Software Enabled Wear-Leveling for Hybrid PCM Main Memory on Embedded Systems

Jingtong Hu[1], Qingfeng Zhuge[3], Chun Jason Xue[2], Wei-Che Tseng[1], and Edwin H.-M. Sha[1,3]
[1]Dept. of Computer Science, University of Texas at Dallas, Richardson, TX, 75080, USA
[2]Dept. of Computer Science, City University of Hong Kong, Tat Chee Ave, Kowloon, Hong Kong.
[3]College of Computer Science, Chongqing University, Chongqing, China.
jthu@utdallas.edu, jasonxue@cityu.edu.hk, qfzhuge@cqu.edu.cn, {wxt043000, edsha}@utdallas.edu

***Abstract*—Phase Change Memory (PCM) is a promising DRAM replacement in embedded systems due to its attractive characteristics. However, relatively low endurance has limited its practical applications. In this paper, in additional to existing hardware level optimizations, we propose software enabled wear-leveling techniques to further extend PCM's lifetime when it is adopted in embedded systems. A polynomial-time algorithm, the Software Wear-Leveling (SWL) algorithm, is proposed in this paper to achieve wear-leveling without hardware overhead. According to the experimental results, the proposed technique can reduce the number of writes on the most-written bits by more than 80% when compared with a greedy algorithm, and by around 60% when compared with the existing Optimal Data Allocation (ODA) algorithm with under 6% memory access overhead.**

I. Introduction

Phase Change Memory (PCM), a type of Non-volatile memories (NVMs), has many appealing characteristics, such as low-cost, shock-resistivity, non-volatility, high density, and power-economy (extreme low leakage power), to be used as main memory in embedded systems. PCM, however, has two drawbacks that hinder its practical adoption as main memory in embedded systems. First, the endurance of PCM is still relatively short compared with DRAM. Second, the write to PCMs is relatively expensive in terms of both energy and time.

Many embedded systems, such as the ARM Cortex-M3 Processor, AVR by Atmel Corp, Freescale's MPC5xx series, etc., adopt Scratch Pad Memory (SPM), software-controlled SRAM, as their on-chip memories due to their small area, low power consumption, time-predictability. SPMs also grant compilers the capability to analyze data access patterns. With profiled data access information, an optimizing compiler can smartly manage data placements and movements. Several works have optimized programs to extend the lifetime of PCM and improve the memory access efficiency. Hu et al. [1], [2] proposed task scheduling, data recomputation, data migration, and dynamic data allocation techniques to reduce the total number of writes to NVMs. However, the lifetime of PCM is not directly proportional to the total number of writes on PCM. The lifetime depends on the address that is written most frequently. Once the number of writes to any part of the PCM exceeds the endurance limit, the whole PCM is considered worn-out. In the Optimal Data Allocation (ODA) algorithm proposed in [1], the addresses in PCM are treated indiscriminately. The first available space will always be allocated to a variable. Therefore, it is possible that the total number of writes to PCM is reduced, but the largest number of writes on an address may not be reduced. Thus, the PCM's lifetime is still limited. In this paper, we keep track of the number of writes to each address in PCM. When a variable is allocated to PCM, an appropriate address is carefully picked so that the writes to PCM can be evenly distributed to all the addresses-while keeping the total memory access cost minimized. Therefore, even with the same total number of writes to PCM, the number of writes to the mostly written address can be greatly reduced, and the lifetime can be extended accordingly. The technique proposed in this paper can be combined with all previous software write reduction techniques to further improve the lifetime of PCM.

In this paper, a polynomial-time algorithm, the Software Wear-Leveling (SWL) algorithm, is proposed to obtain the minimal overhead wear-leveling. The proposed software wear-leveling technique has the following advantages compared with existing wear-leveling techniques: 1) It has low run-time execution and memory overhead. All the optimizations are accomplished in compile-time. 2) It has no hardware overhead since it does not need OS or hardware support. 3) It has finer granularity. The proposed techniques operate at the variable level, while existing software techniques operate at the memory page level. Therefore, the proposed techniques can achieve better wear-leveling. 4) It is not hardware dependent, which means that the software wear-leveling is applicable to all NVMs with with properties similar to PCM.

The major contributions of this paper include:

- We propose a polynomial-time software wear-leveling algorithm to achieve near-optimal results.
- We developed a simulator with hybrid PCM main memory to evaluate the proposed algorithms.

The rest of this paper is organized as follows. Related works are discussed in Section II. Section III presents the hardware

This work is partially supported by NSF CNS-1015802, Texas NHARP 009741-0020-2009, HK GRF 123609, NSFC 61173014, National 863 Program 2013AA013202 and grants from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 123811 and 123210].

978-3-9815370-0-0/DATE13/©2013 EDAA

and software models used in this paper. The SWL algorithm is presented in Section IV. The experiments are presented in Section V. Finally, Section VI concludes this paper.

II. Related Works

Previous works [3], [4], [5], [6], [7] have confirmed that an NVM main memory can achieve significant energy saving with comparable performance to that of a DRAM main memory. However, as mentioned before, there are still two main drawbacks of PCMs that need to be addressed before they can be practically applied as DRAM replacement. Plenty of research has been performed. Zhou et al. [5] and Lee et al. [6] proposed hardware design optimizations. They are all orthogonal to the technique proposed in this paper. Dhiman et al. [3] and Park et al. [8] proposed hybrid PCM and DRAM main memories. In these works, DRAM is used to "absorb" the writes to PCM and wear-leveling techniques for PCM are also proposed. Ferreira et al. [9] also worked on hybrid PCM main memories and proposed writeback minimization with new cache replacement policies, unnecessary write avoidance, and PCM-aware swap algorithm for wear-leveling. Shi et al. [10] proposed a smart victim cache to reduce writes on non-volatile main memory. These works differ from this work in the following three aspects: 1) They assume that hardware-controlled caches are used as the on-chip memory. 2) They need additional complicated hardware enhancements in the memory controller. 3) Most of them need OS support. Since many embedded systems employ software-controlled SPM as their on-chip memories, cannot afford additional complex hardware, and do not have OS running, all these works are not effective for these embedded systems.

Software compiler optimization is an inexpensive and efficient approach for embedded systems that adopt software-controlled SPM. Many software optimization techniques have also been proposed [1], [2], [11], [12], [13], [14], [15]. They only consider reducing the total number of writes on NVM. In this paper, we propose software wear-leveling techniques for hybrid PCM and DRAM main memory. Wear-leveling distributes writes to the PCM more evenly over all the addresses. The techniques proposed in this paper can be combined with all previous software-level write reduction techniques to further improve the lifetime of PCM.

III. Hardware and Software Model

In this section, the hardware model and software model used in this paper will be described.

A. Hardware Model

The targeted system hardware architecture is shown in Figure 1. As shown, the targeted architecture employs pure SRAM-based SPM as its on-chip memory. The data movement and allocation of SPM is purely controlled by software. These kinds of processors normally have explicit instructions to move data among different memory components. The main memory consists of DRAM and PCM. SRAM, DRAM, and PCM are all in the same memory address space. Examples of such architecture include the ARM Cortex-M3 Processor, the AVR by Atmel Corp, and Freescale's MPC5xx series.

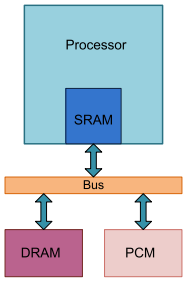

Fig. 1. System Architecture.

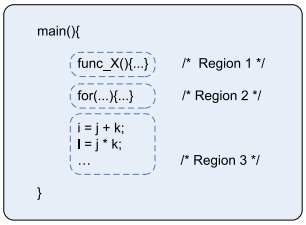

Fig. 2. Example of Program Regions.

B. Software Model and Problem Definition

Under the dynamic SPM data allocation scheme, the program code is divided into regions that are delineated by *program points*: 1) the start and end of each procedure, and 2) the start and end of every loop. Before starting the execution of each region, data allocation code is executed to generate a data allocation which is suitable for this region. Data allocation instructions are inserted into the program either by the programmer or the compiler. During the execution of each region, the data allocation remains the same. Figure 2 shows an example of dividing a program into regions.

Formally, the problem is defined as following: assuming that the number of reads and writes for each variable is known, what is the data allocation for each program region such that the memory access cost is minimized and that the number of writes to each address in the PCM is less than or equal to a given threshold θ?

IV. Software Wear-Leveling Algorithm (SWL)

In this section, we propose the Software Wear-Leveling (SWL) Algorithm, a heuristic that runs in polynomial time.

Hu et al. [1] proposed a dynamic data allocation technique, the Optimal Data Allocation (ODA) algorithm, to reduce the writes on NVM part in hybrid SPM with SRAM and NVM. From the point of view of programs, the hybrid SPM architecture in [1] is the same as the targeted architecture since all the different memory components are in the same memory address space. The difference lies in the physical implementation. Therefore, even though the data allocation algorithm is designed for hybrid SPMs, it is also applicable for the targeted architecture of this paper. As mentioned before, ODA only considers reducing the total number of writes to the PCM. In this paper, rather than reducing the total number of writes, we are focusing on evenly distributing the writes to all addresses of the PCM. Since wear-leveling incurs extra data movement costs, we also want the overhead to be minimized.

The idea of SWL is to use the ODA algorithm to generate the data allocation for each region first. Therefore, the ODA algorithm is used as a pre-processing algorithm. Then we know the variables that are allocated into PCM in each region. We

build up an array to record the number of writes on each address of the PCM. With the help of this array, we find an address assignment in PCM for each variable so that the number of writes on each address is less than or equal to θ and the total moving cost is minimal.

The first step of the SWL algorithm is to build an array W to record the number of writes on each address of PCM. Then for each region r_i, we divide all the data that are allocated to PCM into two groups. The data that were in PCM in previous region are in the first group and the rest are in the second group.

For each data D_j of the first group, we use its address $addr_j$ to look up the number of writes on this address in W. Then we add the number of writes on this address with the number of writes on data D_j in this region. If the summation is less than or equal to θ, this data will stay in address $addr_j$ in this region. The corresponding value in W is updated. Otherwise, find the address in W with the least number of writes. We check if the summation of the least number of writes in W and the number of writes on D_j in this region is less than or equal to θ. If it is true, then D_j is moved to that address. The corresponding writes count in W is updated. If the summation is greater than θ, there is no feasible address to assign to D_j. Therefore, this is θ is too small, the user needs to increase the value of θ.

For each data D_k of the second group, we directly find the smallest number of writes in W and check if the summation of this smallest number and the number of writes on D_k is less than or equal to θ. If it is true, then D_k is moved to that address. The corresponding writes count in W is updated. If the summation is greater than θ, there is no feasible address to assign to D_k. The user needs to increase the value of θ.

In the experiments, the total number of writes to the PCM is divided by the size of PCM to get the lower bound of θ. The lower bound is used as the initial value for θ. Once it is not big enough, the value is increased. Experiments show that θ close to the lower bound can almost always be achieved.

The complexity of the SWL algorithm is $O(n \times m \times p)$, where n is the number of variables, m is the number of regions, and p is the size of PCM.

V. Experiments

In the section, experimental results are presented. First, the experimental setup is presented in Section V-A. Then the experimental results of the SWL algorithm are presented in Section V-B.

A. Experiments Setup

In the experiments, a Simplescalar [16] based custom simulator is implemented. The system specification used to evaluate the proposed algorithm is shown in Table I. CACTI and NVsim [17] are used to used to obtain the parameters for DRAM, SRAM, and PCM. The obtained parameters are integrated into the custom simulator. Mibench [18], a set of commercially representative embedded programs, is used for the experiments as benchmarks. The memory access information is first statically profiled. Then, SWL is used to generate the data allocation and wear-leveling instructions. After that, the instructions are inserted into the program. The new programs run in the custom simulator to obtain the results.

TABLE I
Target System Specification

Component	Description
CPU	Frequency: 1.0 GHz
On-chip SRAM	Size: 16 KB, access latency: 1.41 ns, access energy: 0.004 nJ, leakage power: 29.28 mW
DRAM Main Memory	DDR SDRAM, Size: 16KB, Access latency: 1.78 ns access energy: 0.147 nJ, leakage power: 3.4 mW
PCM Main Memory	Size: 64 KB, read latency: 6.82 ns, write latency (SET/RESET): 152.20/12.20 ns, read energy: 0.064 nJ, write energy(SET/RESET): 0.07/0.876 nJ, leakage power: 0.713 mW

B. Evaluation of the Software Wear-Leveling Algorithms

Experiments with three different algorithms are conducted and compared. The first algorithm is the greedy algorithm, in which data are sorted according to the access time and allocated to first available address. The second algorithm is the ODA algorithm proposed by Hu et al. [1]. The third algorithm is the SWL algorithm proposed in this paper.

TABLE II
Number of writes on most-written bit.

Benchmarks	Greedy	ODA	SWL		
	Writes	Writes	Writes	(G-S)/G	(O-S)/O
basicmath	3349	1374	538	83.94%	60.84%
bitcount	15	3	2	86.67%	33.33%
CRC32	4924	434	59	98.80%	86.41%
dijkstra	3800	1254	555	85.39%	55.74%
FFT	553	446	339	38.70%	23.99%
patricia*	1300	662	382	70.62%	42.30%
qsort*	4421	1557	405	90.84%	73.99%
rijndael*	962	577	210	78.17%	63.60%
sha	2559	327	98	96.17%	70.03%
stringsearch	177	92	53	70.06%	42.39%
susan*	598	170	28	95.32%	83.53%
Average				81.33%	57.83%

The lifetime of PCM is directly related to the most written bit. If the writes to that bit exceeds the endurance limit, the PCM is considered worn-out. An error occurs for all the following writes to that bit. In the experiments, we also recorded the number of writes to the most written bit under different techniques. Since most existing PCMs already incorporated bit-level write avoidance technique [6], [5], we also implemented it in our PCM simulator. Table II shows the results under different algorithms. From the table, we can see that even with hardware optimization presented, the software wear-leveling algorithms can still reduce the writes by more than 80% on average compared with the greedy algorithm, and by almost 60% on average compared with the ODA algorithm.

As mentioned above, the lifetime of PCM is directly related to the bit that is most written. Therefore, as the number of writes on the most-written bit is reduced, the lifetime of PCM is extended. In this paper, we are assuming the

TABLE III
PCM LIFETIME IN DAYS.

Benchmarks	Greedy	ODA	SWL		
	Lifetime	Lifetime	Lifetime	S/G	S/O
basicmath	262.80	640.56	1635.93	6.22	2.55
bitcount	3229.03	16145.15	24217.72	7.50	1.50
CRC32	24.58	278.93	2051.76	83.46	7.36
dijkstra	5.76	17.47	39.47	6.85	2.26
FFT	152.40	188.96	248.61	1.63	1.32
patricia*	101.91	200.12	346.80	3.40	1.73
qsort*	16.80	47.70	183.36	10.92	3.84
rijndael*	726.03	1210.46	3325.89	4.58	2.75
sha	9.33	73.05	243.74	26.11	3.34
stringsearch	18.75	36.07	62.61	3.34	1.74
susan*	96.34	338.87	2057.45	21.36	6.07
Average				15.94	3.13

endurance of PCM is 10^9 rewrites [5]. Let $Endurance_{pcm}$ be the endurance of PCM, $Number_{most}$ be the number of writes on the most-written bit of the PCM, $Time_{exe}$ be the programs' execution time, and $Time_{memory}$ be the memory access time. The lifetime of the PCM for each benchmark is computed according to Eq. (1).

$$Lifetime = \frac{Endurance_{pcm}}{Number_{most}} \times (Time_{exe} + Time_{memory}) \quad (1)$$

With the greedy algorithm, the lifetime of PCM is around 422 days on average. With the ODA algorithm, the lifetime of PCM is extended to around 1743 days on average. With the proposed software wear-leveling algorithm, the lifetime of PCM can be extended to around 3130 days on average, which is about 8 years. On average, the lifetimes obtained with SWL are more than 16 times as long as those obtained with the greedy algorithm and more than three times as long as those obtained with ODA.

TABLE IV
TIME OVERHEAD.

Benchmarks	Greedy	ODA	SWL		
	Time(μs)	Time(μs)	Time(μs)	(G-S)/G	(O-S)/O
basicmath	16306.83	11713.96	12694.42	22.15%	-8.37%
bitcount	2482.19	2105.71	2131.40	14.13%	-1.22%
CRC32	13906.36	7365.64	7800.21	43.91%	-5.9%
dijkstra	9250.19	6635.87	6820.35	26.27%	-2.78%
FFT	12158.66	5150.79	5586.55	54.05%	-8.46%
patricia*	20617.70	13237.28	14484.24	29.75%	-9.42%
qsort*	9620.64	8203.77	8962.62	6.84%	-9.25%
rijndael*	34487.54	18959.06	19935.45	42.20%	-5.15%
sha	14375.63	7171.12	7336.05	48.97%	-2.3%
stringsearch	4672.28	2745.30	2786.48	40.36%	-1.5%
susan*	13738.92	4852.91	5128.07	62.67%	-5.67%
Average	13783.36	8012.86	8515.08	35.57%	-5.46%

Extra reads and writes are caused by the wear-leveling. Table IV shows the memory access cost overhead of the SWL algorithm. Since the SWL algorithm adds the additional wear-leveling instructions into the ODA algorithm, the benchmarks incur longer memory access time under SWL than the memory access time obtained under ODA. On average, the SWL algorithm can still reduce the memory access cost by 35.57% compared with the greedy algorithm. However, the memory access cost increased by 5.46% on average when compared with the ODA algorithm. Considering the lifetime improvement, such a small memory access cost overhead is acceptable.

VI. CONCLUSIONS

This paper proposed a software enabled wear-leveling technique to extend PCM's lifetime when it is adopted in embedded systems. According to the experimental results, the proposed technique can reduce the number of writes on most-written addresses by more than 80% when compared with a greedy algorithm, and by around 60% when compared with the existing ODA algorithm with under 6% memory access overhead.

REFERENCES

[1] J. Hu, C. Xue, Q. Zhuge, W.-C. Tseng, and E.-M. Sha, "Towards energy efficient hybrid on-chip scratch pad memory with non-volatile memory," in *DATE '11*, march 2011, pp. 1–6.

[2] J. Hu, C. J. Xue, W.-C. Tseng, Y. He, M. Qiu, and E. H.-M. Sha, "Reducing write activities on non-volatile memories in embedded cmps via data migration and recomputation," in *DAC '10*, 2010, pp. 350–355.

[3] G. Dhiman, R. Ayoub, and T. Rosing, "Pdram: a hybrid pram and dram main memory system," in *DAC '09*, 2009, pp. 664–469.

[4] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *ISCA '09*, 2009, pp. 24–33.

[5] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ISCA '09*, 2009, pp. 14–23.

[6] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ISCA '09*, 2009, pp. 2–13.

[7] C. Xue, Y. Zhang, Y. Chen, G. Sun, J. Yang, and H. Li, "Emerging non-volatile memories: Opportunities and challenges," in *CODES+ISSS 2011*, pp. 325 –334.

[8] H. Park, S. Yoo, and S. Lee, "Power management of hybrid dram/pram-based main memory," in *DAC '11*, june 2011, pp. 59–64.

[9] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mosse, "Increasing pcm main memory lifetime," in *DATE '10*, 2010, pp. 914–919.

[10] L. Shi, C. J. Xue, J. Hu, W.-C. Tseng, and E. H.-M. Sha, "Write activity reduction on flash main memory via smart victim cache," in *GLVLSI '10*, 2010, pp. 91–94.

[11] J. Hu, C. J. Xue, W.-C. Tseng, Q. Zhuge, and E. H.-M. Sha, "Minimizing write activities to non-volatile memory via scheduling and recomputation," in *SASP '10*, 2010, pp. 7–12.

[12] J. Hu, W.-C. Tseng, C. Xue, Q. Zhuge, Y. Zhao, and E.-M. Sha, "Write activity minimization for nonvolatile main memory via scheduling and recomputation," *IEEE TCAD*, vol. 30, no. 4, pp. 584–592, april 2011.

[13] W.-C. Tseng, C. J. Xue, Q. Zhuge, J. Hu, and E. H.-M. Sha, "Optimal scheduling to minimize non-volatile memory access time with hardware cache," in *VLSI-SOC '10*, 2010, pp. 131–136.

[14] T. Liu, Y. Zhao, C. Xue, and M. Li, "Power-aware variable partitioning for dsps with hybrid pram and dram main memory," in *DAC '11*, june 2011, pp. 405–410.

[15] M. Qiu, M. Guo, M. Liu, C. J. Xue, L. T. Yang, and E. H. M. Sha, "Loop scheduling and bank type assignment for heterogeneous multi-bank memory," *J. Parallel Distrib. Comput.*, vol. 69, no. 6, pp. 546–558, Jun. 2009.

[16] T. Austin, E. Larson, and D. Ernst, "Simplescalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.

[17] X. Dong, N. P. Jouppi, and Y. Xie, "Pcramsim: System-level performance, energy, and area modeling for phase-change ram," in *ICCAD '09*, 2009, pp. 269–275.

[18] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *WWC-4*, dec. 2001, pp. 3–14.