IEICE Electronics Express, Vol.19, No.1, 1–6

LETTER

Optimized fast data migration for hybrid DRAM/STT-MRAM main memory

Chenji Liu[1, 2, 3], Lan Chen[1, 2, 3, a)], Xiaoran Hao[1, 3], and Mao Ni[1, 3]

Abstract In order to reduce the main memory energy of the IoT terminal, STT-MRAM is used to replace DRAM to save refresh energy. However, the write performance of STT-MRAM cells is worse than that of DRAM. Our previous work proposed a hybrid DRAM/STT-MRAM main memory and fast data migration to reduce the adverse effects of poor write performance of STT-MRAM cells with negligible performance overhead. This article optimizes the migration algorithm and experiment scheme: 1. Reduce the storage overhead of the algorithm. 2. Realize the continuous work of the algorithm. 3. Consider the impact of system standby time on main memory energy. The results show that compared with our previous work, the storage overhead of the algorithm is reduced 99.8%. When the system standby time is zero, the energy of the hybrid main memory (including the energy of the algorithm) is reduced by 4% on average compared to DRAM. The longer the system standby time, the more energy saving.

Keywords: STT-MRAM, hybrid main memory, migration, energy saving
Classification: Integrated circuits

1. Introduction

With the development of Internet of Things (IoTs) technology, the number of IoT terminal devices has grown exponentially. Battery-powered terminals require a low-energy main memory. As one of the new non-volatile memories (NVMs), STT-MRAM has many advantages [1]. Compared with other new NVMs, it has the fastest access speed and the strongest endurance. Compared with DRAM, it has no leakage current and does not need refresh. Now, the key task for STT-MRAM to replace DRAM is to expand it to higher densities. In addition, the write performance of STT-MRAM cells needs to be improved [2]. In theory, STT-MRAM can be continuously scaled to below 10-nm. However, this ideal behavior will encounter many challenges in mass production, requiring continuous innovation in manufacturing technology [3].

For standalone STT-MRAM, commercial products have been continuously launched [4, 5, 6], and the current maximum capacity is 1 Gb [7], which can meet the memory capacity requirements of some IoT terminals. A lightweight neural network as a complex application, its weight parameter is about 10 MB [8]. If the adverse effects of poor write performance of STT-MRAM can be controlled, STT-MRAM will be a good choice for terminal low-energy main memory. Our previous work proposed a hybrid DRAM/STT-MRAM main memory and fast data migration to reduce the adverse effects of poor STT-MRAM write performance, with negligible performance overhead [9]. This article further optimizes the migration algorithm and experiment scheme. Optimize the structure of the miss table and the selection of DRAM migration blocks to reduce the storage overhead of the algorithm. Realize the replacement mechanism of migraton table to ensure the continuous work of the algorithm. When calculating the main memory energy, consider different system standby times to get closer to the real scene.

The rest of this article is organized as follows. Section 2 introduces related work. Section 3 introduces the optimized migration algorithm. Section 4 describes the experimental setup. Section 5 discusses the experimental results and Section 6 summarizes the article.

2. Related work

Research on new NVM/DRAM hybrid main memory began to appear in 2009, mainly PCM and DRAM. There are two types of organization: hierarchical and parallel. For hierarchical organization, Qureshi et al. [10] proposed a main memory consisting of PCM coupled with a small DRAM buffer. Lee et al. [11] proposed a novel write-only DRAM cache for PCM. Park et al. [12] addressed the power management of DRAM cache and PCM. Yoon et al. [13] proposed a new caching policy for DRAM/PCM hybrid memory. For parallel organization, Dhiman et al. [14] proposed PDRAM, a novel energy efficient main memory architecture and architecture and system policies. Zhang et al. [15] presented a hybrid PRAM/DRAM memory architecture and exploit an OS-level paging scheme. Ramos et al. [16] proposed a new hybrid design that features a hardware-driven page placement policy. The above managements of parallel hybrid main memory requires the participation of the OS, which involves the division of software and hardware. It is not suitable for embedded systems, especially when the embedded system does not use virtual memory.

STT-MRAM has been widely studied as an on-chip cache [17, 18, 19, 20], and few studies have used it as a main memory. Meza et al. [21] showed that reducing the size of the row buffer can greatly reduce the dynamic energy of the NVM

[1] Institute of Microelectronics of Chinese Academy of Sciences, Beijing 100029, China
[2] University of Chinese Academy of Sciences, Beijing 100049, China
[3] Beijing Key Laboratory of Three-dimensional and Nanometer Integrated Circuit Design Automation Technology, Beijing 100049, China
[a)] chenlan@ime.ac.cn

DOI: 10.1587/elex.18.20210493
Received November 21, 2021
Accepted November 25, 2021
Publicized December 6, 2021
Copyedited January 10, 2022

main memory. Kultursay et al. [22] showed that the energy and performance of STT-RAM without any optimization cannot compete with DRAM. Partial write and row buffer write bypass can significantly improve the performance and energy of STT-RAM main memory. Wang et al. [23] solved performance issues caused by small MRAM page size. The above circuit-level optimization within the MRAM can be combined with the architecture-level optimization proposed in this article. Asifuzzaman et al. [24, 25] investigated the feasibility of using STT-MRAM in high performance computing systems and real-time embedded systems. However, the timing parameters of MRAM come from estimates, and publicly reliable timing parameters are unavailable [26]. We proposed a hybrid DRAM/STT-MRAM main memory and fast data migration [9].

3. Optimized migration algorithm

Standalone STT-MRAM adopts DDRx interface design, which can directly replace DRAM. The structure of the STT-MRAM chip is similar to that of DRAM, and each bank in the chip has a row buffer. When the row buffer hits, read or write the row buffer. When the row buffer misses, the opened page is first precharged back to the array, and then the page to be accessed is activated into the row buffer. Only precharge and activation are related to the array. Therefore, reducing the adverse effects of poor STT-MRAM cell performance requires reducing the number of precharges and activations. In other words, the number of MRAM row buffer misses needs to be reduced. We propose fast data migration, which migrates frequently missed MRAM data to DRAM. The structure of the hybrid memory is shown in Fig. 1.

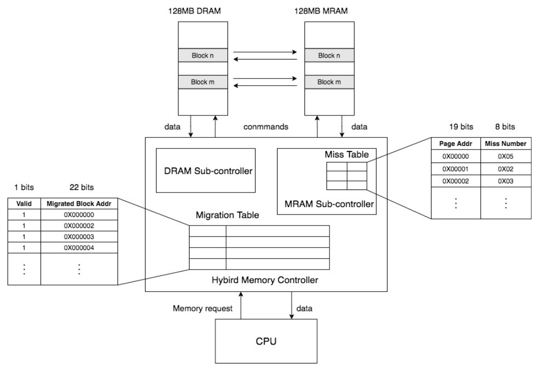

Fig. 1 The structure of the hybrid memory.

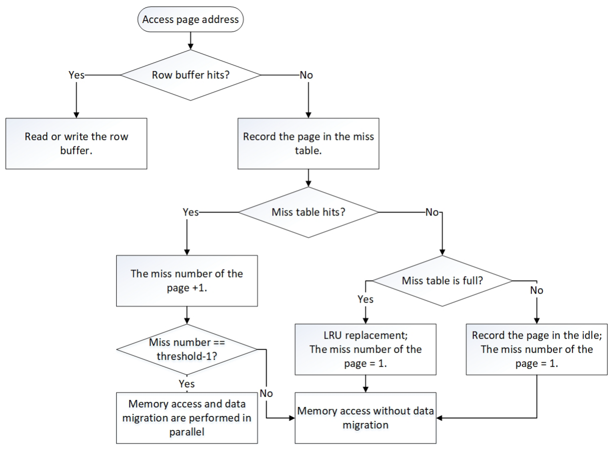
Access page address
Row buffer hits?
Yes
No
Read or write the row buffer.
Record the page in the miss table.
Miss table hits?
Yes
No
The miss number of the page +1.
Miss table is full?
Yes
No
Miss number == threshold-1?
Yes
No
LRU replacement; The miss number of the page = 1.
Record the page in the idle; The miss number of the page = 1.
Memory access and data migration are performed in parallel
Memory access without data migration

Fig. 2 The workflow of the miss table.

3.1 Miss table

As shown in Fig. 1, a miss table needs to be implemented in the MRAM sub-controller to record the number of MRAM row buffer misses caused by different pages. If the miss table records misses caused by all pages, the size of the miss table is 512 KB (the number of pages is 512K, and the width is set to 1 byte). Considering the cost and area overhead, it is not suitable for implementation in an on-chip memory controller. In this article, the miss table only records the most recent MRAM row buffer misses. The depth of the miss table is set to 64 and the width is increased by 19-bit to record the address of the missed page. When the miss table is full, LRU is used to replace the least recently accessed entry in the table. This structure is reasonable, because the program usually only accesses part of the memory pages, and pages that missed a long time ago do not need to be retained. The workflow of the miss table is shown in Fig. 2.

3.2 Migration table

In order to achieve correct access after migration, a migration table needs to be implemented in the hybrid memory controller to record the address of the migration block, as shown in Fig. 1. In our previous work, once the migration table overflows, the algorithm will be disabled. If the program continues to access MRAM and causes a large number of row buffer misses, the performance of hybrid main memory will be greatly reduced. This article implements the replacement mechanism of the migration table. After the MRAM block is migrated to DRAM, a counter is used to record the number of times it has been accessed. When the migration table is full, considering the time locality of the program, the least recently accessed MRAM block will be migrated back to MRAM. The workflow of the migration table is shown in Fig. 3. The time cost of migrating back is tested in the following experiment.

3.3 DRAM migration block

In addition to the miss table, the migration table also introduces storage overhead. In our previous work, the selection of the DRAM migration block is moved up from the end block. One migration requires two migration table entries to record the actual addresses of the MRAM migration block and the DRAM migration block respectively. In this article, the capacity ratio of DRAM and MRAM in hybrid main memory is 1:1, and the two migrated blocks need to have the same offset address in DRAM and MRAM, as shown in Fig. 1. Therefore, only one entry is required for a migration, and the entry only needs to record the same offset address. When accessing the block located at the offset address in DRAM or MRAM, just go to the opposite memory to access it. In this way, the migration table of the same capacity can record more migrations while the program is running, which can better reduce the number of MRAM row buffer misses.

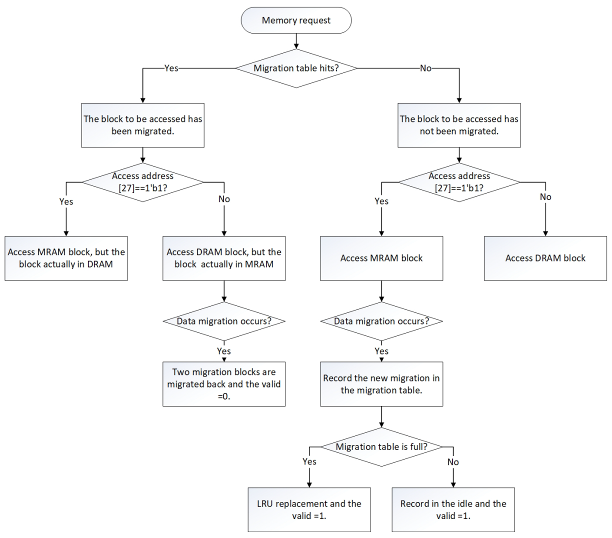
Memory request
Migration table hits?
Yes
No
The block to be accessed has been migrated.
The block to be accessed has not been migrated.
Access address [27]==1'b1?
Yes
No
Access MRAM block, but the block actually in DRAM
Access DRAM block, but the block actually in MRAM
Access MRAM block
Access DRAM block
Data migration occurs?
Yes
Two migration blocks are migrated back and the valid =0.
Record the new migration in the migration table.
Migration table is full?
LRU replacement and the valid =1.
Record in the idle and the valid =1.

Fig. 3 The workflow of the migration table.

4. Experimental setup

We built a hybrid DRAM/STT-MRAM main memory using Micron 1 Gb x8 DDR3 SDRAM [27] and Everspin 256 Mb x8 DDR3 STT-MRAM [28] verilog models. We modified the capacity of the DRAM model to 256 Mb. The model configuration and parameters are shown in Table I. The technology nodes of the DRAM and STT-MRAM models are 2x-nm and 40-nm respectively, making the results more friendly to DRAM. We have implemented three main memory structures: 256 MB pure DRAM, 256 MB hybrid memory composed of 128 MB DRAM and 128 MB STT-MRAM, and 256 MB pure STT-MRAM. Each main memory structure contains two ranks. In the hybrid memory, rank0 is composed of four 256 Mb DRAMs in parallel, and rank1 is composed of four 256 Mb STT-MRAMs in parallel. The system configuration is shown in Table II.

The experiment is divided into three parts:

1. Test the effect of the optimization:
   (a) Miss table size reduction. Compare the number of MRAM row buffer misses under different miss table depths (depth=512K, 64, 32, 16, 8).
   (b) Migration table replacement mechanism. Compare the number of MRAM row buffer misses under the replacement mechanism and the disable mechanism (migration table depth=512, 256, 128, 64).
2. Test the memory performance and energy:
   (a) The memory access time and energy when the program is running, and the total execution time of the program.
   (b) The memory energy when the system standby time is 1 ms, 10 ms, and 100 ms. The processor waits for the specified time before starting to fetch instructions.
3. Test the overhead of the algorithm, including storage overhead, performance overhead and energy overhead.

Table I The model configuration and parameters (256 Mb, 667 MHz).

Configuration	DRAM	MRAM
Bank	4	8
Row	8K	64K
Column	1K	64
Page size	1 KB	64B
Parameters		
ACTIVE to READ or WRITE delay (tRCD)	15 ns	95 ns
PRECHARGE time (tRP)	15 ns	66 ns
READ latency (RL)	10 cycle	10 cycle
WRITE latency (WL)	7 cycle	7 cycle
Write recovery time (tWR)	15 ns	15 ns
Average time between REFRESH (tREFI)	7.8 us	N/A
REFRESH time (tRFC)	110 ns	N/A
One bank active-precharge current (IDD0)	65 mA	220 mA
Burst read operating current (IDD4R)	125 mA	135 mA
Burst write operating current (IDD4W)	125 mA	165 mA
Burst refresh current (IDD5B)	165 mA	N/A

Table II Hardware simulation platform configuration.

Processor	667 MHz, riscv, sequential execution
I/D caches	Private, 32 KB/4KB, 2-way, 32B block, LRU write-back
Memory controller	DRAM controller, Hybrid memory controller, MRAM controller
Main memory	DRAM memory, Hybrid memory, MRAM memory

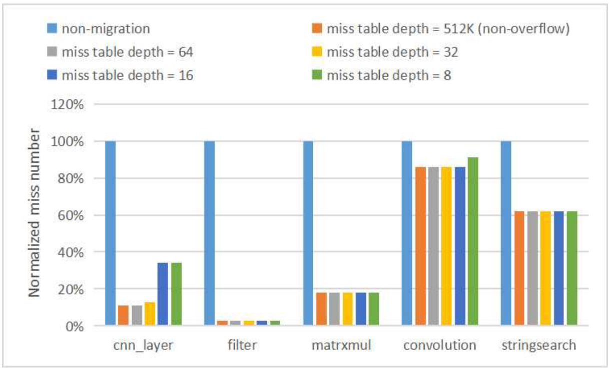

Fig. 4 The impact of miss table size on MRAM row buffer misses.

5. Experimental results

5.1 The effect of the optimization

Fig. 4 shows the impact of miss table size on MRAM row buffer misses. It can be seen that for cnn_layer and convolution, as the depth of the miss table decreases, the miss table overflows and the number of misses increases. However, compared with non-migration, the number of misses is still effectively reduced. For other programs, the miss table does not overflow, so the number of misses remains unchanged. Therefore, the optimized miss table structure can greatly reduce storage overhead without affecting the migration effect. In addition, it can be seen that the migration algorithm has different effects for different programs. For convolution, the effect is poor. The greater the number of misses and the more concentrated the miss distribution, the better the migration effect.

Fig. 5 shows the impact of the migration table replace-

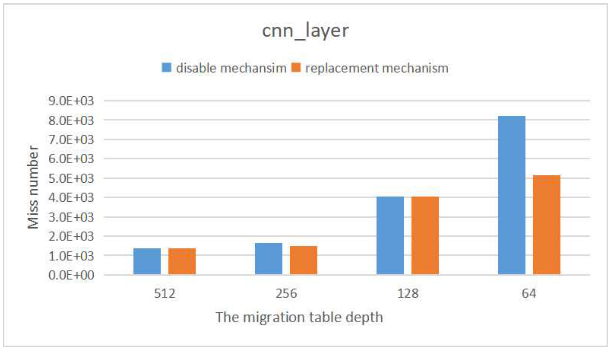

Fig. 5 The impact of the migration table replacement mechanism on MRAM row buffer misses.

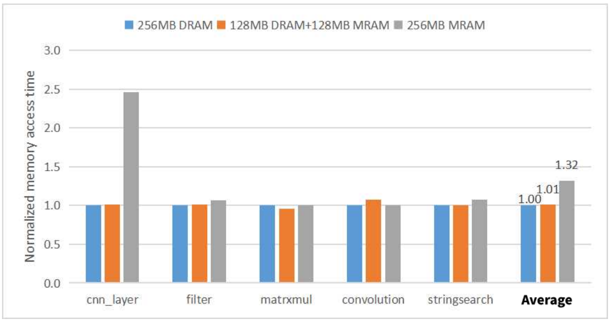

Fig. 6 Memory access time normalized to DRAM.

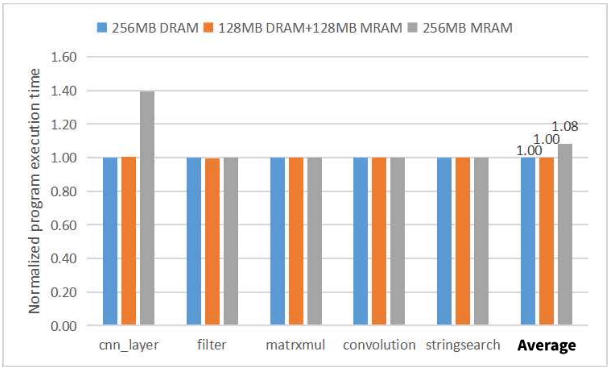

Fig. 7 Program execution time normalized to DRAM.

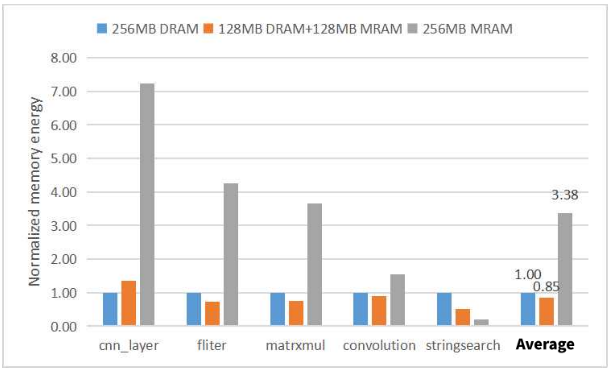

Fig. 8 Memory energy normalized to DRAM.

ment mechanism on MRAM row buffer misses. When the depth of the migration table is 512, the migration table does not overflow, and the number of misses under the two mechanisms is the same. As the depth of the migration table decreases, the table overflows. When the depth is 256 and 64, the number of misses under the replacement mechanism is reduced by 8% and 38% compared with that under the disabled mechanism. When the depth is 128, the number of misses is close. This is related to the program's access to MRAM after the migration table overflows. The more misses, the more effective the replacement mechanism. Migrating back will bring additional reads and writes to DRAM and MRAM. But in general, the reduction in the number of MRAM row buffer misses caused by the replacement mechanism can completely offset this overhead.

5.2 The memory performance and energy

Fig. 6 shows the normalized memory access time. It can be seen that, compared with DRAM, the access time of hybrid memory increases by 1% on average, while the access time of MRAM increases by 32% on average. This is because MRAM has higher activation and precharge delays and more activation and precharge operations (the capacity of the MRAM row buffer is one-sixteenth of the DRAM row buffer, which is easier to miss). For matrxmul and convolution, the access time of MRAM does not increase compared with DRAM. This is because the program generates fewer activations and precharges, and DRAM refresh will increase the access time of the DRAM. However, the access time of hybrid memory is always comparable to that of DRAM, because the migration algorithm can effectively control the precharge and activation times of MRAM, and the time overhead of migration is negligible. Fig. 7 shows the normalized program execution time. It can be seen that the program execution time is less sensitive to the increased delay of MRAM. Compared with DRAM, the program execution time of hybrid memory does not increase, and the average increase of MRAM is 8%.

Fig. 8 shows the normalized memory energy. It can be seen that compared with DRAM, the energy of hybrid memory is reduced by 15% on average, while the energy of MRAM becomes 3.38 times on average. This is because the activation-precharge energy of the MRAM is too high, which completely exceeds the saved refresh energy. In hybrid memory, the migration algorithm can well control the activation-precharge energy, and refresh energy is reduced by half. However, for stringsearch, the energy of MRAM is the lowest. For cnn_layer, the energy of hybrid memory cannot be reduced. This is related to the ratio of refresh energy to total memory energy. The higher the ratio, the more energy MRAM saves. Fig. 9 shows the ratio of memory sub-energy to total memory energy in DRAM. Memory energy can be divided into read energy, write energy, activation-precharge energy and refresh energy [29]. It can be seen that for stringsearch, Eref accounts for 96%, and Eact-pre accounts for 2%. The refresh energy saved by the MRAM completely covers the increased activation-precharge energy. For cnn_layer, Eref accounts for 30%, and worse, Eact-pre accounts for 46%. The refresh energy saved by the hybrid memory cannot keep up with the increased activation-precharge energy. In addition, for convolution, its Eref accounts for 86%, and Eact-pre accounts for 5%. The hybrid memory energy dropped by only 11% (as shown

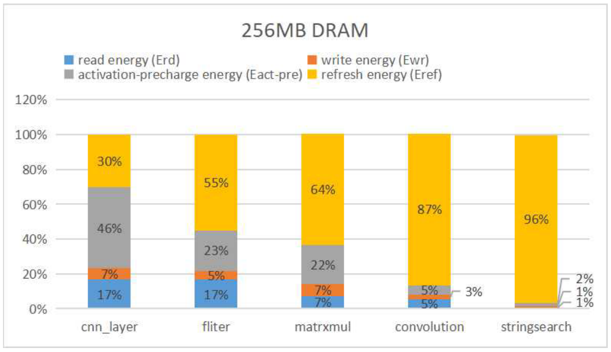

Fig. 9 The ratio of memory sub energy to total memory energy.

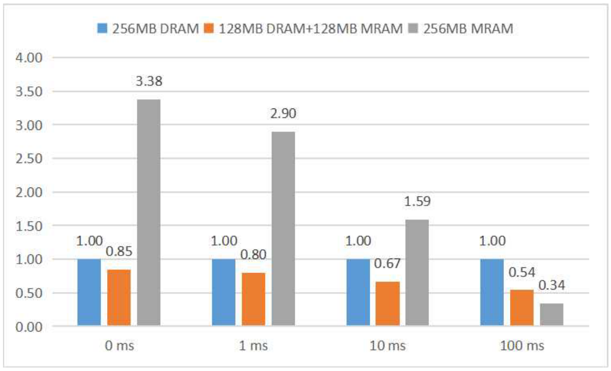

Fig. 10 Average memory energy under different system standby times.

in Fig. 8). This is because the migration algorithm does not work well for convolution (as shown in Fig. 4).

Fig. 10 shows the average memory energy under different system standby times. It can be seen that as the system standby time increases, hybrid memory is more and more energy-efficient than DRAM. When the system standby time is 100 ms, the energy of MRAM is the lowest. This is because when the system is in standby, only DRAM consumes refresh energy. Compared with DRAM, refresh energy of hybrid memory is halved, while MRAM has no refresh energy at all. The longer the system standby time, the greater the advantage of MRAM. It should be noted that in the 1:1 hybrid memory, the energy reduction ratio of the hybrid memory can only be infinitely close to 50%.

5.3 The overhead of the algorithm

The storage overhead of the algorithm mainly comes from the migration table and the miss table. These two tables are constructed as fully associative LRU replacement caches. The block size is 1 byte. The size of the migration table is about 1 KB, and the size of the miss table is about 0.2 KB. Compared with our previous work (the size of the migration table is about 2 KB and the size of the miss table is 512 KB), the storage overhead of the algorithm is reduced by 99.8%. Table III is the estimated performance and energy of the migration table and miss table using CACTI 7.0 [30].

The performance overhead of the algorithm is very small, which is the contribution of the fast data migration proposed in our previous article [9]. Normally, only one clock (tCK=1.5 ns) is added to the critical path of memory access to query the migration table to determine whether the block to be accessed is migrated. When the MRAM is accessed and the row buffer misses, a delay of two clocks will be added to read and write the miss table. When the migration occurs, the migration process and the registration of the migration table after the migration is completed, are executed in parallel with the memory access. The delay of the algorithm is already included in the memory access time (Fig. 6).

The energy overhead of the algorithm mainly comes from the leakage current energy of the migration table and the miss table. The miss table is read and written only when the MRAM row buffer misses, and the migration table is written only when a migration occurs. Although the migration table is read every time the memory is accessed, it is very small compared with the leakage current energy that has always existed. The additional DRAM and MRAM read and write energy caused by the migration has been included in the memory energy (Fig. 8). Fig. 11 shows the impact of algorithm energy overhead under different system standby times. After adding algorithm energy, the energy of hybrid memory is reduced by 4% on average than DRAM (system standby time = 0 ms). In the future, we will try to implement a table based on embedded STT-MRAM instead of SRAM to solve the problem of large leakage current.

Table III The estimated results from CACTI 7.0 (22-nm).

Parameters	migration table	miss table
Access time	0.46 ns	0.14 ns
Read energy per access	0.26 pJ	0.14 pJ
Write energy per access	1.19 pJ	0.23 pJ
Leakage power	2.42 mW	0.32 mW

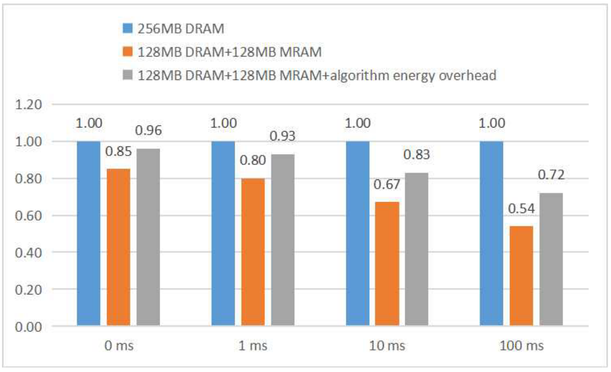

Fig. 11 The impact of algorithm energy overhead under different system standby times.

6. Conclusion

This article first reduces the storage overhead of the migration algorithm without affecting the migration effect. Then the continuous work of the algorithm is realized, which can better control the number of MRAM row buffer misses. After considering the overhead of the algorithm, hybrid memory can still reduce memory energy without affecting system performance. And as the system standby time increases, the energy saved by the hybrid memory is increasing. Therefore, compared with DRAM, hybrid memory is more suitable for battery-powered IoT terminals, especially in scenarios with a long standby time, such as smart homes and smart agriculture.

Acknowledgments

This work was supported by National Key R&D Program of China (2019YFB2102400).

References

[1] S.W. Chung, *et al.*: "4 Gbit density STT-MRAM using perpendicular MTJ realized with compact cell structure," 2016 IEEE International Electron Devices Meeting (2017) (DOI: 10.1109/iedm.2016.7838490).
[2] S. Ikegawa, *et al.*: "Magnetoresistive random access memory: present and future," IEEE Trans. Electron Devices **67** (2020) 1407 (DOI: 10.1109/ted.2020.2965403).
[3] J.M. Slaughter, *et al.*: "High density ST-MRAM technology," 2012 International Electron Devices Meeting (2012) (DOI: 10.1109/iedm.2012.6479128).
[4] N.D. Rizzo, *et al.*: "A fully functional 64 Mb DDR3 ST-MRAM built on 90 nm CMOS technology," IEEE Trans. Magn. **49** (2013) 4441 (DOI: 10.1109/tmag.2013.2243133).
[5] J.M. Slaughter, *et al.*: "Technology for reliable spin-torque MRAM products," 2016 IEEE International Electron Devices Meeting (2016) (DOI: 10.1109/iedm.2016.7838467).
[6] S. Aggarwal, *et al.*: "Demonstration of a reliable 1 Gb standalone spin-transfer torque MRAM for industrial applications," 2019 IEEE International Electron Devices Meeting (2019) (DOI: 10.1109/iedm19573.2019.8993516).
[7] "EMD4E001G - 1 Gb Spin-transfer Torque MRAM," https://www.everspin.com.
[8] M. Sandler, *et al.*: "MobileNetV2: inverted residuals and linear bottlenecks," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (2018) (DOI: 10.1109/cvpr.2018.00474).
[9] C. Liu, *et al.*: "Fast cacheline-based data replacement for hybrid DRAM and STT-MRAM main memory," IEICE Electron. Express **17** (2020) 20200090 (DOI: 10.1587/elex.17.20200090).
[10] M.K. Qureshi, *et al.*: "Scalable high performance main memory system using phase-change memory technology," Computer Architecture News **37** (2009) 24 (DOI: 10.1145/1555815.1555760).
[11] H.G. Lee, *et al.*: "An energy- and performance-aware DRAM cache architecture for hybrid DRAM/PCM main memory systems," IEEE Computer Society (2011) (DOI: 10.1109/iccd.2011.6081427).
[12] H. Park, *et al.*: "Power management of hybrid DRAM/PRAM-based main memory," IEEE Design Automation Conference (2011) 59 (DOI: 10.1145/2024724.2024738).
[13] H.B. Yoon, *et al.*: "Row buffer locality aware caching policies for hybrid memories," 2012 IEEE 30th International Conference on Computer Design (2012) (DOI: 10.1109/iccd.2012.6378661).
[14] G. Dhiman, *et al.*: "PDRAM: a hybrid PRAM and DRAM main memory system," IEEE Design Automation Conference ACM (2009) (DOI: 10.1145/1629911.1630086).
[15] W. Zhang and L. Tao: "Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures," IEEE International Conference on Parallel Architectures & Compilation Techniques (2009) (DOI: 10.1109/pact.2009.30).
[16] L.E. Ramos, *et al.*: "Page placement in hybrid memory systems," Proceedings of the 25th International Conference on Supercomputing (2011) (DOI: 10.1145/1995896.1995911).
[17] G. Sun, *et al.*: "A novel architecture of the 3D stacked MRAM L2 cache for CMPs," 2009 IEEE 15th International Symposium on High Performance Computer Architecture (2009) (DOI: 10.1109/hpca.2009.4798259).
[18] M. Rasquinha, *et al.*: "An energy efficient cache design using spin torque transfer (STT) RAM," Proc. 16th ACM/IEEE International Symposium on Low Power Electronics and Design - ISLPED'10 (2010) 389 (DOI: 10.1145/1840845.1840931).
[19] A. Jog, *et al.*: "Cache revive: architecting volatile STT-RAM caches for enhanced performance in CMPs," Proc. 49th Annual Design Automation Conference on - DAC'12 (2012) (DOI: 10.1145/2228360.2228406).
[20] M.H. Samavatian, *et al.*: "An efficient STT-RAM last level cache architecture for GPUs," 2014 51st ACM/EDAC/IEEE Design Automation Conference ACM (2014) (DOI: 10.1109/dac.2014.6881524).
[21] E. Kultursay, *et al.*: "Evaluating STT-RAM as an energy-efficient main memory alternative," 2013 IEEE International Symposium on Performance Analysis of Systems and Software (2013) (DOI: 10.1109/ispass.2013.6557176).
[22] J. Meza, *et al.*: "A case for small row buffers in non-volatile main memories," 2012 IEEE 30th International Conference on Computer Design (2012) (DOI: 10.1109/iccd.2012.6378685).
[23] J. Wang, *et al.*: "Enabling high-performance LPDDRx-compatible MRAM," Proc. 2014 International Symposium on Low Power Electronics and Design (2014) 339 (DOI: 10.1145/2627369.2627610).
[24] K. Asifuzzaman, *et al.*: "Performance impact of a slower main memory: a case study of STT-MRAM in HPC," International Symposium on Memory Systems ACM (2016) 40 (DOI: 10.1145/2989081.2989082).
[25] K. Asifuzzaman, *et al.*: "STT-MRAM for real-time embedded systems: performance and WCET implications," International Symposium (2019) 195 (DOI: 10.1145/3357526.3357531).
[26] K. Asifuzzaman, *et al.*: "Enabling a reliable STT-MRAM main memory simulation," International Symposium (2017) 283 (DOI: 10.1145/3132402.3132416).
[27] "1 Gb: x4, x8, x16 DDR3 SDRAM," https://www.micron.com.
[28] "EMD3D256M - 256 Mb Spin-transfer Torque MRAM," https://www.everspin.com.
[29] "TN-41-01: calculating memory system power for DDR3," https://www.micron.com.
[30] A.B. Kahng, *et al.*: "CACTI 7: new tools for interconnect exploration in innovative off-chip memories," ACM Trans. Archit. Code Optim. **14** (2017) 1 (DOI: 10.1145/3085572).