A Content-Aware Block Placement Algorithm for Reducing PRAM Storage Bit Writes

Brian Wongchaowart   Marian K. Iskander   Sangyeun Cho
Department of Computer Science
University of Pittsburgh
{bpw5,marianky,cho}@cs.pitt.edu

Abstract

Phase-change random access memory (PRAM) is a promising storage-class memory technology that has the potential to replace flash memory and DRAM in many applications. Because individual cells in a PRAM can be written independently, only data cells whose current values differ from the corresponding bits in a write request need to be updated. Furthermore, when a block write request is received, the PRAM may contain many free blocks that are available for overwriting, and these free blocks will generally have different contents. For this reason, the number of bit programming operations required to write new data to the PRAM (and consequently power consumption and write bandwidth) depends on the location that is chosen to be overwritten. This paper describes a block placement algorithm for reducing PRAM bit writes based on the idea of indexing free blocks using a content-based signature; computing the signature value of a new block of data to be written allows a free block with similar contents to be located quickly. While the benefit that can be realized by the use of any block placement algorithm is heavily dependent on the workload, our evaluation results show that block placement using content-based signatures is able to reduce the number of PRAM bit programming operations by as much as an order of magnitude.

1. Introduction

Phase-change random access memory (PRAM) is a storage-class memory technology with many features that make it a viable candidate for replacing flash memory and even DRAM: nonvolatility, byte-addressability, low power operation, fast random read speed, and improved write endurance [1], [2]. Like flash memory, however, there is a significant asymmetry between PRAM read and write performance. A PRAM cell encodes a bit by being in one of two states with very different electrical resistivities. Reading the cell merely involves sensing its resistivity, which is a fast, nondestructive, low-power operation, whereas updating the bit value stored in a cell requires changing the state of the cell's phase-change material, which is an expensive operation in both time and energy compared to reading a cell.

Several researchers have observed that because individual PRAM cells can be written independently, only the memory cells whose current values differ from the corresponding bits in a write request need to be programmed [3], [4]. This technique, which we will call data-comparison write (DCW) after [3], reduces power consumption, improves write bandwidth, and also contributes to a longer device lifetime by reducing wear on the cells. While other architectural mechanisms may improve write bandwidth and wear leveling, essentially the only way to reduce the energy used in writing data, given a certain PRAM technology, is to write the same data using fewer bit programming operations.

In this paper we investigate the possibility of reducing the number of PRAM bit programming operations by intelligently choosing where a block of data is written. When a block write request arrives at the PRAM, there may be many free blocks that are available for overwriting with new data. Because these free blocks will generally have different contents, the number of bit programming operations required to write the new data depends on the location that is chosen to be overwritten, assuming that DCW is used. This block placement decision provides an opportunity to reduce the overall number of bit writes required to store a given amount of data in the PRAM. To the best of our knowledge, no previous work on PRAM block placement has been published aside from wear leveling techniques whose goals are very different from ours.

While it may be desirable to write a new data block to the free location whose contents are already most similar to it, the cost of searching for this optimal location is unlikely to be acceptable if there are a large number of free blocks to choose from. An algorithm is thus needed that can quickly locate free blocks that are similar to a given data block. This paper describes such a block placement algorithm, based on the idea of indexing free blocks using a content-based signature. Computing the signature value of a new block of data allows a free block with similar contents to be quickly located.

In experiments involving a variety of disk access traces, we simulate the number of bit writes required to store a given amount of data using DCW without block placement optimization. We then compare this to the number of bit writes required when using our placement algorithm and when using an ideal nonclairvoyant placement algorithm that always finds the most similar free block in the PRAM. While the benefit of using our placement algorithm is heavily dependent on the workload, our evaluation results show that block placement optimization using content-based signatures can reduce the number of bits that need to be written by an order of magnitude in a realistic scenario.

The remainder of this paper is organized as follows. Section 2 reviews key facts about PRAM storage technology. Section 3 describes our signature-based block placement algorithm. In Section 4, we describe our evaluation methodology and report our results. Section 5 discusses some practical considerations. Finally, Section 6 reviews related work and Section 7 summarizes the paper.

2. PRAM Background

Phase-change random access memory technology is based on the use of a memory cell material that has two phases with very different electrical properties. An "amorphous phase" exhibits high resistivity. In the "crystalline phase," however, the resistivity drops by as much as five orders of magnitude [2]. These contrasting resistivity levels can be used to store a bit of information in each PRAM cell. Heating the phase-change material to its crystallization temperature for a sufficiently long period of time causes it to assume its crystalline state, while heating it to a yet higher temperature for a short period of time makes the material amorphous. Both of these operations require high-power current pulses. Reading the state of a cell is done with very low power by sensing its resistivity.

The properties of a PRAM device that are crucial to our work are that the state of individual cells can be changed independently and that testing the state of a cell is a much less expensive operation (in time and power) than setting it. Since only a fraction of the bits in a memory block stored in the PRAM will typically be changed by a write operation, it is generally advantageous to first read the existing contents of the entire block, compare it with the new data to be written, and then update only the bits that need to change. The cost of the read and comparison operations is offset by avoiding the unnecessary bit writes. Any memory technology where this "differential write" mechanism is effective can benefit from block placement optimization, since the number of bits that need to be written (and thus the cost of the write operation) depends on the location that is chosen to be overwritten.

3. Block Placement Algorithm

At a high level, our proposed block placement algorithm is very simple. Every free block in the PRAM is indexed by a signature value computed from its contents. When a new data block needs to be written, its signature is computed in the same manner. Blocks with matching signatures are assumed to have similar contents, so a free location can be chosen for a new data block by looking up the signature of the new data in the free block index. The following subsections describe the process of computing a block signature and choosing a free location in greater detail.

3.1. Block Signature Computation

The goal of our block signature computation algorithm is to succinctly describe a block's contents using a short, fixed-length bit string. Perhaps the simplest way to describe the contents of a block of data is to treat it as an unordered collection of bits and count the number of 1-bits in this collection. If all information about the positions of bits is discarded in this way, however, then a block consisting of n 1-bits followed by n 0-bits would not be distinguishable from a block consisting of n 0-bits followed by n 1-bits. Changing one of these two blocks into the other requires updating all of the bit values, so these two blocks should not share the same signature.

Our solution to this problem is to partition a block into multiple unordered collections of bits ("sets") and count the number of 1-bits in each set. Using two sets, this approach is able to distinguish a block of 0-bits followed by 1-bits from a block of 1-bits followed by 0-bits. Increasing the number of sets produces a higher-resolution picture of the block contents, at the cost of an increase in the signature size or a decrease in the number of bits that can be allotted to storing the (approximate) number of 1-bits in each set. In our evaluation, we will explore the trade-off between having many sets per block versus many bits per set.

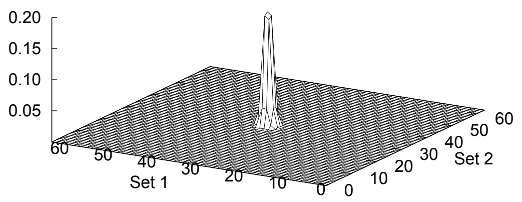


Figure 1. Distribution of signature values for uniformly distributed random data.

There is an additional reason why dividing a block into many sets is desirable. If the number of 1-bits and 0-bits in a set is equal, then in the worst case overwriting such a set with a different set that has the same count of 1-bits and 0-bits will cause all of the bits in the first set to change, since all of the 1-bits can change to 0-bits and vice versa. If a set that consists of 90% 1-bits is overwritten with another set consisting of 90% 1-bits, however, then at most 20% of the bits in the first set change (10% of the bits change from 1 to 0, while 10% of the bits change from 0 to 1). A similar result holds for two sets consisting of 90% 0-bits. When blocks whose corresponding sets have a similar number of 1-bits overwrite each other, there is thus a better guarantee on the number of bits that change if these sets are heavily biased towards either 0-bits or 1-bits. This is intuitively more likely when sets are small.

Our approach to block signature computation therefore consists of partitioning a block into n equal-size sets of bits and using the number of 1-bits in each set as one component of a vector of length n that serves as a signature for the block as a whole. In order to limit the size of a block signature, we fix the number of bits allotted to representing the number of 1-bits in each set. To be precise, suppose that a set contains $s = 2^k$ bits and that the count of 1-bits in the set is b, where $0 \leq b \leq s$. If m bits are reserved for each set in the block signature and $m \leq k$, then the interval $[0, s]$ is divided into 2^m regions, each of size $s/2^m$ except for the last region, which is wider than the others by 1: $[0, s/2^m - 1], [s/2^m, 2s/2^m - 1], \ldots, [s - s/2^m, s]$. The index of the region in which b lies (counting from 0) is an m-bit integer that represents the approximate value of b—we call this m-bit integer the *signature value* of the set. If $m > k$, then the signature value of the set is simply b, the count of 1-bits in the set. The concatenation of the n set signature values forms an mn-bit signature value for a block.

For example, Figure 1 shows the distribution of signature values for 512-byte blocks of uniformly distributed random data when there are 2 sets per block and 6 bits are used in the signature to represent the approximate number of 1-bits in each set. The x- and y-axes correspond to the 6-bit signature values for the two sets, while the z-axis gives the probability that a block has a particular combination of set signature values. The distribution peaks in the center of the plot since each set can be expected to contain approximately as many 1-bits as 0-bits. We will use similar plots in Section 4 to give some indication of the contents of the data blocks in our experiments, with the change that the z-axis will give the actual frequency of a particular block signature value.

3.2. Block Placement Decision

We assume that there is a mechanism by which the operating system notifies the PRAM controller that certain blocks can be regarded as free. Signature values for these blocks are computed as described above and the address of each block is stored in an index structure that maps a given signature value to a list of free blocks whose contents have that signature value. Newly freed blocks have their addresses appended to the end of these lists. When a request to write a new data block arrives at the PRAM, the signature of the new block is computed. There are now two cases to consider: the PRAM either contains at least one free block with the same signature value as the new block, or there are no free blocks with a matching signature in the PRAM:

- If the PRAM contains a free block with a matching signature value, then a lookup operation on the free block index using the new block's signature returns a list of free blocks with the same signature. This list of block addresses is searched for a free block that is most similar to the new data block to be written. (Free block a is defined to be more similar to the new block than free block b if writing the new block on top of a would require fewer bit writes than writing the new block on top of b.) Since there may be a large number of free blocks with the same signature value, our algorithm is parameterized by a search distance limit that determines the maximum number of free blocks that are considered per placement decision. The search for the best match among the free blocks that have the same signature as the new data starts at the head of the list of free block addresses. When the search distance limit is reached, or there are no more free blocks in the list, the free block that was found to be most similar to the block to be written is chosen for overwriting and its address is removed from the list.

- If there are no free blocks in the index with the same signature value as the block to be written, then a free block list for another signature value is retrieved and a free block is selected from it. Although it is possible to search the free block index for signature values that are close to the signature of the block to be written according to some distance metric, our simulated implementation picks an arbitrary free block list from the index and chooses a free block from it as described in the preceding case. This avoids the cost of looking up many signature values in the index.

4. Evaluation

In this section we evaluate the effectiveness of our block placement algorithm using different parameter settings on disk traces chosen to provide a variety of distributions of write data blocks: two simple synthetic traces using uniformly distributed random data, two traces in which the write request data consists of digital camera images (compressed and uncompressed), a trace of source code compilation, a suspend-to-disk trace, and finally a trace of data snapshots from benchmark programs that represent typical high-performance computing applications.

4.1. Experimental Methodology

Each of our traces specifies the initial contents of the free blocks in the PRAM and a stream of block write requests from the operating system. We simulate the placement decision made by our algorithm for each data block that is written and report the number of bit writes that would be caused by the write requests in each trace if DCW is used without block placement optimization, if our block placement algorithm is used in combination with DCW, and if a greedy exhaustive search is always carried out to find the free block in the PRAM that would require the smallest number of bit writes using DCW. No wear leveling is performed, so there is a fixed mapping between logical block addresses and physical PRAM cells when no block placement optimization is used.

In our experiments we use a fixed block size of 512 bytes and block signature sizes of 16 and 32 bits. For each of the two signature sizes, we varied the number of sets per block from 1 to the number of bits in the signature. The number of bits used to encode the number of 1-bits in each set is determined by the signature size and the number of sets per block. For each combination of sets per block and bits per set in a block signature, we compare the effectiveness of search distance limits of 1, 5, and 10 within a list of free block addresses. A search distance limit of 1 is particularly attractive because it means that the block placement decision is made using the free block signature index alone without the need to read and compare the contents of multiple free blocks in the PRAM.

			Search distance limit		
Sets/block	Bits/set	Total bits	1	5	10
1	16	16	49.99	49.08	48.79
2	8	16	49.97	49.07	48.78
4	4	16	49.97	49.06	48.77
8	2	16	49.94	49.03	48.73
16	1	16	49.88	49.28	49.25
1	32	32	49.99	49.08	48.79
2	16	32	49.98	49.11	48.86
4	8	32	49.96	49.38	49.30
8	4	32	49.93	49.04	48.75
16	2	32	49.88	49.28	49.25
32	1	32	49.89	49.89	49.89

Table 1. Percentage of the random trace requiring a bit write.

4.2. Simple Synthetic Traces

We begin with two simple synthetic traces using uniformly distributed random data.

4.2.1. random Trace

In this trace the PRAM initially has 128 MB of free space containing uniformly distributed random data. The stream of write requests consists of sequentially writing 64 MB of uniformly distributed random data to the PRAM. Table 1 shows our experimental results. The number of bit writes for each combination of algorithm parameter settings is reported as a percentage of the total size in bits of the write requests in the trace.

All of the results in Table 1 are close to 50%, which is the expected number of bit updates required to change a data block generated uniformly at random into another data block generated uniformly at random. Using DCW without placement optimization in fact caused 50.00% of the bits in the trace to be written, while using greedy exhaustive search only improved this to 46.47% of the bits in the trace being written. Increasing our algorithm's search distance limit slightly reduces the number of bit writes. In general, we do not expect placement optimization to be effective for random or compressed data where the data blocks are uniformly distributed over the space of possible data blocks because the PRAM is unlikely to contain a free block that is very similar to a new block of data to be written.

Sets/block	Bits/set	Total bits	Search distance limit 1	5	10
1	16	16	49.94	48.84	48.34
2	8	16	49.90	48.71	48.07
4	4	16	49.96	49.04	48.73
8	2	16	49.88	48.80	48.27
16	1	16	37.68	3.81	0.05
1	32	32	49.94	48.84	48.34
2	16	32	46.96	36.86	28.43
4	8	32	34.17	10.57	2.91
8	4	32	49.48	47.19	46.48
16	2	32	37.68	3.81	0.05
32	1	32	0.00	0.00	0.00

Table 2. Percentage of the permutation trace requiring a bit write.

Sets/block	Bits/set	Total bits	Search distance limit 1	5	10
1	16	16	49.53	48.64	48.35
2	8	16	49.54	48.64	48.36
4	4	16	49.55	48.65	48.36
8	2	16	49.56	48.66	48.37
16	1	16	49.57	48.83	48.65
1	32	32	49.53	48.64	48.35
2	16	32	49.54	48.74	48.54
4	8	32	49.54	49.06	49.01
8	4	32	49.52	48.64	48.36
16	2	32	49.50	48.77	48.59
32	1	32	49.54	49.52	49.52

Table 3. Percentage of the jpeg trace requiring a bit write.

4.2.2. permutation Trace

The initial state of the PRAM in this experiment is the same as for the random trace (128 MB of free space containing uniformly distributed random data). The trace of write requests was generated by choosing a random sample of half of the free blocks in the PRAM, and then sequentially writing these blocks back to the PRAM in random order.

Table 2 reports our results for this trace. Using exhaustive search, no bits are written, since all the blocks in the write requests are already contained in the PRAM. Our placement algorithm can match this result when there are 32 sets per block, 1 bit per set, and a search distance limit of at least 2 is used (the figure of 0.00% shown in Table 2 for a search distance limit of 1 represents a very small, but nonzero, number of bit writes). DCW without placement optimization caused 50.00% of the bits in the trace to be written, since it consistently writes one block of random data drawn from a uniform distribution on top of another block of random data drawn from a uniform distribution.

4.3. Digital Camera Image Traces

The next two traces simulate writing digital camera images to a PRAM memory card that contains no useful data, but previously contained another set of images that have now been deleted (but not overwritten). The images used in these traces are test shots taken from a review of the Nikon D3X digital SLR camera.[1]

4.3.1. jpeg Trace

The initial state of the PRAM in this trace represents a 256 MB digital camera memory card that contains a FAT32 file system populated with 188 MB of JPEG images (free blocks are zeroed). In this trace we assume that these images have been copied to more permanent storage and simulate the number of bit writes required to overwrite them with a new set of images. In particular, the trace captures the block writes caused by reformatting the PRAM with a FAT32 file system and writing a different 170 MB set of JPEG images into the file system to simulate a user taking new photos. Because the file system will be reformatted, all of the blocks in the PRAM are considered free at the beginning of the trace.

1. http://www.imaging-resource.com/PRODS/D3X/D3XTHMB.HTM

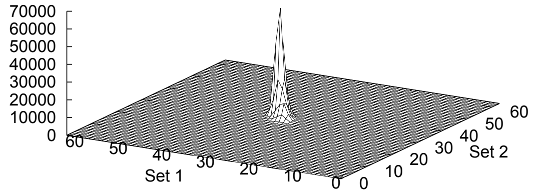


Figure 2. Distribution of signature values for the write request data blocks of the jpeg trace.

Greedy exhaustive search resulted in 46.06% of the bits in this trace being written, which is only slightly lower than our results for uniformly distributed random data. DCW with no placement optimization also behaves similarly to the case of uniformly distributed random data: 49.78% of the bits in the trace were written. Table 3 gives the performance of our placement algorithm for this trace. Once again, compressed data is little different from random data from the point of view of block placement optimization. This is confirmed by Figure 2, which shows the distribution of block signature values for this trace when there are 2 sets per block and 6 bits are used in the signature per set. It is similar to Figure 1, which shows the distribution of block signature values for uniformly distributed random data.

Sets/block	Bits/set	Total bits	Search distance limit 1	5	10
1	16	16	30.34	29.39	28.99
2	8	16	30.26	29.31	28.92
4	4	16	30.29	29.54	29.31
8	2	16	30.57	30.00	29.78
16	1	16	32.55	31.86	31.36
1	32	32	30.34	29.39	28.99
2	16	32	30.25	29.00	28.66
4	8	32	30.29	29.64	29.57
8	4	32	30.31	29.43	29.10
16	2	32	30.44	29.68	29.33
32	1	32	32.52	31.61	31.29

Table 4. Percentage of the dng trace requiring a bit write.

4.3.2. dng Trace

The dng trace is similar to the JPEG trace, except that uncompressed Adobe Digital Negative (DNG) versions of the same images are used to simulate the raw image formats favored by professional photographers. The initial state of the PRAM represents a 1 GB memory card that contains a FAT32 file system populated with 946 MB of DNG images. The trace captures the block writes caused by reformatting the file system and writing 757 MB of new DNG images to simulate a user taking new photos. All of the blocks in the PRAM are considered free at the beginning of the trace. Figure 3 shows the distribution of block signature values for this trace when there are 2 sets per block. The fact that the distribution peaks along the diagonal indicates that the bias towards zeros or ones tends to be similar for the first and second halves of each data block. Most of the blocks are moderately biased towards having more zeros than ones.

Table 4 gives our experimental results for this trace. Using DCW alone resulted in 32.51% of the bits in the trace being written, so our block placement algorithm yields only a small additional gain over DCW, even with the considerable overhead of reading up to 10 free blocks for comparison. This poor performance would not be a sign of any flaw specific to our placement algorithm, however, if even the "ideal" approach of searching through all of the available free blocks before making a placement decision does not perform much better. Because it was not feasible to simulate placement by exhaustive search for a trace of this size, we randomly sampled 1/8 of the blocks from the initial PRAM contents and writes trace to obtain a "scaled down" version of the dng trace that involves writing 95 MB of data to a 128 MB PRAM. Greedy exhaustive search on this trace shows 25.02% of the trace being written, suggesting that DCW alone already exploits most of the available bit write savings, leaving our placement algorithm with little room for improvement.

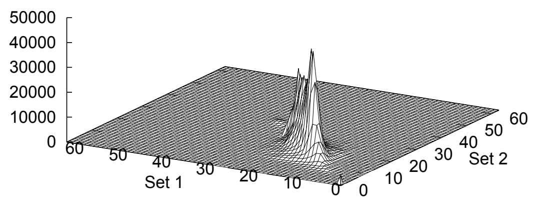


Figure 3. Distribution of signature values for the write request data blocks of the dng trace.

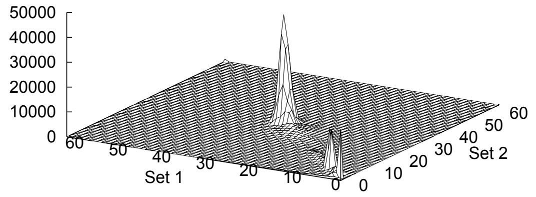


Figure 4. Distribution of signature values for the write request data blocks of the kernelbuild trace.

4.4. Kernel Build Trace

The scenario captured by this trace is that a developer or system administrator compiles five different Linux 2.6.31 kernels from the source code, using different configuration settings. Each of the five kernels is built in the same 256 MB ext4 file system, which is initially empty and contains only free blocks filled with 0-bits. Each build generates an average of 58 MB of block write requests, but the files generated by each build are deleted after the build completes, at which point the next build is started. When the generated files are deleted after each build, the data blocks that they occupied are marked as free for the block placement algorithm. Figure 4 shows the distribution of block signature values for this trace when there are 2 sets per block. This plot resembles that of the distributions for random data and JPEG images, showing that the number of ones and zeros is roughly balanced for most data blocks.

Using DCW without placement optimization resulted in 38.21% of the bits in the trace being written, whereas only 23.82% of the trace was written using exhaustive search, indicating that placement optimization is potentially helpful for this trace. This can be expected because similar files are generated by each build, and the data blocks for one build's files are available for overwriting during the following build.

Sets/block	Bits/set	Total bits	Search distance limit		
			1	5	10
1	16	16	33.87	32.63	32.08
2	8	16	33.20	32.19	31.68
4	4	16	33.20	32.39	32.01
8	2	16	33.94	32.83	32.29
16	1	16	32.03	30.68	30.27
1	32	32	33.87	32.63	32.08
2	16	32	32.35	31.29	31.10
4	8	32	31.57	30.97	30.90
8	4	32	32.53	31.51	30.98
16	2	32	31.96	30.60	30.20
32	1	32	30.46	30.13	30.09

Table 5. Percentage of the kernelbuild trace requiring a bit write.

Table 5 shows the results for our block placement algorithm. The best results lie roughly midway between the lower bound provided by exhaustive search and the upper bound of DCW without placement optimization; using a search distance limit greater than 1 has little benefit.

4.5. Suspend-to-Disk (swsusp) Trace

In this experiment we exhibit a scenario in which our block placement algorithm is very effective compared to using DCW alone. The experiment simulates a small 128 MB PRAM used solely for storing suspend-to-disk images (this would be a reasonable size for a Linux thin client). A Linux system was booted, the user checked her email, and then the system was suspended to disk using the mainline 2.6.31 kernel's swsusp (software suspend) implementation. The contents of the suspend partition at this point are used as the initial state of the PRAM. The system was then resumed, the user checked her email again, and the system was suspended a second time. The trace of write requests consists of the data written sequentially to disk by this second suspend operation. Since the existing contents of the partition (from the first suspend) are meaningless, all of the blocks in the PRAM are marked as free at the beginning of the trace. Figure 5 shows the distribution of block signature values for this trace when there are 2 sets per block; in this trace most data blocks contain more 0-bits than 1-bits.

Table 6 reports our experimental results. DCW without placement optimization caused 16.09% of the bits in the trace to be written. Using 4 sets per block, 8 bits per set, and a search distance limit of 1, our placement algorithm was able to reduce the number of bit writes required to 2.01% of the trace, or less than 12.5% of the number required when using DCW alone. Of course, we recognize that this trace is exceptionally well suited for block placement optimization. Placement by a greedy exhaustive search through all of the available free blocks resulted in just 0.32% of the bits in the trace being written, whereas picking a free block uniformly at random for each placement decision resulted in 26.20% of the bits in the trace being written. Nevertheless, we conclude from this experiment that block placement optimization can be a very effective technique in a realistic scenario compared to using DCW alone.

Sets/block	Bits/set	Total bits	Search distance limit		
			1	5	10
1	16	16	7.75	3.05	2.54
2	8	16	4.46	2.27	2.08
4	4	16	9.44	4.26	3.26
8	2	16	12.52	8.57	6.77
16	1	16	12.26	8.57	8.00
1	32	32	7.75	3.05	2.54
2	16	32	2.27	2.17	2.16
4	8	32	2.01	1.96	1.95
8	4	32	3.11	2.02	1.87
16	2	32	6.52	4.46	3.85
32	1	32	7.08	5.44	4.91

Table 6. Percentage of the swsusp trace requiring a bit write.

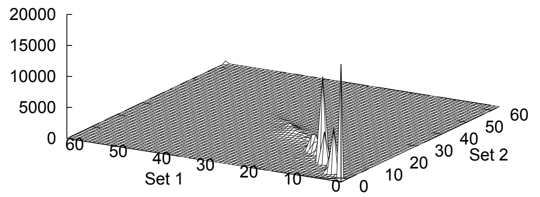
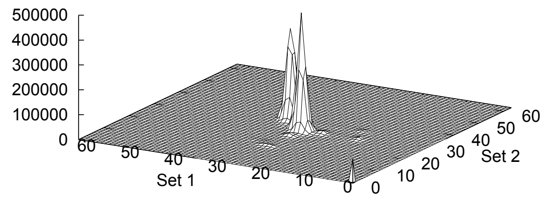


Figure 5. Distribution of signature values for the write request data blocks of the swsusp trace.

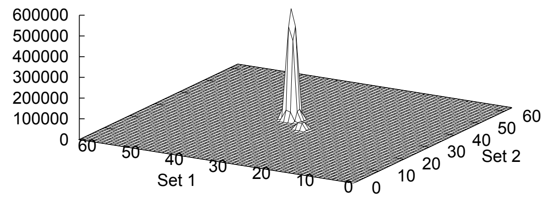
4.6. NAS Parallel Benchmark Snapshot Traces

This experiment studies data snapshot traces from high-performance computing applications. It is common to generate data snapshots as such applications run so that precious computing cycles can be salvaged in the event of a system failure. We employ data snapshots from four NAS Parallel Benchmark programs [5]: BT (block tridiagonal solver), CG (conjugate gradient), FT (FFT), and MG (multigrid). Snapshots of main data structures are captured and stored after the 2nd, 6th, 10th, and 14th iterations. The snapshot sizes range from 317 MB (CG) to 1.25 GB (FT).

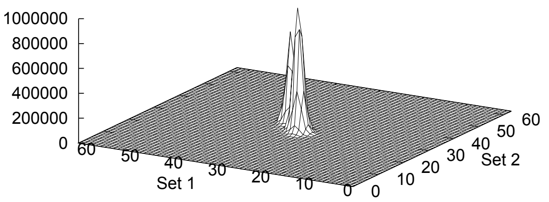
We assume that only the most recently generated snapshot needs to be kept to enable recovery. Each benchmark program therefore produces a trace in



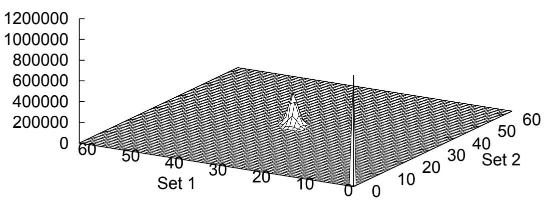
(a) BT



(b) CG



(c) FT



(d) MG

Figure 6. Distribution of signature values for the write request data blocks of the NAS snapshot traces.

which the first and second snapshots are written to the PRAM storage device, the space occupied by the first snapshot is freed before the third snapshot is written, and the space occupied by the second snapshot is freed before the fourth snapshot is written. Figure 6 shows the distribution of block signature values for these four traces when there are 2 sets per block.

We compare three block placement algorithms in this experiment: our signature-based block placement algorithm, an algorithm that chooses uniformly at random from the available free blocks, and a manual block placement strategy. The signature-based placement algorithm was configured to use 4 sets per block, 8 bits per set, and a search distance limit of 1, since this combination of parameter settings performed well on the kernelbuild and swsusp traces. Our manual block placement strategy was to write the first and second snapshots of each trace sequentially to the PRAM, and then exactly overwrite the first snapshot with the third snapshot and the second snapshot with the fourth snapshot (this is possible because all of the snapshots generated by a single program have the same size).

In this experiment we assume that there are initially 3 GB of free space in the PRAM and consider five possibilities for the contents of this space: all 0-bits, and as many copies of the four snapshots produced by the BT, CG, FT, and MG programs, respectively, as are required to fill up the 3 GB of free PRAM space. The intuition behind these choices is that we would like to learn the number of bit writes caused by writing one benchmark program's data snapshots onto a PRAM where another program has just written its snapshots.

Table 7 reports the percentage of the four snapshot traces requiring a bit write when DCW is used with random block placement for each of the five choices for the initial PRAM contents. Tables 8 and 9 report the corresponding results for our signature-based block placement algorithm and manual block placement. The signature-based block placement algorithm clearly outperforms random block placement, but in most cases the manual block placement strategy did still better. The reason for this is that writing the third and fourth data snapshots of each trace on top of the corresponding freed blocks of the first and second snapshots turned out to be an excellent optimization that the signature-based placement algorithm fails to make. It can thus be concluded that a programmer with knowledge of the semantics of the data written by an application can potentially make better data placement decisions than our automatic algorithm, a fact that we do not find very surprising.

5. Discussion

The experiments described in the preceding section show that while the signature-based block placement technique presented in this paper can be very beneficial for some applications, its effectiveness in any particular setting needs to be evaluated on a case-by-case basis. Although further research is required to more exactly quantify the cost-benefit trade-off of using our block placement strategy, the work described in this paper

Trace	Initial PRAM contents				
	Zeros	BT	CG	FT	MG
BT	47.81	49.65	50.37	49.67	49.31
CG	51.16	49.13	43.46	49.47	49.57
FT	48.93	49.38	50.00	48.81	49.35
MG	36.29	48.38	49.81	48.84	43.93

Table 7. Percentage of the NAS snapshot traces requiring a bit write when DCW is used with random block placement.

Trace	Initial PRAM contents				
	Zeros	BT	CG	FT	MG
BT	42.24	0.00	43.89	43.27	42.48
CG	32.96	39.45	0.00	34.38	40.14
FT	42.84	41.34	43.56	16.06	42.67
MG	33.65	40.29	41.85	41.55	0.00

Table 8. Percentage of the NAS snapshot traces requiring a bit write when signature-based block placement is used with 4 sets per block, 8 bits per set, and a search distance limit of 1.

Trace	Initial PRAM contents				
	Zeros	BT	CG	FT	MG
BT	38.27	14.70	39.91	38.83	39.15
CG	26.67	25.66	0.38	25.14	25.81
FT	42.27	42.05	43.46	17.90	42.78
MG	33.15	39.64	40.42	40.03	15.04

Table 9. Percentage of the NAS snapshot traces requiring a bit write when DCW is used with a manual block placement strategy.

opens up a promising line of research. In particular, the fact that the energy required to write a block of data to a PRAM storage device depends on the location to which this data is written presents a novel opportunity for reducing PRAM energy consumption.

In the remainder of this section, we discuss some practical considerations relevant to our block placement algorithm: the utility of block placement optimization beyond conventional mass storage applications, the cost in time and space of maintaining and using the free block signature index, and the hardware and operating system support that our algorithm can benefit from.

5.1. Applications Beyond Mass Storage

In this paper we have focused on applications in which the PRAM replaces a block storage device such as magnetic disk or flash memory, rather than DRAM. There are two primary reasons for this. First, block placement optimization only makes sense when a placement decision can be made for a relatively large block of data at a time, since any benefit gained from making a placement decision at the granularity of individual memory words is almost certainly not worth the time and storage overhead of making the placement decision. Second, the additional latency introduced by our placement algorithm is more likely to be tolerable when writes are asynchronous (that is, the system does not block waiting for the write to complete).

We believe that our techniques are also applicable, however, when PRAM is used as a large main memory behind a fast DRAM cache. In this case only blocks that are evicted from the DRAM cache are written to the PRAM, and a large write queue can be provided to prevent stalls caused by PRAM write delays. We also note that if low write latency is required at certain times, then the placement optimization step can be skipped without any impact on the correctness of the PRAM's operation. Placement optimization trades a reduction in the number of bit writes for increased latency, and the decision about whether this trade-off is desirable can be made dynamically at run time. In fact, the search distance limit parameter of our algorithm allows the trade-off between placement latency and the number of bit writes to be fine-tuned.

5.2. Time and Space Complexity

We require an index data structure that supports efficient retrieval of a list of free blocks that have a specified block signature value, deletion of free blocks that are chosen for overwriting, and addition of newly freed blocks. A standard data structure for this purpose is a balanced search tree that maps block signature values to a linked list of block addresses. A balanced search tree such as a red-black tree guarantees a worst-case time complexity of $O(\lg n)$ for the operations of search, insertion, and deletion, where n is the number of nodes in the tree. In our case the number of nodes is equal to the number of distinct signature values for the free blocks in the PRAM (assuming that when the last free block with a particular signature value is used up, the node corresponding to that signature value is deleted from the tree). In practice, the number of distinct signature values may be considerably smaller than the number of free blocks in the PRAM, as the plots of block signature distributions in Section 4 suggest.

Once the node corresponding to a signature value is found during the processing of a write request, the time complexity of searching for the best match within the linked list of free blocks with that signature value depends on the search distance limit parameter of our algorithm. A search distance limit of 1 means that the block address at the head of the list is always chosen and removed in constant time. A search distance limit of $d > 1$ corresponds to at most d block read and comparison operations, followed by the removal of one of the first d addresses from the linked list. If the only entry in the linked list is removed, then the entire node is deleted from the search tree in $O(\lg n)$ time. Inserting the address of a newly freed block into the index consists of either a tree search or a tree insertion operation, followed by appending a new element to a linked list of free block addresses; this also requires $O(\lg n)$ time.

The amount of memory needed to store a balanced tree of linked lists of free block addresses is linear in the sum of the number of tree nodes and the number of linked list entries. Both of these values are bounded by the number of free blocks in the PRAM, so the total storage requirement of our algorithm is linear in the number of free blocks. The free block signature index will perform optimally if it is kept in DRAM, but if this is not feasible, then it can be stored in PRAM using a standard database index structure such as a B-tree, parts of which can be cached in DRAM.

5.3. System Support

Finally, our proposed scheme will benefit from additional support from the operating system or the hardware. In our experiments the smallest write request received from the operating system overwrites an entire 512-byte block. Two optimizations are possible if the operating system knows that only a small fraction of a block has been modified. First, the operating system can give a hint to the PRAM controller that the best location for the block is probably its current physical location so that the placement optimization step can be skipped. Second, if the operating system knows exactly which words in the block have been modified, then it is wasteful for the operating system to send the entire block to the PRAM and have it compare the old data block with the new data to see which bits need to be updated. The operating system can instead send only the words that have changed and their addresses to the PRAM. This idea is similar to the "partial write" scheme of [6].

If there is an "auxiliary" or "spare" area associated with each PRAM block (as in the case of NAND flash memory [7]), the signature of the block's contents can be recorded there by the PRAM controller when it updates the block (this signature must already be computed in order to decide where a new block of data should be written). Whenever a block becomes free, its signature can be retrieved from the auxiliary area. Such hardware support would eliminate the need to recompute the signature of a newly freed block in order to insert it into the free block index.

6. Related Work

Techniques for reducing the number of PRAM bit writes can be employed at several levels. First, optimizations at the application and operating system levels can reduce the amount of memory and disk storage needed to perform a certain task. Most obviously, data that is not immediately needed can be compressed both in memory and on disk. Memory pages or disk blocks with identical contents can also be shared via a copy-on-write mechanism [8].

At the level of hardware architecture, several optimizations are possible when a DRAM buffer or cache is placed between the processor and a PRAM-based main memory. The mechanism of "partial writes" [6] tracks which memory words in a cache block or which blocks in a memory page are modified by a processor write instruction. When the dirty block or page is evicted from the DRAM buffer, only the modified part needs to be written back to the PRAM. On a page fault, the "lazy write" scheme [9] places the page in the DRAM cache without writing it to the PRAM. The page is only written to the PRAM main memory when it is evicted from the DRAM cache.

Within the PRAM module itself, data-comparison write (DCW) [3] and Flip-N-Write [10] reduce the number of bit write operations needed to write a memory word. They both rely on the idea of replacing a write operation with a read, modify, and write sequence. DCW compares the new data word to be written with the old data word and only updates the bits that differ. If the number of bits in a memory word is N, however, DCW still updates all N bits in the worst case. Flip-N-Write improves significantly on this by limiting the maximum number of bit writes to $N/2$ in all cases. An auxiliary flip bit is added to each word that inverts the logical meaning of the bits stored in the PRAM when it is set. If writing the new data word in noninverted form would require more than $N/2$ bit updates (including setting the flip bit to 0 if it is currently 1), then the flip bit is set to 1 if it is currently 0 and the memory word is updated to match the bitwise complement of the new data word to be written. Both DCW and Flip-N-Write are compatible with our block placement algorithm; we chose to use DCW in this paper for a more intuitive presentation.

The idea of writing a data block at a physical location different from the logical address supplied in a write request is at the heart of PRAM wear-leveling schemes (see [11] and [4] for recent proposals). Although this can be viewed as a form of block placement optimization, the goal of wear leveling is to minimize the number of times that the most frequently written bit in the PRAM is overwritten with a new value, whereas our goal is to minimize the total number of bit writes (for lower energy usage). If both wear leveling and content-based block placement optimization are to be used simultaneously, then the address translation performed by the block placement algorithm should be done first, and the resulting address given as input to the wear leveling algorithm. This prevents the block placement decision from defeating the purpose of the wear leveling scheme.

7. Conclusions

In this paper we have proposed a new approach to reducing the number of bit programming operations required to write a sequence of data blocks to a PRAM storage device. When a differential write scheme such as DCW is employed, the number of bit writes needed to store a new data block in the PRAM depends on the current contents of the location at which the block is written. This paper describes an efficient block placement algorithm for choosing a free block whose contents are already similar to a new data block. We compared the number of bits written by our algorithm to the number that would be written using DCW without placement optimization on a variety of disk traces. With the right parameter settings, our block placement algorithm was able to reduce the number of bit writes needed to as low as 12.5% of the number needed when DCW alone is used. This figure was achieved without reading and comparing multiple free blocks, demonstrating that block placement using content-based signatures is a promising approach to reducing PRAM energy consumption.

Acknowledgments

This research was supported in part by the National Science Foundation under grant numbers CCF-0702236 and CCF-0952273.

References

[1] R. F. Freitas and W. W. Wilcke, “Storage-class memory: The next storage system technology,” *IBM Journal of Research and Development*, vol. 52, no. 4/5, pp. 439–447, 2008.

[2] S. Raoux *et al.*, “Phase-change random access memory: A scalable technology,” *IBM Journal of Research and Development*, vol. 52, no. 4/5, pp. 465–479, 2008.

[3] B.-D. Yang, J.-E. Lee, J.-S. Kim, J. Cho, S.-Y. Lee, and B.-G. Yu, “A low power phase-change random access memory using a data-comparison write scheme,” in *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS 2007)*, 2007, pp. 3014–3017.

[4] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, “A durable and energy efficient main memory using phase change memory technology,” in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA’09)*, 2009, pp. 14–23.

[5] D. Bailey *et al.*, “The NAS parallel benchmarks,” NASA RNR Technical Report RNR-94-007, Mar. 1994.

[6] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, “Architecting phase change memory as a scalable DRAM alternative,” in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA’09)*, 2009, pp. 2–13.

[7] Micron Technology, “NAND flash 101: An introduction to NAND flash and how to design it in to your next product,” Micron Technical Note TN-29-19, Nov. 2006.

[8] C. A. Waldspurger, “Memory resource management in VMware ESX server,” in *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI’02)*, 2002, pp. 181–194.

[9] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, “Scalable high performance main memory system using phase-change memory technology,” in *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA’09)*, 2009, pp. 24–33.

[10] S. Cho and H. Lee, “Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance,” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’09)*, 2009, pp. 347–357.

[11] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, “Enhancing lifetime and security of PCM-based main memory with Start-Gap wear leveling,” in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO’09)*, 2009, pp. 14–23.