electronics

MDPI

Article

A Cache Fill and Migration Policy for STT-RAM-Based Multi-Level Hybrid Cache in 3D CMPs

Fen Ge *, Lei Wang, Ning Wu and Fang Zhou

College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China; leiwang.study@foxmail.com (L.W.); wunee@nuaa.edu.cn (N.W.); zfnuaa@nuaa.edu.cn (F.Z.)

* Correspondence: gefen@nuaa.edu.cn; Tel.: +86-139-1397-2742

Received: 10 May 2019; Accepted: 4 June 2019; Published: 6 June 2019

Abstract: Recently, in 3D Chip-Multiprocessors (CMPs), a hybrid cache architecture of SRAM and Non-Volatile Memory (NVM) is generally used to exploit high density and low leakage power of NVM and a low write overhead of SRAM. The conventional access policy does not consider the hybrid cache and cannot make good use of the characteristics of both NVM and SRAM technology. This paper proposes a Cache Fill and Migration policy (CFM) for multi-level hybrid cache. In CFM, data access was optimized in three aspects: Cache fill, cache eviction, and dirty data migration. The CFM reduces unnecessary cache fill, write operations to NVM, and optimizes the victim cache line selection in cache eviction. The results of experiments show that the CFM can improve performance by 24.1% and reduce power consumption by 18% when compared to conventional writeback access policy.

Keywords: hybrid cache architecture; 3D CMP; NVM; cache fill; data migration

1. Introduction

Technology scaling has enabled an increasing number of cores on a chip in 3D Chip-Multiprocessors (CMP). As a result, more cache resources are needed to feed all the cores. With the increase in cache capacity, traditional SRAM consumes a large chip area and has high power consumption. The new Non-Volatile Memory (NVM) has the advantages of low static power consumption, high density and non-volatility, and is expected to replace the traditional SRAM [1–4]. However, NVM also suffers from challenges, such as limited write endurance and large write power consumption [5,6]. In order to solve these problems, hybrid cache architecture of SRAM and NVM is generally adopted in 3D CMPs [7–11]. Wu et al. [8] proposed a read-write-aware hybrid cache architecture, in which the cache is divided into read and write portions. Each portion contains STT-RAM and SRAM to improve the system performance. Lin et al. [11] used hybrid cache partitioning to improve write operations in Non-Uniform Cache Architecture (NUCA) architectures. It can be seen that hybrid cache architecture can effectively avoid the shortcomings of both NVM and SRAM technology.

A hybrid cache design relies on an intelligent data access policy that makes good use of the characteristics of both NVM and SRAM technology. Recent work has focused on optimizing the data access policy of hybrid cache, mainly to reduce the write overhead in NVM [12–15]. Ahn et al. [13] proposed a read-write mechanism in the hybrid cache architecture. This read–write mechanism can predict and bypass dead write operations, thereby reducing write overhead. They did not solve the problem of possible frequent write operations to NVM. Wang et al. [14] and Khan [15] proposed line placement and migration policy to improve system performance and reduce power consumption overhead. However, they did not consider data migration between different cache levels. In multi-level hybrid cache, the cost of migration between different cache levels is large and should be considered.

This paper proposes a cache fill and migration policy (CFM) for multi-level hybrid cache in 3D CMPs. The CFM optimizes data access in three aspects: Cache fill, cache eviction and data migration compared to the conventional writeback access policy. Firstly, the CFM can effectively reduce the unnecessary cache fill and the write operations to the NVM if a cache fill is required. Secondly, the CFM optimizes the victim cache line selection in cache eviction to reduce the data migration overhead of the dirty victim. Finally, the CFM analyzes the migration cost in multi-level hybrid cache architecture and proposes two migration principles to minimize the migration cost. These two migration principles are suitable in dark silicon era, in which case some open cache banks become closed and some closed cache banks become open (we call this cache architecture dynamically reconfiguration). The results show that in multi-level hybrid cache architecture, the CFM can achieve performance improvement and power saving effectively.

The rest of the paper is organized as follows. Section 2 analyzes the problems in conventional writeback access policy in hybrid cache. Section 3 presents the proposed cache fill and migration policy. Section 4 shows the experimental results and finally the conclusion is given in Section 5.

2. Problems with Conventional Access Policy

In conventional writeback cache access policy, data access can be divided into write access and read access. For write access, if it is missed on the cache, it will access the main memory. After that, the accessed data line is fetched and will be written back to the cache, we call this cache fill. If the cache is full, a victim cache line will be selected to be replaced according to the replacement strategy. If the victim cache line is dirty, the dirty data need a writeback to the lower level cache or main memory. The read access is basically the same as write access.

From the above access process, it can be seen that in the hybrid cache architecture, the conventional writeback access policy has the following problems:

- After an access miss on the cache, a cache fill is required, and a writeback might be required (if cache eviction occurs). If the fetched data line will not be accessed in the future, the cache fill is redundant. If the dirty data in the victim cache line will not be accessed as well, the writeback is unnecessary.
- In a cache fill, for the hybrid cache the conventional access policy does not consider the type of the cache. If the fetched data line will be frequently written, it is inappropriate to place the line in the NVM, otherwise it will cause a large write overhead.
- As mentioned in Section 1, in 3D CMPs cache resource is increased steadily. However, all of them cannot be simultaneously used within the peak power budgets. This phenomenon is called the dark silicon [16–18]. In the dark silicon era, the hybrid cache hierarchy might be dynamically reconfigured [19]. In this case, partially cache banks open will be closed. The conventional access policy does not support this situation, so it might cause data loss.

3. The Proposed Cache Fill and Migration Policy

3.1. Target Hybrid Cache Architecture

The cache fill and migration policy in this paper is based on the 3D CMP architecture similar to Figure 1.

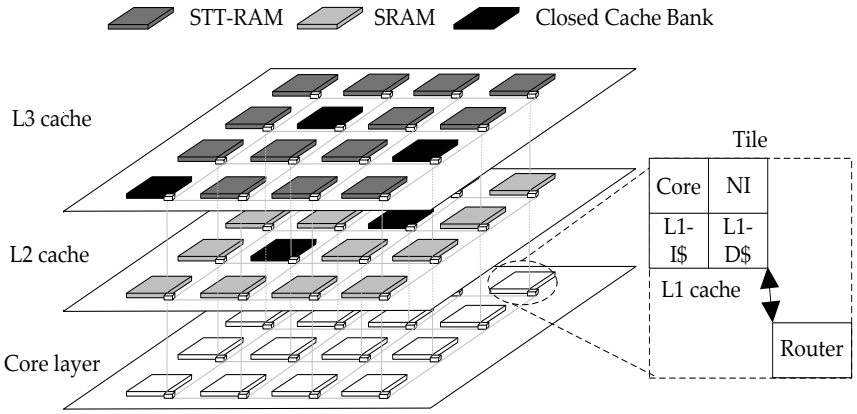

Figure 1. The target 3D Chip-Multiprocessor (CMP) architecture.

As shown in this figure, each tile in the core layer consists of core, network interface (NI) and L1 cache. The L1 cache is divided into L1 instruction cache (L1-I$) and L1 data cache (L1-D$), and they are private. There are two cache level implemented stacked on the core layer using different memory technology: L2 cache uses SRAM while L3 cache uses STT-RAM. We choose STT-RAM because STT-RAM is the most promising candidate of NVM [20]. SRAM is close to the core layer and can take advantage of its low write overhead. As the last level cache (LLC), STT-RAM can make the most of its advantages of high density and near-zero leakage power consumption.

In Figure 1, the black portion in L2 and L3 cache represents the cache banks that are dynamically closed under the limit of peak power budgets in dark silicon era. The unmarked portion represents the normally opened cache bank. The number and the location of the closed cache banks are only used as an illustration and are dynamically changed in the actual cache architecture reconfiguration.

3.2. Dead Line and Write-Burst Line

We used dead line and write-burst line for these two types of cache lines as indicators in CFM. When optimizing cache fill and migration, the cache line will determine whether it is dead or write-burst, and then the corresponding policy will be executed.

Dead line represents the cache line that will never be used again prior to eviction. Write-burst line represents the cache line that is written frequently. In the hybrid cache architecture, the write-burst line should not be placed in STT-RAM. Because STT-RAM has a large write overhead compared to SRAM, if write-burst lines are placed in STT-RAM, the system power consumption will have a rapid increase. STT-RAM has a limited write life. When the number of write operations to STT-RAM reaches a threshold, STT-RAM will be damaged.

3.3. Determination of Dead Line and Write-Burst Line

We used the method of counting cache accesses to determine whether the cache line was dead or write-burst, as shown in Figure 2. We added a lookup table in the cache controller to store the dead counter value and write-burst counter value to monitor the cache. When a cache line was accessed or evicted, the dead counter value and write-burst count value of the corresponding tag in the lookup table was updated. Since there are a lot of cache lines in the cache, monitoring each of them created a lot of hardware overhead and delay. We monitored the cache lines in the cache bank of only 1/64 sets to reduce the hardware overhead. This sampling-based monitoring was performed to ensure 80% accuracy of application load to cache access [21].

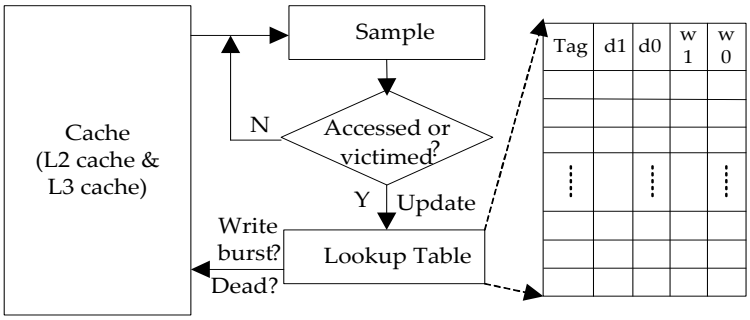

Figure 2. The method of counting cache accesses.

When determining whether the cache line is dead or write-burst, the corresponding counter value in the lookup table is found (if not found, the cache line is neither dead nor write-burst). If the dead counter value reaches the threshold Td, then the cache line is dead. If the write-burst counter value reaches the threshold Tw, the cache line is write-burst. We set the threshold Td and Tw both to four by simulation (the simulation result is mentioned in Section 4.2), so the dead counter occupies 2 bits (d1 and d0) and the write-burst counter occupies 2 bits (w1 and w0) in the lookup table. In addition, we set the maximum size of the lookup table to 1 KB.

The steps of updating the dead counter value and write-burst counter value of the lookup table are as follows:

- When an access occurs on the monitored cache line, the dead counter value is decreased by one hashed by the corresponding tag. If the access is a write access, the corresponding write-burst counter value in the lookup table is increased by one as well. If the tag of this cache line does not exist in the lookup table, the tag is written to the lookup table. When the lookup table is full, a victim cache line will be replaced by a random replacement strategy.
- When the monitored cache line is replaced or evicted, the dead counter value is increased by one and the write-burst counter value is decreased by one hashed by the corresponding tag in the lookup table. If the tag of this cache line does not exist in the lookup table, no action is taken.

3.4. The Optimization in Cache Fill

We optimize the cache fill as shown in Figure 3.

Firstly, when a data line is fetched from the main memory, if it is dead, it will be bypassed; otherwise, it will be placed in STT-RAM. In our hybrid cache hierarchy, the STT-RAM is the LLC. When a line fetched from the main memory fills the cache, it should be placed in the STT-RAM at first. In this case, the unnecessary cache fill of dead line can be avoided.

Secondly, the line should be migrated to the SRAM if it is write-burst. We place the write-burst line in the SRAM to reduce the write operations to STT-RAM. If the line is not write-burst, we place it in the STT-RAM to exploit the high density of STT-RAM and to reduce the pressure on the SRAM.

3.5. The Optimization in Cache Eviction

We optimize the selection of the victim cache line in cache eviction, as shown in Figure 4. When selecting the victim cache line, the dead line is preferred. If a dead line is found, this line is directly replaced. Otherwise, a victim cache line is selected according to the replacement strategy, Least Recent Used (LRU) strategy is usually used.

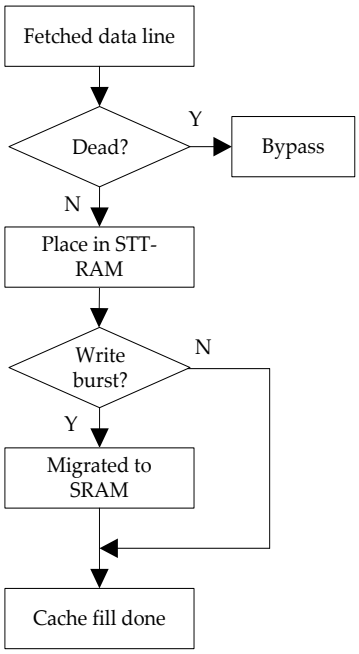

Figure 3. The optimization in cache fill.

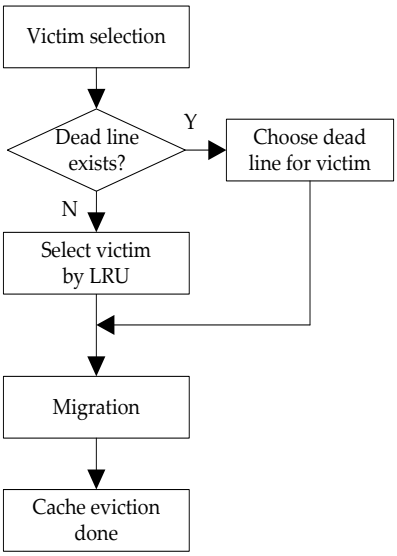

Figure 4. The optimization in cache eviction.

We prefer the dead line in cache eviction because the dead line will never be used again so that we do not need to consider the migration of dirty data.

3.6. The Optimization in Data Migration

In conventional writeback access policy, migrations usually occur in the cache eviction. Besides, in our work migrations might occur when the open cache banks become closed due to the reconfiguration of cache hierarchy in dark silicon era, so we have an optimization for two situations. In our work, migrations are categorized into two classes: Migrations in L2 cache and migrations in L3 cache, as shown in Figure 5.

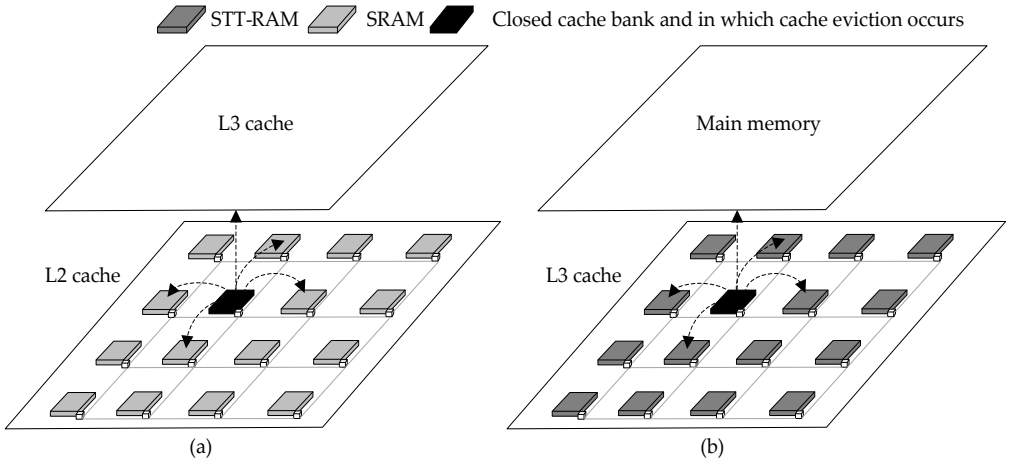

Figure 5. Two classes of migrations. (**a**) Migrations in L2 cache, and (**b**) migrations in L3 cache.

In Figure 5, the data line might be migrated to other cache banks in the same level or lower cache level or main memory as indicated by the dashed arrows.

For the migrations shown in Figure 5a, the optimizations made are as follows:

- If the cache line is not dirty, no migration is required. If the dirty line is dead, no migration is required as well.
- If L2 cache is not full, the dirty line is preferentially migrated to the cache bank which is closer and open. If the cache bank closer is full, it will be migrated to the cache bank that is far away and open in ascending order of the hops.
- If L2 cache level is full, the dirty line is written back to the L3 cache level.

For the migrations shown in Figure 5b, the optimizations made are as follows:

- If the cache line is not dirty, no migration is required. If the dirty line is dead, no migration is required as well.
- If L3 cache level is not full, the dirty line is preferentially migrated to the cache bank which is closer and open. If the cache bank closer is full, it will be migrated to the cache bank that is far away and open in ascending order of the hops.
- If L3 cache level is full, the dirty line is written back to the main memory.

In summary, our optimization in migration is concluded in Figure 6. In this figure, the gray portion represents the closed cache bank or the cache bank where cache eviction occurs that needs migration. Number 1 to 6 represents the migration priority based on the hops, and the migration to lower cache or main memory always has the lowest priority.

The migration policy optimization proposed in this paper considers the migration cost of the cache line. In order to reduce the migration cost, we follow these two principles:

- When dirty lines need to be migrated, they are preferentially migrated to the same level of cache. In the multi-level hybrid cache architecture, the access latency between different levels is large. The access request to the L3 cache occurs only when the L2 cache request misses, including the transaction delay of one additional access request and the transmission line delay in the vertical direction. Prioritizing migration to the same cache level can reduce these additional delays.
- In the same cache level, the dirty line is preferentially migrated to the cache line with a short distance. A short distance means low access latency. Priority migration to the close cache bank can reduce access latency and migration cost. Only if the cache level is full, the dirty line will be migrated to the lower cache level or main memory. When compared to the writeback to the lower cache level or main memory, the access latency caused by the migration to a distant cache bank is small.

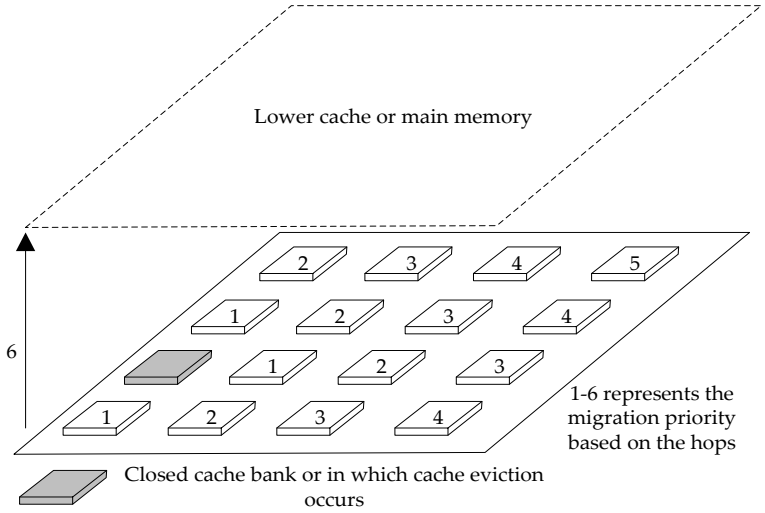

Figure 6. The optimization in migration.

4. Experimental Results

4.1. Platform Setup

We used a Gem5 full system simulator to setup the experimental platform. We modeled a 3D CMP architecture similar to Figure 1. For the core layer, there are 16 x86 cores connected by a mesh network at 32 nm technology. L2 cache and L3 cache were stacked on the core layer. They were both divided into 16 cache banks connected by mesh network. The L2 cache used SRAM technology, and each cache bank had a capacity of 256 KB. The L3 cache used STT-RAM technology and each cache bank had a capacity of 1 MB. The detailed platform information is listed in Table 1. We used McPAT for power consumption estimation. The applications were selected from the SPEC2006 benchmarks suites. We chose these five applications for simulations, 401.bzip2, 462.libquantum, 456.hmmer, 470.lbm and 429.mcf because they are of different types, including cache-intensive, compute-intensive and balanced applications. The experimental results can illustrate the influence of CFM on different types of applications. These applications are run on 16 cores in a multi-threaded manner. The detailed application information is shown in the Table 2.

Table 1. System configuration.

Core	16 nuclear out of order execution processors, 2 GHz, x86, 4 × 4 Mesh
L2 cache	Shared, 16-way, LRU replacement, 20-cycle latency, 4 MB
L3 cache	Shared, 16-way, LRU replacement, 50-cycle latency, 16 MB
Cache hierarchy	D-NUCA, MESI
Main memory	4 G, 300-cycle latency

Table 2. The detailed application information.

Application	Read Ratio	Write Ratio	Type
401.bzip2	86.2%	13.8%	Compute-intensive
462.libquantum	100%	0%	Cache-intensive
456.hmmer	63.6%	36.4%	Compute-intensive
470.lbm	15.7%	84.3%	Cache-intensive
429.mcf	94.5%	5.5%	Balanced

In order to simulate the reconfiguration of cache hierarchy, we referred to the EECache proposed in the literature [21], and realized the dynamic close and open of cache banks. We setup 4 experimental groups to compare the impact of CFM on the power consumption and system performance as shown in Table 3.

Table 3. Experimental group simulation parameters configuration.

Group	Access Policy	Cache Hierarchy
Group 1	Conventional	Static
Group 2	CFM	Static
Group 3	Conventional	Dynamic
Group 4	CFM	Dynamic

Group 1 uses the conventional writeback access policy and the cache hierarchy is static. Group 2 uses the CFM, and the cache hierarchy is static. Group 3 uses the conventional writeback access policy and the cache hierarchy is dynamically reconfigured. Group 4 uses CFM and the cache hierarchy is dynamically reconfigured.

4.2. Experimental Results

To simplify the design, we set both Tw and Td to Th. Figure 7 shows the Instruction Per Cycle (IPC) of Group 4 normalized to Group 3 in different threshold of Th. We chose a Th of 4 (2 bits), 8 (3 bits), 16 (4 bits), 32 (5 bits) and 64 (6 bits) for the experiment. As shown in this figure, the CFM using a Th of 4, 8 and 16 improved the IPC, but the CFM using a Th of 32 and 64 did not improve the system performance. The IPC increased at a Th of 4, 8 and 16, but the gap was small. The threshold of 4 balanced IPC and hardware overhead well. We chose 4 for the threshold of Td and Tw, and the following experimental results were carried out under the threshold of 4.

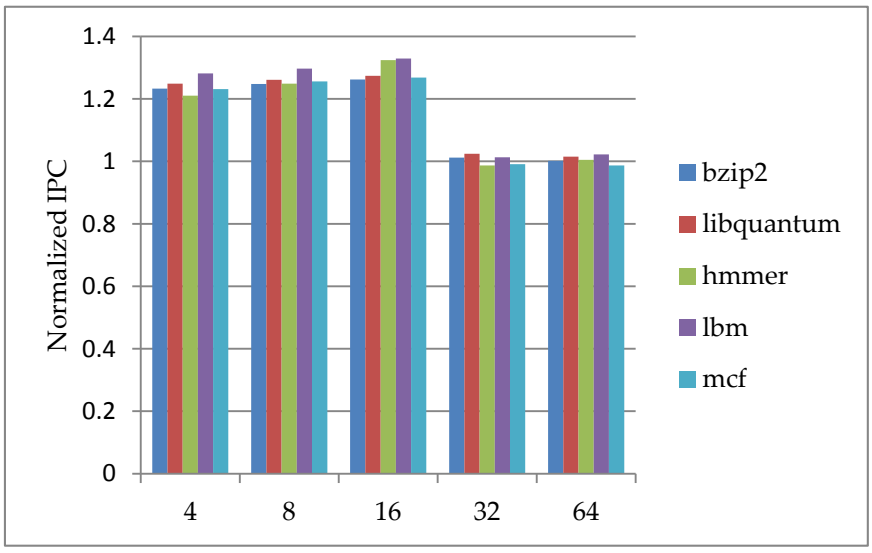

Figure 7. Comparison of IPC of Group 4 normalized to Group 3 in different Th.

Figures 8 and 9 show the normalized Energy and the normalized System Delay comparison for each experimental group. In Figure 8, Group 4 consumed more energy than Group 3. However, in Figure 9, Group 4 had a smaller system delay than Group 3. Figure 10 shows the normalized Energy-Delay Product (EDP) comparison for each experimental group. As shown in this figure, Group 4 reduced the EDP by about 16% when compared to Group 3. Thus, it can be seen from Figures 8–10, the combination of Energy and Delay makes the system EDP lower in CFM. This is because the CFM reduces the unnecessary cache fill, write operations to NVM and optimizes the victim selection in cache eviction, thereby reducing the write operations of the system. The CFM migrates the dirty data with a minimum migration cost, reducing the power consumption of the accesses between different levels.

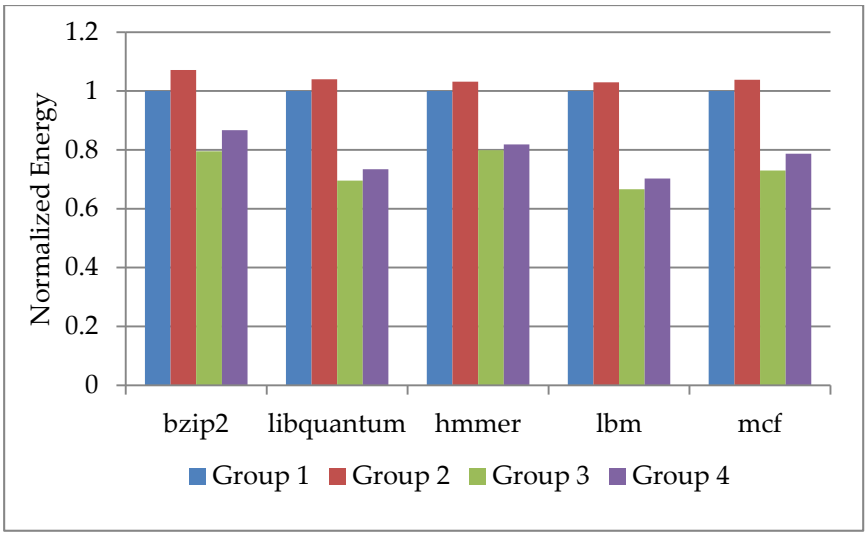

Figure 8. Comparison of Energy normalized to Group 1.

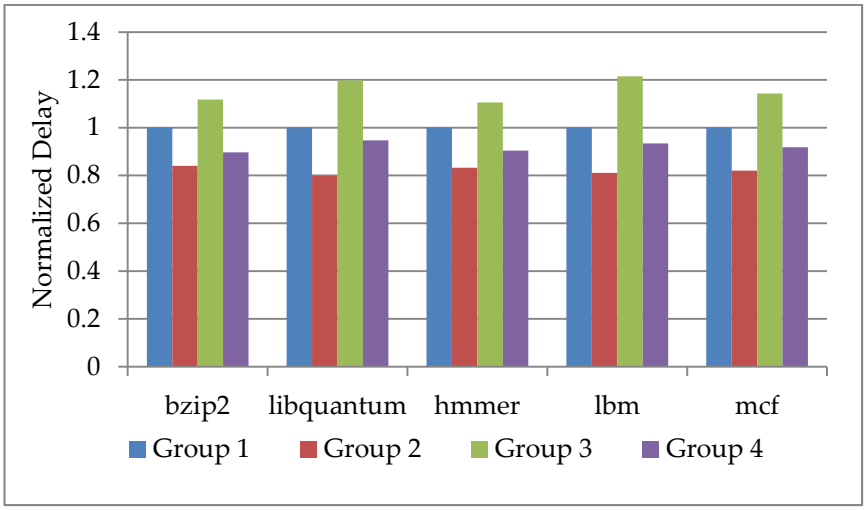

Figure 9. Comparison of System Delay normalized to Group 1.

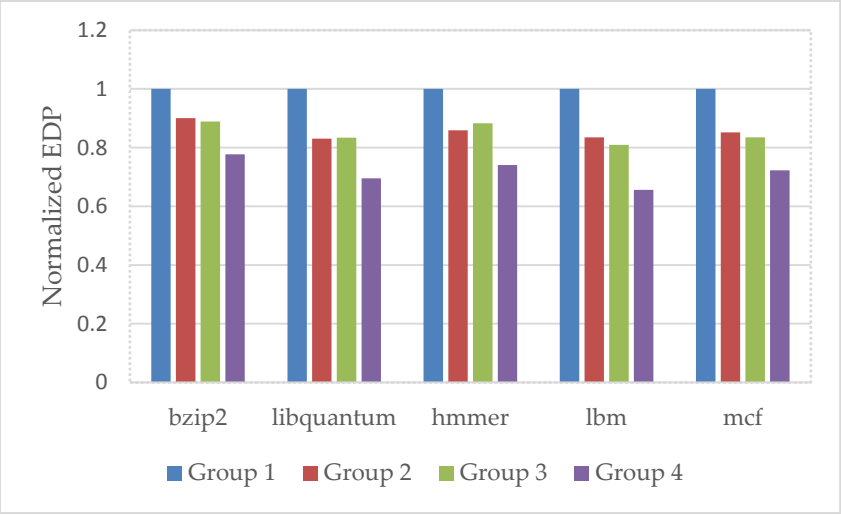

Figure 10. Comparison of Energy-Delay Product normalized to Group 1.

Figure 11 shows the distribution of write operations to STT-RAM normalized to all write operations for different applications. It can be seen that the CFM reduces write operations to the STT-RAM portion to 32% on average of the total number of write operations. In libquantum and lbm applications, the rate is reduced to 24% and 25% respectively. Because libquantum and lbm are cache-intensive applications, the write burst counters in the lookup table can easily reach the threshold to migrate the data line to SRAM.

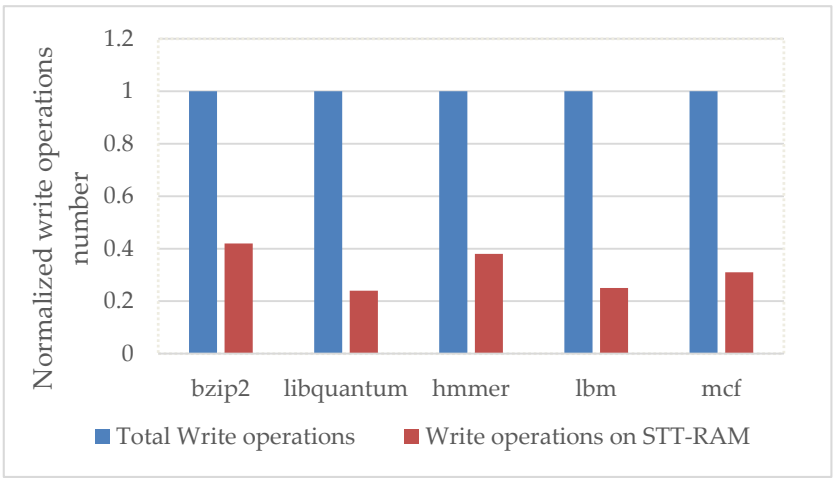

Figure 11. Comparison of the number of write operations normalized to the total write operations.

Figure 12 shows the comparison of normalized IPC for each experimental group. As shown in the figure, Group 2 improves the IPC by about 21.9% when compared to the Group 1. Group 4 improves the IPC by about 25.6% when compared to Group 3. The CFM supports the dynamic reconfiguration of cache hierarchy, so the CFM can take advantages of it to improve the system performance.

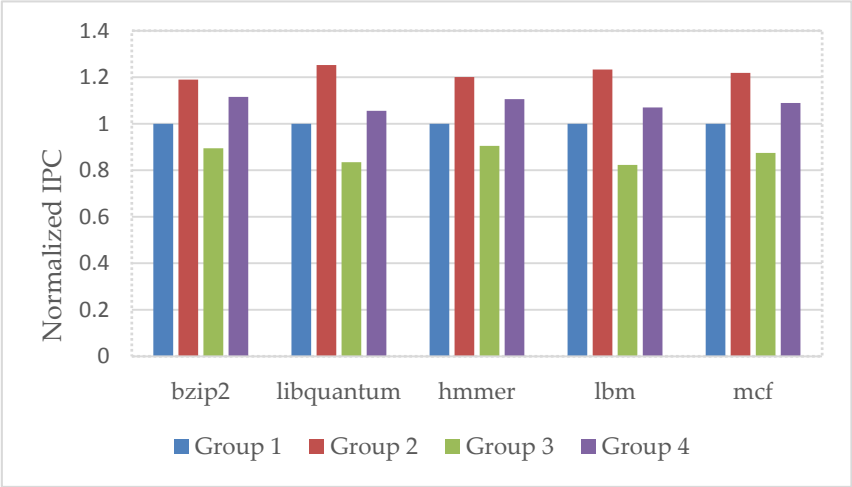

Figure 12. Comparison of IPC normalized to group 1.

5. Conclusions and Future Work

In this paper, we analyzed the problems in conventional writeback access policy for hybrid cache architecture. Then we proposed a cache fill and migration policy (CFM) for multi-level hybrid cache in 3D CMP. The CFM reduces unnecessary cache fill, write operations to the NVM and optimizes victim cache line selection. In addition, the CFM optimizes the data migration between different cache levels. The experiments carried out with SPEC2006 benchmarks showed that the CFM can achieve 16% power savings and 25.6% performance improvement, compared to the conventional writeback cache access policy which does not consider the hybrid cache architecture and the cache reconfiguration in dark silicon era.

This paper was mainly focused on the optimization of write operations in some different cache access situations. However, there are some other cache access situations such as prefetch write operations and hit write operations in our cache architecture that we did not consider, and they make up a large part of the write operations. We will improve this in future work.

Author Contributions: Conception and structure of the concept of this paper, F.G.; Resources, F.Z.; Supervision, N.W.; Writing-original draft, L.W.; Writing-review and editing, L.W. and F.G.

Funding: This work is supported by the Fundamental Research Funds for the Central Universities under Grant NS2016041.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Emre, K.; Mahmut, K.; Anand, S.; Onur, M. Evaluating STT-RAM as an energy-efficient main memory alternative. In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX, USA, 21–23 April 2013; pp. 256–257.
2. Xu, W.; Sun, H.; Wang, X.; Chen, Y.; Zhang, T. Design of Last-Level On-Chip Cache Using Spin-Torque Transfer Ram (STT-RAM). *IEEE Trans. VLSI Syst.* **2009**, *19*, 483–493. [CrossRef]
3. Hanbin, Y.; Justin, M.; Naveen, M.; Norman, P.; Onur, M. Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories. *ACM Trans Archit. Code Optim. (TACO)* **2015**, *11*. [CrossRef]
4. Asit, K.; Dong, X.; Sun, G.; Xie, Y.; NVijaykrishnan Chita, R. Architecting on-chip interconnects for stacked 3D STT-RAM caches in CMPs. In Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA), San Jose, CA, USA, 4–8 June 2011; pp. 69–80.
5. Li, Q.; He, Y.; Li, J.; Shi, L.; Chen, Y.; Chun, J. Compiler-Assisted Refresh Minimization for Volatile STT-RAM Cache. *IEEE Trans. Comput.* **2015**, *64*, 2169–2181. [CrossRef]

6. Wang, J.; Tim, Y.; Wong, W.; Ong, Z.; Sun, Z.; Li, H. A coherent hybrid SRAM and STT-RAM L1 cache architecture for shared memory multicores. In Proceedings of the 19th Asia and South Pacific Design Automation Conference (ASP-DAC), Singapore, 20–23 January 2014; pp. 610–615.
7. Cong, J.; Karthik, G.; Huang, H.; Liu, C.; Glenn, R.; Zou, Y. An energy-efficient adaptive hybrid cache. In Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design, Fukuoka, Japan, 1–3 August 2011; pp. 67–72.
8. Wu, X.; Li, J.; Zhang, L.; Evan, S.; Xie, Y. Power and performance of read-write aware Hybrid Caches with non-volatile memories. In Proceedings of the 2009 Design, Automation and Test in Europe Conference and Exhibition, Nice, France, 20–24 April 2009; pp. 737–742.
9. Li, J.; Xue, C.; Xu, Y. STT-RAM based energy-efficiency hybrid cache for CMPs. In Proceedings of the IEEE/IFIP 19th International Conference on VLSI and System-on-Chip, Hong Kong, China, 3–5 October 2011; pp. 31–36.
10. Lee, S.; Jung, J.; Kyung, C. Hybrid cache architecture replacing SRAM cache with future memory technology. In Proceedings of the IEEE International Symposium on Circuits and Systems, Seoul, Korea, 20–23 May 2012; pp. 2481–2484.
11. Chao, L.; Chiou, J. High-Endurance Hybrid Cache Design in CMP Architecture with Cache Partitioning and Access-Aware Policies. *IEEE Trans. VLSI Syst.* **2015**, *23*, 2149–2161.
12. Nikos, H.; Michael, F.; Babak, F.; Anastasia, A. Reactive NUCA: Near-optimal block placement and replication in distributed caches. In Proceedings of the 36th Annual International Symposium on Computer Architecture, Austin, TX, USA, 20–24 June 2009; Volume 37, pp. 184–195.
13. Ahn, J.; Yoo, S.; Choi, K. DASCA: Dead Write Prediction Assisted STT-RAM Cache Architecture. In Proceedings of the IEEE 20th International Symposium on High Performance Computer Architecture, Orlando, FL, USA, 15–19 February 2014; pp. 25–36.
14. Wang, Z.; Daniel, A.; Xu, C.; Sun, G.; Xie, Y. Adaptive placement and migration policy for an STT-RAM-based hybrid cache. In Proceedings of the IEEE 20th International Symposium on High Performance Computer Architecture, Orlando, FL, USA, 15–19 February 2014; pp. 13–24.
15. Khan, S.M.; Tian, Y.; Daniel, A. Sampling Dead Block Prediction for Last-Level Caches. In Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture, Atlanta, GA, USA, 4–8 December 2010; pp. 175–186.
16. Hadi, E.; Emily, B.; Amant, R.; Karthikeyan, S.; Burger, D. Dark silicon and the end of multicore scaling. In Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA), San Jose, CA, USA, 4–8 June 2011; pp. 365–376.
17. Turakhia, Y.; Raghunathan, B.; Garg, S.; Marculescu, D. HaDes: Architectural synthesis for heterogeneous dark silicon chip multi-processors. In Proceedings of the 50th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 29 May–7 June 2013; pp. 1–7.
18. Raghunathan, B.; Turakhia, Y.; Garg, S.; Marculescu, D. Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 18–22 March 2013; pp. 39–44.
19. Niknam, S.; Asad, A.; Fathy, M.; Rahmani, A. Energy efficient 3D Hybrid processor-memory architecture for the dark silicon age. In Proceedings of the 10th International Symposium on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC), Bremen, Germany, 29 June–1 July 2015; pp. 1–8.
20. Salman, O.; Arghavan, A.; Kaamran, R.; Mahmood, F. An Energy-Efficient Heterogeneous Memory Architecture for Future Dark Silicon Embedded Chip-Multiprocessors. *IEEE Trans. Emerg. Top. Comput.* **2016**. [CrossRef]
21. Cheng, H.; Matt, P.; Narges, S.; Ivan, S.; Mary, J.; Mahmut, K.; Jack, S.; Xie, Y. EECache: A Comprehensive Study on the Architecture Design for Energy-Efficient Last-Level Caches in Chip Multiprocessors. *J. ACM Trans. Archit. Code Optim. (TACO)* **2015**, *12*. [CrossRef]