results, an enhanced implementation is further proposed for unprotected memories that further extends the range of tolerated BERs and enables power savings of up to 61.31% for SRAMs. In conclusion, this paper shows that the use of approximate memories in sketches for similarity estimation provides significant benefits with a negligible impact on accuracy.

Index Terms—Similarity, Approximate Computing, Minhash, Memories

1 Introduction

Computing the similarity of sets of data is a fundamental problem in computer science [1] and is needed in many applications, for example to detect documents that are similar [2]. An exact computation of the similarity requires a computational effort that is at least linear with the size of the sets and also the storage of all sets. Therefore, when the sets are large, an exact computation may not be feasible with reasonable computational overheads. For many applications, a nearly accurate estimate of the similarity is still feasible. For example, when comparing two documents, knowing that they are 90% similar with a 1% accuracy loss is sufficient in many cases [2]. This can be exploited to design algorithms for similarity estimation that significantly reduce the computation and storage complexities. Estimates are based on probabilistic guarantees for accuracy. Many algorithms have been proposed over the last decades and are widely used in computing systems [1], [2], [3], [4], [5]. Most of these algorithms for similarity estimation compute fixed-sized signatures on each set and then compare the signatures to estimate the similarity. This makes the computation and storage needed for similarity estimation constant and independent of the size of the sets. The signature is formed by several signature components that are typically obtained by computing a value on each element of the set and taking the minimum value across all the elements. The value can be a hash function on the element [2], [5] or more complex functions that depend on random distributions [3]. The functions used are different for each signature component, so that they are also different. For a given signature component, the value is determined by a single element (the one that takes the minimum value) and if that element is the same on both sets, then the signature components match. Therefore, the more elements the two sets have in common, the more likely that the signature components match. Typically, hundreds of signature components or even a few thousands are used [3], each having a fixed size (ranging from 64 bits to just a few bits or even a single bit [5]). As the number of sets to compare grows, current algorithms become inadequate in terms of memory, especially when signatures must be stored in fast memories (as they need to be frequently accessed); therefore, it is of interest to reduce the memory used to store the signatures of the sets.

Several techniques to reduce the memory in terms of both area and power dissipation have been proposed in technical literatures over the years. Recently, some algorithms have been proposed by allowing errors in the stored data [6]; for example, either the refresh rate of DRAMs can be reduced [7], or the supply voltage can be scaled down for SRAMs [8] to save a significant amount of power. However, these techniques also introduce additional increments in the

[1]*Pedro Reviriego is with Universidad Carlos III de Madrid, Leganés 28911, Madrid, Spain. email:* revirieg@it.uc3m.es
[2]*Shanshan Liu, Farzad Niknia and Fabrizio Lombardi are with Northeastern University, Dept. of ECE, Boston, MA 02115, USA. emails:* ssliu@ece.neu.edu, niknia.f@northeastern.edu, lombardi@ece.neu.edu
[3]*Otmar Ertl is affiliated with Dynatrace Research, Linz, Austria. email:* otmar.ertl@dynatrace.com

observed bit error rate when accessing the memory, because the read data is no longer reliable and/or correct. These memories (also known as approximate memories) trade off power/size overhead for an acceptable loss of accuracy; they are commonly used together with other approximate computing blocks in applications that are inherently error tolerant [9], [10].

An alternative is to use Error Correction Codes (ECCs) to mitigate errors, such that the memory is no longer approximate and can be used in traditional (exact) computing systems [11]. This approach is effectively provided when the overhead introduced using ECCs is significantly smaller than the savings due to reducing the refresh rate, or the supply voltage of the memory. However, for applications that are error-tolerant, the use of approximate memories usually results in larger overhead reductions. Examples of such applications are signal/image processing [12] and machine learning [13] in which errors on the least significant bits can be tolerated by the algorithm with a small impact on the result. From the previous discussion, the use of approximate memories for finding a similarity estimate may significantly reduce overheads, for example in terms of power dissipation. However, to the best of the authors' knowledge, the use of approximate memories has never been considered for this application (i.e., similarity estimation data sketches) as well as for other probabilistic data structures. The closest related works have studied the impact of radiation induced soft errors on probabilistic data structures used for counting [14] and cardinality estimate [15], so showing that the algorithmic features of the data sketches can be used to protect against errors in the memory. The use of approximate memories for similarity estimation of large graphs has been reported in [16] but utilizing exact computation; moreover no sketches were used.

In this paper, the use of approximate memories for similarity estimation is studied for the first time. Errors on the signatures stored in memory are considered; the analysis and error injection results show that sketches for similarity estimation intrinsically tolerate large bit error rates when using an unprotected memory. The range of bit error rates that have a small impact on the similarity estimate, can also be further extended by using a parity protected memory. After these findings, a scheme to mitigate the impact of errors is presented; the proposed scheme tolerates even larger bit error rates and does not need additional memory. These results mean that the use of an approximate memory significantly reduces overheads, for example the power dissipation needed for computing the similarity estimate. In more general terms, they also indicate that there is a significant potential in using approximate computing in probabilistic data structures and motivates further research in this area.

The rest of the paper is organized as follows. Section 2 covers the background on similarity estimation and approximate memories. Section 3 first analyzes the impact of errors on algorithms for similarity estimation and then considers two sketches (minhash and b-bit minhash) as examples. The impact of errors is evaluated by error injection in section 4 to validate the previously presented theoretical analysis and provide insights into the bit error rates that can be tolerated by widely used configurations of similarity sketches. A protection scheme that further reduces the impact of errors on the similarity estimate is proposed in section 5. The benefits in terms of memory performance (size and power dissipation) that can be obtained by exploiting the error tolerance for different configurations are studied in detail in section 6 for different memories. The paper ends with the conclusion and some proposals for future work.

2 Preliminaries

This section initially provides a brief overview of existing sketches for similarity estimation and then, it discusses different alternatives to compute the similarity estimate using approximate memories and the trade-offs that are involved in such implementations.

2.1 Sketches

Most sketches for finding the similarity estimate use two steps/processes: i) the signature pre-computation process, in which several signature components on each set are computed and then stored in a memory; ii) the similarity process, which is performed by comparing the signature components of different sets. Figure 1 illustrates these two processes for finding the similarity estimate on two sets; to describe them more formally, the notation that is used in this paper is defined as follows:

: function used for the signature component.
: value used for the signature component.
: signature component.
: number of signature components.
: number of elements in the set.
: number of bits in each signature component.
: number of matching signature components when comparing two sets.
: Jaccard similarity coefficient.
: bit error rate of the memory.

In the signature pre-computation process, each signature component is typically obtained by computing a hash function across the elements of the set and keeping the minimum value of the function across all of them; thus, by using different hash functions, several signature components can be computed; they are used for finding the similarity estimate. As shown in Figure 1, to compute the signature component for a set , the function is computed for all the elements in and a value typically for all in is selected from those . Then, the signature is extracted from . In some cases, the signature component is the same as the value but in other cases, it can be different; for example, contains only the lower bits of [5]. Each set needs to store signature components of bits each, so memory bits are required. Typically, to provide a good estimate, is in the range of a few

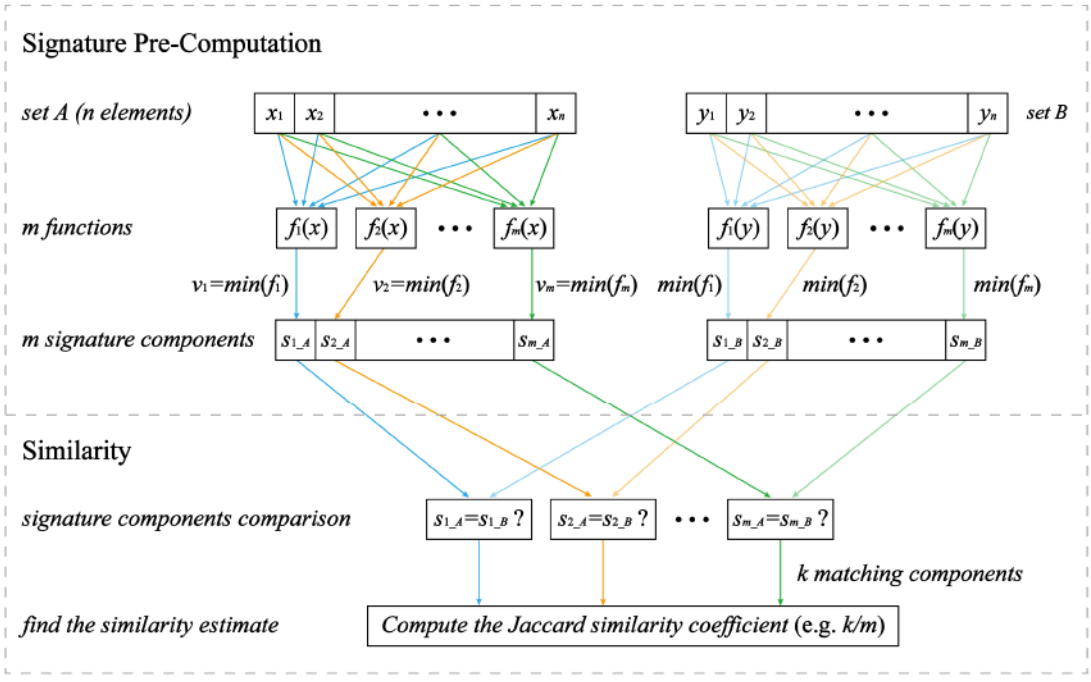

Fig. 1: Illustration of the process for finding a similarity estimate on two sets.

hundreds to a few thousands and b has from a few bits to 64 bits; therefore, the signature of a set requires few Kbits of memory. In many applications, the signature computation is performed once and all components are stored; once the similarity estimate is required by an application, the signature components of the related set/sets are read from the memory for comparison.

As shown in Figure 1, for estimating the similarity of two sets, the number of matching signature components is checked first and then used to compute an estimate of the Jaccard similarity coefficient [2]; therefore, when comparing a set against a group of existing sets, the signatures of the sets can be pre-computed and stored. When a new set is available for comparison, its signature components are computed and compared with those of the stored sets. To better illustrate the similarity sketches, next minwise hashing [2] and b-bit minwise hashing [5] are discussed in more detail and will be used as examples in the rest of the paper.

2.1.1 Minwise Hashing

In minwise hashing or simply minhash [2], the functions f_i are hash functions, the value v_i is computed as the minimum over the hashes of all the elements in the set and the signature component is equal to v_i. Consider a set S with four elements x, y, w, z and eight bit hash functions that for the first hash function f_1 take the values $f_1(x) = 01011111, f_1(y) = 11011001, f_1(w) = 01100001, f_1(z) = 00110101$. Then, the minimum value is $f_1(z) = 00110101$ and thus, the first signature component is $s_1 = 00110101$. Hash functions typically have 32 or 64 bits to ensure that the probability that two different elements have the same hash value, is very small. The first bits of the signature components tend to be zero as the minimum hash value is selected, while the lower ones are uniformly distributed if the hash functions behave properly. To estimate the Jaccard similarity coefficient of two sets, the number of matching signature components k is divided by m, i.e., $J_{est} = \frac{k}{m}$.

2.1.2 b-bit Minwise hashing

To reduce the number of bits per signature component and thus, the memory size, b-bit minwise hashing uses also hash functions. However, it takes as value the minimum but then uses as signature components only the lower b-bits of the value. In the example considered previously, a 4-bit minwise hashing would use as signature component the lowest four bits of $f_1(z) = 00110101$ such that $s_1 = 0101$. As discussed previously, the lower bits tend to be uniformly distributed, so that the two signature components created by two different elements have a probability $1/2^b$ of taking the same value. This probability is no longer negligible when b has a small value, for example four; therefore, this has to be compensated when estimating the Jaccard similarity coefficient of two sets. In more detail, the estimate of the Jaccard similarity coefficient of two sets that have k matching signature components is given by: $J_{est} = \frac{\frac{k}{m} - 2^{-b}}{1 - 2^{-b}}$.

2.2 Approximate Memories

For some error-resilient applications that deal with a very large amount of data (such as signal and image processing or neural networks), the use of approximate memories is an alternative solution to address the challenging issue of storage overhead. As introduced previously, approximate

memories can reduce both the area and power of a storage system by introducing errors in the stored data. Therefore, they have a great potential to reduce the implementation complexity for finding a similarity estimate when an acceptable accuracy loss is permitted in the comparison and the number of stored signatures is very large.

There are several approximate storage techniques suitable for different types of memories to meet these performance requirements. For example, in SRAMs (as usually utilized as caches), efficient access operations and low power dissipation are required. Scaling down the supply voltage Vdd in SRAMs can significantly reduce the power [8], [17], therefore some works have studied the relationship between the voltage and the bit error rate (BER) in data (due to the failure of some read/write operations) to determine an appropriate scaling factor, thus establishing the best arrangement for potential power savings. For example, Figures 2 and 3 show the dynamic/static power saving when different read BERs are introduced due to scaling in the supply voltage of a 28 nm SRAM (with a nominal Vdd of 1.0 V) and a 32 nm SRAM (with a nominall Vdd of 0.9 V), respectively. Note that these plots are obtained by extracting the simulation results[1] reported in [18] and [19] as well as considering the general scaling factor between power and supply voltage in CMOS technology (i.e., the dynamic power scales quadratically with Vdd, while the static power scales linearly with Vdd [20], [21]). As shown in Figures 2 and 3, 51.00% (30.00%) dynamic (static) power can be reduced in the 28 nm SRAM and 43.42% (24.78%) dynamic (static) power reduced in the 32 nm SRAM, by introducing a read BER of 10^{-4}. In addition to voltage scaling, a scheme to further reduce power dissipation consists of disabling the bit lines for some of the least significant bits, because they have a very small impact on the data integrity [22]. In addition, since the required cache size is increasing in today's processors to improve the performance, the cache hierarchy can also be enlarged; however, this solution is fast reaching its limit due to the large volume of hardware involved. In some chip multiprocessors (such as those by Intel), caches can account for nearly 30% of the area of the entire chip [23]); therefore, some other approximate methods have been utilized to increase the storage density by organizing the data stored in the caches [24], [25].

For DRAMs that are usually employed as main memory, approximate techniques mainly focus on the refresh operation to save energy [7]. Since the refresh operation is executed periodically in a DRAM to hold the stored data, this operation accounts for 10% to 25% of the entire power dissipation for a 2 GB to 32 GB DRAM, as reported in [26], [27], [16] and it may account for a smaller fraction in more advanced DRAMs [16]; moreover, the ratio also depends on the specific application (e.g., 12% when used for a LDPC decoder and 1.6% when used for image processing [16]). Overall, it is clear that an increase in the refresh period is helpful for reducing the entire DRAM power. However, when the refresh interval increases, the logic state stored in some memory cells may erroneously change, and so introducing retention errors. The BER when changing the refresh period (thus power savings) has been extensively studied. Figure 4 presents the refresh power saving at different BERs in a 20 nm approximate 4 GB DDR4 DRAM (with a nominal refresh interval of 64 ms); this plot is obtained by extracting the measurement results[2]

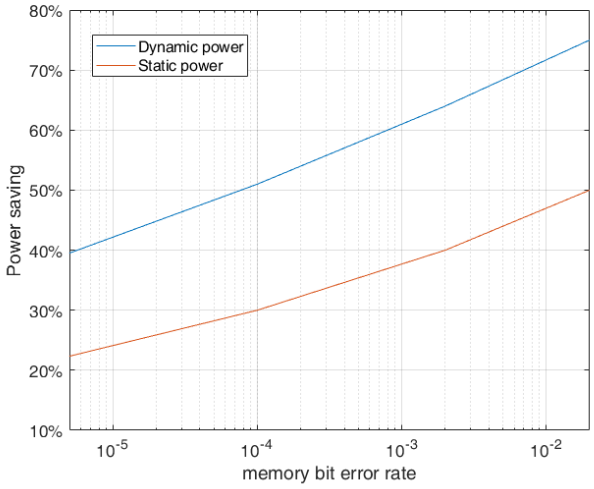

Fig. 2: Power saving for different memory bit error rates due to supply voltage scaling in a 28 nm approximate SRAM.

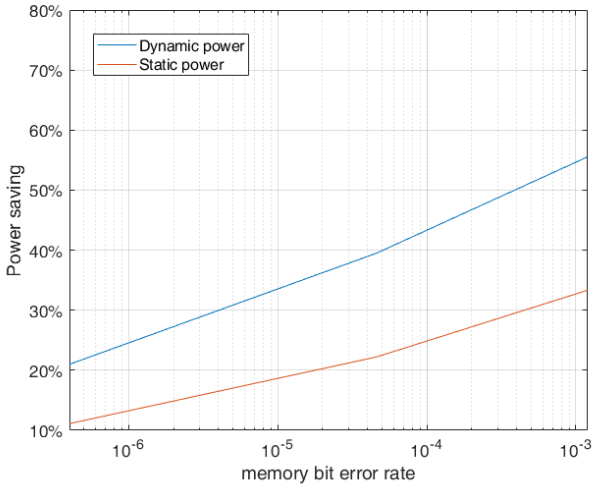

Fig. 3: Power saving for different memory bit error rates due to supply voltage scaling in a 32 nm approximate SRAM.

1. The simulation reported in [18] is at the read-critical process corner with room temperature; the simulation in [19] uses an in-house simulator based on ModelSim with PTM models. For additional simulation results please refer to the corresponding references.

2. The measurements reported in [28] for DDR4 SO-DIMM modules use a custom platform with room temperature; a true-cell bit storage is utilized, and the worst-case is considered when analyzing the BER. More details are available in [28].

reported in [28] as well as considering the percentage of power saving as proportional to the increase in the refresh period. As per Figure 4, when a BER higher than ^{9} is introduced, the refresh power is saved by over 97.44%; since in such a DRAM the refresh power accounts for 8.26% of the entire power dissipation (estimated by using Micron Power Calculator [29]), a saving of at least 8.05% is achieved for the entire DRAM power when introducing the BERs considered in Figure 4. As an alternative solution, the refresh period can also be customized for different DRAM cells. For example, a framework presented in [30] allocates critical and non-critical data in different parts of the DRAM; the regular refresh rate is then set for the critical data part, while the non-critical data part is refreshed at a substantially shorter rate. In this case, the power dissipation can be reduced by generating some inexact results for the non-critical data. This method can also be implemented by dividing the memory pages into different quality bins (so associate error rates) [31]. Moreover, the refresh operation can even be disabled for some dedicated applications (e.g., a wide I/O DRAM for similarity computation of graphs [16]), in which errors have negligible impact.

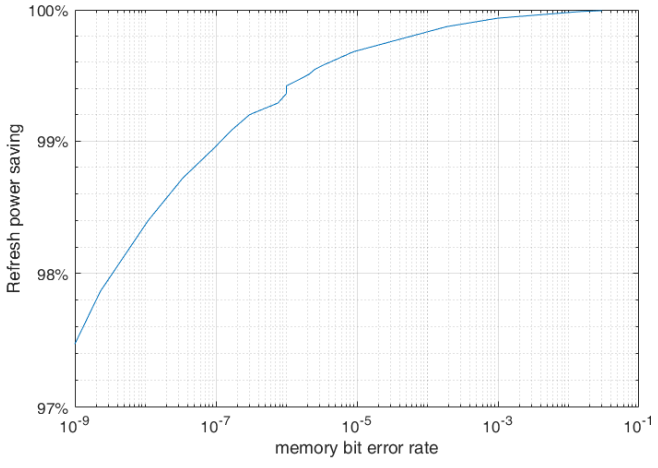

Fig. 4: Refresh power saving for different memory bit error rates due to refresh period relaxing in a 20 nm approximate DRAM.

In addition to conventional CMOS memories, approximate techniques have also been considered for emerging memories (e.g., phase-change memories and magnetic RAMs). These emerging memories store multiple data bits in a single cell, offering high storage density, so they are promising candidates for on-chip memories (e.g., replacing SRAMs and DRAMs in some cases). However, they have a disadvantage in terms of the latency and energy for the write operations. Therefore, some approximate techniques are focusing on reducing the write precision to improve the performance and endurance of emerging memories [32], [33].

Overall, if the introduced errors in the stored data can be tolerated by the application (in this case, the similarity estimate), approximate memories should have an excellent potential to reduce the area and power overheads when storing a large volume of data. This will be analyzed in more detail in the following sections.

3 Impact of Errors

To evaluate the use of approximate memories for computing the similarity estimate, an understanding of the impact of memory errors on the similarity sketches is initially required. The error model of this paper considers only a memory that when reading, a bit has a certain probability of returning an incorrect value (so, errors or faults in computational/arithmetic units such adders, are not considered). This is given by the Bit Error Rate (BER). As discussed in the previous section, the BER of memories increases as the supply voltage or refresh rate are reduced. Therefore, a knowledge of the impact of different BERs on the accuracy of the similarity estimate is needed to trade-off power dissipation for accuracy. The main reason to focus on read errors is that when computing the similarity estimate, once the signatures have been stored in memory, only read operations are needed. However, the analysis can be easily extended to consider errors in write operations as those would have a similar impact and therefore, they can be modeled by increasing the BER to account for both read and write errors.

Consider comparing two signature components and when a signature component has an error; so without loss of generality instead of , is read. If originally , the error causes the signature components to be mismatched as now . This reduces the similarity estimate. Conversely, if originally , then the error can make the signature components match, such that now . However, this only happens when the error flips (changes) exactly the bits on which and are different. This is highly unlikely in the original minhash as signature components have many bits (at least 32); two different components will have several bits that are different with high probability and therefore, having errors on all those bits is highly unlikely[3]. Instead, if the signatures have only a few bits (like in b-bit minhash), the probability will be larger. From this discussion, errors can both increase and decrease the number of matching signatures, thus affecting the similarity estimate.

The impact of errors also depends on whether the memory has any error detection or correction mechanism. For example, parity or Single Error Detection Double Error Correction (SEC-DED) are widely used to protect memories from errors [34], [35]. For a parity protected memory, when a signature component has an odd number of bit errors, the error is detected and thus, that pair of signature components can be discarded. Next, the impact of errors is analyzed first for an unprotected memory and then for a protected memory.

3. For d different bits the probability is approximately $(\text{BER})^d$

3.1 Unprotected Memory

To better analyze the impact of the errors, consider two sets that have matching signature components for an error free memory of which are true matches and are false matches (due to for example, the use of a small number of bits for the signature components as in b-bit minwise hashing). Then, when using an approximate memory, the probability that there is an error in a pair of signature components, is given by ; so, the number of matching signature components is approximately given by:

(1)

where is the probability that an error makes two signatures that were different, equal.

This general formula can be simplified for each specific similarity sketch. For example, for minhash, is negligible as discussed in the previous section. As it is also very unlikely that two signatures are the same (except in one or two bits), is also negligible for the BER values of interest. Therefore, for minhash the estimate can be simplified to:

(2)

The value of for signature components of bits and a BER can be estimated as follows:

$$^{2} \tag{3}$$

These equations show that minhash can tolerate significant bit error rates with limited impact on accuracy. For example, for a BER of ^{4} and a signature size of bits, the loss given by equations 2 and 3 is less than one percent. This suggests that the sketch is quite resilient to errors.

For b-bit minwise hashing, can be approximately given by $\frac{}{2^{b}}^{t}$ and can also be approximated by assuming single bit errors are dominant (as applicable to the case for the BERs of interest). This leads to the following approximation:

(4)

Like minhash, the sketch can tolerate errors; in b-bit minwise hashing, the signatures are smaller, so, is lower for the same BER. As an example, for a BER of ^{4} and a signature size of 64 bits minhash has a of approximately 1.27% while for b-bit hashing with signatures of 8 bits, is only 0.16%.

In summary, it seems that similarity sketches can tolerate large BERs even when the memory is unprotected. This is valid, because errors affect only a fraction of the signature components and if that fraction is small, the impact on accuracy is also small. However, if the BER increases, then, the impact on the accuracy is significant. For those BERs, a protected memory should be used.

3.2 Protected Memory

In the analysis so far, only unprotected memories have been considered; however, it is possible to detect and correct errors in memories by adding protection. For example, a single parity check bit can be used to detect single bit errors and a Single Error Correction Double Error Detection (SEC-DED) code corrects single bit errors and detects double bit errors [36]. Implementing protection incurs in an overhead; parity check bits must be added to each protected memory word and then encoding and decoding circuitry is needed to write and read from the memory. The memory overhead of a parity check is a single bit per word, while a SEC-DED code requires typically $_{2}$ bits for a data word of bits, i.e., for example, 8 bits for words of 64 bits. Therefore, in this case, a given protection technique is viable only if it provides larger savings (by enabling the use of larger BERs) than the overheads it introduces. This is evaluated in the next sections.

For now, consider that a protection (e.g. a single parity check) is implemented that enables us to detect signature components that have incurred in an error. Then the similarity estimate can be computed by using only the pairs of signature components that are error-free. So, the pairs of signature components for which there is an error are discarded and not used in the estimate; hence, instead of , is used in the formulas and includes only matching signatures that are error-free. As an example, if 1% of the signature component pairs have an error, we would basically have a sketch with 99% of the signatures that marginally reduces the accuracy. This can be used to extend the BER that the sketch can tolerate. Consider a memory that is protected with parity; in such case only when both signatures have no errors or an even number of errors, the pair is used for the estimate.

As per [37], a signature component has a probability of no error or an even number of errors given by:

$$- \quad - \qquad ^{+1} \tag{5}$$

The probability that a signature component has no error is

$$^{+1} \tag{6}$$

Therefore, the probability that a signature component has no error after detection is

$$\frac{^{+1}}{\frac{1}{2} \quad \frac{1}{2} \qquad ^{+1}} \tag{7}$$

Finally, the probability that a pair has an error after detection is

$$\left(\frac{^{+1}}{\frac{1}{2} \quad \frac{1}{2} \qquad ^{+1}} \right)^{2} \qquad ^{2} \tag{8}$$

The approximation is a second order Taylor approximation at .

In the case of protection that corrects errors, the analysis is even simpler as the errors are effectively removed, and the similarity sketch does not consider them. However, correction comes at a large overhead in terms of memory overhead and as shown in the next sections, it does not seem to be interesting for similarity sketches implemented on approximate memories for the scenarios considered.

3.3 Discussion

As per the analysis of the impact of errors on similarity sketches considering both the protected and unprotected memories, errors tend to reduce the estimated similarity. This occurs because in general, an error that reduces the number of matching signature components, is more likely than one that increases it; for single bit errors, an error affecting a pair of matching signature components always makes them different. Instead, for an error that makes two mismatched signature components equal, all different bits and only these bits have to be affected by errors; this occurs in most cases with an extremely low probability, and particularly it is negligible when the signatures consist of a large number of bits as explained previously. Therefore, errors tend to introduce a negative bias in the estimated similarity that is proportional to the real similarity, as seen for example in equation 2.

As the impact of an error tends to be one sided and can to some extent be predicted, it is then possible to compensate it. For example, consider again equation 2 if the BER is known, its effect on the estimate can be compensated by simply dividing by . This can further improve the accuracy of the estimate, so reducing the impact of errors. However, to provide an effective compensation, the BER must be accurately estimated. Although the BER can be estimated by simulation, for a given memory in the field it depends on factors such as temperature or supply voltage that cannot be fully controlled. Therefore, accurately compensating the impact of errors seems challenging. The first option is to perform partial compensation using a best-case BER that corresponds to a combination of several parameters (so resulting in the lowest BER). Another option is to measure the BER experimentally by for example checking the number of parity errors and use the result for compensation; this enables a good compensation if the BER measurements are accurate. In the remaining part of this paper, error compensation is not considered and is left for future work. The main reason to do so is that even without error compensation, significant savings are achieved by using approximate memories to implement sketches for similarity estimation as shown next.

4 Evaluation

In this section, the impact of errors on the computation of the similarity estimate is evaluated by simulation with two objectives:

- Validate the analysis presented in the previous section.
- Identify the values of the bit error rate that can be tolerated by similarity sketches.

These results will then be used in the next sections to study the potential benefits when employing different types of approximate memory. As in the previous section, two sketches are considered: minhash [2] and b-bit minwise hashing [5]. They have both been implemented using ideal hash functions and random sets of elements. Then, error injection simulation has been performed to assess the impact of errors. In more detail, the first two random sets of ^{16} are generated with a known Jaccard similarity coefficient and the signature components for each of the similarity sketches are computed and stored. Then, errors are randomly injected on the stored signatures with a given bit error rate and the similarity estimate is computed.

The average results over 100 runs for signature components and are shown in Figure 5 for minhash with 32-bit signatures and in Figures 6, 7, 8 for b-bit minwise hashing with 8, 4 and 2 bits respectively. The figures also include the values of the theoretical estimates derived in section 3 labeled as "theo" in the plots. The first observation is that the theoretical estimates provided by equations 2 for minhash and 4 for b-bit minwise hashing mostly match the simulation results in all cases. Therefore, those equations can be used to estimate the impact of errors on both sketches. A second observation is that large bit error rates can be tolerated in all cases. So, for an unprotected memory, minhash tolerates a BER of ^{4} with almost no impact on the similarity estimate, while for b-bit minhash the value goes up to ^{3}. Those values are even larger when the memory is protected with parity, reaching ^{3} and ^{2} respectively.

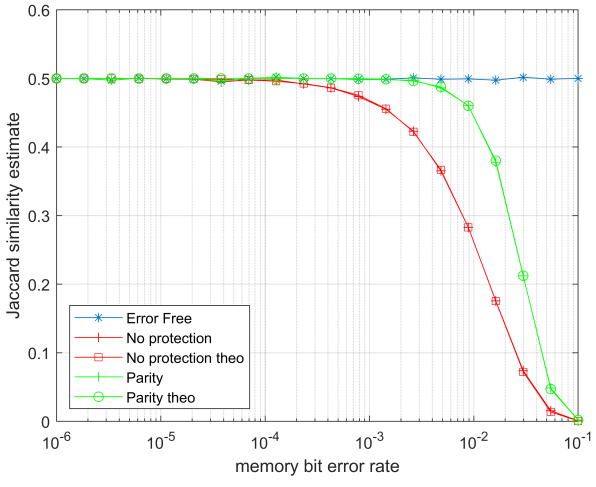

Fig. 5: Impact of errors on a minhash sketch with and signature components of bits.

In addition to looking into the average estimate, it is of interest to analyze the worst-case relative deviation introduced by errors. The maximum relative error between the Jaccard coefficient estimator obtained in the error-free case and the one obtained with a given bit error rate (over 1000

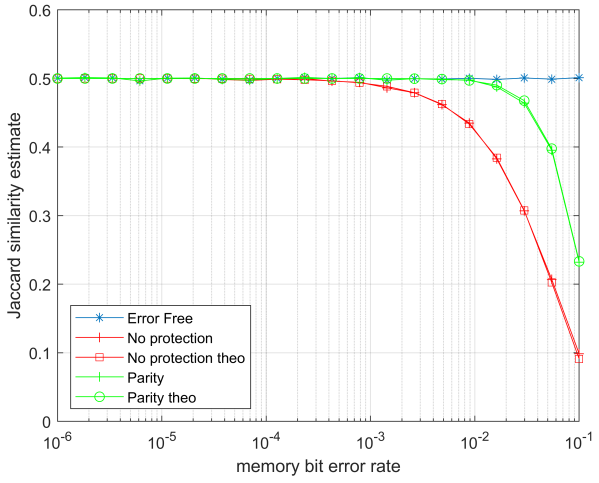

Fig. 6: Impact of errors on an 8-bit minwise sketch with .

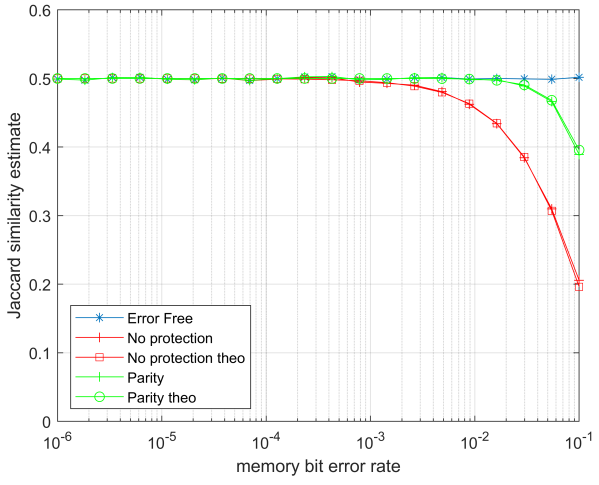

Fig. 7: Impact of errors on a 4-bit minwise sketch with .

runs) are recorded and the results are shown in Table 1 for an unprotected memory and in Table 2 for a parity protected memory. Consider that in the worst case, the goal is to have a 1% deviation in the similarity estimate. Then the results show that the minhash sketch can tolerate a bit error rate of ^{5} and the b-bit minwise hashing can go up to ^{4} when the memory is unprotected. For a parity protected memory, the tolerated bit error rates increase to ^{4} for minhash and to ^{3} (although the deviation is slightly above 1%) for b-bit minhash. However, the implementation overhead of a parity per signature for a b-bit minwise hashing is larger because is significantly smaller. Therefore, the use of parity is less interesting for b-bit minwise hashing as confirmed in the case studies discussed in the next sections.

Finally, it is worth looking at how errors affect the distribution of the Jaccard similarity estimate values for unprotected and parity protected memories. To this end, the values for each run are recorded and the histograms for

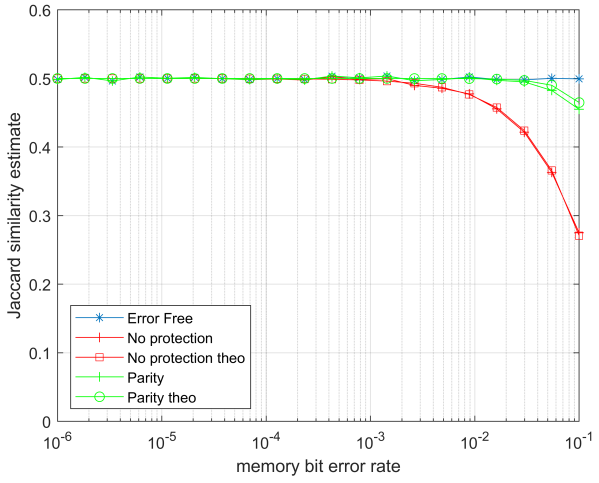

Fig. 8: Impact of errors on a 2-bit minwise sketch with .

TABLE 1: Worst case relative deviation of the Jaccard similarity coefficient estimator for an unprotected memory

bit error rate	^{5}	^{4}	^{3}	^{2}
minhash, b= 32	-0.60%	-2.21%	-9.83%	-54.42%
b-bit, b= 8	-0.41%	-0.82%	-3.58%	-20.12%
b-bit, b= 4	-0.40%	-0.81%	-2.62%	-12.65%
b-bit, b= 2	-0.29%	-0.83%	-2.58%	-11.02%

the different configurations are analyzed. For minhash, the results for different bit error rates are shown in Figures 9, 10, 11, 12. As per these figures, all the histograms are almost the same when the bit error rate is small. As the error rate increases, the unprotected and parity protected histograms start to move to the left due to the bias introduced by the errors. This effect is more pronounced for an unprotected memory. Without considering the bias, the shapes of these histograms are similar for all configurations unless when the error rate is large (i.e., ^{2}). In this case, the parity protected memory seems to have a wider distribution. This occurs because when a parity error is detected, the pair of signature components affected by the error are removed from the estimate, which effectively reduces the value of and thus the accuracy of the estimate. Instead, the unprotected memory seems to have a narrower distribution; this is due to most of the signature components pairs having errors and thus not contributing to in the similarity estimate. Therefore, their contribution is fixed to zero and that reduces the variability of the estimate. The results for b-bit minwise hashing are similar and are omitted for brevity.

5 Distance-One Comparison

The results presented in the previous section show that similarity estimation sketches have some inherent resilience to errors. This resilience is lower as the number of bits in the signature increases (i.e., because it is more likely to be

TABLE 2: Worst case relative deviation of the Jaccard similarity coefficient estimator for a parity protected memory

bit error rate	5	4	3	2
minhash, b= 32	-0.31%	-0.86%	-2.83%	-19.55%
b-bit, b= 8	-0.21%	-0.53%	-1.35%	-4.96%
b-bit, b= 4	-0.18%	-0.38%	-1.27%	-3.30%
b-bit, b= 2	-0.12%	-0.36%	-1.04%	-3.07%

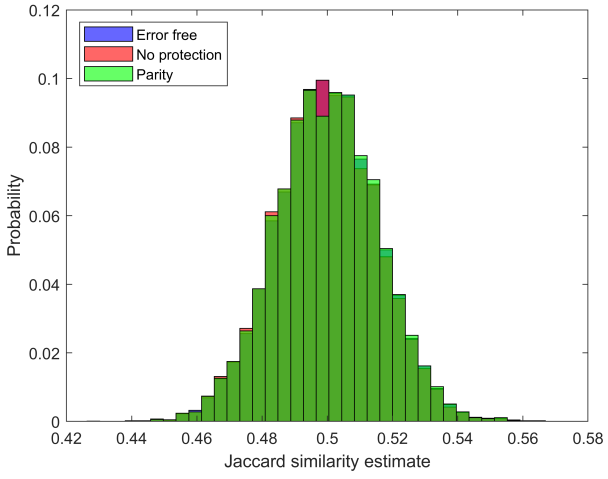

Fig. 9: Distribution of the similarity estimates over 10,000 runs for the minhash sketch with and a bit error rate of 5.

in error). At the same time, the larger the signature, the smaller the probability that two signatures are different in just one, or two bits. This can be exploited to further reduce the impact of errors on the similarity sketch as discussed next.

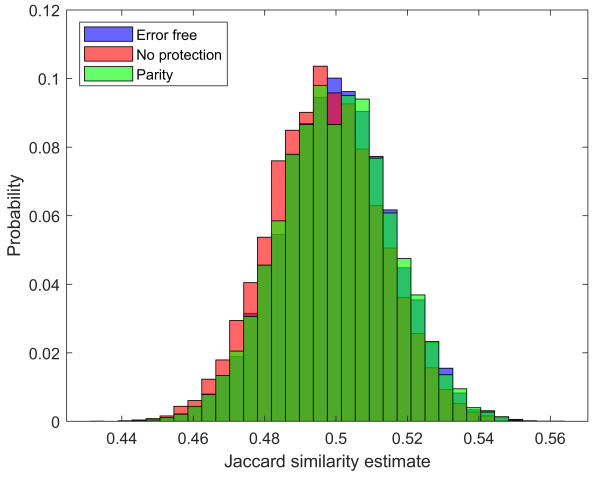

Fig. 10: Distribution of the similarity estimates over 10,000 runs for the minhash sketch with and a bit error rate of 4.

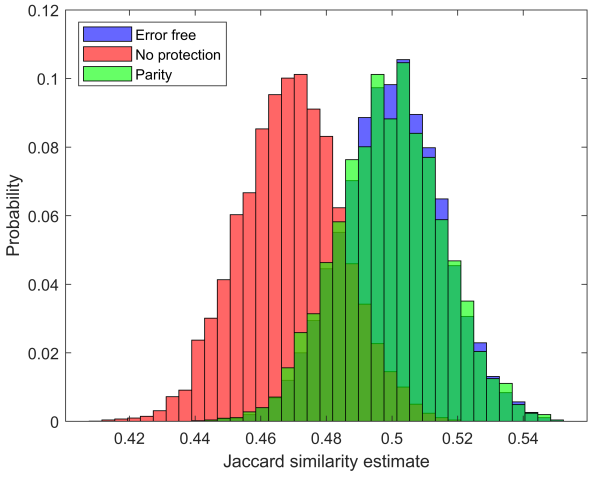

Fig. 11: Distribution of the similarity estimates over 10,000 runs for the minhash sketch with and a bit error rate of 3.

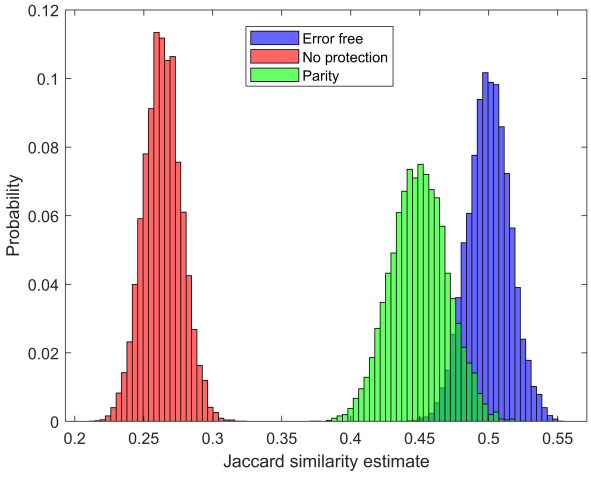

Fig. 12: Distribution of the similarity estimates over 10,000 runs for the minhash sketch with and a bit error rate of 2.

The proposed scheme relies on the feature that instead of using an exact match for the signature components, a distance-one comparison can be used, i.e., two signature components match when they are equal or differ in a single bit. The goal is that when an error changes a bit on a pair of matching signature components, the distance-one comparison will still generate a match.

However, the distance-one comparison would match two error free signatures that differ only in a single bit; therefore, for the scheme to be useful the probability that two error free signatures are at a distance-one should be negligible. This should be the case for example for minhash because the number of signature bits is large.

Same as in the previous section, simulations have been run for the sketches using an unprotected memory and distance-one comparators to match the signatures. The re-

sults are shown in Table 3 that also includes the results from Tables 1 and 2 for minhash with for comparison purposes.

The use of distance-one comparison is very effective in reducing the impact of errors on the cardinality estimate; also, it is better than parity protection. This is interesting as the proposed distance-one scheme does not need to add additional memory bits. The BER that can be tolerated with less than 1% impact on the accuracy of the estimate, is ^{3}, so better than for parity. However, it is important to note that this only holds when the probability of having two different signatures different in only one bit is negligible. This is only the case when is large and thus the proposed distance-one technique is not applicable to b-bit minwise hashing unless is large.

TABLE 3: Worst case relative deviation of the Jaccard similarity coefficient estimator of minhash with b = 32 bits when using the proposed distance-one scheme

bit error rate	^{5}	^{4}	^{3}	^{2}
exact match	-0.60%	-2.21%	-9.83%	-54.42%
parity protected	-0.31%	-0.86%	-2.83%	-19.55%
distance-one	-0%	-0.20%	-1.01%	-17.86%

6 Case Studies and Applicability

In this section, the power dissipation required for implementing similarity sketches is initially evaluated showing that the memory is the largest contributor to power dissipation; then, the advantage of using different approximate memories for reducing the power dissipation is analyzed.

6.1 Power Dissipation

As discussed previously, for many applications, once the signature components for different sets are computed and stored in a memory (i.e., the signature pre-computation process, that is usually performed once), the process to find the similarity estimate is executed frequently if there is such a need in the system/application. Therefore, the power dissipation mainly consists of two parts:

The power dissipation due to the memory read operations to provide the stored signatures of the sets being compared.
The power dissipation due to the circuity implementing signature comparison and for calculating the Jaccard coefficient estimate.

To illustrate the relative importance of memories on the power dissipation, the minhash with and (i.e., each set has 1024 32-bit signature components) is taken as a case study to evaluate the power required for estimating the similarity of two sets. A 2K words SRAM with a word width of 32-bit (i.e., for storing all signatures of two sets) is implemented by using Synopsys Memory Compiler and the circuit for signature comparison (with both the traditional exact match scheme and the proposed distance-one scheme) and the Jaccard coefficient estimator is implemented in Verilog; the designs are mapped to a 32 nm library using the Synopsys Design Compiler with the default toggle rate of 0.5. The place-and-route for the design has been implemented using Cadence Innovus. Table 4 shows the synthesis results (corresponding to the final circuits obtained after place-and-route) of the power dissipation required for the different hardware parts; the reported dynamic power is for an entire similarity estimate computation (instead of for each clock cycle). The estimation circuit includes a comparison module, a decision module, and a coefficient computation module. In the estimation process, each pair of 32-bit signature components are first read out from the memory and then compared in the comparison module of the estimation circuit; if they match, increases (so performed in the decision module). Once all 1024 pairs of signature components have been checked sequentially, the Jaccard similarity coefficient is obtained in the coefficient computation module. Therefore, the dynamic power for the memory reported in Table 4 is obtained by multiplying the power for each read operation by 2048; for the estimation circuit it is obtained by multiplying the power for both the comparison module and the decision module by 1024 and then adding the power for the coefficient computation module. As per Table 4, the memory requires over 98% of the entire power dissipation. Therefore, the use of approximate memories is an efficient solution to reduce power dissipation when implementing similarity sketches.

TABLE 4: Synthesis results of power dissipation (mW) required for the minhash with and

power	memory	estimation circuit	
		exact match	distance-one
dynamic power	3642.81	29.23	50.23
static power	1.09	0.03	0.04

6.2 Power Saving Using Approximate Memory

The benefits of using approximate memories for the similarity sketches can be assessed by combining the power savings versus the bit error rates (given by Figures 2 to 4 for SRAMs and DRAMs which are extracted from the simulation/measurement results reported in [18], [19], [27], [28]) and the tolerated bit error rates obtained by simulation results provided in previous sections. Since most operations to the memory that stores the signatures, are read (for comparison), the approximate emerging memories that focus on write operations are not considered in this section.

For example, assume that a worst case deviation in the similarity estimate of 1% is allowed; then, the minhash with , can tolerate a bit error rate (BER) of ^{5} for an unprotected memory when using an exact match and of ^{3} when using distance-one comparisons. In the unprotected scheme, the dynamic (static) power savings are 42.24% (24.00%) for the 28 nm SRAM (as per Figure 2),

and 33.31% (18.33%) for the 32 nm SRAM (as per Figure 3), respectively; in the distance-one scheme, the dynamic (static) power savings are 61.31% (37.80%) for the 28 nm SRAM (as per Figure 2), and 54.66% (32.67%) for the 32 nm SRAM (as per Figure 3), respectively. For the 20 nm DRAMs, the refresh (entire) power dissipation can be reduced by more than 97.44% (8.05%) when the BER is as low as 10^{-9} (as per Figure 4). Therefore, the power dissipation can be dramatically reduced in both memory types.

For b-bit minhash, a bit error rate of 10^{-4} is tolerated for an unprotected implementation, leading to savings of 51.00% dynamic power (30.00% static power) for the 28 nm SRAM, 43.42% dynamic power (24.78% static power) for the 32 nm SRAM, and 99.84% (8.25%) refresh power (entire DRAM power) for the 20 nm DRAM, respectively. For DRAMs, there is less incentive to provide protection, because the reduction of power is similar for all considered bit error rates; instead, for SRAMs it is of interest to consider parity protection. Adding parity protection permits the use of larger bit error rates, close to 10^{-3}. However, to assess the benefits of such scheme, the overhead due to parity must be further analyzed; an additional overhead of 12.5%, 25% and 50% in memory power dissipation is introduced for b = 8, 4 and 2 when using parity protection, that can even be larger than the saving achieved by using an approximate memory. Therefore, adding parity is not effective when b is small and results in a larger overall power dissipation. This is due to the small size of the signatures that implies a large relative overhead for parity protection. The situation is worse if more powerful error correction codes (such as the SEC-DED codes) are used. In this case, the additional overhead for 32-bit signature components is over 21% and for 8-bit signature components over 62%. This clearly shows that the use of error correction codes is not attractive.

As a summary, the analysis presented in this paper on the benefits of using approximate SRAMs and DRAMs for similarity estimation confirms that significant reductions in power dissipation can be obtained.

7 Conclusion and Future Work

This paper has considered for the first time the use of approximate memories to implement sketches for similarity estimation. Results (based on a theoretical analysis and simulation) show that similarity sketches can tolerate large bit error rates and thus, they can benefit from using approximate memories without substantially compromising the accuracy of the similarity estimate. This paper has shown that bit error rates of up to 10^{-4} have less than a 1% impact on the accuracy of the estimate when the memory is unprotected, and larger bit errors rates can be tolerated if the memory is parity protected. This can be used for supply voltage scaling and for increasing the refresh time in SRAMs and DRAMs respectively; reductions of power dissipation by up to 51.00% and 8.25% have been accomplished. Based on those initial results, an enhanced implementation has also been proposed for unprotected memories that further extends the range of tolerated BERs and enables power savings of up to 61.31% for SRAMs. This paper has shown that the use of approximate memories for data sketches for similarity estimation provides significant benefits with a negligible impact on accuracy.

Since the goal of this paper targets a scheme using approximate memories for data sketches for similarity estimation, the detailed memory design with a given sketch and the related evaluation are left for future work. Moreover, there will be more interesting topics to be investigated: i) exploring the compensation of the impact of errors to achieve even larger savings in memory overhead; ii) similarly integrating the proposed scheme on a memory that needs to also support the storage of exact data; iii) possibly existing schemes that support different regions in a memory with different accuracy can be used to integrate the proposed approximate similarity estimator in a general-purpose computing system. More generally, considering the use of approximate memories to implement other sketches and probabilistic data structures seems also an interesting area for future work.

Acknowledgments

This work was supported by the ACHILLES project PID2019-104207RB-I00 and the Go2Edge network RED2018-102585-T funded by the Spanish Ministry of Science and Innovation and by the Madrid Community research project TAPIR-CM grant no. P2018/TCS-4496. The research of S. Liu and F. Lombardi is supported by NSF under CCF-1953961 and 1812467 grants.

References

[1] S. Dahlgaard, M. B. T. Knudsen, and M. Thorup, "Fast similarity sketching," *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, Oct 2017.

[2] A. Broder, "On the resemblance and containment of documents," *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*.

[3] O. Ertl, "Probminhash – a class of locality-sensitive hash algorithms for the (probability) jaccard similarity," *IEEE Transactions on Knowledge and Data Engineering*, 2020 (Ealy Access).

[4] A. Shrivastava, "Simple and efficient weighted minwise hashing," in *Advances in Neural Information Processing Systems 29*, pp. 1498–1506, 2016.

[5] P. Li and C. König, "b-bit minwise hashing," *Proceedings of the 19th international conference on World wide web - WWW '10*, 2010.

[6] Y. Chen, X. Yang, F. Qiao, J. Han, Q. Wei, and H. Yang, "A multi-accuracy-level approximate memory architecture based on data significance analysis," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 385–390, 2016.

[7] I. Bhati, M. Chang, Z. Chishti, S. Lu, and B. Jacob, "Dram refresh mechanisms, penalties, and trade-offs," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 108–121, 2016.

[8] S. Kirolos and Y. Massoud, "Adaptive sram design for dynamic voltage scaling vlsi systems," in *2007 50th Midwest Symposium on Circuits and Systems*, pp. 1297–1300, 2007.

[9] W. Liu, F. Lombardi, and M. Shulte, "A retrospective and prospective view of approximate computing: Point of view," *Proceedings of the IEEE*, vol. 108, no. 3, pp. 394–399, 2020.

[10] K. Nepal, S. Hashemi, H. Tann, R. I. Bahar, and S. Reda, "Automated high-level generation of low-power approximate computing circuits," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 1, pp. 18–30, 2019.

[11] G. Yalcin, E. Islek, O. Tozlu, P. Reviriego, A. Cristal, O. S. Unsal, and O. Ergin, "Exploiting a fast and simple ecc for scaling supply voltage in level-1 caches," in *2014 IEEE 20th International On-Line Testing Symposium (IOLTS)*, pp. 1–6, 2014.

[12] S. Ataei and J. E. Stine, "A 64 kb approximate sram architecture for low-power video applications," *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 10–13, 2018.

[13] D. T. Nguyen, N. H. Hung, H. Kim, and H. Lee, "An approximate memory architecture for energy saving in deep learning applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 5, pp. 1588–1601, 2020.

[14] P. Reviriego, J. Martínez, and M. Ottavi, "Soft error tolerant count min sketches," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 284–290, 2021.

[15] P. Reviriego, J. Martinez, O. Rottenstreich, S. Liu, and F. Lombardi, "Remove minimum (rm): An error-tolerant scheme for cardinality estimate by hyperloglog," *IEEE Transactions on Dependable and Secure Computing*, 2020 (Early Access).

[16] M. Jung, É. Zulian, D. M. Mathew, M. Herrmann, C. Brugger, C. Weis, and N. Wehn, "Omitting refresh: A case study for commodity and wide i/o drams," in *Proceedings of the 2015 International Symposium on Memory Systems*, pp. 85–91, 2015.

[17] A. Kumar, J. Rabaey, and K. Ramchandran, "Sram supply voltage scaling: A reliability perspective," in *2009 10th International Symposium on Quality Electronic Design*, pp. 782–787, IEEE, 2009.

[18] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester, and M. Alioto, "Sram for error-tolerant applications with dynamic energy-quality management in 28 nm cmos," *IEEE Journal of Solid-state circuits*, vol. 50, no. 5, pp. 1310–1323, 2015.

[19] Y. Emre and C. Chakrabarti, "Energy and quality-aware multimedia signal processing," *IEEE transactions on Multimedia*, vol. 15, no. 7, pp. 1579–1593, 2013.

[20] T. Mudger, "Power: A first-class architectural design constraint," *Computer*, vol. 34, no. 4, pp. 52–58, 2001.

[21] N. H. Weste and D. Harris, *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[22] F. Frustaci, D. Blaauw, D. Sylvester, and M. Alioto, "Approximate srams with dynamic energy-quality management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 6, pp. 2128–2141, 2016.

[23] D. Kadjo, H. Kim, P. Gratz, J. Hu, and R. Ayoub, "Power gating with block migration in chip-multiprocessor last-level caches," in *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pp. 93–99, IEEE, 2013.

[24] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelgänger: a cache for approximate computing," in *Proceedings of the 48th International Symposium on Microarchitecture*, pp. 50–61, 2015.

[25] M. Sutherland, J. San Miguel, and N. E. Jerger, "Texture cache approximation on gpus," in *Workshop on Approximate Computing Across the Stack*, 2015.

[26] W.-K. Cheng, P.-Y. Shen, and X.-L. Li, "Retention-aware dram auto-refresh scheme for energy and performance efficiency," *Micromachines*, vol. 10, no. 9, p. 590, 2019.

[27] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "Raidr: Retention-aware intelligent dram refresh," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 1–12, 2012.

[28] D. M. Mathew, M. Schultheis, C. C. Rheinländer, C. Sudarshan, C. Weis, N. Wehn and M. Jung, "An Analysis on Retention Error Behavior and Power Consumption of Recent DDR4 DRAMs," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 293–296, IEEE, 2018.

[29] Micron, DRAM, *System Power Calculators*. 2014.

[30] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: saving dram refresh-power through critical data partitioning," in *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, pp. 213–224, 2011.

[31] A. Raha, H. Jayakumar, S. Sutar, and V. Raghunathan, "Quality-aware data allocation in approximate dram," in *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pp. 89–98, IEEE, 2015.

[32] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 3, pp. 1–23, 2014.

[33] A. Ranjan, S. Venkataramani, Z. Pajouhi, R. Venkatesan, K. Roy, and A. Raghunathan, "Staxcache: An approximate, energy efficient stt-mram cache," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pp. 356–361, IEEE, 2017.

[34] Z. Azad, H. Farbeh, A. M. H. Monazzah, and S. G. Miremadi, "Aware: Adaptive way allocation for reconfigurable eccs to protect write errors in stt-ram caches," *IEEE Transactions on Emerging Topics in Computing*, vol. 7, no. 3, pp. 481–492, 2019.

[35] M. Ottavi, S. Pontarelli, D. Gizopoulos, C. Bolchini, M. K. Michael, L. Anghel, M. Tahoori, A. Paschalis, P. Reviriego, O. Bringmann, *et al.*, "Dependable multicore architectures at nanoscale: The view from europe," *IEEE Design & Test*, vol. 32, no. 2, pp. 17–28, 2015.

[36] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 124–134, 1984.

[37] S. Ross, *A First Course in Probability 8th Edition, (theoretical exercise 15 in chapter 4)*. Pearson, 2009.