Eliminating Energy of Same-Content-Cell-Columns of On-Chip SRAM Arrays

Bushra Ahsan*, Lorena Ndreu*, Isidoros Sideris*, Yiannakis Sazeides*, Sachin Idgunji† and Emre Özer†
*University of Cyprus †ARM

***Abstract*—[1] This work proposes to reduce energy by avoiding access to columns of on-chip SRAM arrays whose cell contents are all 1s or all 0s. We refer to this dynamic phenomenon as the Same-Cell-Content-Column (SCC-column). Analysis reveals that SCC-columns occur frequently in several processor arrays, such as tag arrays of L1 caches, TLBs and predictors. An interval based scheme that employs one bit per column is proposed to track whether we have a SCC-column. We explain how a SCC-column can be leveraged to reduce the energy needed for SRAM read and write accesses. Experimental analysis for a specific processor configuration reveals that the proposed scheme detects SCC-columns effectively. The potential energy savings of the proposed approach at 32nm often exceeds 40% for several processor arrays.**

I. Introduction

The continuous miniaturization of devices on silicon chips has provided designers the opportunity to place more functionality per unit area. Unfortunately, the scaling of other key design parameters has not followed suit. In particular, to avoid power increase, voltage and frequency have scaled slower than area scaling [1], [2], [3]. These conflicting trends have rendered paramount the development of effective techniques that harness power across all computing layers. As a result, power minimization has been at the forefront of architectural research for more than a decade [4].

On-chip array structures such as caches, TLBs and predictors consume significant die area and power and have received the attention of a plethora of microarchitectural proposals for power reduction. A category of such techniques, most relevant to our work, aim to leverage the properties of the dynamic content of the arrays to reduce power [5], [6], [7], [8].

In this work we identify a new dynamic phenomenon: array columns whose cells contents are the same. We refer to this as Same-Cell-Content-Column (SCC-Column). The key qualitative difference from previous work is that we consider non-uniformity across vertical array columns whereas all previous work, as far as we know, considered it horizontally for blocks/entries or at the granularity of a cell. Related work and its comparison with our technique is given in Section VI.

For motivation, we measure the frequency of SCC-column during the execution of various SPEC benchmarks for several arrays of a given processor configuration (presented in Section IV). Fig. 1 shows the average fraction of time a column is the same for a given array and program execution. For each array, results are sorted in ascending order to make the graph more readable [2]. The results vary depending on the benchmark and array but overall the data suggest the phenomenon is frequent. It occurs more than 80% for many cases, at least 35% of the time for half the benchmarks and all structures except the data cache and DTLB for which SCC-column is more rare.

We propose to leverage SCC-column to reduce read and write access energy. An interval based scheme that requires one bit per column is used to detect a SCC-column and the value it stores. When accessing a SCC-column some of the actions needed to read or write can be avoided thus saving energy. Experimental analysis reveals energy savings of more than 40% for several arrays.

The main contributions of this paper are: (i) the identification of the SCC-Column phenomenon, (ii) the introduction of an efficient mechanism for detecting and exploiting SCC-columns, (iii) a circuit explanation of how a SCC-column can save access energy, and (iv) a characterization of the potential savings of the proposed scheme for various benchmarks and arrays.

II. Mechanism

Let's assume that we have an array with N entries and B bits per entry, and each column is initialized to all 0's or all 1's. For each column in the array we use two extra bits initialized as follows: the *Same-Bit* that is set to 1 and the *Initial-Bit* set to the initial column value that is the same for all column cells. When the value of the *Same-Bit* is 1 it means that every cell in its corresponding column has the same value, and the value is equal to its *Initial-Bit* value.

On a read from a column with the *Same-Bit* set we do not read from the column in the array but read through a multiplexer the value from the column's *Initial-Bit*. Otherwise, when the *Same-Bit* is 0 we perform a normal column read. The access path for a read is shown in Fig. 2.a.

On a write access when the *Same-Bit* of a column is set, we check if the bit value to write to that column is equal to the column's *Initial-Bit*. If it is the same we inhibit the write access, otherwise, we reset the corresponding *Same-Bit* and perform a normal write. A normal write is also performed when the *Same-Bit* is not set. The write path is illustrated in Fig. 2.b.

[1]The research leading to this paper is supported by the European Commission FP7 project "Energy-conscious 3D Server-on-Chip for Green Cloud Services (Project No:247779 "EuroCloud")"

[2]The x-axis is not common between curves, i.e. a benchmark may correspond to different x-points

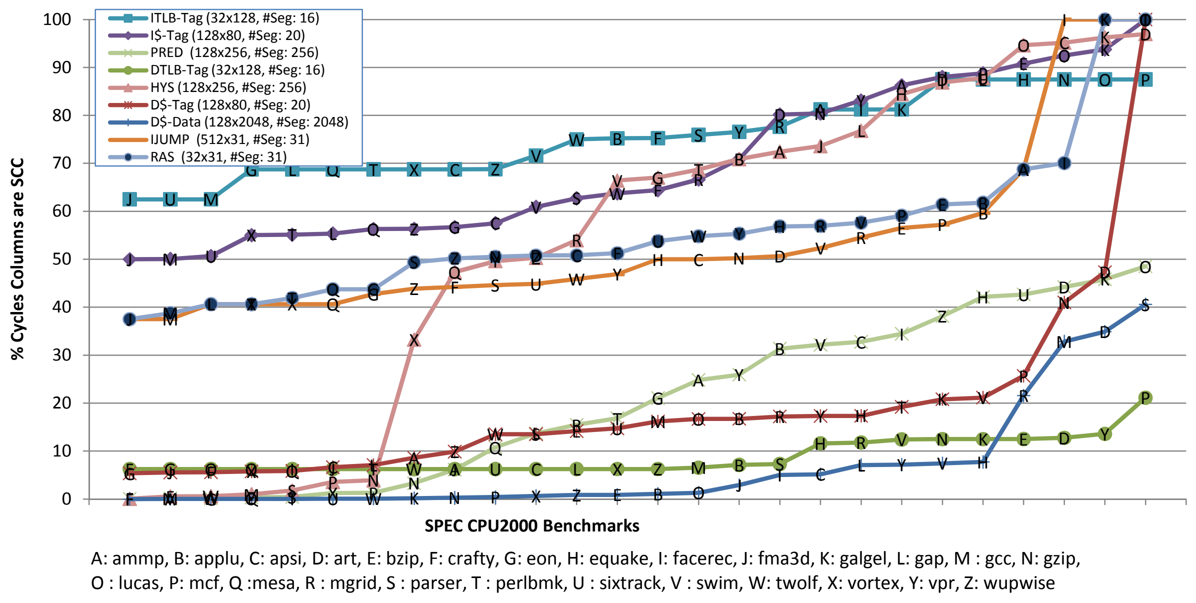


Fig. 1. Frequency of SCC-column for various Benchmarks and Arrays

Fig. 2.c shows a flow diagram that summarizes the different access scenarios.

A. How to Set Regularly the Same-Bit

The description of the mechanism so far implies that once a *Same-Bit* is reset there is no way for it to be set again which of course is extremely limiting. This can be overcome by reinitializing the array and the *Same-bit* vector at regular time intervals. The intuition behind such an approach is that a column may temporarily lose its uniformity and by reinitializing it we enable the mechanism to detect if it became a SCC-column again.

We illustrate the workings of the interval based scheme used to initialize an array multiple times with the aid of the example in Fig 3 that describes the behavior for one array column.

Let's assume that at the *Start Point* of an interval the column is logically initialized to all 0s, indicated by the *Initial-Bit* set to 0, and the *Same-Bit* for that column is set. Lets assume at a point X cycles into the interval the *Same-Bit* is reset, we refer to this as the *Change Point*. This means that 1 is written into one of the entries in that column, which violates the SCC property of the column.

All the accesses to the column that occurred between *Start Point* and *Change Point* are either reads or writes of a 0 value. During the interval between *Start Point* and *Change Point* the column is a SCC-column since it contains all 0s and we refer to it as *SCC-column interval*. Naturally during a *SCC-column interval* the *Same-Bit* remains set and read and writes are prevented from accessing the column.

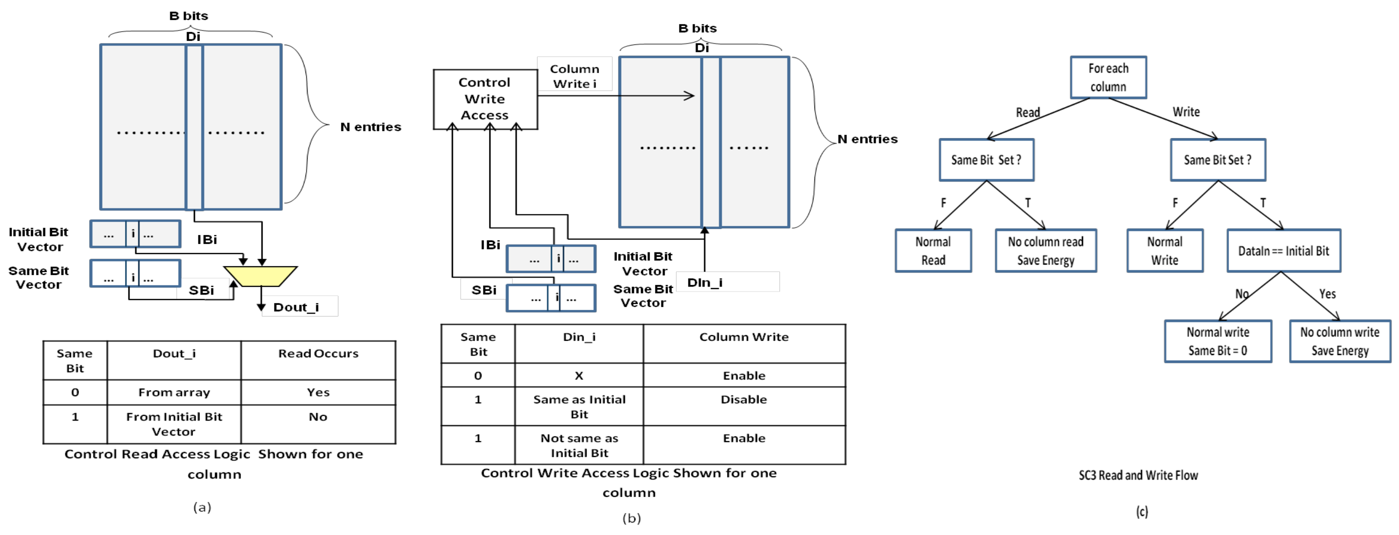
After the *Change Point* the *Same-Bit* is reset and remains reset until the *End Point*, the end of an interval. During this period all the accesses to the column are performed normally. Therefore, during this period we lose the opportunity of detecting a transition back to a SCC-column. Thus the proposed scheme can lose some of the potential in Fig. 1. However, reinitializing all the columns of the array at the end of each interval, facilitates the detection of columns that have transition to a SCC-column. This can occur because we reset cell content in a column that can contain a rarely accessed bit that breaks the uniformity in the column. Section 5 compares the performance obtained with continuous tracking for SCC-columns, shown in Fig. 1, vs interval-based tracking.

The key parameter of interest is the length between initialization intervals. If the interval is too short the mechanism will be able to detect quickly the possible transition of a column back to SCC-column but this may entail major performance penalty since reinitializing arrays may imply cold starts. Alternatively, if the interval is too long SCC-columns opportunities are lost but the impact of cold starts due to array reinitialization will be small.

Previous work [9] has shown that reinitializing core resources (flushing L1 caches and TLBs, resetting predictors state, but maintaining L2) has a negligible impact on performance if it is done every 1 million or more cycles. The reason for this is that usually the number of entries in these structures is in the order of 100 to 1000 entries and they can be warmed up relatively quickly as compared to the interval length. This is particularly true for flushed L1 caches that are backed-up by an inclusive L2. Furthermore, as we argue in the next subsection, an array reinitialization does not require an actual physical reinitialization of the content but only a semantic one.

B. How to Reinitialize On-Chip SRAM Arrays

To reinitialize L1 data and instruction caches, both data and tag arrays, and TLBs, we reset at the end of each interval their per block valid bits without changing the actual cache contents. Writeback caches need to write back their dirty blocks before they get invalidated. Additionally, the per column *Same-Bit* is also set to indicate a same column. The array content needs no reinitialization because when a write does not match a column's *Initial-Bit* it will perform a normal write and overwrite the cache content. Therefore, by block invalidation



Same Bit	Dout_i	Read Occurs
0	From array	Yes
1	From Initial Bit Vector	No

Control Read Access Logic Shown for one column

Same Bit	Din_i	Column Write
0	X	Enable
1	Same as Initial Bit	Disable
1	Not same as Initial Bit	Enable

Control Write Access Logic Shown for one column

Fig. 2. *(a) Read Access (b) Write Access (c) SCC Read and Write Flow*

we can assume the column initial state, indicated by the *Initial-Bit vector*, to be either all 0s or all 1s. The best value of the initial-bit vector depends on program phase, input data, and structure, however, analysis we performed suggests that a good compromise is to use an initial-bit vector that always contain 0s. Future work can explore a more dynamic scheme for selecting the *initial-bit vector* at each interval. However, for this work we assume the *initial-bit vector* value to be always zero and hence the *initial-bit* overhead can be completely eliminated resulting in just one bit overhead (*Same-Bit*) per column.

For prediction arrays reinitialization is very simple: just set the *Same-Bit* vector. This is sufficient since we do not need to worry about reading stale values (as incorrect predictions do not affect correctness). This has the interesting property that when a SCC-column of a predictor transitions to non-SCC its previous content becomes available for reading and thus the training overhead for the predictors is kept minimal.

C. Overheads

The detection of SCC-column in any structure results in area, performance and energy overheads.

Extra hardware is needed to implement the counter that keeps track of the number of cycles of an interval, but this overhead is small due to small interval length and can possibly have one counter shared for all structures. Also extra hardware is needed for the *Same-Bit* Vector and the *Initial Bit* Vector. As mentioned earlier the size of the two vectors depends on the number of columns that an array has which typically is small as compared to the array size. Additionally, it is possible for a *Same-Bit* to be shared by multiple physical columns. For the array configurations used in this paper (see Section IV), the state overhead of the vectors for each array we consider is never more than 1% as compared to each array size. In this paper the *Initial Bit* Vector is always set to zero, therefore, the *Initial Bit* overhead can be completely eliminated resulting in *only one bit per logical column overhead.*

On a read access a 2:1 Mux delay is added for each column to select whether to read from the array or not (see Fig. 2.a). On a write access extra delay is also added due to the need to check the *Same-Bit* value, the *Initial Bit* value and the new value before performing the write access (see Fig. 2.b). In order to account for the delay of the added muxes and latches, we do a breakdown of the delay using CACTI [10]. Subarray to latch to output driver is about 20-30% of the $CLK->Q$ delay for the memory, with about 40-20-40 split across each component. The latch can add 4-6% of the total delay and this delay can be reduced if output stage is tapered. However, this is a tradeoff in area, leakage and delay. Our experiments have shown that the multiplexer will impact the overall timing by adding 0.2-2% to the pre-mux delay. Larger size for the pass transistors in the mux will improve timing (lower resistance) but at the expense of area overhead.
Another overhead is due to write-backs of dirty blocks. But this is not needed often due to large intervals and limited number of writeback arrays. For example for structures like TLBs, I$ and predictors we do not need to write-back data. Furthermore, in data caches performing an early write back in some cases helps performance because if a miss occurs at an entry we do not need to write back data. Finally, write-through caches do not need to perform a write-back.
One more overhead added by our mechanism is a degradation at the beginning of each interval. Because we invalidate data at the beginning of each interval the first accesses to a structure are going to be misses or result in a misprediction. The performance implication of these extra misses and mispredictions depend on the size of the interval after which we should reset. As was shown in [9] resetting processor arrays around every 1 million or more cycles has low performance overhead. We have reconfirmed this observation and reported it in Section V.

III. Leveraging SCC to Save Energy

We first explain a normal column access and then describe access to a SCC column.

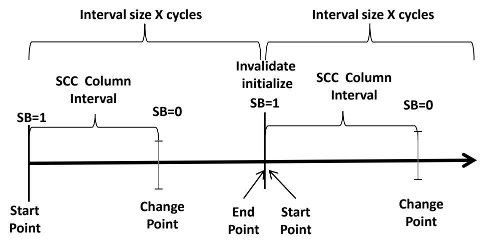


Fig. 3. *Interval Based Behavior*

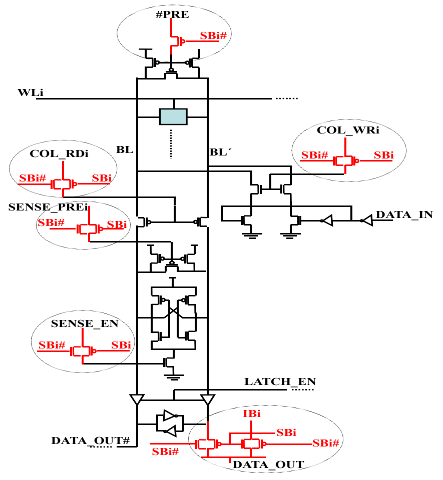


Fig. 4. *The control signals, multiplexer and associated drivers shown for a single cell [11]. The dotted circles show the Same-Bit and the added transistors that are used to control the signals when accessing SCC column. Energy is saved when precharging is disabled in columns and senseamps. Also, not driving data during write to SCC column saves energy.*

A. Normal Access

An array consists of rows and columns. In an SRAM based array, we have columns known as differential pair columns (bitlines BL and BL') and an associated wordline. This is shown in Fig. 4. The figure also shows the control signals that are necessary to perform the access, the sense amplifiers and the write drivers. For simplicity we show just one cell and one column pair.

The address and data are transferred to the array through the buses. Before a normal access starts, the columns are presumed to be in precharged state and ready to be accessed. The decoders decode the row address from the address bits and activate the associated wordline.

On a read access, the COL_RD selects the column to be read (since many columns can be sharing the sense amps and the associated drivers). Depending on the value of the cell, either BL or BL' is discharged. The $SENSE\_EN$ signal allows the sense amps to sense the values and output the data. Once the value has been read, the PRE and $SENSE\_PREi$ signals precharge the column and the sense amps back to the precharge voltage.

In a write operation, the COL_WR operation selects the column and data values are input through the $DATA\_IN$ drivers. Unlike read, in a write access, a complete differential has to occur for the value to be written (full swing as compared to half swing). Once the write is done, the discharged column is brought back to precharge voltage.

B. Access to SCC-column

If SCC column is detected, the normal access is bypassed. We explain the mechanism for read and write to SCC column and show the added circuitry to control them.

In a read, the Same-Bit indicates whether a column is SCC or non-SCC. If the column is in SCC state, we bypass normal read and use the Initial-Bit of the column as the read value. Since the read is not done, none of the two bitlines are discharged and hence they do not need to be precharged. Also there is no need for sensing the data, thus the sense amp is gated off. Thus no energy is spent in bitlines, senseamps and the associated multiplexers. The Same-Bit is used to turn off COL_RD, $SENSE\_EN$ and PRE signals as shown in the dotted circles at the left side of Fig. 4. If the column is non-SCC, the read is done in a regular fashion and bitlines are discharged.

In a write operation the following scenarios can occur: (i) The write is to a non-SCC column. In this case a regular write access is done. If after the write the column becomes SCC, it is not detected until the start of the next interval, (ii) The write is to a SCC column and the written value is the same as Initial-Bit. In this case the normal write is bypassed. The Same-Bit is used to deactivate the COL_WR signal, (iii) The write is to a SCC column and the written value is not the same as the Same-Bit. In this case a normal write has to be done and the control signals are resumed and data is driven in. In scenario (i) no energy is saved since column is accessed in a normal way. In scenario (ii) energy is saved since there is no need to write and drive the data in. Since full swing discharge is avoided to write the data, the resultant precharge is also avoided. In scenario (iii), a regular access is done and hence no energy is saved.

There are some implementation issues related to deactivating sense and write drivers during a write to SCC column. The contents of the cell can be corrupted if we lower the precharge for a SCC column and the associated wordline is activated. We thus propose to keep the columns floating in the write state, that is, one column at precharge voltage and the other discharged to avoid corrupting the cell contents. This is done during the resetting of the column to SCC at the start of the interval.

Another implementation issue could be that of leakage current corrupting the cell value in an SCC column since the bitline is not precharged. In traditional SRAM designs, bitlines remain precharged when bitcells are not accessed, resulting in subthreshold leakage current into the logical 0 sides of the SRAM cells and junction leakage current from the N+ regions connected to the bitlines. By allowing the bitline voltages to float in the write state, these leakage components are decreased and there is no path to leakage and hence the values are not corrupted.

C. Energy Savings

The dynamic energy breakdown of an SRAM array access is given by:

$$Edecoder + Ewordline + Eoutputdriver + Eamps + Eprecharge$$

Three components of energy that can not be eliminated by SCC are $Edecoder$, $Ewordline$ and $Eoutputdriver$. On an access to SCC column, $Eamps$ and $Eprecharge$ are eliminated.

The SCC-column energy overhead includes energy for updating the Same-Bit vector, and the switching energy for gating at the start of the intervals and for un-gating when SCC is terminated. Thus, this overhead is negligible when compared to the overall array energy over a large time interval. We do not determine this overhead in detail but rather assume it is a fixed fraction of the array energy and we report its implications on overall energy of the scheme in Section V.

IV. Framework

We use sim-alpha simulator [12] to evaluate SCC columns for different arrays: D$-tag, D$-data, RAS, JUMP, I$-tag, DTLB-tag, ITLB-tag and branch predictors. For benchmarks, we use the SPEC2000 benchmark suite. Each benchmark is run for 100 million instructions after fast forwarding to the representative regions. All benchmarks are used with their reference input sets.

For all structures except D$-data and branch predictor, we show results for logical columns which means the bits of different blocks at the same bit position belong to the same column. This means the logical column can span across blocks of different physical columns if the array is segmented into smaller sub arrays. This allows the Same-Bit and Initial-Bit per multiple physical columns (one logical column). The tradeoff is that we lose some potential since it is more likely for the column to be non-SCC.

For D$-data and branch predictor we show results by segmenting the logical columns. The D$-data is segmented according to physical ways. Branch predictor array is divided into 256 segments. This means we have 256 columns and 128 wordlines for prediction and hysteresis bit respectively. An entry is selected by using the upper 8 bits to choose the segment number and the lower 7 bits to choose the entry within the segment. We use segmentation for these structures since it is closer to real implementation in which the arrays are mostly squares. Segmenting the array increases the opportunity for more SCC columns but also increases the overheads. The size of each of the structure and the segmentation details are given in the labels of Fig. 1 and Fig. 5.

For the interval based scheme we initialize all arrays to zeroes (supported by the analysis of different structures and benchmarks)and experiment with an interval length of 1Million.

V. Results

A. SCC Column Detection

Figure 5 shows the number of cycles the columns in a structure are SCC using the interval based scheme. The structures fall into two categories. (i) The D$ and DTLB that lose the potential as compared to Fig. 1. (ii) The I$, ITLBs and predictors for which the potential is maintained. For structures falling in category (i), some of the potential is lost because columns switch between SCC and non-SCC during intervals and thus detecting SCC column only at the beginning of the interval means the potential is lost for the remaining interval. For structures falling in category (ii), the columns are mostly zeroes throughout execution and hence with the reset based scheme they preserve their SCC column potential.

B. SCC Energy Savings Results

We use CACTI version 6.0 [10] to calculate the energy savings per access at 32nm. We concentrate on dynamic energy.

To determine the implications of the energy overhead of the proposed scheme we show the average energy savings per access with three different fractions of fixed overhead 0%, 1% and 5% in Figure 6.

The structures for which the SCC phenomenon occurs a lot there are large energy savings. The savings can go as high as 60% for structures like TLBs. For several arrays the savings exceed 40%. This is considerable as these arrays contribute typically, according to our in-house power simulator based on CACTI[10] and WATTCH[13], to 35% of the total power consumption in a contemporary core including L1 caches. For data cache data array, the energy savings is small. Also for data caches, the invalidating of the cache after every interval incurs more misses which account for more energy. However, we observe only 0.3 additional misses/perKiloInstructions due to reinitialization at 1M intervals (analysis not shown due to space constraints), which agrees with the findings of [9].

VI. Related Work

Several previous works aim to reduce energy by leveraging cache dynamic properties. One such technique exploits the non-uniformity in the values stored in arrays where a small number of unique values account for most of the array content [14] whereas another exploits the sparsity of used entries [15]. Authors in [5] have shown how to reduce the energy on reading or writing a 0 byte. For every byte in the array they attached an extra bit, ZIB, which indicates whether the byte is 0 or not. The approach is to prevent the bitline discharging when reading or writing a 0 byte, preventing in this way the bitline swing. They add various gating circuits, similar to our scheme, to disable part of the access activity. The main difference between the two techniques is that we use extra information for each bitline instead of an extra bit per byte in a wordline. Thus the two schemes exploit different array phenomena caused by non-uniform cache content.

The frequent-value-cache [7] stores frequently occurring block values encoded in a small table that is accessed first to filter accesses to the larger cache for less frequent values. Asymmetric-cell SRAM [8] leverages finer grain non-uniformity to reduce the leakage of cells that contain zero.

The decay cache approach turns off cache lines which are not used for a large number of cycles to decrease leakage power [6]. In contrast we reduce dynamic energy when an access occurs to a SCC-column. The decay-cache and SCC-column share some structural similarity because both schemes

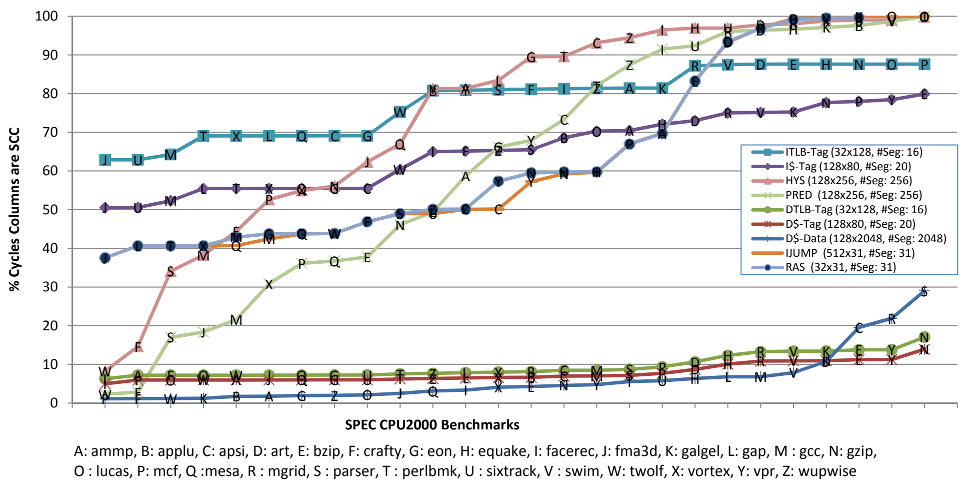


Fig. 5. *Percentage of Cycles Columns are SCC as detected by Interval Based Scheme*

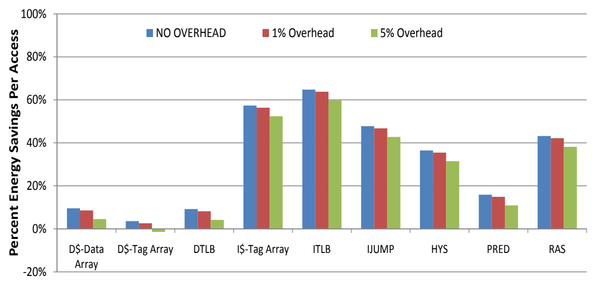


Fig. 6. *Energy savings per access in each array structure*

rely on a global counter and a few bits of state per line for the decay-cache and two bits per column for the SCC-column. However, the dynamic behavior that is exploited is quite distinct: unused lines vs same cell content column.

One other work leverages the higher frequency of cells storing 0 bits in caches to design low-leakage asymmetric-sram cells [8]. The key difference from our work is that we exploit same content at the granularity of columns instead of individual cells.

An excellent discussion for the breakdown of energy consumption in SRAM arrays is provided in [11].

VII. Conclusions and Future Work

This paper shows how to reduce the energy to access an array by exploiting the dynamic phenomenon of SSC-column that occurs when all cells of an array column store the same bit value. A characterization of SCC-column reveals that it occurs frequently for many benchmarks and arrays. An interval based scheme is introduced that requires one bit per column to detect a SCC-column. This scheme can be used to gate some of the actions needed to read or write a SCC-column thus reducing the array access energy. An experimental evaluation shows that the proposed interval based scheme detects SCC-columns frequently and the energy savings often exceed 40%. The findings of this paper point to future work aiming to make the compiler and microarchitecture SCC-column aware to increase the frequency of SCC-columns.

References

[1] S. Borkar, "Design Challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, Jul. 1999.
[2] Y. Taur, "CMOS design near to the Limit of Scaling," *IBM Journal of Research and Development*, vol. 46, no. 2/3.
[3] The International Technology Roadmap for Semiconductors, "Edition 2009," ITRS, Tech. Rep., 2009. [Online]. Available: http://www.itrs.net
[4] S. Kaxiras and M. Martonosi. Morgan & Claypool Publishers, 2008.
[5] L. Villa, M. Zhang, and K. Asanovic, "Dynamic zero compression for cache energy reduction," in *MICRO*, 2000.
[6] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache decay: exploiting generational behavior to reduce cache leakage power," in *ISCA*, 2001.
[7] J. Yang and R. Gupta, "Energy efficient frequent value data cache design," in *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, ser. MICRO 35, 2002.
[8] N. Azizi, A. Moshovos, and F. N. Najm, "Low-leakage asymmetric-cell sram," in *Proceedings of the 2002 international symposium on Low power electronics and design*, ser. ISLPED '02, 2002.
[9] T. Constantinou, Y. Sazeides, P. Michaud, D. Fetis, and A. Seznec, "Performance implications of single thread migration on a chip multi-core," in *SIGARCH Computer Architecture News*, 2005, p. 2005.
[10] N. Muralimanohar and R. Balasubramonian, "Cacti 6.0: A tool to understand large caches."
[11] B. Jacob, S. Ng, and D. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann, 2007.
[12] R. Desikan, D. Burger, S. Keckler, and T. Austin, "Sim-alpha: a validated execution driven Alpha 21264 simulator," CS Dept., University of Texas at Austin, Tech. Rep. TR-01-23, 2001.
[13] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proceedings of the 27th annual international symposium on Computer architecture*, ser. ISCA '00. New York, NY, USA: ACM, 2000, pp. 83–94. [Online]. Available: http://doi.acm.org/10.1145/339647.339657
[14] Y. Zhang, J. Yang, and R. Gupta, "Frequent value locality and value-centric data cache design," in *ASPLOS*, 2000.
[15] P. Juang, K. Skadron, M. Martonosi, Z. Hu, D. W. Clark, P. W. Diodato, and S. Kaxiras, "Implementing branch-predictor decay using quasi-static memory cells," *ACM Trans. Archit. Code Optim.*, vol. 1, June 2004.