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

Brian Wongchaowart Marian K. Iskander Sangyeun Cho

March 17, 2010

Motivation

Suppose that you had a block storage device with the following characteristics:

- There are many free blocks that can be overwritten.
- The cost of writing a new block depends on the choice of which free block is overwritten.

How can you minimize the cost of writing a sequence of new blocks to such a device?

Offline Problem

- The offline version of the problem can easily be reduced to maximum weight bipartite matching.
- We need to match new blocks to be written with free blocks to overwrite; the edge between a new block x and a free block a has cost $W - d(x, a)$, where $d(x, a)$ is the cost of overwriting a with x (Hamming distance or Flip-N-Write cost) and W is a constant chosen so that $W - d(x, a)$ is always positive.
- Efficient polynomial time algorithms for maximum weight bipartite matching are known: for a table of results, see Piotr Sankowski, "Maximum weight bipartite matching in matrix multiplication time," *Theoretical Computer Science* 410, no. 44 (2009): 4480–4488.

Online Problem

- In the online version of the problem, new blocks to be written arrive one at a time.
- The natural greedy algorithm is to write a new block to the free location with the least cost.
- This greedy algorithm is not optimal, however.

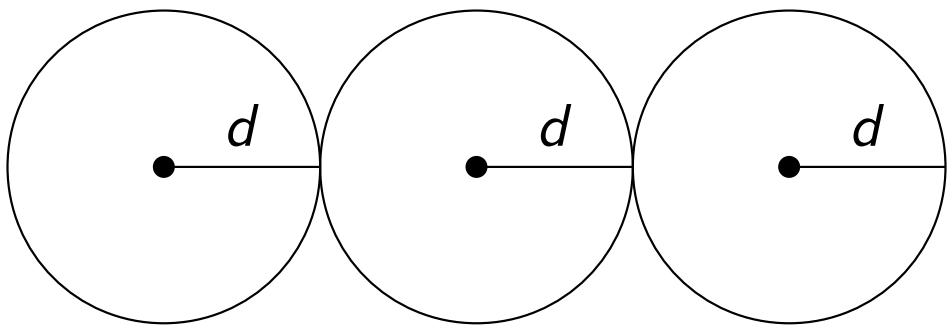
[Everything on competitiveness has been removed from this version of my slides.]

Implementing the Greedy Algorithm

- Given a set of free blocks and a new block to be written, how can one quickly find the free block closest to the new block (the one that costs the least to overwrite)?
- This is the well studied nearest-neighbor search problem: given a set S of points, find the point in S that is closest to a specified point (not necessarily in S).
- Since the points that we are interested in are binary vectors, Manhattan distance (here equivalent to Hamming distance) is the right distance metric; in the case of Flip-N-Write, one can search for the nearest neighbor of each of the two possible representations of a new block.

Elias's Algorithm[1]

- If we want to locate a point close to a given point in constant time, we can try using a hash function.
- Specifically, our goal is to partition the space of possible bit strings (points) into regions such that two points that lie in the same region are close to each other; the hash value of a point is an identifier for the region in which it lies.
- An ideal partitioning is a set of spheres of radius d that completely fill the space, which guarantees that points that lie in the same sphere will be within distance $2d$ of one another.



[1] Ronald L. Rivest, "On the Optimality of Elias's Algorithm for Performing Best-Match Searches," *Information Processing* 74 (1974): 678–681.

- Elias observed that given a perfect error-correcting code with length n and minimum distance $2d + 1$ between codewords, the codewords are precisely the centers of spheres of radius d that completely fill the space of possible binary vectors of length n.
- Finding the unique codeword within Hamming distance d of an arbitrary point is just the decoding problem for an error-correcting code that can correct up to d errors.
- Given a new block to be written, the sphere in which it lies and possibly neighboring spheres can be searched until a free block is found.
- The major problem with this scheme is that it is difficult to decode a binary vector of 512 bytes efficiently.

Block Signature Computation Algorithm in My Paper

Divide a block into equal-size sets of bits and use a vector containing the approximate count of 1-bits in each set as the block signature (hash value).
Problems:

- Blocks with the same signature may not be similar.
- Similar blocks may not have the same signature.
- Nonuniform distribution of signature values even for uniform random data.

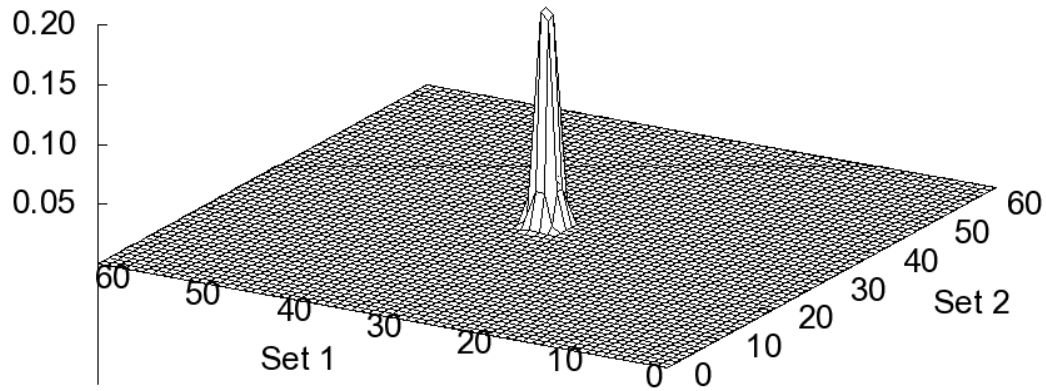


Figure: Distribution of signature values for uniformly distributed random data.

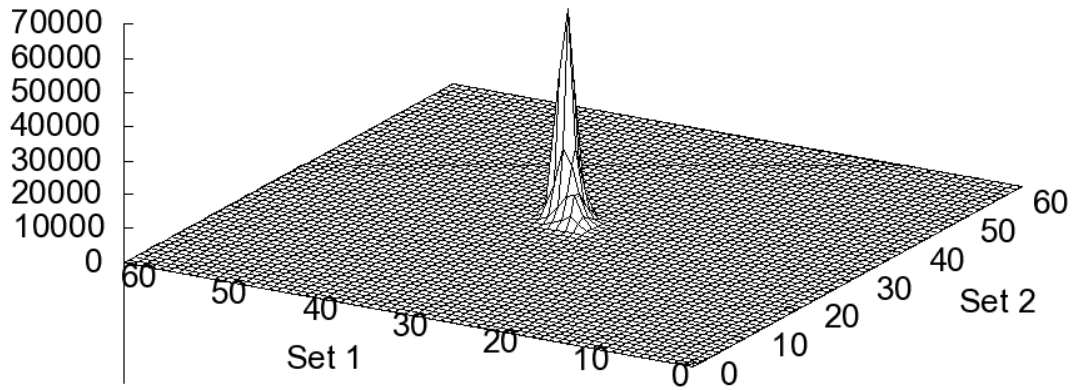


Figure: Distribution of signature values for the write request data blocks of the jpeg trace.

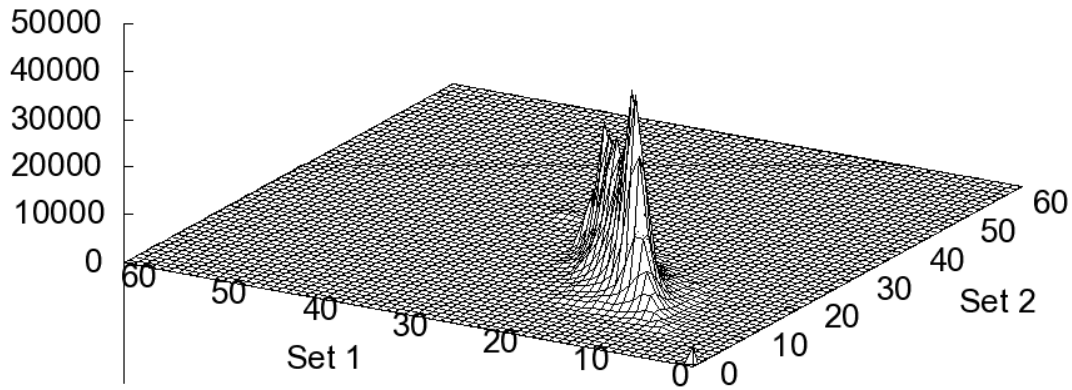


Figure: Distribution of signature values for the write request data blocks of the dng trace.

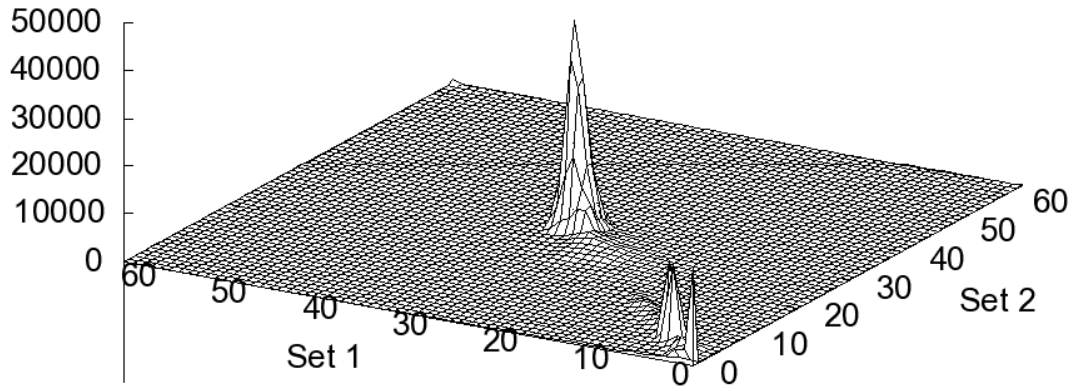


Figure: Distribution of signature values for the write request data blocks of the kernelbuild trace.

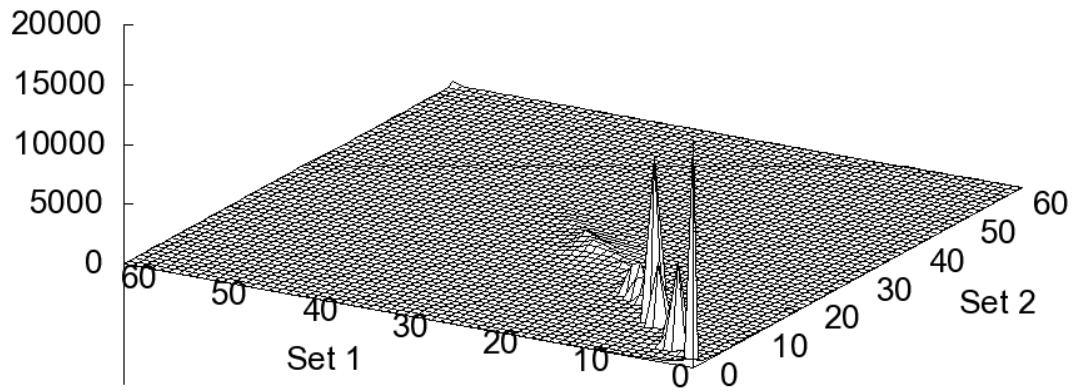


Figure: Distribution of signature values for the write request data blocks of the swsusp trace.

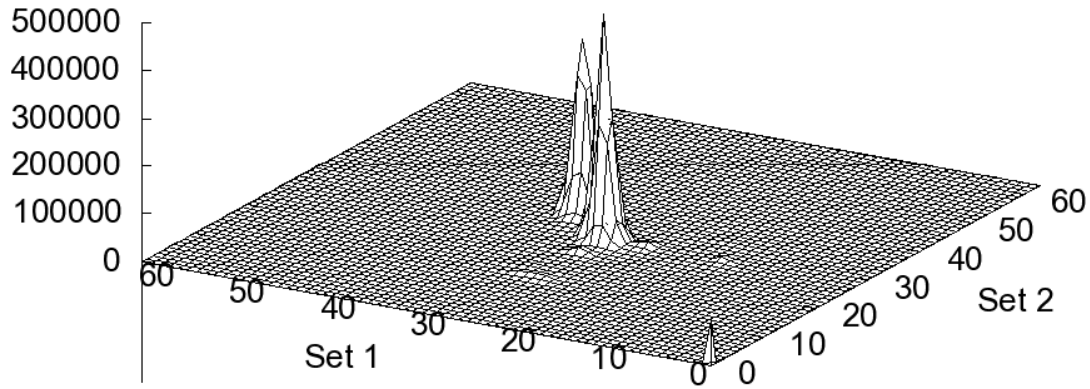


Figure: Distribution of signature values for the write request data blocks of the BT trace.

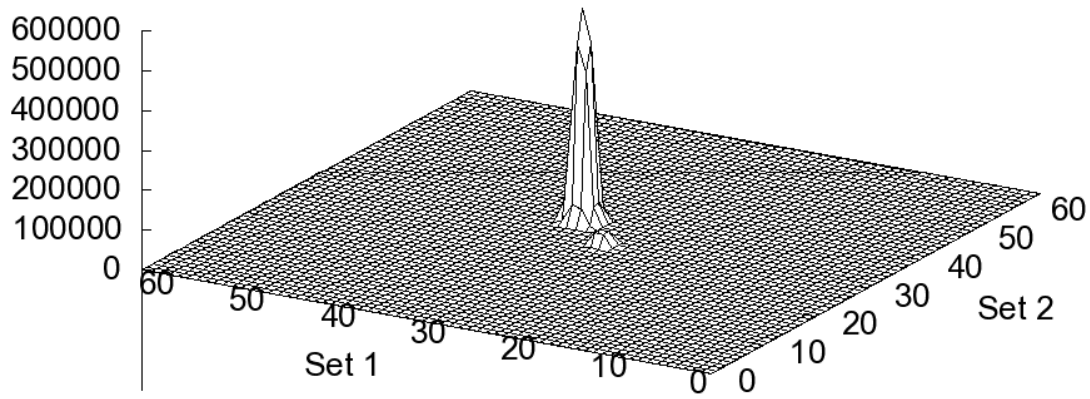


Figure: Distribution of signature values for the write request data blocks of the CG trace.

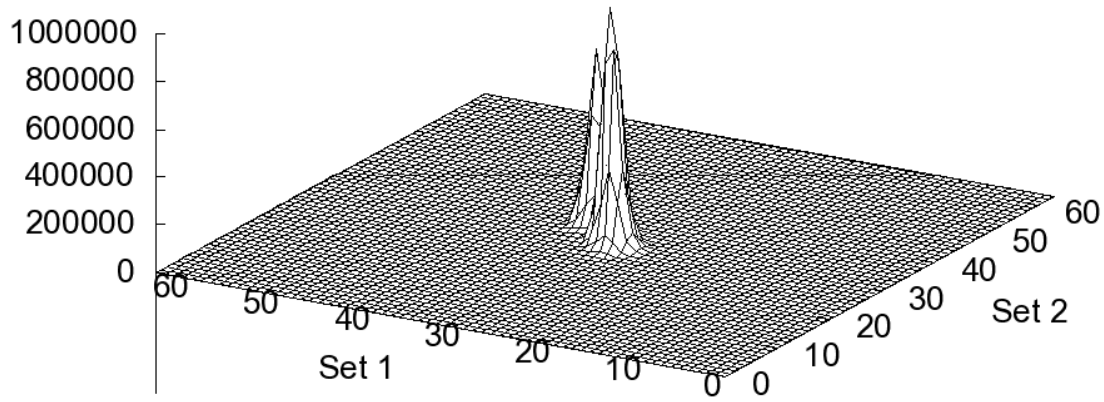


Figure: Distribution of signature values for the write request data blocks of the FT trace.

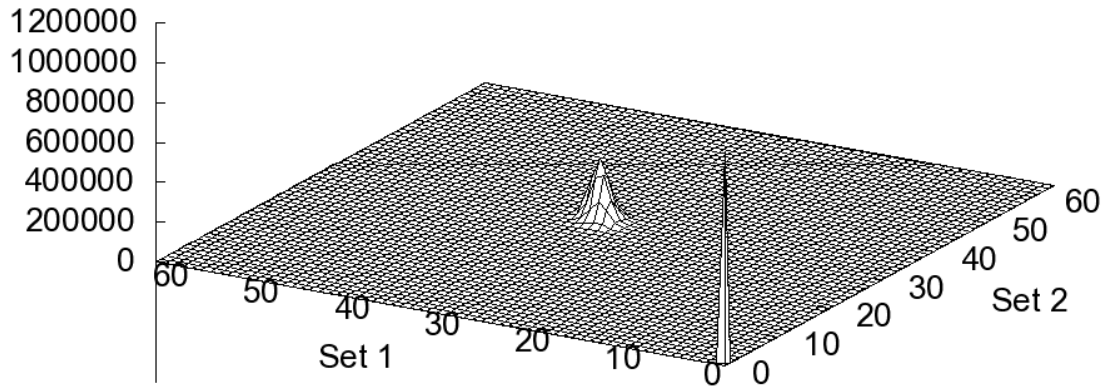


Figure: Distribution of signature values for the write request data blocks of the MG trace.

Sets/block	Bits/set	Total bits	Search distance limit		
			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: Percentage of the random trace requiring a bit write.

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: 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: Percentage of the jpeg trace requiring a bit write.

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: Percentage of the dng trace requiring a bit write.

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: Percentage of the kernelbuild trace requiring a bit write.

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: Percentage of the swsusp trace requiring a bit write.

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

Summary

- The number of bit programming operations needed to store a new data block in a PRAM storage device depends on the current contents of the location at which the block is written.
- We proposed a signature-based block placement algorithm for reducing the number of bit writes required to store a sequence of new data blocks, which saves energy.
- 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 (data-comparison write) alone is used. This figure was achieved without reading and comparing multiple free blocks.

Questions/Comments