Open access • Proceedings Article • DOI:10.1109/HPCA.2014.6835978

Mosaic: Exploiting the spatial locality of process variation to reduce refresh energy in on-chip eDRAM modules — Source link

Aditya Agrawal, Amin Ansari, Josep Torrellas

Institutions: University of Illinois at Urbana–Champaign

Published on: 01 Feb 2014 - High-Performance Computer Architecture

Topics: eDRAM

Mosaic: Exploiting the Spatial Locality of Process Variation to Reduce Refresh Energy in On-Chip eDRAM Modules *

Aditya Agrawal, Amin Ansari and Josep Torrellas
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

Abstract

EDRAM cells require periodic refresh, which ends up consuming substantial energy for large last-level caches. In practice, it is well known that different eDRAM cells can exhibit very different charge-retention properties. Unfortunately, current systems pessimistically assume worst-case retention times, and end up refreshing all the cells at a conservatively-high rate. In this paper, we propose an alternative approach. We use known facts about the factors that determine the retention properties of cells to build a new model of eDRAM retention times. The model is called Mosaic. *The model shows that the retention times of cells in large eDRAM modules exhibit* spatial correlation. *Therefore, we logically divide the eDRAM module into regions or* tiles, *profile the retention properties of each tile, and program their refresh requirements in small counters in the cache controller. With this architecture, also called Mosaic, we refresh each tile at a different rate. The result is a 20x reduction in the number of refreshes in large eDRAM modules — practically eliminating refresh as a source of energy consumption.*

1. Introduction

An attractive approach to reduce the energy wasted to leakage in the cache hierarchy of multicores is to use embedded DRAM (eDRAM) for the lower levels of caches. EDRAM is a capacitor-based RAM that is compatible with a logic process, has high density and leaks very little [13]. While it has higher access times than SRAM, this is not a big concern for large lower-level caches. As a result, eDRAM is being adopted into mainstream products. For example, the IBM POWER7 processor includes a 32 MB on-chip eDRAM L3 cache [32], while the POWER8 processor will include a 96 MB on-chip eDRAM L3 cache and, potentially, an up to 128 MB off-chip eDRAM L4 cache [28]. Similarly, Intel has announced a 128 MB off-chip eDRAM L4 cache for its Haswell processor [2].

EDRAM cells require periodic refresh, which can also consume substantial energy for large caches [1, 34]. In reality, it is well known that different eDRAM cells can exhibit very different charge-retention properties and, therefore, have different refresh needs. However, current designs pessimistically assume worst-case retention times, and end up refreshing all the eDRAM cells in a module at the same, conservatively-high rate. For example, they use a refresh period of around 40 µs [3]. This naive approach is wasteful.

Since eDRAM refresh is an important problem, there is significant work trying to understand the characteristics of eDRAM charge retention (e.g., [11, 16, 17]). Recent experimental work from IBM has shown that the retention time of an eDRAM cell strongly depends on the threshold voltage (V_t) of its access transistor [17].

In this paper, we note that, since the values of V_t within a die have spatial correlation, then eDRAM retention times will also necessarily exhibit spatial correlation. This suggests that architectural mechanisms designed to exploit such correlation can easily save refresh energy.

Consequently, in this paper, we first develop a new model of the retention times in large on-chip eDRAM modules. The model, called *Mosaic*, builds on process-variation concepts. It shows that the retention properties of cells in large eDRAM modules do exhibit spatial correlation. Then, based on the model, we develop a low-cost architectural mechanism to exploit such correlation and eliminate most of the refreshes.

Our architectural technique, also called Mosaic, consists of logically dividing an eDRAM module into logical regions or *tiles*, profiling the retention characteristics of each tile, and programming their refresh requirements in small counters in the cache controller. Such counters then trigger refreshes when it is time to refresh their corresponding tiles. With this architecture, we refresh each tile at a different rate.

There is prior work on solutions that exploit the non-uniformity of the retention time of cells in dynamic memories to reduce refreshes. Examples include RAPID [31], Hi-ECC [34], the 3T1D-based cache [19], and RAIDR [21]. We discuss them in detail in a later section. Fundamentally, our contribution is at a different level, in that we investigate and identify the main source of this variation, and build a mathematical model of the variation. The model shows the presence of spatial correlation in retention times. Building on this novel observation, we propose a targeted solution to minimize refreshes.

Our results show that the Mosaic tiled architecture is both inexpensive and very effective. An eDRAM L3 cache augmented with Mosaic tiles increases its area by 2% and reduces the number of refreshes by 20 times. This reduction is 5 times the one obtained by taking the RAIDR scheme for main memory DRAM [21] and applying it to cache eDRAM. With Mosaic, we get very close to the lower bound in refresh energy, and end up saving 43% of the total energy in the L3 cache.

This paper is organized as follows: Section 2 discusses the

*This work was supported in part by NSF under grant CCF-1012759; Intel through an Intel Ph.D. Fellowship to Aditya Agrawal; DARPA under UHPC Contract HR0011-10-3-0007 and PERFECT Contract HR0011-12-2-0019; and DOE ASCR under Award Numbers DE-FC02-10ER2599 and DE-SC0008717. Dr. Amin Ansari is now with Qualcomm Inc., San Diego, CA.

problem addressed; Section 3 introduces the Mosaic model; Sections 4 and 5 present the Mosaic architecture; Sections 6 and 7 evaluate them; and Section 8 covers related work.

2. Problem Addressed

In this section, we discuss how eDRAM cells retain charge. We observe that the expected retention time and the one assumed in practice are off by orders of magnitude. We then present the distribution of the retention time and discuss its sources.

2.1. eDRAM Cell Retention Time

Fig. 1 shows an eDRAM cell. It consists of an access transistor and a storage capacitor. The logic state is stored as electrical charge in the capacitor. The capacitor loses charge over time through the access transistor — shown as I_{off} in the figure. Therefore, an eDRAM cell requires periodic refresh to maintain the correct logic state.

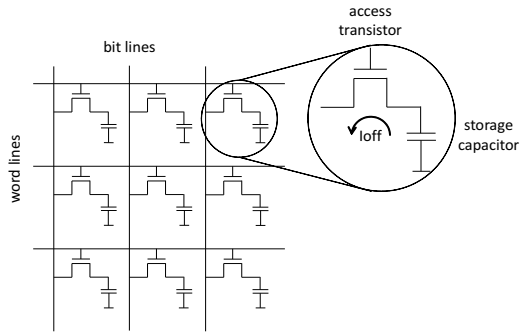

Figure 1: An eDRAM cell.

The leakage through the transistor depends on the threshold voltage (V_t) of the transistor. The higher the V_t is, the lower the leakage is and, therefore, the cell retains its logic value for longer. Conversely, a low V_t results in more leakage and, hence, the cell loses its logic value sooner. On the other hand, a higher V_t reduces the overdrive of the transistor and increases the access time of the cell. Therefore, there is a tradeoff between the cell access time and how long it retains its value.

We now derive a closed-form mathematical equation relating the parameters of the cell to its retention time. Let C be the storage capacitance, W and L the width and length of the access transistor, V the voltage applied to the gate of the access transistor, S_t the subthreshold slope (defined below), I_{off} the off drain current through the access transistor, and T_{ret} the retention time of the eDRAM cell. T_{ret} is defined as the time until the capacitor loses 6/10th of the stored charge [17], that is,

$$T_{ret} = \frac{0.6 \times C}{I_{off(V=0)}} \tag{1}$$

The definition of V_t is empirical. The definition varies from foundry to foundry, and across technology nodes. Kong et al. [17] define it as the gate voltage at which the current becomes the expression on the right in Eq. 2.

$$I_{off(V=V_t)} = 300 \times \frac{W}{L}\ nAmps \tag{2}$$

The subthreshold slope is defined as the inverse of the slope of the semi–logarithmic I_{off}-V curve, that is,

$$S_t = \frac{V_t - 0}{log_{10}(I_{off(V=V_t)}) - log_{10}(I_{off(V=0)})} \tag{3a}$$

$$= \frac{V_t}{log_{10}(\frac{I_{off(V=Vt)}}{I_{off(V=0)}})} \tag{3b}$$

Re-arranging and substituting,

$$I_{off(V=0)} = I_{off(V=Vt)} \times 10^{-V_t/S_t} \tag{4a}$$

$$= 300 \times \frac{W}{L} \times 10^{-V_t/S_t}\ nAmps \tag{4b}$$

Substituting Eq. 4b in Eq. 1 gives

$$T_{ret} = 0.6 \times C \times \frac{L}{W} \times 10^{V_t/S_t} \times 10^9/300\ sec \tag{5}$$

From [17], at 65nm technology, we get $C = 20\ fF$, $L = W = 100\ nm$, $V_t = 0.65\ V$ and $S_t = 112\ mV/dec$. Substituting these values in Eq. 5, we get $T_{ret} = 25.44\ ms$.

Therefore, we expect eDRAM cell retention times to be of the order of a few tens of milliseconds. However, in practice, eDRAM cells are refreshed with a period of the order of a few tens of *microseconds*. For example, Barth et al. [3] report a time of 40 µs. This is because manufacturing process variations result in a distribution of retention times and, to attain a high yield, manufacturers choose the retention time for the entire memory module to be the one of the leakiest cells.

2.2. Retention Time Variation

It is well known that there is variation in the retention time of eDRAM and DRAM cells (e.g., [11, 16, 17]). The overall distribution and the sources of variation have also been identified. Fig. 2 shows a typical eDRAM retention time distribution [17]. The X axis is $log_{10}\ T_{ret}$. The Y-axis is the cumulative density function of the number of cells under a given retention time. The Y axis uses a normal distribution scale — that is, 0 represents the fraction 0.500, -1σ represents the fraction 0.158, and so on as shown in Table 1.

Sigma (σ)	Fraction
0.0	0.500000
-1.0	0.158655
-2.0	0.022750
-3.0	0.001349
-4.0	0.000031
-4.5	0.000003

Table 1: Area under the curve for a normal distribution.

The figure shows that the retention time distribution has two components, namely the *Bulk Distribution* and the *Defect Tail Distribution*. The Bulk Distribution includes the majority of cells. Specifically, since the figure shows that the Bulk Distribution goes from approx. -4σ to ∞, it includes the 0.999968

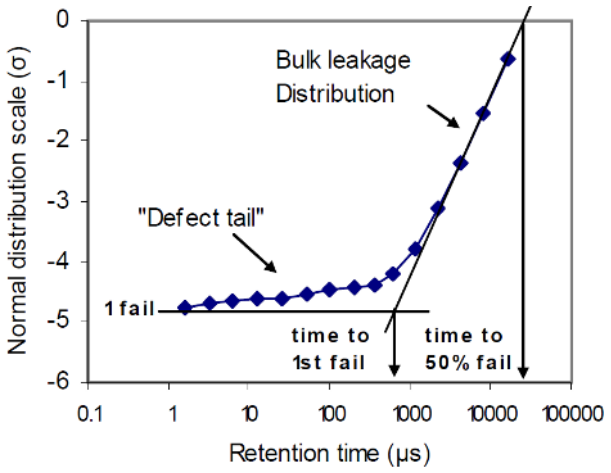

Figure 2: Typical eDRAM retention time distribution [17].

fraction of the cells — as given by the area under the curve of a normal distribution from -4σ to ∞. In addition, the fact that it appears as a straight line in the log-normal plot of Fig. 2 indicates that $log_{10}\ T_{ret}$ follows a normal distribution for the Bulk — or that T_{ret} follows a log-normal one for the Bulk.

Based on experimental data, Kong et al. [17] from IBM say "We demonstrate that the T_{ret} (Bulk) Distribution can be attributed to array (i.e., access transistor) V_t variation". This is a *key observation*, and is consistent with what we know about V_t's process variation distribution. Indeed, it is accepted that process variation in V_t follows a normal distribution [17]. If we take the log of Eq. 5, we obtain,

$$log_{10}\ T_{ret} = \frac{V_t}{S_t} + expression \tag{6}$$

which shows that a normal distribution of V_t results in a normal distribution of $log_{10}\ T_{ret}$ and, hence, a log-normal distribution of T_{ret}. This agrees with the straight line in Fig. 2.

The Tail Distribution includes very few cells. Since it covers the area under the curve from $-\infty$ to approx. -4σ in Fig. 2, it includes only the 0.000031 fraction of the cells, or 31 ppm (parts per million). The fact that it appears as a straight line in Fig. 2 indicates that $log_{10}\ T_{ret}$ follows a normal distribution for the Tail — hence, T_{ret} follows a log-normal one for the Tail.

The source of the T_{ret} Tail Distribution has been attributed to random manufacturing defects. These defects manifest themselves as leaky cells. However, not all the cells following the Tail Distribution are considered defective. Only the cells in the region $-\infty$ to -4.5σ (about 3 ppm) are considered defective and are handled by redundant lines provided by ordinary designs [12, 17].

In the distribution above, the -4.5σ point represents a retention time of 45 μs. Barth et al. [3] have reported retention times of 40 μs as well. Therefore, it is clear that, overall, eDRAMs are refreshed at a very pessimistic rate. Since process variation in the V_t of the access transistor governs the distribution of almost all the cells, we look at it in more detail next.

2.3. Process Variation in the Threshold Voltage

Process variation in the V_t has two components, namely, *systematic* and *random*. Systematic variation is introduced by lithographic tools such as steppers, and exhibits high spatial correlation [9, 22, 27] — i.e., nearby transistors have similar V_t. Random variation is the result of material defects, dopant fluctuation, and line edge roughness, and is essentially white noise. The total variation is a superposition of the systematic and random components.

VARIUS [26] and other variation-modeling tools model the two components with normal distributions. Each distribution has its own sigma, namely, σ_{sys} and σ_{rand}. The superposition of both components results in an overall normal distribution for V_t's variation, with a sigma σ_{tot} equal to $\sqrt{\sigma_{sys}^2 + \sigma_{rand}^2}$. It is the *combined systematic and random components* of V_t's variation that induce the Bulk Distribution in Fig. 2.

From Eq. 6, we observe that the spatial correlation in V_t will result in spatial correlation in the retention times of eDRAM cells — i.e., eDRAM cells that are spatially close to each other will have similar retention time values. In this paper, we exploit this property to eliminate most refreshes.

3. The Mosaic Retention Time Model

We want to develop a new model of eDRAM retention time that can help us to understand and optimize eDRAM refreshing. This section describes our model, which we call *Mosaic*.

3.1. Extracting the Values of Retention Parameters

To build the model, we first need to obtain the values for the key parameters of the T_{ret} Bulk and Tail Distributions in Fig. 2. Specifically, we need: (i) the mean and sigma of the Bulk Distribution $(\mu_{Bulk}, \sigma_{Bulk})$, (ii) the mean and sigma of the Tail Distribution $(\mu_{Tail}, \sigma_{Tail})$, and (iii) the fraction of cells that follow the Tail Distribution (ρ). From Kong et al. [17], we obtain that $\mu(V_t) = 0.65\ V$, $\sigma(V_t) = 0.042\ V$, and $S_t = 112\ mV/dec$. Therefore, from Eq. 6, and computing *expression* based on Eq. 5, we get the parameter values for the Bulk Distribution:

$$\mu_{Bulk}(log_{10}\ T_{ret}) = \frac{\mu(V_t)}{S_t} - 7.40 = -1.594$$

$$\sigma_{Bulk}(log_{10}\ T_{ret}) = \frac{\sigma(V_t)}{S_t} = 0.375$$

Kim and Lee [16] observe that the peak of the Bulk Distribution and the peak of the Tail Distribution in DRAMs are off by approximately one order of magnitude. This is 1 in log_{10} scale, which is approximately 3 times the value that we have just obtained for σ_{Bulk}. Hence, in our model, we estimate the peak of the Tail Distribution $(\mu_{Tail}(log_{10}\ T_{ret}))$ to be $3 \times \sigma_{Bulk}(log_{10}\ T_{ret})$ to the left of the peak of the Bulk Distribution $(\mu_{Bulk}(log_{10}\ T_{ret}))$:

$$\mu_{Tail}(log_{10}\ T_{ret}) - \mu_{Bulk}(log_{10}\ T_{ret}) = -3 \times \sigma_{Bulk}(log_{10}\ T_{ret})$$
$$= -1.125$$
$$hence, \quad \mu_{Tail}(log_{10}\ T_{ret}) = -2.719$$

We obtain the last two parameters, namely $\sigma_{Tail}(log_{10}\ T_{ret})$ and ρ, by curve-fitting the data in Fig. 2. We obtain,

$$\sigma_{Tail}(log_{10}\ T_{ret}) = 1.8$$
$$\rho = 0.000020\ (20\ ppm)$$

This value of ρ is more accurate than our initial estimation of 31 ppm for the Tail Distribution in Sec. 2.2. The final parameter values are summarized in Table 2. With these values, we generate the curve shown in Fig. 3. On this same graph, we superpose the experimental data from Fig. 2 in circles. We can see that the curve fits the experimental data. Moreover, our parameters are in agreement with an observation made in [11] that the σ of the Tail and the Bulk Distributions have a ratio of 4.87. This ratio for Mosaic is 4.80. Finally, in this curve, the -4.5σ point (3 ppm) corresponds to a retention time of about 45 μs.

Parameter	Value
$\mu_{Bulk}(log_{10}\ T_{ret})$	-1.594
$\sigma_{Bulk}(log_{10}\ T_{ret})$	0.375
$\mu_{Tail}(log_{10}\ T_{ret})$	-2.719
$\sigma_{Tail}(log_{10}\ T_{ret})$	1.8
ρ	20 ppm

Table 2: Parameter values extracted from the data in [17].

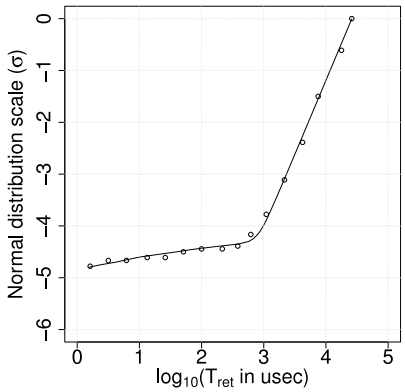

Figure 3: Retention time curve generated by Mosaic compared to the experimental data in [17].

3.2. Generating a Spatial Map of Retention Times

The parameter values for the Bulk and Tail Distributions extracted in the previous section are not enough to generate a spatial map of the T_{ret} values in a given memory module. The reason is that the $\sigma_{Bulk}(log_{10}\ T_{ret})$ has a random and a systematic component, and the latter has a spatial correlation function. These effects are caused by the V_t distribution, as per Eq. 6. Different values for the σ breakdown into systematic and random, and the spatial correlation do not change the Bulk line in Fig. 2, as long as the total σ stays constant. However, they produce very different spatial maps of T_{ret} values. For example, if the fraction of σ coming from its systematic component is high and the correlation distance is long, there will be spatial clusters of eDRAM cells with similar T_{ret}. For the Tail Distribution, since it does not have any systematic component, we do not need any more information.

Overall, to generate a spatial map of the T_{ret} in an eDRAM memory module, we need to know: (i) the values in Table 2, (ii) the breakdown of $\sigma_{Bulk}(log_{10}\ T_{ret})$ into random and systematic components, and (iii) the correlation function for the systematic component. Our observation is that we can obtain (ii) and (iii) based on published data on the V_t *variation of the access transistors*. Specifically, as per Eq. 6, the breakdown of $\sigma_{Bulk}(log_{10}\ T_{ret})$ into random and systematic components is the same as the breakdown of $\sigma(V_t)$. Similarly, the correlation of $log_{10}\ T_{ret}$'s systematic component follows the correlation of V_t's systematic component.

3.3. Procedure to Generate a Spatial T_{ret} Map

Let us now generate a spatial map of T_{ret} values for an on-chip eDRAM memory module of 1024×1024 cells that distribute according to the parameter values of Table 2. For the V_t variation parameters of the access transistors, we will assume the following. First, following Karnik et al. [14], the $\sigma(V_t)$ has equal components of systematic and random components — i.e., $\sigma_{sys} = \sigma_{rand} = \sigma/\sqrt{2}$. Secondly, the correlation function for V_t's systematic component follows the Spherical function described in VARIUS [26] with a correlation distance ϕ of 0.4.

To generate the spatial T_{ret} map, we first generate the one for its Bulk Distribution and then superpose the one for its Tail Distribution. For the Bulk one, we first generate the spatial map for the systematic component and then superpose the one for the random component. To generate the T_{ret}'s Bulk Distribution maps, we will first proceed with the intermediate step of generating the V_t maps and then use Equation 6 to generate T_{ret}'s Bulk maps. This is done for pedagogical reasons, since we could instead directly generate T_{ret}'s Bulk maps using (i) the μ_{Bulk} and σ_{Bulk} of Table 2, (ii) the breakdown of $\sigma(V_t)$, and (iii) the correlation of V_t's systematic component.

3.3.1. Spatial Map for $log_{10}\ T_{ret}$'s Bulk Distribution. Following the VARIUS methodology [26], we lay out an imaginary grid of $N_x \times N_y$ points on top of the eDRAM memory module. We then invoke VARIUS with $\mu(V_t)$=0.65 V, $\sigma(V_t)$=0.042 V (these two values are from Kong et al. [17]), and correlation distance ϕ=0.4 (this is one of our assumptions). We obtain a spatial map of V_t's systematic component, as shown in Fig. 4a. We then obtain $N_x \times N_y$ samples of a normal distribution with $\mu = 0$ and $\sigma_{rand}(V_t)$, without correlation, as shown in Fig. 4b. As expected, the spatial map looks like white noise. We then superpose both maps, point per point, and obtain the total V_t map in Fig. 4c. Finally, the spatial map for $log_{10}\ T_{ret}$'s Bulk Distribution is obtained from the spatial map of the total V_t by applying Eq. 6 to every point in the grid. The resulting map is shown in Fig. 4d.

3.3.2. Spatial Map for $log_{10}\ T_{ret}$'s Tail Distribution. We use a normal distribution with the μ_{Tail} and σ_{Tail} of Table 2, and

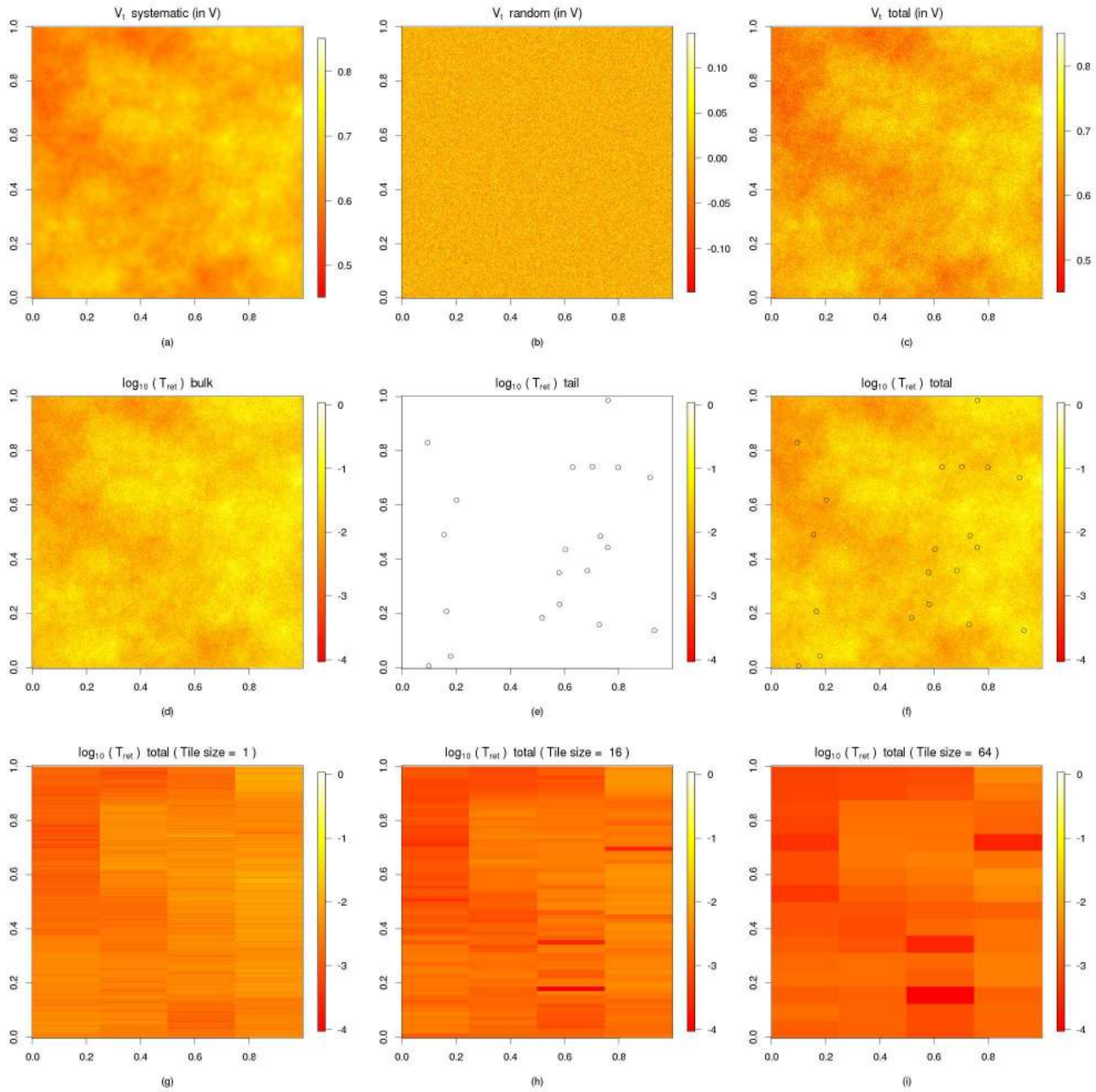

Figure 4: Various spatial maps of V_t and $log_{10}\ T_{ret}$ for an on-chip eDRAM memory module.

take $N_x \times N_y \times \rho$ samples with no correlation. We place them randomly on the $N_x \times N_y$ grid. The locations of the Tail cells are marked by circles in Fig. 4e.

3.3.3. Spatial Map for Overall $log_{10}\ T_{ret}$'s Distribution. We obtain the spatial map for the overall $log_{10}\ T_{ret}$ by replacing points in the Bulk Distribution with those in the Tail Distribution. The result is Fig. 4f, where the locations of the Tail cells are marked in circles. The map is almost the same as the one with only Bulk, since only a fraction ρ is replaced by Tail cells.

3.4. Using the Mosaic Retention Model Across Generations

As we move from one eDRAM generation to the next, we need to recompute the new distribution of T_{ret}. The Bulk Distribution of T_{ret} can be easily computed from the distribution of the access transistor's V_t — obtained with V_t's μ, σ, σ breakdown, and ϕ. This information on V_t for the new eDRAM generation may be easily available from the manufacturing process. The Tail Distribution of T_{ret}, however, needs to be characterized with direct measurements that extract $\mu_{Tail}(log_{10}\ T_{ret})$, $\sigma_{Tail}(log_{10}\ T_{ret})$, and ρ.

In some cases, it may be too expensive or difficult to characterize the Tail Distribution of T_{ret} for a new generation. In these cases, it may be possible to reuse the known Tail Distribution from the older eDRAM generation (if it has changed relatively little), and combine it with the V_t information of the new eDRAM generation, to generate the distribution of T_{ret} for the new eDRAM generation.

4. The Mosaic Architecture

4.1. Insight and Opportunity

The analysis in the previous section has provided a useful insight: since the retention time of an eDRAM cell is highly dependent on its access transistor's V_t, and V_t has well-known spatial correlation properties, then the retention time also has spatial correlation. This fact offers an opportunity to save refresh energy. Specifically, we can logically group cells into regions, profile their retention time, and set-up time counters to refresh the regions only at the frequency that each one requires. With reasonable spatial correlation, the hardware cost of the counters will be minimal.

As an example, consider the eDRAM of Fig. 4f. Let us organize it as a 4-way set associative memory with 256-bit lines. Since eDRAM reads, writes and refreshes are performed at line granularity, we need the *line-level* distribution of $log_{10}\ T_{ret}$. For this, we set a line's T_{ret} to the minimum of the T_{ret} of the bit cells constituting the line. The result is shown in Fig. 4g.

If *Nlines* is the number of lines in the memory module, the per-line retention time map gives an absolute lower bound of the number of refreshes required to prevent data loss:

$$Min.\ refreshes/sec = \sum_{i=1}^{Nlines} \frac{1}{T_{ret\_line\_i}} \tag{7}$$

This lower bound would be hard to attain. First, the memory has a limited number of ports, and only a number of lines equal to the number of ports can be refreshed simultaneously. Hence, some of the refreshes may need to be delayed. As a result, to ensure correctness when multiple lines need to be refreshed at the same time, and some refreshes need to be delayed, we need to provide a timing guardband.

In addition, providing a counter per line is too expensive. Kaxiras et al. [15] have estimated the cost of an N-bit counter to be $40N+20$ transistors. Therefore, even a 2-bit counter per 256-bit line would amount to a 40% area overhead. Solutions using Bloom filters lose accuracy.

The striations seen in Fig. 4g indicate that there is spatial correlation in the T_{ret} of adjacent lines — especially within the same way. Hence, we group sets of contiguous lines from the same way into *Tiles*, setting T_{ret} for the tile to the minimum of the T_{ret} of the lines constituting the tile. Figs. 4h and 4i show the tile-level distribution of $log_{10}\ T_{ret}$ for tile sizes of 16 and 64 lines, respectively. We can see that there are regions of spatial locality. The resulting distribution looks like a mosaic of tiles. With this design, the hardware cost of a counter is amortized over a whole tile. However, we have to refresh the whole tile whenever the counter rolls down to zero.

4.2. Profiling the Retention Times

Mosaic needs to profile the retention times of the tiles, for example at boot time. Literature from industry such as IBM [13, 17], Intel [34], Samsung [16], and Toshiba [11], and academic groups [6, 21, 31] have proposed and/or used T_{ret} profiling schemes. However, recent work [20] has pointed out that T_{ret} profiling has to deal with Data Pattern Dependence (DPD) and Variable Retention Time (VRT) effects. As suggested in [20] and [13], profiling in the presence of DPD can be best done by using a variety of manufacturer-provided test patterns — e.g., all 0s/1s, checkerboard, walk and random. One of the papers [20] also points out that VRT changes are slow (in the order of hours and sometimes a day) and, therefore, one could profile periodically. Note that a tile in Mosaic will include over 10,000 cells. With many cells, it is possible that, macroscopically, these effects exhibit relatively less external variation across measurements. In reality, sophisticated profiling techniques are still a subject of research in this area, and paramount to dynamic-memory manufacturers. Once the tiles are profiled at boot time, the per-tile T_{ret} count in cycles is stored in a small SRAM in the cache controller.

4.3. Temperature Adaptation

The V_t of a transistor is a function of temperature (T) [33]. Therefore, the access transistor's leakage current (Eq. 4b) and T_{ret} (Eq. 5) vary strongly with T. Empirical data [5, 6] suggests the following exponential relationship between T_{ret} and T: there is approximately a 1.3x reduction in T_{ret} for every 10 °C increase in T. Specifically, let T_1 and T_2 be the T in °C and T_{ret_1} and T_{ret_2} their corresponding T_{ret}, then

$$T_{ret\_2} = T_{ret\_1} \times e^{-0.0268 \times (T_2 - T_1)} \tag{8}$$

The boot-time profiling temperature (T_0) and the corresponding T_{ret_0} values are stored in the SRAM. At run time, a thermal sensor measures T. If the T is T', then when Mosaic reads the SRAM, it computes the new T'_{ret} as $f(T_0, T_{ret\_0}, T' + Guardband)$, using Eqn. 8. We add a small T guardband as a safeguard against T changes between consecutive refreshes of the tile. This guardband is only a few degrees, as the time between refreshes is at most a few ms, and T changes slowly.

For these computations, we need a lookup table (LUT) and an 8-bit adder and multiplier. The exponential portion of Eqn. 8 is stored in a 32-entry LUT (from -40 °C to 120 °C in 5 °C steps). Each LUT entry is 8 bits.

In an advanced design, we may want to consider a per-tile T, rather than a single T for the whole eDRAM module. In this case, we would need to store a per-tile reference temperature T_0 in addition to a per-tile T_{ret_0}. At run time, T sensors would provide the T' of each tile. Then, for each tile, the algorithm would use the local T' and the local T_0 to apply the correction to the local T_{ret_0}.

4.4. Designing the Refresh Counters

Assuming that we have accurately profiled each tile's T_{ret}, we now consider the design of the refresh counters. A refresh operation is akin to a read or a write access to the cache line. However, the refresh operation takes precedence over normal accesses. In addition, we assume that a refresh operation takes one cycle when done in a pipelined fashion. If a memory module is organized into banks, then the banks can be refreshed in parallel. In a given bank, the number of ports sets the number

of refreshes that can proceed in parallel; if the bank has a single port, then only one refresh can be done in the bank per cycle.

In this paper, we assume one port per bank. Therefore, the minimum number of cycles required to refresh a whole bank is *Nlines*, which is the number of lines per tile times the number of tiles per bank. In the worst case, all the lines of the bank might require a refresh at the same time. To handle this corner case, we need to use a time guardband equal to the maximum time between requesting a refresh and being serviced. This guardband is equal to the time required to refresh all the lines of the bank, that is,

$$Guardband = \frac{Nlines}{f} \tag{9}$$

where f is the frequency of the cache. Therefore, the corrected value of the retention time to be used for a tile, T'_{ret}, is

$$T'_{ret} = T_{ret} - Guardband \tag{10}$$

A second correction comes from the fact that Mosaic uses counters to track time. A counter increments in units of *Step*. Therefore, T'_{ret} has to be rounded to the highest multiple of Step that is less than T'_{ret}. Combining guardbanding and rounding off, the value of the retention time to be used, T''_{ret}, is

$$T''_{ret} = n \times step \mid n \times step \leq T'_{ret} < (n+1) \times step \tag{11}$$

4.5. Overall Mosaic Design & Operation

Fig. 5 shows the hardware needed by Mosaic to refresh a cache bank. The cache controller for the bank has a programmable clock divider, a down-counter for each of the tiles in the bank, a sequencer, an LUT with an adder and multiplier, and a small Retention Profile SRAM. The latter stores data obtained from profiling during boot time: T_{ret} for each of the tiles in the bank, and the bank's temperature (T_0). The LUT with the adder and multiplier are used for T adaptation (Section 4.3).

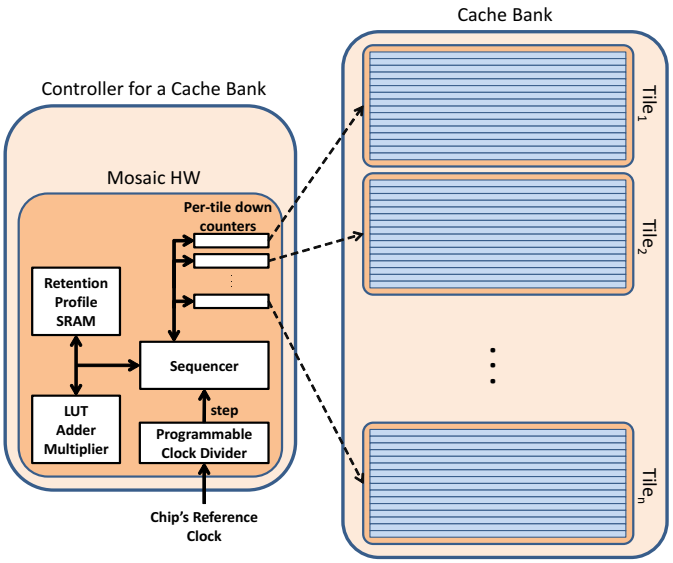

Figure 5: Hardware required by Mosaic to refresh a cache bank.

The programmable clock divider takes a reference clock as input and creates a clock pulse of Step, which is typically the -4.5σ point, i.e., about 45 μs. This signal acts as the clock for all the per-tile counters. Each counter is an N-bit down-counter that tracks the retention time of one tile. All counters are initialized to the values in the SRAM followed by the adjustments due to T (Section 4.3) and other corrections (Section 4.4).

At every Step, the sequencer rolls down all the counters one by one. For a given counter, it first decrements it and then compares its value to zero. If it is zero, the sequencer schedules refreshes for all the lines in the corresponding tile. Next, it reloads the value of the counter after reading the SRAM and adjusting it for T and the other corrections. Then, the sequencer moves to the next tile. The process continues until all the counters get decremented.

4.6. Area and Energy Overheads

Mosaic induces little area and energy overhead. To see why, assume that Fig. 5 corresponds to a 1 MB cache bank with 64-byte lines and 16-line tiles. Such an organization has 1024 tiles in the bank. We can use down-counters with 8 bits which, if they step at 45 μs, can cover a T_{ret} of nearly 12 ms.

Consider first the area and energy overhead of the Retention Profile SRAM. Its storage capacity is about 1KB, which is 0.1% of the data array in the cache bank. Hence, it takes an insignificant area. The SRAM is accessed only when a counter rolls down to zero. Given the T_{ret} distribution, this happens infrequently — typically, every few μs, since we have 1024 tiles and typical T_{ret} values are equal to a few ms. Hence, the overall dynamic energy associated with reading the SRAM is insignificant. The same is the case for its leakage power.

Next, consider the area and energy overhead of the counters. If we assume that an N-bit counter needs about $40N+20$ transistors [15], then adding an 8-bit counter per each 16-line tile adds 4% more transistors. In addition, since the counters are only accessed every 45 μs, their dynamic energy contribution is very small. Similarly, their leakage power is negligible. In Section 7, we estimate their area and energy overhead accurately using the Synopsys Design Compiler.

Finally, consider the 32-byte LUT and the 8-bit adder and multiplier. Using McPAT [18], we estimate that their area is 0.2% of the area of the cache bank, which is negligible. Also, the energy overhead of these structures is very small, as they are accessed only every few μs as the SRAM.

5. Discussion

We can approach the lower bound of refresh energy (Eq. 7) if we can afford per-line counters with an arbitrary number of bits and Step sizes. However, this solution is very expensive in both area and energy. On the other hand, we can avoid the area and energy overheads altogether if we refresh the entire eDRAM module at a constant rate, as it is currently done. There is a clear tradeoff between the potential refresh-energy savings and the overheads we are willing to incur.

As the number of lines per tile increases, the area and energy

overheads of the counters decrease. However, the lines constituting the tile are refreshed at the rate needed by the line with the lowest retention in the tile. Therefore, we perform more refreshes than required and move away from the lower bound.

As the number of bits in the counter increases, the area and energy overhead increases, but in return we create more bins and therefore reduce the number of refreshes. The benefits saturate as soon as the range of the counter approaches the higher retention times of the distribution.

We do not experiment with counter Step sizes. A Step size larger than the -4.5σ point, such as 100 μs, would require disabling the lines with retention times in the range 45 μs - 100 μs. It would allow us to investigate solutions that trade-off cache capacity for refresh energy gains. A Step size less than the -4.5σ point would increase the granularity of the counter, but decrease the range of the counter. Initial results suggested that, given the same number of bits, the benefits of a larger range with coarser granularity outweigh the benefits of a smaller range with finer granularity.

We can potentially attain higher energy savings by using variable-sized tiles. For example, if there is a line or a small region of lines that have very different T_{ret} than their neighbors, we can save refreshes by placing them in a tile of their own. In contrast, if there is a very large region of lines that have similar T_{ret}, then we can save counter overhead and energy by placing all of them in a giant tile. However, while having variable-sized tiles may be attractive, it has hardware reconfiguration costs and design complexity. Further, as we will see in Section 7, Mosaic with fixed-sized tiles already eliminates the majority of refresh energy at a very modest hardware cost. Hence, there is little room to improve.

Finally, given an eDRAM module, the fraction of refresh power that Mosaic eliminates does not depend on the application running or its input data size. Mosaic's refresh savings depend on the T_{ret} variation profile, and the hardware parameters of the eDRAM module and Mosaic.

6. Evaluation Setup

We evaluate Mosaic with simulations of a chip multiprocessor (CMP) with 16 cores. Each core is a dual-issue, out-of-order engine. The cache hierarchy consists of per-core private L1 instruction and data caches, a per-core private L2 cache, and a shared L3 cache. The L3 is divided into 16 banks. Each bank is close to one core and has a statically-mapped range of addresses. A 4x4 torus on-chip-network connects the 16 nodes. The CMP uses a MESI directory coherence protocol maintained at the L3. The architectural parameters are shown in Table 3.

The L1 and L2 caches are modeled as SRAMs. The L3 cache is modeled as an eDRAM. Each L3 bank has a Mosaic module as in Fig. 5. A refresh operation is like a cache access, consuming an energy equal to a cache line hit. It takes one cycle when done in a pipelined fashion.

To evaluate Mosaic, we use a variety of tools. To estimate performance, we use the SESC [25] architecture simulator. To estimate the area and the dynamic and leakage energies of cores, caches and interconnect, we use McPAT [18]. Since McPAT does not model eDRAM caches, we use CACTI [30] to estimate the area and energy of the L3. We use the Synopsys Design Compiler to estimate the area and energy of the counters introduced by Mosaic. Finally, we use VARIUS [26] to model V_t variation and R [29] for fast statistical prototyping and analysis.

Architectural Parameters	
Chip	16-core CMP
Core	2-issue out-of-order
Instr. L1 cache	32 KB, 2 way. Round trip (RT): 2 ns
Data L1 cache	32 KB, 4 way, WT, private. RT: 2 ns
L2 cache	256 KB, 8 way, WB, private. RT: 6 ns
L3 cache	16 MB, 16 banks, WB, shared
L3 bank	1 MB, 8 way. RT: 14 ns (local), 26 ns (farthest)
Line size	64 bytes
DRAM	Idle RT from controller: $\approx$ 40 ns
Network	4 x 4 torus
Coherence	MESI directory protocol at L3
Step size	50 μs
Technology Parameters	
Tech. node	32 nm
Frequency	1000 MHz
Device type	Low operating power (LOP)
Temperature	330 Kelvin
eDRAM variation	$\mu(V_t)$=0.65V, $\sigma(V_t)$=0.042V (equal contrib. systematic and random), ϕ=0.4
Tools	
Architecture	SESC [25]
Time & Power	McPAT [18] and CACTI [30]
Synthesis	Synopsys Design Compiler
Statistics	R [29]
Variation	VARIUS [26]

Table 3: Evaluation parameters & tools.

We target a platform where energy efficiency is critical. Hence, we use a modest frequency. We use area, energy, and timing estimates for 32 nm technology — although the T_{ret} distribution from IBM used in this paper (Fig. 2) is for 65 nm technology. We use this distribution for lack of corresponding data at 32 nm. However, at 32 nm, the distribution changes only marginally [16]. To generate spatial maps for T_{ret}, we use the distribution parameters of Table 2. For each experiment, we average out the results of 20 T_{ret} maps. The V_t variation parameters used are shown in Table 3. The $\mu(V_t)$ and $\sigma(V_t)$ values are obtained from Kong et al. [17]. We assume an equal contribution of systematic and random components in $\sigma(V_t)$, but later we vary the breakdown.

We evaluate Mosaic designs with different combinations of tile size (T_{size}) and counter size. The parameter sweep is summarized in Table 4. A 1-bit counter rolls down every Step, and corresponds to the baseline (i.e., conventional) implementation of periodic refresh every Step. We assume a constant T of 330 K. We do not experiment with T variation (spatial or temporal) and the corresponding refresh rate adaptation.

Tile size (lines)	1, 2, 4, 8, 16, 32, 64
Counter size (bits)	1, 2, 3, 4, 5, 6, 7, 8

Table 4: Parameter sweep. We use 56 total combinations.

We compare Mosaic against: (i) the baseline (i.e., conventional) periodic refresh, (ii) a proposed scheme that uses mul-

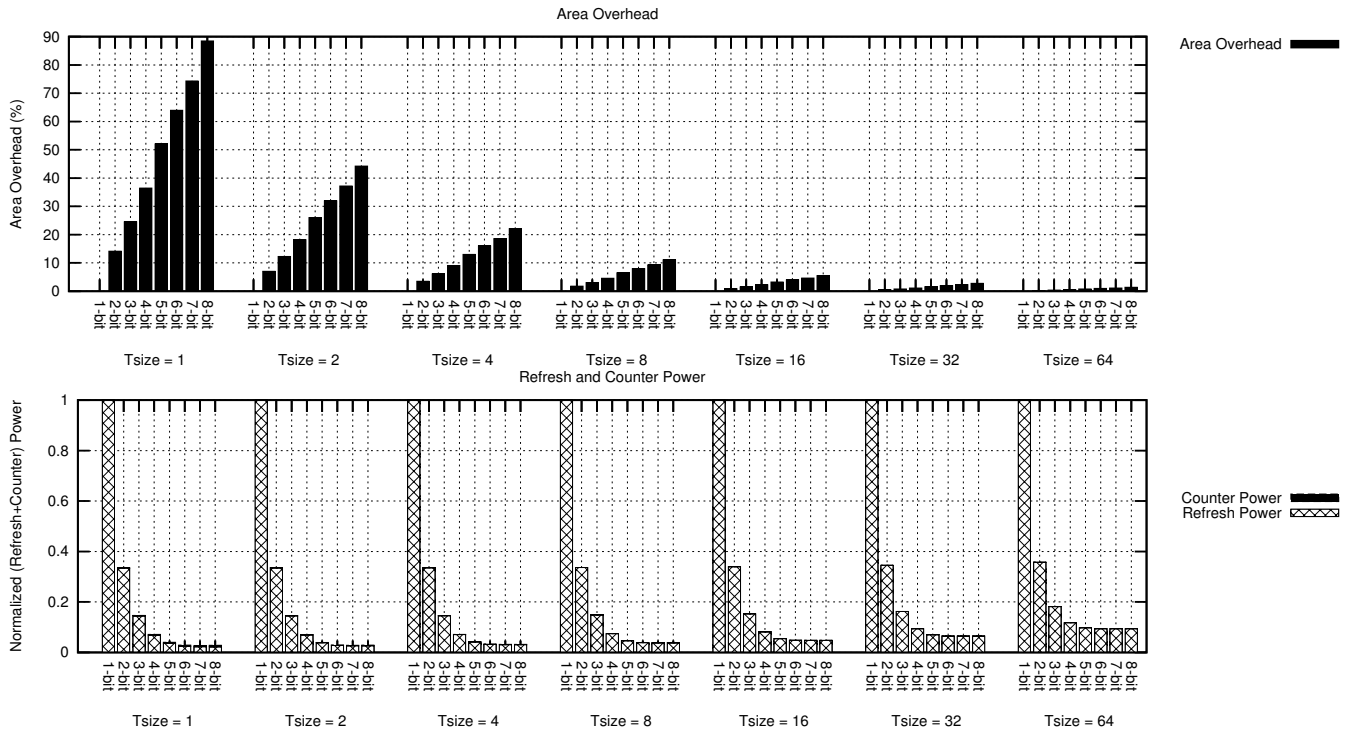

Figure 6: Area overhead of the counters (top), and power consumed refreshing L3 and operating the counters (bottom).

tiple refresh periods (RAIDR [21]), and (iii) an ideal design with the lower-bound refresh as given by Eq. 7. The ideal design is subjected to the guardband of Section 4.4, but not to the rounding-off constraint. Guardbanding is required because of port constraints, but rounding-off is an artifact of using counters.

For simplicity, we use a Step size of 50 μs. The baseline design refreshes at every Step and has no counter overhead. RAIDR [21] was proposed to refresh pages in DRAM main memories with bins that are in a geometric progression with a ratio of 2 i.e., either 64 *ms*, 128 *ms* or 256 *ms*, depending on the pages' retention times. It uses a Bloom filter per bin. Since we apply it to eDRAMs, we set it to refresh cache lines at the Step size, 2 $\times$ the Step size, or 4 $\times$ the Step size — namely, 50 μs, 100 μs or 200 μs. As we discuss in Section 8, we do not enable more bins to be faithful to the original design. In practice, more bins could be supported but, with many bins, the algorithm becomes inefficient: the bins for the higher refresh times quickly become too coarsed-grained to be useful. Moreover, with many Bloom filters, the algorithm quickly becomes slow. For Mosaic and for the ideal design, we use fixed-distance bins: 50 μs, 100 μs, 150 μs, 200 μs, etc.

We evaluate these designs with the following 16-threaded applications from SPLASH-2 and PARSEC, where the problem sizes are in parenthesis: FFT (2^{20}), LU (512x512), Radix (2M keys), Cholesky (tk29.O), Barnes (16K particles), FMM (16K), Radiosity (batch), Raytrace (teapot), Streamcluster (sim small), Blackscholes (sim medium), and Fluidanimate (sim small).

7. Evaluation

In Section 7.1, we evaluate the merit of several combinations of tile sizes and counter sizes in saving refresh energy. We choose the combination that minimizes the area and energy overheads of the counters, and maximizes refresh energy savings. In Section 7.2, we compare the resulting best Mosaic against the baseline, RAIDR, and ideal designs. We examine reduction in refreshes, system performance, and L3 energy savings. Finally, in Section 7.3, we perform a sensistivity analysis of the breakdown of $\sigma(V_t)$ into systematic and random components.

7.1. Finding the Best Mosaic

Figure 6 shows the area overhead of the counters (top), and the power consumed refreshing L3 and operating the counters (bottom) for different parameter combinations. In both plots, the X-axis is divided into 7 sets. Each set corresponds to a tile size (T_{size}) in number of L3 cache lines. Within each set, there are 8 bars for different counter sizes in bits. The area overhead of the counters is shown as a percentage of the L3 data array area. The power consumed is normalized to that of the baseline (i.e., conventional) design. It is broken down into contributions from refreshing the L3 cache, and operating the counters.

In both plots, across all tile sizes, a 1-bit counter is equivalent to not having a counter at all, and corresponds to the baseline. Therefore, in the area overhead plot, the 1-bit counter is marked zero. Likewise, in the power plot, its counter power component is zero. All the 1-bit combinations are equivalent and correspond to the baseline.

For a fixed T_{size}, the area overhead of the counters increases as the size of the counters increases. For a given counter size, its area overhead decreases as the T_{size} increases. This is because the same counter is now being shared amongst more lines.

For a given T_{size}, the refresh power decreases with the counter size. This is because the retention time of the tiles can be tracked at a much finer granularity. However, the benefits flatten out as the range of the counter approaches the maximum retention

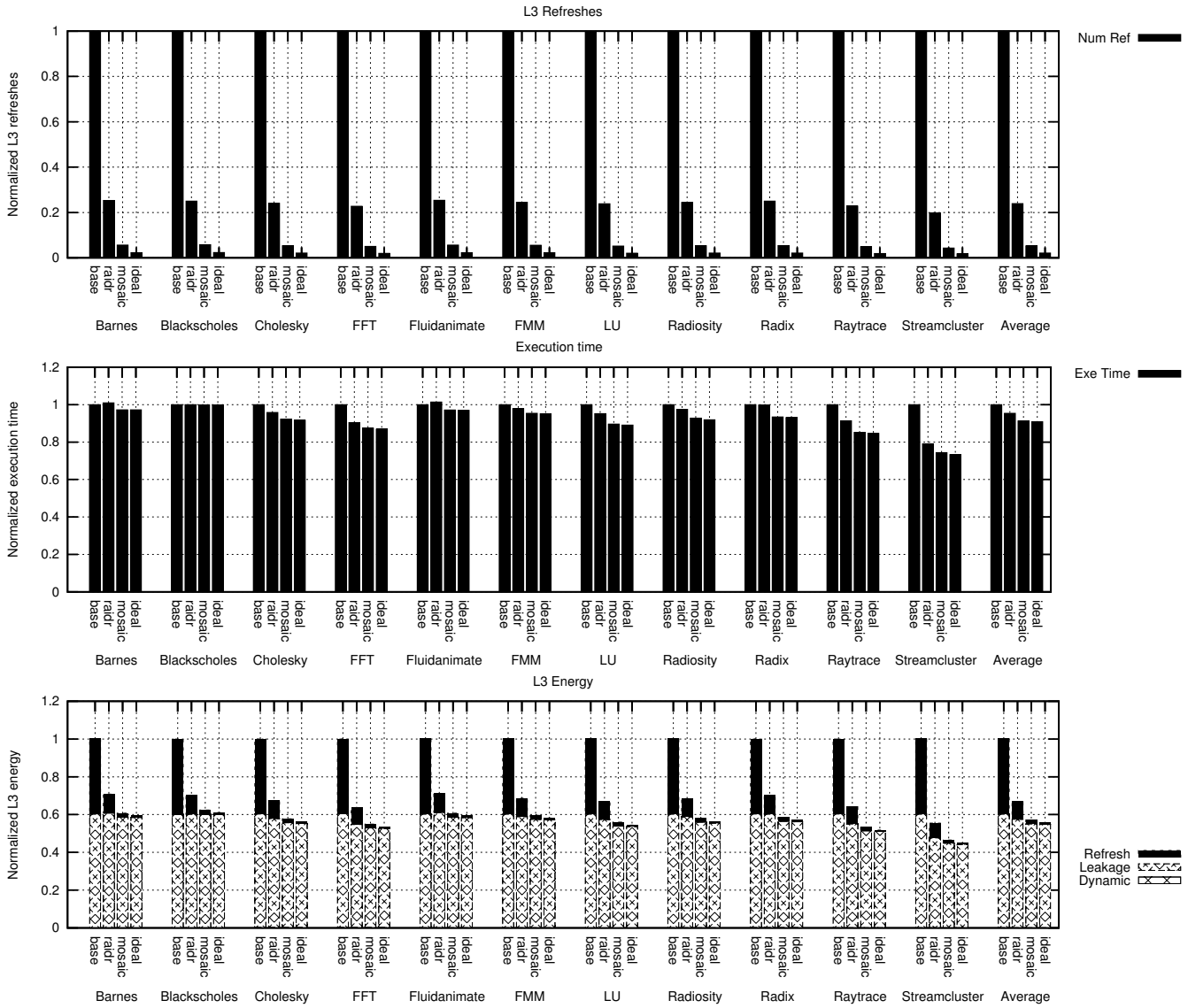

Figure 7: Number of L3 refreshes (top), execution time (center), and L3 energy consumption (bottom).

time of the distribution. As the T_{size} increases, the refresh power goes up. This is because all the lines in a tile are refreshed at the rate of the weakest line in the tile. We also see that the counter power is negligible compared to the L3 refresh power.

Therefore, there is a clear tradeoff in Mosaic between area overhead and refresh power savings. To help choose the best design, we only consider combinations with an area overhead of less than 2% and refresh power savings of at least 90%. Amongst the few candidate solutions, a tile size of 32 lines with a 6-bit counter is the best combination. It has an area overhead of 2% and refresh power savings of 93.5%. Henceforth, we call this combination the *Mosaic*.

7.2. Refresh Count, Performance & Energy

Figure 7 shows the total number of L3 refreshes (top), the execution time (center), and the L3 energy consumption (bottom) for different designs running the applications. In all plots, the X-axis is divided into 12 sets (11 for the applications and 1 for the average). Each application is run on the baseline, RAIDR, Mosaic, and ideal designs, and the result is normalized to the application's baseline design. In the L3 energy plot, each bar is broken down into dynamic, leakage and refresh energies from bottom to top. The dynamic energy is too small to be seen.

Total Refresh Count. As we go from baseline to RAIDR, to Mosaic, and to ideal, we see that the number of L3 refreshes decreases. There is little difference across applications; the difference is affected by how much the optimized designs speed-up the particular application over baseline. Overall, on average, RAIDR reduces the number of L3 refreshes to a quarter (i.e., a reduction of 4x). This is expected from the statistical distribution of T_{ret}, where most of the lines have a T_{ret} of over 200 μs. In contrast, on average, Mosaic reduces the number of refreshes by 20x. In addition, Mosaic is within 2.5x of ideal. Recall that ideal has not been subjected to the rounding-off constraint. Any practical implementation, using counters or otherwise, will have additional overheads (e.g., area or precision loss), and will close the gap between Mosaic and ideal.

Performance. Across most applications, we see that the different optimized designs perform better than the baseline. The reason for the faster execution is that the reduction in the number of refreshes reduces L3 cache blocking. The applications with most L3 accesses are the ones that benefit the most. On average, RAIDR reduces the execution time by 5%, Mosaic by 9%, and ideal by 10%. Mosaic comes to within one percent of the execution time of ideal.

L3 Energy. Across all the applications, we see that the different optimized designs significantly reduce the L3 energy compared to baseline. The reduction comes from savings in refresh energy and (to a much lesser extent) leakage energy. As is generally the case for last level caches, the fraction of dynamic energy is very small. The savings due to refresh energy reduction are the most significant. The designs reduce refresh energy by significantly reducing the number of refreshes. We can see that Mosaic eliminates practically all of the refresh energy; its effectiveness is practically the same as the ideal design.

The leakage energy is directly proportional to the execution time. Since these optimized designs reduce the execution time, they also save leakage energy. Overall, on average, RAIDR saves 33% of the L3 energy. Mosaic saves 43% of the L3 energy and is within one percent of the ideal design.

7.3. Sensitivity Analysis

Up until now, we have assumed that $\sigma(V_t)$ has equal systematic and random components — i.e., $\sigma_{rand} : \sigma_{sys}$ is 1:1. In future technology nodes, the breakdown into systematic and random components may be different. Hence, we perform a sensitivity analysis, keeping the total $\sigma(V_t)$ constant, and varying its breakdown into the σ_{rand} and σ_{sys} components. We measure the power consumed by the Mosaic configuration chosen in Section 7.1, as it refreshes L3 and operates the counters. Fig. 8 compares the resulting power. The X-axis shows different designs, as we vary the ratio $\sigma_{rand} : \sigma_{sys}$, with the random component increasing to the right. The bars are normalized to the case for $\sigma_{rand} : \sigma_{sys} = 1 : 1$, and broken down into power consumed refreshing L3 and operating the counters.

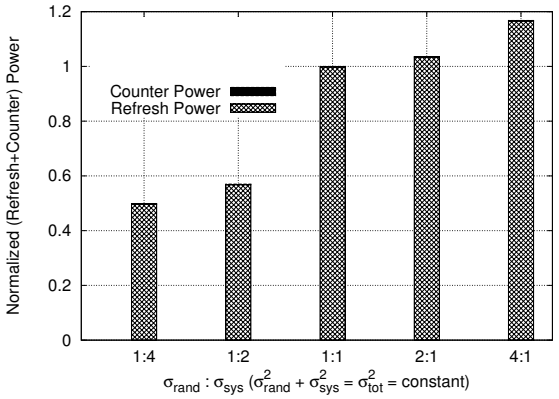

Figure 8: Power consumed refreshing L3 and operating the counters, as we change the breakdown of $\sigma(V_t)$.

The refresh power increases as the random component gains more weight. The reason is that, with relatively lower systematic component, the spatial correlation of T_{ret} decreases, eroding away some of the benefits of tiling. However, it is important to note that the increase in refresh needs is modest. Specifically, for a $\sigma_{rand} : \sigma_{sys} = 4 : 1$, the increase in power over the 1:1 configuration is only about 20%. With this increase, the power consumed refreshing L3 and operating the counters is still 92% lower than the baseline.

8. Related Work

Several approaches have been proposed to reduce the leakage (in SRAMs) or refresh (in eDRAMs/DRAMs) power in memory subsystems. One approach is to exploit the access patterns to the cache or memory. A second one is to take advantage of the intrinsic variation in the retention time of eDRAM lines or DRAM rows to save refresh power. A third one involves the use of error-correction codes (ECC) and tolerating errors.

As examples of the first class of approaches targetting SRAMs, we have Gated-Vdd [24] and Cache Decay [15, 35]. These schemes turn off cache lines that are not likely to be accessed in the near future, and thereby save leakage power. Cache Decay relies on fine-grained logic counters, which are expensive, especially for large lower-level caches. Drowsy Caches [8, 23] periodically move inactive lines to a low power mode in which they cannot be read or written. However, this scheme is less applicable in deep-nm technology nodes, where the difference between V_{dd} and V_t will be smaller.

Ghosh et al. [10] propose SmartRefresh, which reduces refresh power in DRAMs by adding timeout counters per line. This avoids unnecessary refreshes of lines that were recently accessed. Refrint [1] uses count bits instead of counters to reduce the refresh power in eDRAM-based caches in two ways. First, it avoids refreshing recently-accessed lines. Second, it reduces unnecessary refreshes of idle data in the cache. Such data is detected and then written back to main memory and invalidated from the cache. Chang et al. [4] identify dead lines in the last-level cache (LLC) using a predictor and eliminate refreshes of it.

As part of the second class of approaches, there is work focused on reducing the refresh power of dynamic memories by exploiting variation in retention time. It includes RAPID [31], the 3T1D-based cache [19], and RAIDR [21]. RAPID [31] proposes a software-based mechanism that allocates blocks with longer retention time before allocating the ones with a shorter retention time. With RAPID, the refresh period of the whole cache is determined only by the used portion.

The 3T1D-based cache [19] is an L1 cache proposal that uses a special type of dynamic memory cell where device variations manifest as variations in the data retention time. To track retention times, the authors use a 3-bit counter per line, which introduces a 10% area overhead. Using this counter, they propose refresh and line replacement schemes to reduce refreshes.

RAIDR [21] is a technique to reduce the refresh power in DRAM main memories. The idea is to profile the retention time of DRAM rows and classify the rows into bins. A Bloom filter is used to group the rows with similar retention times. There are

several differences between Mosaic and RAIDR. First, Mosaic observes and exploits the spatial correlation of retention times, while RAIDR does not. In DRAMs, an access or a refresh operates on a row that is spread over multiple chips, which have unknown correlation. Mosaic can be applied to DRAMs if the interface is augmented to support per-chip refresh.

Second, RAIDR classifies rows in a coarse manner, working with bins that are powers of 2 of the baseline (i.e., bins of t, tx2, tx4, tx8, etc.). Therefore, many bins are not helpful because the bins for the higher retention times quickly become too coarsed-grained to be useful. Mosaic tracks the retention time of lines in a fine-grained manner, using fixed-distance bins (i.e., t, tx2, tx3, tx4, etc.). This allows it to have tens of bins (64 with a 6-bit counter) and hence enables more savings in refresh power.

Finally, the RAIDR algorithm takes longer to execute with increasing numbers of bins. With 8 bins, in the worst case, it requires 7 Bloom filter checks for every line. Hence, RAIDR only uses 3 bins. The Mosaic implementation using a counter is simple and scalable.

The third class of approaches involves using ECC to enable a reduction in the refresh power [7]. ECC can tolerate some failures and, hence, allow an increase in the refresh time — despite weak cells. As a result, it reduces refresh power. One example of this approach is Hi-ECC [34], which reduces the refresh power of last-level eDRAM caches by 93%. The area overheads and refresh power reduction achieved by Hi-ECC and Mosaic are similar. However, Mosaic improves execution time by 9%, while Hi-ECC does not affect the execution time.

9. Conclusion

This paper has presented a new model of the retention times in large on-chip eDRAM modules. This model, called Mosaic, showed that the retention times of cells in large eDRAM modules exhibit spatial correlation. Based on the model, we proposed the simple Mosaic tiled organization of eDRAM modules, which exploits this correlation to save much of the refresh energy at a low cost.

We evaluated Mosaic on a 16-core multicore running 16-threaded applications. We found that Mosaic is both inexpensive and very effective. An eDRAM L3 cache augmented with Mosaic tiles increased its area by 2% and reduced the number of refreshes by 20 times. This reduction is 5 times the one obtained by taking the RAIDR scheme for main memory DRAM and applying it to cache eDRAM. With Mosaic, we saved 43% of the total energy in the L3 cache, and got very close to the lower bound in refresh energy.

References

[1] A. Agrawal, P. Jain, A. Ansari, and J. Torrellas, "Refrint: Intelligent Refresh to Minimize Power in On-Chip Multiprocessor Cache Hierarchies," in *HPCA*, Feb. 2013.

[2] "Intel eDRAM attacks graphics in pre 3-D IC days," Jun. 2013, http://www.eetimes.com/document.asp?doc_id=1263303.

[3] J. Barth *et al.*, "A 500 MHz Random Cycle 1.5ns Latency, SOI Embedded DRAM Macro Featuring a 3T Micro Sense Amplifier," *ISSCC*, Feb. 2008.

[4] M.-T. Chang *et al.*, "Technology Comparison for Large Last-Level Caches (L3Cs): Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM," in *HPCA*, Feb. 2013.

[5] J.-H. Choi, K.-S. Noh, and Y.-H. Seo, "Methods of Operating DRAM Devices Having Adjustable Internal Refresh Cycles that Vary in Response to On-chip Temperature Changes," Patent US 8 218 137, Jul., 2012.

[6] K. C. Chun, W. Zhang, P. Jain, and C. Kim, "A 700 MHz 2T1C Embedded DRAM Macro in a Generic Logic Process with No Boosted Supplies," in *ISSCC*, Feb. 2011.

[7] P. Emma, W. Reohr, and M. Meterelliyoz, "Rethinking Refresh: Increasing Availability and Reducing Power in DRAM for Cache Applications," *IEEE Micro*, Nov.-Dec. 2008.

[8] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, "Drowsy Caches: Simple Techniques for Reducing Leakage Power," in *ISCA*, May 2002.

[9] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos, "Modeling Within-Die Spatial Correlation Effects for Process-Design Co-Optimization," in *ISQED*, Mar. 2005.

[10] M. Ghosh and H.-H. Lee, "Smart Refresh: An Enhanced Memory Controller Design for Reducing Energy in Conventional and 3D Die-Stacked DRAMs," in *MICRO*, Dec. 2007.

[11] T. Hamamoto, S. Sugiura, and S. Sawada, "On the Retention Time Distribution of Dynamic Random Access Memory (DRAM)," *TED*, Jun. 1998.

[12] M. Horiguchi, "Redundancy Techniques for High-Density DRAMs," in *ISIS*, Oct. 1997.

[13] S. S. Iyer, J. E. B. Jr., P. C. Parries, J. P. Norum, J. P. Rice, L. R. Logan, and D. Hoyniak, "Embedded DRAM: Technology Platform for the Blue Gene/L Chip," *IBM Journal of Research and Development*, Mar. 2005.

[14] T. Karnik, S. Borkar, and V. De, "Probabilistic and Variation-Tolerant Design: Key to Continued Moore's Law," in *TAU Workshop*, Feb. 2004.

[15] S. Kaxiras, Z. Hu, and M. Martonosi, "Cache Decay: Exploiting Generational Behavior to Reduce Cache Leakage Power," in *ISCA*, Jun. 2001.

[16] K. Kim and J. Lee, "A New Investigation of Data Retention Time in Truly Nanoscaled DRAMs," *EDL*, Aug. 2009.

[17] W. Kong, P. Parries, G. Wang, and S. Iyer, "Analysis of Retention Time Distribution of Embedded DRAM - A New Method to Characterize Across-Chip Threshold Voltage Variation," in *ITC*, Oct. 2008.

[18] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *MICRO*, Dec. 2009.

[19] X. Liang, R. Canal, G.-Y. Wei, and D. Brooks, "Process Variation Tolerant 3T1D-Based Cache Architectures," in *MICRO*, Dec. 2007.

[20] J. Liu, B. Jaiyen, Y. Kim, C. Wilkerson, and O. Mutlu, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *ISCA*, Jun. 2013.

[21] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *ISCA*, Jun. 2012.

[22] M. Orshansky, L. Milor, and C. Hu, "Characterization of Spatial Intrafield Gate CD Variability, its Impact on Circuit Performance, and Spatial Mask-Level Correction," *TSM*, Feb. 2004.

[23] S. Petit, J. Sahuquillo, J. M. Such, and D. Kaeli, "Exploiting Temporal Locality in Drowsy Cache Policies," in *CF*, May 2005.

[24] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," in *ISLPED*, Jul. 2000.

[25] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos, "SESC Simulator," Jan. 2005, http://sesc.sourceforge.net.

[26] S. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A Model of Process Variation and Resulting Timing Errors for Microarchitects," *IEEE Trans. on TSM*, Feb. 2008.

[27] B. Stine, D. Boning, and J. Chung, "Analysis and Decomposition of Spatial Variation in Integrated Circuit Processes and Devices," *IEEE Trans. on TSM*, Feb 1997.

[28] J. Stuecheli, "POWER8," in *Hot Chips*, Aug. 2013.

[29] The R project for statistical computing, *http://www.r-project.org/*.

[30] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. Jouppi, "CACTI 5.1. Technical Report," Hewlett Packard Labs, Tech. Rep., Apr. 2008.

[31] R. K. Venkatesan, S. Herr, and E. Rotenberg, "Retention-Aware Placement in DRAM (RAPID): Software Methods for Quasi-Non-Volatile DRAM," in *HPCA*, Feb. 2006.

[32] D. F. Wendel *et al.*, "POWER7: A Highly Parallel, Scalable Multi-Core High End Server Processor," *JSSC*, Jan. 2011.

[33] N. Weste, K. Eshraghian, and M. Smith, *Principles of CMOS VLSI Design: A Systems Perspective*. Prentice Hall, 2000.

[34] C. Wilkerson, A. R. Alameldeen, Z. Chishti, W. Wu, D. Somasekhar, and S.-L. Lu, "Reducing Cache Power with Low-Cost, Multi-bit Error-Correcting Codes," in *ISCA*, Jun. 2010.

[35] H. Zhou, M. C. Toburen, E. Rotenberg, and T. M. Conte, "Adaptive Mode Control: A Static-Power-Efficient Cache Design," in *PACT*, Sep. 2001.