PAPER *Special Section on Low-Power and High-Speed Chips*

A Metadata Prefetching Mechanism for Hybrid Memory Architectures

Shunsuke TSUKADA[†a)], **Hikaru TAKAYASHIKI**[†], *Nonmembers*, **Masayuki SATO**[†b)], *Member*, **Kazuhiko KOMATSU**[†], *Nonmember*, *and* **Hiroaki KOBAYASHI**[†], *Member*

SUMMARY A hybrid memory architecture (HMA) that consists of some distinct memory devices is expected to achieve a good balance between high performance and large capacity. Unlike conventional memory architectures, the HMA needs the metadata for data management since the data are migrated between the memory devices during the execution of an application. The memory controller caches the metadata to avoid accessing the memory devices for the metadata reference. However, as the amount of the metadata increases in proportion to the size of the HMA, the memory controller needs to handle a large amount of metadata. As a result, the memory controller cannot cache all the metadata and increases the number of metadata references. This results in an increase in the access latency to reach the target data and degrades the performance. To solve this problem, this paper proposes a metadata prefetching mechanism for HMAs. The proposed mechanism loads the metadata needed in the near future by prefetching. Moreover, to increase the effect of the metadata prefetching, the proposed mechanism predicts the metadata used in the near future based on an address difference that is the difference between two consecutive access addresses. The evaluation results show that the proposed metadata prefetching mechanism can improve the instructions per cycle by up to 44% and 9% on average.

key words: *hybrid memory architecture, metadata, prefetch, address difference, performance*

1. Introduction

Recent computers are equipped with a large number of cores due to requests for significant amount of computing resource. Such computers execute applications that handle large amounts of data, e.g., extensive data analysis and machine learning, which requires a larger memory capacity. One of the promising memory technologies for large capacity is Phase Change Memory (PCM). PCM memorizes data by using the difference in resistance value caused by the phase change of chalcogenide glass [1]. Since the resistance value changes in different medium conditions, PCM allows a large amount of data to be stored in a single memory cell. Thus, PCM is expected to achieve a larger memory capacity than DRAM. However, since PCM uses physical change for write operations, the latencies for the operations are much higher than those of DRAM. Therefore, it is challenging to use PCM alone and realizes the high-speed and large-capacity memory system.

Manuscript received June 30, 2021.
Manuscript revised October 20, 2021.
Manuscript publicized December 3, 2021.

[†]The authors are with Tohoku University, Sendai-shi, 980–8579 Japan.
a) E-mail: shunsuke.tsukada.p1@dc.tohoku.ac.jp
b) E-mail: masa@tohoku.ac.jp

DOI: 10.1587/transele.2021LHP0004

To realize both high speed and large capacity, hybrid memory architectures (HMAs) have been proposed [2]–[5]. The HMA consists of the high-performance memory called the *near memory* (NM) and the large-capacity memory called *far memory* (FM). By allocating the data between the NM and the FM according to the data access pattern, the HMA can take advantage of the fast data access of the NM and the large capacity of the FM. Currently, real hardware systems including HMAs become available, e.g., Intel Knights Landing processor [6] and systems including Intel Optane DC Persistent Memory [7].

From a data management perspective, three approaches have been proposed to effectively use the NM and the FM. The first one uses the NM as cache [8]–[19]. The second one uses the NM and the FM as a flat address-space memory [3]–[5], [20]–[27]. The third one is the hybrid approach that uses a part of the NM and the FM as a flat address-space and the rest of the NM as a cache [28]. When the NM and the FM used as a flat address-space, the HMA needs an information to manage data between the NM and the FM. This information is called *metadata*. The metadata has the information of the data location. To access the data correctly, the memory controller always has to refer to the metadata before accessing the data.

Since the memory controller is on a chip, there is insufficient space to store all of the metadata in the memory controller. The state-of-the-art HMA stores the metadata in the NM [28]. In this case, when a data request comes, the memory controller has to access the NM to refer to the metadata. It causes a long access latency to access the target data and results in the performance degradation. To avoid the performance degradation, the HMA has the small memory array to cache the metadata in the memory controller, called *metadata memory array* (MMA). However, as the total capacity of the HMA increases, the amount of the metadata also increases so that the MMA cannot hold all the metadata. Therefore, it becomes difficult to hide the long access latency to the target data in the MMA.

To solve this problem, this paper proposes a metadata prefetching mechanism for HMAs. The metadata prefetching can reduce the metadata access latency by improving the hit rate of MMA. This paper first conducts a preliminary experiment to discuss a strategy to determine the metadata to be prefetched. Based on the experimental results, this paper proposes that the prefetching mechanism should consider the prefetching based on the *address difference that* is a regularity of the difference between two successive accesses. This paper implements and evaluates the proposed metadata prefetching mechanism based on the address difference and confirms the performance improvement.

The contributions of this paper are the following two points.

1. This paper shows the address difference is effective for the metadata prefetching in a preliminary experiment.
2. This paper proposes a metadata prefetching mechanism by using the knowledge of the preliminary experiment and verifies its performance improvement for the HMA through evaluations.

The remaining of this paper is organized as follows. Section 2 shows the metadata problem for the HMA. Section 3 presents related work. Section 4 preliminarily evaluates the memory access behavior of examined benchmarks and analyzes their address differences. Section 5 proposes a metadata prefetching mechanism based on the address differences. Section 6 evaluates the performance of the proposed mechanism. Finally, Sect. 7 summarizes this paper.

2. Metadata Problem in an HMA

In contrast to conventional memory systems, an HMA consists of two memories with entirely different performances and capacities. To take advantage of the high performance of the NM and the large capacity of the FM, the memory controller has to move data between the NM and the FM according to data access behavior. In the HMA, the data is moved and managed at sector granularity. The sector is a data management unit for the HMA. Some previous researches record the number of accesses to each sector, and take a strategy to place the sectors with the highest number of accesses in the NM and the sectors with the lowest number of accesses in the FM [3], [13].

To access the same sector even after migrating it the memory controller should have a mapping information that is called metadata. At least, the metadata has to contain two addresses. The first address is the original physical address before the migration. The processors issue their requests based on this address. The second address is the current physical address, which may be different from the original address because the HMA can migrate the data. Since it is necessary to hold two addresses for the entire capacity of the HMA, the number of metadata tends to become significant. Therefore, it is not easy to store such a large amount of metadata on a processor chip. Thus, the HMA has to store all of the metadata in the NM. The memory controller always accesses the metadata before accessing the data. Therefore, the latency for accessing the data is longer than that of using a conventional memory. To avoid accessing the NM for the metadata lookup, the HMA caches some metadata in an on-chip memory controller. A memory that caches the metadata temporarily is called the Metadata Memory Array (MMA).

The HMA faces two problems regarding metadata reference latency and the amount of the metadata. Figure

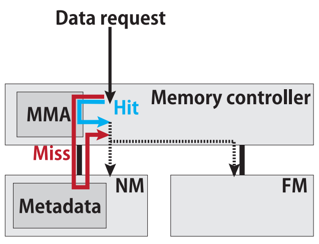


Fig. 1 MMA hit & MMA miss.

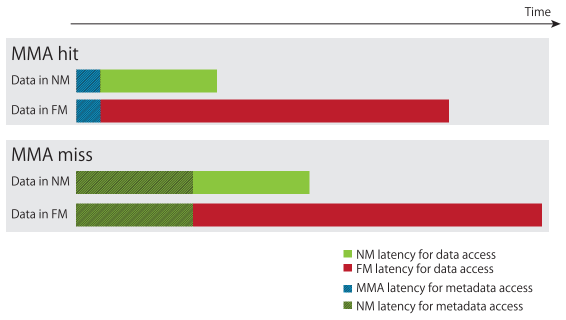


Fig. 2 MMA hit latency & MMA miss latency.

1 shows paths to access data in the HMA. Depending on whether caching the metadata in the MMA or not, there are two access paths to the data. The first path is the blue line in Fig. 1 showing the case that the metadata exists in the MMA when the memory controller checks the metadata, which treats an MMA hit. The second path is the red line in Fig. 1 showing the case that the metadata does not exist in the MMA when the memory controller checks the metadata, which treats an MMA miss. When an MMA miss occurs, the memory controller has to access the metadata in the NM. After referring to the metadata, the memory controller finally accesses the data. The MMA miss causes a long latency compared with the case with the MMA hit. For example, we assume that the MMA access latency is several nanoseconds, the NM access latency is about 100 nanoseconds, and the FM latency is about three times that of the NM. Figure 2 shows the latency at the time of the MMA hit and at the time of the MMA miss when data is stored in the NM or the FM. From Fig. 2, the latency of the MMA miss is longer than that of the MMA hit, regardless of whether the data is stored in the NM or the FM. Thus, the longer latency caused by the MMA miss is a penalty for accessing the data.

In addition to the metadata access latency, the HMA has the other problem of increasing metadata. The HMA needs to increase the capacity of the NM and the FM as a result of the development of applications that handle large data, such as big-data analysis, machine learning, and high-performance computing. Additionally, the amount of the metadata increases in proportion to the capacity of an HMA. Since the memory controller cannot store all of the metadata on a chip, the increase in the metadata will increase the MMA capacity miss during the execution. Therefore, it is essential to reduce the number of the MMA misses efficiently even when the amount of metadata on the MMA is limited.

This paper aims at reducing the number of accesses to the NM for the metadata reference. In addition, it is also necessary to have a mechanism to reduce the latency even when the capacity of the MMA is much smaller than the total amount of the metadata in the NM.

3. Related Work

The conventional HMAs are mainly categorized into three approaches. The first HMA uses the NM as a cache. The second HMA uses the NM and the FM as a flat address-space memory. The third HMA is a hybrid of the first one and the second one. Thus, the third one uses a portion of the NM as a cache and also uses the rest of the NM as a flat address-space memory combined with the FM. This paper refers to them as caching-based HMAs, flat address-space HMAs, mixed HMAs, respectively. Since all these approaches have its data management mechanisms, this section briefly reviews these proposals and their metadata management.

3.1 Currently Available HMAs

Some systems including HMAs have become available. Intel Knights Landing processor (KNL) [6] have on-package DRAM modules called MCDRAM and combine them with external DRAM DIMM modules as a single memory system. KNL has three modes for different usage of both MCDRAM and external DRAM DIMMs, cache, flat, and hybrid. The cache mode can transparently use MCDRAM as a cache memory, and the data are managed by hardware. In the flat mode, both MCDRAM and DRAM DIMMs are organized as a single memory address space. The flat mode has an advantage that a user can choice of the placement of data to MCDRAM or DRAM DIMMs as they like. On the other hand, a user needs additional efforts of managing the placement of data by command line operations and/or program libraries. The hybrid mode can divide MCDRAM into two regions and can use one part as MCDRAM and the other part as the cache mode and the flat mode.

Intel Optane [7] is a DDR4-compatible DIMM module of non-volatile memory. The system consisting of Optane DIMMs and DRAM DIMMs can have two modes. The memory mode uses DRAM DIMMs as a cache memory and manages data transparently from a user. The app direct mode needs management of data by users and programmers, as the flat mode of KNL. However, Optane DIMMs by the app direct mode are accessed as a file system. Therefore, accessing Optane DIMMs in the app direct mode is a relatively high cost due to software overheads rather than that of the memory mode.

3.2 Caching-Based HMAs

The first case is the HMA using the NM as a cache. By using a part of the NM as a cache, the memory controller can immediately respond to the change of the application behavior. In the case, the memory controller requires tags enough to manage the whole area of the NM, which increases the number of tags in proportion to the size of the NM. Thus, the memory controller stores all the tags in the NM. However, storing all the tags in the NM causes a long tag-lookup latency because the memory controller needs to access the NM two times before accessing the data by checking whether the data requested from the CPU is cached in the NM or not.

In the Alloy cache [9], tags and data are managed as a single chunk that is called TAD. When a TAD is accessed, data of the TAD is consecutively accessed when searching for tags. If the information in the tag matches the address of the requested data, the data can be used as is. Therefore, the latency for a tag search can be reduced. In addition, to prepare for DRAM cache misses, the data in the FM is accessed in parallel with the tag search. In this way, the tag-referencing latency in case of DRAM cache misses can be covered up.

Banshee [13] adds metadata functionality to the page table and also provides a tag buffer to store a part of the metadata information to keep the page table coherent with the TLB. Because of this linkage with the page table, adding metadata information to the page table requires an operation by the OS, which incurs a large overhead. Furthermore, this overhead limits the number of re-locations between the NM and the FM, making Banshee unable to flexibly respond to the behavior of applications.

On the other hand, the apparent disadvantage of the caching-based HMAs is that the HMA cannot increase the address space of the memory system. Since the NM is used as a cache memory, the NM keeps copies of the contents in the FM. Although the capacity of the HBM modules generally small compared with the conventional external memories, a single module of the HBM can realize a gigabyte-class memory. Therefore, it can be unignorable as the resource to increase the total memory capacity of the HMA.

3.3 Flat Address-Space HMAs

The second case is the HMA using the NM as a flat address-space memory. In this case, the memory controller has a larger flat address-space. The HMA stores the data only in either the NM or the FM, and supports migration between the NM and the FM unlike. Even after the data migration, the memory controller of the HMA needs to correctly access the current data position. To access correctly, mapping information is required as metadata in the HMA. Usually, the memory controller has to prepare the metadata for all data in both the NM and the FM. Therefore, the amount of the metadata tends to be larger than the tags of the case using the NM as a cache. To solve the problem, the memory controller stores all the metadata in the NM in bulk. In some cases, the memory controller stores all the metadata in the page table with the help of the OS.

Dong *et al.* have proposed the flat address-space HMA that devises a relocation method so that all relocation information can be recorded in the translation table [3]. Therefore, there is no need to access the NM for metadata references. However, in order to keep the translation table as small as possible, the size of a data management unit is increased to 4 MB. In this case, the overhead of moving the data is large, and it is difficult to relocate the data with a high frequency. Therefore, the system cannot respond quickly to the access status of applications.

In CAMEO [23], data relocation is managed by limiting the range of data relocation for each Congruence Group. By limiting the range of relocation for each group, CAMEO can reduce the size of the location information stored in the Line Location Table (LLT). Furthermore, since CAMEO was able to reduce the size of the LLT, CAMEO placed the LLT together with the data in NM so that the data can be accessed together when accessing the LLT. In this way, the LLT access latency can be hidden. Furthermore, by using Line Location Predictor (LLP) to predict the data location within the congruence group, CAMEO can hide the latency even if the data accessed with LLT is missed.

The drawback of the flat address-space HMA is mainly due to the migration. Since the data are migrated between the NM and the FM, it causes memory requests for migrations in addition to those from processing cores. Therefore, the data transfer capability of the memory system cannot be fully used for the performance improvement of applications.

3.4 Mixed HMAs

The third case is the HMA using a part of the NM and the FM as a flat address-space memory and the rest of the NM as a cache. Therefore, the mixed HMA takes the trade-off between the caching-Based HMAs and the flat address-space HMAs.

Hybrid^2 [28] is the state-of-the-art HMA. In Hybrid^2, to mitigate the overhead of storing the metadata (mapping information and tags), the memory controller stores the metadata of only the data of the NM cache in eXtended Tag Array (XTA) as MMA. Only the metadata of the data in the cache area is stored in the MMA. Other metadata for data in flat address is stored in the NM. The XTA is provided for some of the metadata in Hybrid^2. If the metadata for the data to be accessed is not in the XTA (XTA miss), it is necessary to access the metadata in the NM to determine the current data location. Therefore, in the case of an XTA miss, the data access latency increases due to the NM access associated with the metadata reference. However, Hybrid^2 does not take any countermeasures against the overhead of XTA misses. Therefore, there might be room to reduce this overhead and further improve the performance.

As discussed above, the previous researches focus on the data management and increase the flexibility of the HMAs. On the other hand, the flexibility of the data management increases the amount of metadata. Moreover, the capacity of an HMA is expected to increase. The amount of metadata will grow more and more, and the memory controller of a future HMA cannot cover the whole metadata with the small MMA. Comparing an HMA with a conventional memory architecture, additional references are required for metadata. Therefore, the latency overhead for accessing the target data will become significant. Therefore, this paper focuses on reducing access latency for accessing the metadata and tries to prefetch the metadata before they are required.

4. Analysis of Metadata Access Characteristics

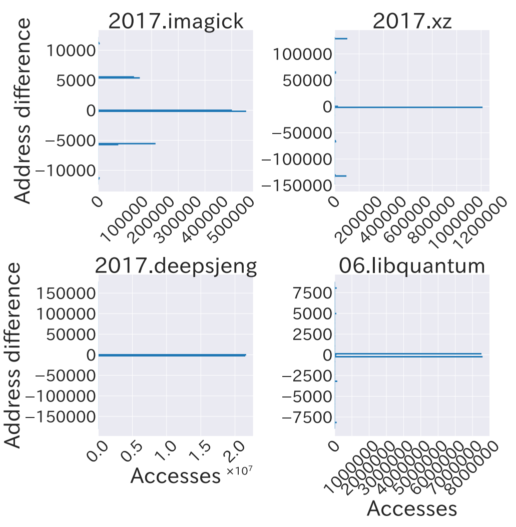
This paper focuses on metadata prefetching that can reduce the MMA misses even when the MMA size becomes relatively smaller than the total amount of the metadata. To realize a metadata prefetching mechanism, the mechanism must predict the metadata to be accessed next. To predict the necessary metadata for the prefetching, This paper investigates and analyzes patterns of metadata accesses to explore the access characteristics useful for the data management, especially prefetching.

For the analysis, this paper uses Gem5 simulator [29]. Table 1 shows the parameters used for this analysis. We assume a single core of a modern multi-core processor with a three-level cache hierarchy and the state-of-the-art HMA [28]. The HMA of this paper uses DRAM as the NM and Non-Volatile Memory (NVM) as the FM to simulate a difference in access latency between the NM and the FM. As for the NVM, the parameters of PCM [30] are assumed. The HMA manages and migrates the data sector-by-sector, whose size is 4 KB. The parameter of the sector size is based on the original work [28]. As benchmarks, this paper uses the benchmarks with large memory footprints from the SPEC CPU 2006 [31] and 2017 [32] suites. Each benchmark is executed by the first one billion instructions for warm-up and the following one billion instructions for performance measurements.

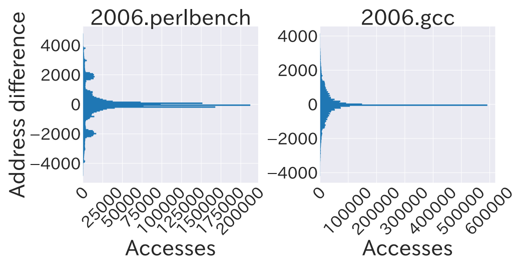
Based on the experimental results, this paper focuses on the address difference, defined as the difference between addresses of two consecutive accesses. Figure 3 shows the histograms of the address differences in the case of whole execution of six benchmarks. The vertical axis shows the address difference that is calculated from two consecutive-

Table 1 Simulation parameters.

Parameter	Value
Core	3 GHz, 4-issue, Out-of-Order
L1-I Cache	64 KB, 64 B line, 4 ways, 2 cycles latency
L1-D Cache	64 KB, 64 B line, 4 ways, 2 cycles latency
L2 Cache	256 KB, 64 B line, 8 ways, 20 cycles latency
L3 Shared Cache	1 MB, 64 B line, 16 ways, 32 cycles latency
Hybrid Main Memory	DRAM 256 MB, tCAS-tRCD-tRP 14-14-14 NVM 4 GB, read time 150 ns, write time 500 ns MMA 512 KB



(a) Benchmarks in the constant group



(b) Benchmarks in the not-constant group

Fig. 3 Address difference.

accessed sector addresses. The horizontal axis shows the number of samples.

From Fig. 3(a), it is observed that the address differences are almost constant. As the address difference is continuously constant, the metadata necessary in the future can be determined based on the address differences. Therefore, it is expected that the metadata to be prefetched can be predicted based on the current access location. Furthermore, 2017.imagick and 2017.xz have other peaks except the peak at the position of 0 in address difference. This fact indicates the metadata exist the positions of these peaks have high possibilities to be used. Therefore, these metadata can be another target of the prefetch.

On the other hand, Fig. 3(b) shows the histograms of two benchmarks: 2006.perlbench and 2006.gcc. The address differences of them are not constant and tend to be widespread. In such applications, it is challenging to predict metadata by only the address differences. However, the behavior of the address differences will change in a particular time for those same benchmarks.

Here, the characteristics of the benchmarks are quantified to categorize the benchmarks as follows. At first, the histogram of each benchmark is converted to cumulative frequency distribution as shown in Fig. 4. The horizontal axis indicates the percentile, and the vertical axis shows the address difference. If the slope of the cumulative frequency

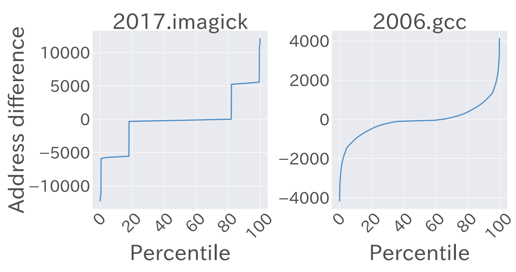


Fig. 4 The cumulative distribution.

Table 2 Constant group and Not-constant group.

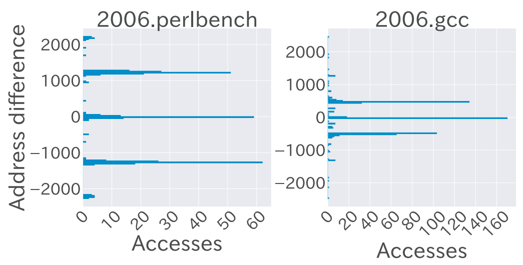
Group	Benchmark
Constant	2006.cactusADM (0.76), 2006.libquantum (0.92), 2006.sjeng (0.96), 2017.cam4 (0.98), 2017.deepsjeng (0.98), 2017.imagick (0.9), 2017.lbm (0.96)
Not-constant	2006.bzip2 (0.68), 2006.gcc (0.24), 2006.gobmk (0.66), 2017.perlbench (0.47)

distribution becomes smaller than 0.001, and the occupancy of such a small slope to the horizontal axis becomes 70% or more, the benchmark is categorized into the constant group. Otherwise, it is categorized to the the not-constant group.

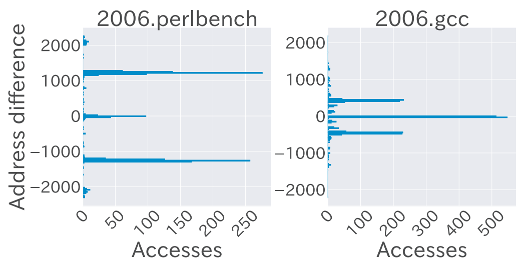
Since the four benchmarks show characteristics in two groups, the four benchmarks can be used to discuss the analysis of all of the benchmarks. Table 2 shows the results of the categorization. The value following each benchmark name is the occupation ratio of the part where the slope is smaller than 0.001. Since the constant group has clear peaks in the histogram, the prefetching of the metadata can be effective for the benchmarks in the constant group. On the other hand, peaks of the benchmark of the not-constant group are ambiguous. Therefore, the benchmarks in the not-constant group have difficulties to predict the metadata used in near future compared with those in the constant group.

Even in the case of the benchmarks in the not-constant group, there may be some phases in which the histogram shows the characteristics of the constant group. Therefore, address differences are sampled by some intervals. Figure 5 shows the address differences in the different time period for 2006.perlbench and 2006.gcc. From this figure, it is observed that the address difference becomes constant by focusing on a certain time period even in the not-constant group. Thus, it is considered that the memory controller can identify the necessary metadata by using the address differences when the controller continuously updates the address differences in a certain interval.

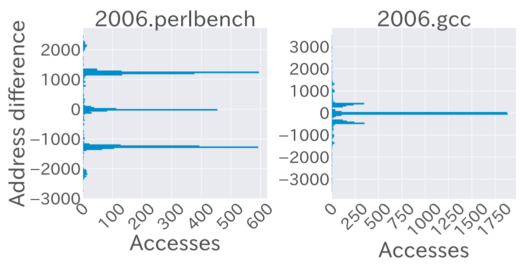
However, the histograms of all the intervals listed in Fig. 5 show the characteristics of the constant group. Therefore, the appropriate interval cannot be determined from these results. Discussions regarding the interval are also shown in the experimental conditions described later, in



(a) Address differences during a specific 10 microseconds.



(b) Address differences during a specific 40 microseconds.



(c) Address differences during a specific 100 microseconds.

Fig. 5 Address differences during each interval.

Sect. 6.1. From the experiments in Sect. 6.5, it is concluded that the interval to sample the histogram does not give an impact on the performance of the prefetching mechanism.

5. Metadata Prefetching Mechanism for HMAs

The discussions in Sect. 4 show a regularity in address differences, each of which is a difference between the address of an access and that of the subsequent access. Based on these discussions, this paper proposes a metadata prefetching mechanism based on the address difference. When an MMA miss occurs, the proposed metadata prefetching mechanism loads the target metadata around the currently-used metadata to the MMA. Furthermore, the proposed mechanism prefetches the metadata predicted to be used in the near future based on the address differences. For this purpose, the proposed mechanism monitors the access requests continually, calculates and records the address differences for each access, and finds the most observed address difference for prefetching.

There are mainly three types of HMAs as discussed in Sect. 3. On the other hand, this proposal is applicable if the metadata is originally stored in NM and/or FM, and cached in a dedicated metadata cache such as MMA. Therefore, the proposal can be applicable regardless of the types of HMAs.

To realize the proposed mechanism, the memory controller of the HMA is modified by adding the metadata prefetching mechanism to the metadata management logic. The metadata prefetching mechanism has a memory, which stores the address differences. After deciding which metadata to be prefetched by using the recorded address differences, the mechanism issues the metadata-prefetching requests.

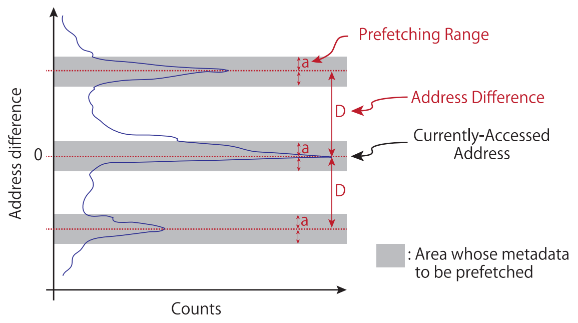

Prefetching Range
Address Difference
Currently-Accessed Address
a
D
0
Address difference
Counts
: Area whose metadata to be prefetched

Fig. 6 Overview of how the proposed mechanism prefetch the metadata by the address difference and the prefetching range.

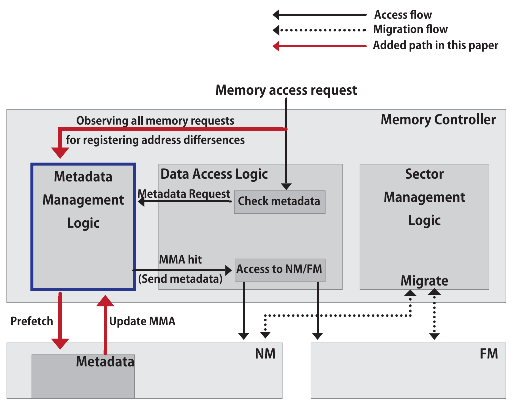
5.1 Prefetching Based on Address Difference

In the proposed mechanism, the metadata is prefetched on an MMA miss. Figure 6 shows the overview of the method to determine the metadata prefetching by the address differences. Since data in the HMA are managed on a sector-by-sector basis, the candidate metadata to be prefetched is also specified by the sector address. When an MMA miss occur at a place whose sector address is *Addr*, the proposed mechanism prefetches the metadata in ranges of $Addr \pm a$, $(Addr+D)\pm a$, and $(Addr-D)\pm a$. Here, D is the address difference, and a means the *prefetching range*. The prefetching range a specifies the number of neighboring sectors whose metadata should be prefetched.

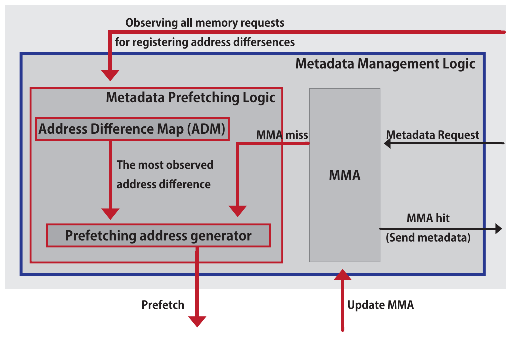
The proposed mechanism needs to determine two parameters D and a to prefetch the metadata expected to be accessed. The address difference D is highly depending on applications, as shown in Figs. 3 and 4. Hence, the proposed mechanism can identify an appropriate D in an online manner. The value of prefetching range a is set to a fixed value at runtime. Thus, a is decided should be carefully determined to avoid the polution of the MMA. If the number of prefetched metadata increases by enlarging a, the MMA hit is expected to be increased. On the other hand, when the amount of prefetching is significantly increased, the amount of useless prefetched metadata in the MMA may also increases, resulting in a situation where the useless metadata evict useful ones from the MMA. In the evaluation section, the performance of the proposed mechanism depending on a will be discussed based on the evaluation results.

5.2 Overview of the Proposed Mechanism

Figure 7(a) shows the overview of the proposed mechanism including the memory controller of the HMA and the



(a) Overview of the HMA with the metadata prefetching mechanism.



Observing all memory requests
for registering address differences
Metadata Management Logic
Metadata Prefetching Logic
Address Difference Map (ADM)
MMA miss
Metadata Request
The most observed
address difference
MMA
MMA hit
(Send metadata)
Prefetching address generator
Prefetch
Update MMA

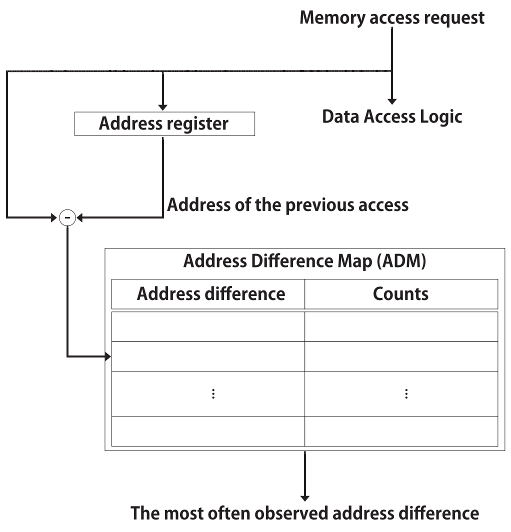
(b) Overview of the metadata prefetching mechanism

Fig. 7 The metadata prefetching mechanism in the HMA.

paths to access data through the memory controller. From Fig. 7(a), the memory controller of the HMA has three components. The first one is the data access logic that manages the requests from processor cores. The second one is the sector management logic that controls the allocation of each sector in the NM/FM. The third one is the metadata management logic that manages the metadata and performs the metadata prefetching. The first and the second components are inherited from the prior work [28]. The third component is a main contribution of the proposed mechanism.

The prior work, Hybrid^2, manages the HMA by using the data access logic and the sector management logic without the metadata prefetching. Hybrid^2 controls two cases: the case where an MMA hit occurs and the case where an MMA miss occurs to manage the HMA. When an MMA hit occurs, if there are data in the cache area of the NM, it accesses the NM. If the cache does not store the data, it accesses the FM. When the MMA miss occurs, the metadata in the NM is first referenced and stored in the MMA. If the data is in the flat address space of the NM, it can be accessed to the NM by simply updating the MMA. On the other hand, if the data exist in the flat address space of the FM, it is migrated between the infrequently accessed data in the NM and the requested data in the FM and then accessed to the NM.

Figure 7(b) shows the metadata prefetching flow in the metadata management logic. The memory controller monitors memory accesses from the CPU side and records an address difference between two consecutive accesses. The registered address differences are used for the metadata prefetching later. The memory controller performs the metadata prefetching only when an MMA miss occurs. If the memory controller recognizes an MMA miss, the memory controller confirms the address that caused the MMA miss. Using this address and the most often observed address difference from the *address difference map* (ADM), the memory controller decides some candidates for the metadata prefetching. Then, prefetching requests for these candidates are issued to the NM. After receiving the prefetched metadata, the memory controller updates the MMA to cache them. This metadata prefetching process is repeated during the execution of an application.



Memory access request
Address register
Data Access Logic
Address of the previous access
-
Address Difference Map (ADM)
Address difference
Counts
The most often observed address difference

Fig. 8 Address difference map and its related data paths.

5.3 Address Difference Map

To realize the metadata prefetching by using the address differences, the proposed mechanism must determine the address difference that is most frequently observed. Therefore, in the proposed mechanism, the memory controller has a memory called an ADM to record the address difference observed in each access.

Figure 8 shows the structure of the ADM and the data paths related to the ADM. When the memory controller receives a memory access request, the request goes to both the ADM and the data access logic. Then, the address of the request is extracted and stored in an address register to use the address at the next time. Furthermore, the address of the current request and the address that was stored in the address register at the previous access are used to calculate the address difference. The calculated result is sent to the ADM.

The ADM is the memory array to store the address differences and its counts observed in this mechanism. A single entry of the ADM includes a value of the address difference and its counts. Every time an address difference is calculated, the count of the corresponding entry is incremented one by one. In this way, the number of observations for each address difference is recorded. When the ADM becomes full, the entry of the address difference with the fewest observed counts is evicted.

The most-frequently-observed address difference is determined dynamically according to the execution status of the application. Since the address difference changes in a short time as discussed in Sect. 4, the aggregation of address differences is reset in a fixed and shorter interval compared with the execution time of the applications. However, the interval between resets is also crucial because a large number of resets results in a large number of predictions without sufficient aggregation. Therefore, an appropriate interval should be set carefully.

Note that the values of the address differences stored in the ADM are converted to the absolute values. In the preliminary experiments, the address difference values were a mixture of positive and negative values when the address differences were recorded. From these results, it is observed that the histograms of the address differences are almost symmetrical. Thus, it is enough that the address differences are recorded by only the absolute values rather than both the positive and negative values, which may double the number of entries in the ADM to assure the same monitoring performance.

6. Evaluations

6.1 Evaluation Methodology

For the evaluation of the proposed mechanism for the HMA, this paper developed a simulator of the proposed mechanism with the state-of-the-art HMA, Hybrid^2, on top of Gem5 simulator [29]. The sector size of the HMA, which is a unit of data migration between the NM and the FM, is set to 4 KB because the previous research has used 4 KB.

The parameters unique to the proposed mechanism are set as follows. The monitoring unit of address differences, ADM, has two parameters, the reset interval and the size. These parameters are preliminarily examined for this evaluation. The preliminary results show that these parameters do not give an impact on the performance. Therefore, the reset interval and the size of the ADM are set to every 100 microseconds and 32 entries to alleviate the excessive overhead. While the prefetch parameter D is automatically determined in the proposed mechanism, the prefetch range a is set to a fixed value. In this section, the prefetch range a is varied from 0 to 4 to examine the effect of changing the prefetch range.

Table 1 shows the other parameters that are used for this evaluation. These parameters are not changed from the preliminary evaluations in Sect. 4. As benchmarks, this paper uses the benchmarks with large memory footprints from the SPEC CPU 2006 [31] and 2017 [32] suites. Each benchmark is executed by the first one billion instructions for warm-up and the following one billion instructions for performance measurements.

6.2 Hardware Overhead Related to the Address Difference Map

There are limitations to the amounts of hardware resources to record the address difference in the ADM because the ADM is a newly added bit-storage component from the baseline HMA, Hybrid^2. Therefore, this section analyzes the hardware overhead of the ADM.

From the evaluation methodology shown in Sect. 6.1, the ADM size is 32. Each ADM entry includes an address difference and a sampled count. The number of bits of the address difference should be smaller than that of the original address. Therefore, the address differences requires 52 bits, which comes from the physical address range of the major industrial microprocessors [33]. In addition, a sampled count does not exceed the number represented by 18 bits because the ADM is reset every 100 microseconds in a 3 GHz processor. Therefore, a single entry of the ADM needs 70 bits, and the ADM totally needs 2240 bits. This bit-storage cost corresponds to 0.05% of the 512 KB MMA. From these observations, the proposed metadata prefetching mechanism has less impact on the hardware cost of the HMA and the microprocessor.

6.3 Experimental Results and Discussion

6.3.1 Effects of the Proposed Mechanism

Figure 9 shows Instructions Per Cycle (IPC) of evaluated three configurations. *No_prefetching* is the baseline Hybrid^2, *Prefetching* prefetches the metadata based on the prefetching only the metadata at the address ±1 from the current access position, and *Proposal* is the proposed mechanism that prefetches the metadata based on both the address differences and the prefetching range a. The a is set to 4. The vertical axis shows IPC normalized by that of the *No_prefetching*, and the horizontal axis shows the benchmarks.

Figure 9 shows that the constant group with prefetching

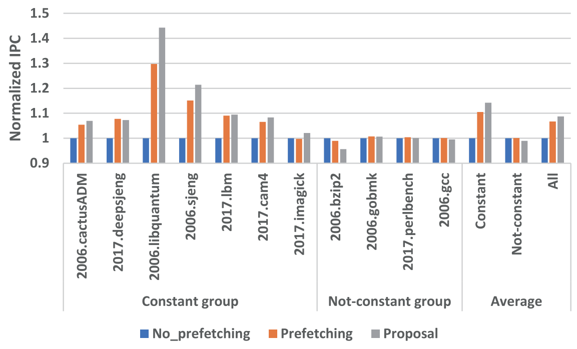


Fig. 9 Evaluation results of the metadata prefetching mechanism.

only around the current accessed address increases the IPC by 10% on average compared with the baseline Hybrid^2. The proposed mechanism can increase the IPC by 14% on average compared with the baseline. In particular, the IPC of *2006.libquantum* is improved by 44%. The reason of the most improvement is because of the most significant reduction in the round trip time of memory requests. Therefore, it is observed that the proposed metadata prefetching mechanism based on the address difference can improve the performance of the benchmarks in the constant group.

On the other hand, *Prefetching* slightly increases IPC in the case of the not-constant group by 0.1% on average, while the proposed mechanism slightly decreases the IPC by 1% on average. In particular, 2006.bzip2 significantly degrades the performance, despite having the same MMA miss rate as 2006.gobmk. This is because the number of MMA misses in 2006.bzip2 is 40 times larger than that in 2006.gobmk, and thus 2006.bzip2 has a more significant impact on the same MMA. There is no much performance difference among the three cases in the not-constant group. In these cases, the effects of the prefetching are still more significant than the disadvantage of the proposal for the not-constant group.

To discuss the evaluation results in more details, Fig. 10 shows the MMA misses per kilo instructions (MMA-MPKI). Each MMA-MPKI is normalized to that of the *No_prefetching*. From Fig. 10, each group of three bars represents the performances of *No_prefetching*, *Prefetching* and *Proposal* from left to right. Figure 10 indicates that, in the constant group, the MMA-MPKI of the *Prefetching* decreases by 37% on average compared with the baseline Hybrid^2. Furthermore, the MMA-MPKI of the proposal decreases by 47% on average compared with the baseline Hybrid^2. Therefore, the proposed metadata prefetching mechanism can successfully reduce the number of the MMA misses, which results in the performance improvements.

On the other hand, *Prefetching* increases the MMA-MPKI in the not-constant group by 8% on average, and the proposed mechanism increases the MMA-MPKI by 32% on average. The reason why the proposed mechanism increases the MMA-MPKI more than the *Prefetching* is the proposed mechanism prefetches many useless metadata by the address differences compared with the *Prefetching*. Consequently, the metadata prefetching by using the address difference degrades the number of MMA hits in the not-constant group.

6.3.2 Prefetching Range

The prefetching range *a* is an important parameter to decide the number of prefetched metadata and affects the performance. Therefore, this subsection evaluates how the performance changes depending on the prefetching range.

Figure 11 shows the evaluation results of the IPC when the prefetching range is varied as 0, 1, 2, and 4. The vertical axis shows the IPC normalized by that of $a = 0$. Figure 11 indicates that the constant group increases the IPC as the prefetching range is increased from 0 to 4. Especially,

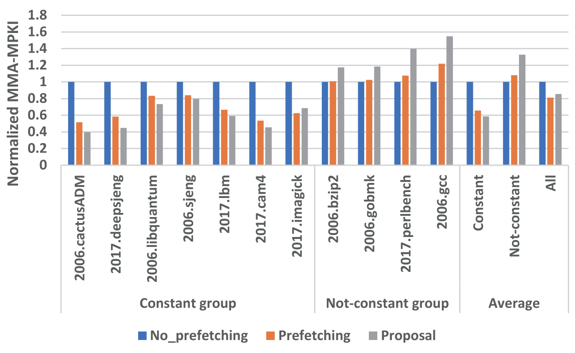


Fig. 10 Evaluation results of MMA-MPKI for the metadata prefetching mechanism.

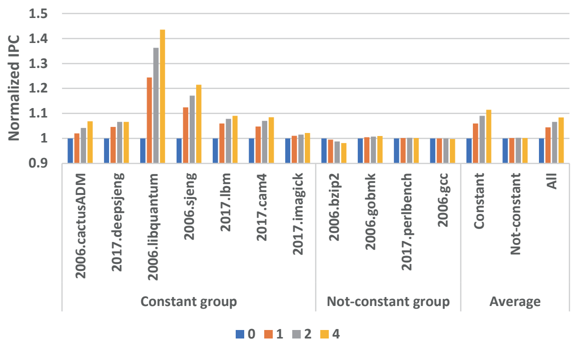


Fig. 11 Evaluation results of IPC depending on the prefetch range.

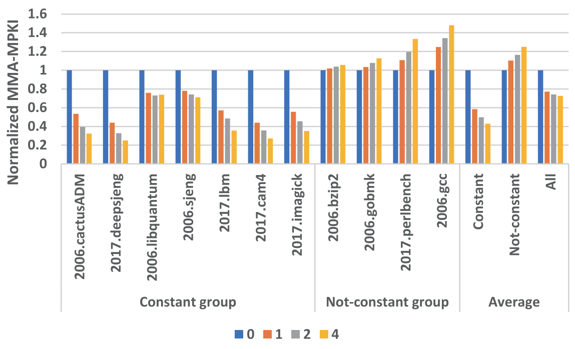


Fig. 12 Evaluation results of MMA-MPKI depending on the prefetching range.

the case of the prefetching range 4 increases the IPC by 11% on average. Among the constant groups, *2006.libquantum* shows the most significant improvement of the IPC. On the other hand, the not-constant group does not change the IPC even as the prefetching range is increased. Therefore, increasing the prefetch range has less impact on the not-constant group.

To discuss the evaluation results in more details, Fig. 12 shows the MMA-MPKI when the prefetching range is varied from 0 to 4. Each MMA-MPKI is normalized by that of the range 0. From Fig. 12, each group of four bars represents the performance of the range 0, 1, 2, and 4 from left to right. Figure 12 indicates that, in the constant group, the MMA-MPKI decreases as the prefetching range increases from 0 to 4. In the case of the prefetching range is 4, the MMA-MPKI of the constant-group decreases by 57% on average. Therefore, the proposed metadata prefetching mechanism can successfully reduce the number of the MMA misses in applications categorized into the constant group, which results in their performance improvements. Furthermore, even in *2006.libquantum* of the constant group, the MMA-MPKI also slightly increases when the prefetching range becomes large.

On the other hand, the not-constant group increases the MMA-MPKI as the prefetching range is increased, and the case of the prefetching range 4 increases the MMA-MPKI by 25% on average. This is because, for the not-constant group, the proposed metadata prefetching mechanism just increases the number of useless metadata coming into the MMA. This fact indicates that the performance degradation of increasing the prefetching range is slight while the MMA-MPKI increases.

6.4 Effects of the Metadata Prefetching Depending on the MMA Size

Figure 13 shows the evaluation results of IPC depending on the sizes of the MMA. The vertical axis means the IPC normalized by that without prefetching for each MMA size. Figure 13 shows that, in the constant group, the proposed metadata prefetching mechanism improves the IPC for smaller MMA sizes up to 512 KB. This is because the metadata prefetching can compensate for the performance loss even when the MMA size is small. On the other hand, in the cases where the MMA size is 1 MB or more, the performance improvement by the proposed mechanism is limited. This is because the MMA-MPKI is already low without the prefetching mechanism in these cases. From these observations, the proposed metadata prefetching mechanism can improve the performance especially when the amount of hardware resources for the MMA is limited.

6.5 Effects of Varying the Reset Interval of the ADM

The Fig. 14 shows the change in the average IPC when the interval is varied. From Fig. 14, we can see almost no change in IPC when the interval is varied from 10 microseconds to 100 microseconds. Therefore, it is clear that the size of the interval does not have a significant effect on the performance.

6.6 Effects of the Metadata Prefetching with Multi Cores

Figure 15 shows the evaluation results on multi-core processors. In this experiment, two benchmarks are chosen from the eleven benchmark to run together and show the average performance improvement for each of the three types of pairs: the constant/constant pairs, the constant/not-constant pairs, and the not-constant/not-constant pairs. From Fig. 15, the constant/constant pairs achieves the highest performance improvement by up to 8% and 1.8% on average. The

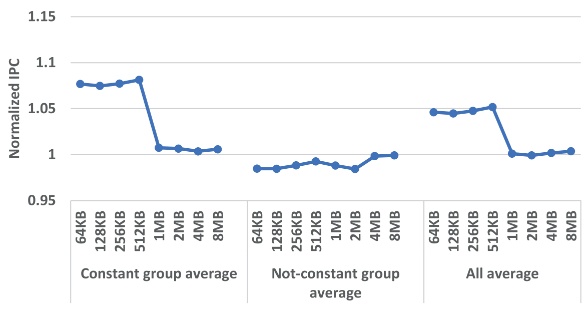


Fig. 13 Evaluation results of IPC depending on the MMA size.

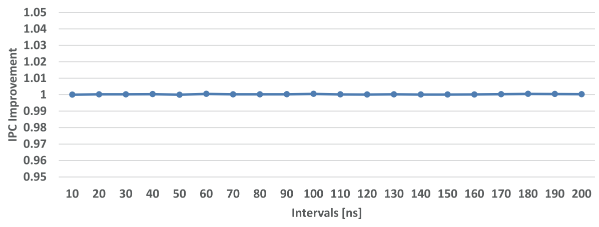


Fig. 14 Evaluation results of changing the ADM reset interval.

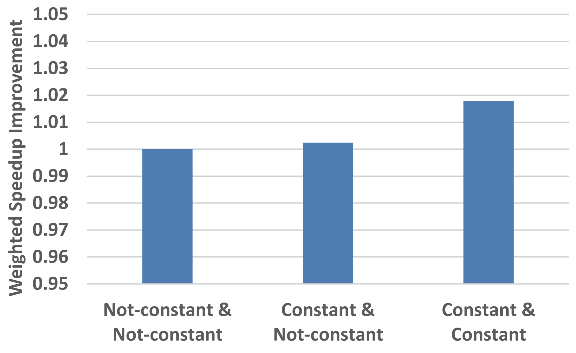


Fig. 15 Wasted speedup improvement of each combinations.

second best performance improvement is observed for the constant/no-constant pairs, with an improvement by up to 4.9% and 0.2% on average. The not-constant/not-constant pairs slightly improve the performance by up to 0.14% and 0.004% on average. Therefore, the proposed mechanism can be effective for the multi-core processors.

7. Conclusions

HMAs are expected to realize the memory system with good balances between high performance and large capacity. On the other hand, since the data in an HMA are managed by metadata, the access latency to the metadata causes the overhead to access the actual target data. While the state-of-the-art HMA caches the metadata in the memory controller, it will become difficult to hide the access latency in the future because the metadata increase as the capacity of an HMA increases. To tackle this problem, this paper proposes a new metadata prefetching mechanism for HMAs, which aims at reducing the long latency to access the target data in an HMA. The proposed mechanism prefetches metadata based on the address differences between two consecutive access addresses. The evaluation results indicate that the proposed metadata prefetching mechanism can improve the performance of the system with Hybrid^2, which is the state-of-the-art HMA management mechanism, by up to 44% and 8.7% on average. As future work, the prefetching mechanism will be further evaluated from the viewpoint of the energy consumption.

Acknowledgements

This work is partially supported by MEXT Next Generation High-Performance Computing Infrastructures and Applications R&D Program, entitled "R&D of A Quantum-Annealing-Assisted Next Generation HPC Infrastructure and its Applications" and JSPS KAKENHI Grand Number JP19K20232.

References

[1] H.S.P. Wong, S. Raoux, S. Kim, J. Liang, J.P. Reifenberg, B. Rajendran, M. Asheghi, and K.E. Goodson, "Phase change memory," Proc. IEEE, vol.98, no.12, pp.2201–2227, Dec. 2010.

[2] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," 2009 46th ACM/IEEE Design Automation Conference, pp.664–669, IEEE, July 2009.

[3] X. Dong, Y. Xie, N. Muralimanohar, and N.P. Jouppi, "Simple but effective heterogeneous main memory with on-chip memory controller support," SC'10: Proc. 2010 ACM/IEEE Int. Conf. for High Performance Computing, Networking, Storage and Analysis, pp.1–11, IEEE, 2010.

[4] L.E. Ramos, E. Gorbatov, and R. Bianchini, "Page placement in hybrid memory systems," Proc. Int. Conf. Supercomputing, pp.85–95, May 2011.

[5] J. Sim, A.R. Alameldeen, Z. Chishti, C. Wilkerson, and H. Kim, "Transparent hardware management of stacked dram as part of memory," 2014 47th Annual IEEE/ACM Int. Symp. Microarchitecture, pp.13–24, IEEE, 2014.

[6] A. Sodani, R. Gramunt, J. Corbal, H.S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y.C. Liu, "Knights landing: Second-generation intel xeon phi product," IEEE Micro, vol.36, no.2, pp.34–46, March–April 2016.

[7] V. Mironov, I. Chernykh, I. Kulikov, A. Moskovsky, E. Epifanovsky, and A. Kudryavtsev, "Performance evaluation of the intel optane dc memory with scientific benchmarks," 2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC), pp.1–6, 2019.

[8] G.H. Loh, Y. Xie, and B. Black, "Processor design in 3D die-stacking technologies," IEEE Micro, vol.27, no.3, pp.31–48, May–June 2007.

[9] M.K. Qureshi and G.H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," 2012 45th Annual IEEE/ACM Int. Symp. Microarchitecture, pp.235–246, IEEE, 2012.

[10] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: Hit ratio, latency, or bandwidth? have it all with footprint cache," ACM SIGARCH Computer Architecture News, vol.41, no.3, pp.404–415, June 2013.

[11] D. Jevdjic, G.H. Loh, C. Kaynak, and B. Falsafi, "Unison cache: A scalable and effective die-stacked DRAM cache," 2014 47th Annual IEEE/ACM Int. Symp. Microarchitecture, pp.25–37, IEEE, 2014.

[12] Y. Lee, J. Kim, H. Jang, H. Yang, J. Kim, J. Jeong, and J.W. Lee, "A fully associative, tagless dram cache," ACM SIGARCH Computer Architecture News, vol.43, no.3S, pp.211–222, June 2015.

[13] X. Yu, C.J. Hughes, N. Satish, O. Mutlu, and S. Devadas, "Banshee: Bandwidth-efficient dram caching via software/hardware cooperation," 2017 50th Annual IEEE/ACM Int. Symp. Microarchitecture (MICRO), pp.1–14, IEEE, Oct. 2017.

[14] N. Gulur, M. Mehendale, R. Manikantan, and R. Govindarajan, "Bimodal dram cache: Improving hit rate, hit latency and bandwidth," 2014 47th Annual IEEE/ACM Int. Symp. Microarchitecture, pp.38–50, IEEE, 2014.

[15] S. Franey and M. Lipasti, "Tag tables," 2015 IEEE 21st international symposium on high performance computer architecture (hpca), pp.514–525, IEEE, 2015.

[16] H. Liu, Y. Chen, X. Liao, H. Jin, B. He, L. Zheng, and R. Guo, "Hardware/software cooperative caching for hybrid dram/nvm memory architectures," Proc. Int. Conf. Supercomputing, pp.1–10, June 2017.

[17] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, "Enabling efficient and scalable hybrid memories using fine-granularity dram cache management," IEEE Comput. Archit. Lett., vol.11, no.2, pp.61–64, July–Dec. 2012.

[18] H. Jang, Y. Lee, J. Kim, Y. Kim, J. Kim, J. Jeong, and J.W. Lee, "Efficient footprint caching for tagless dram caches," 2016 IEEE Int. Symp. High Performance Computer Architecture (HPCA), pp.237–248, IEEE, 2016.

[19] C. Chou, A. Jaleel, and M.K. Qureshi, "Bear: Techniques for mitigating bandwidth bloat in gigascale dram caches," ACM SIGARCH Computer Architecture News, vol.43, no.3S, pp.198–210, June 2015.

[20] M.R. Meswani, S. Blagodurov, D. Roberts, J. Slice, M. Ignatowski, and G.H. Loh, "Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories," 2015 IEEE 21st Int. Symp. High Performance Computer Architecture (HPCA), pp.126–136, IEEE, 2015.

[21] A. Prodromou, M. Meswani, N. Jayasena, G. Loh, and D.M. Tullsen, "Mempod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories," 2017 IEEE Int. Symp. High Performance Computer Architecture (HPCA), pp.433–444, IEEE, 2017.

[22] J.B. Kotra, H. Zhang, A.R. Alameldeen, C. Wilkerson, and M.T. Kandemir, "Chameleon: A dynamically reconfigurable heterogeneous memory system," 2018 51st Annual IEEE/ACM Int. Symp. Microarchitecture (MICRO), pp.533–545, IEEE, 2018.

[23] C.C. Chou, A. Jaleel, and M.K. Qureshi, "Cameo: A two-level memory organization with capacity of main memory and flexibility of hardware-managed cache," Proc. Micro, pp.1–12, 2014.

[24] J.H. Ryoo, M.R. Meswani, A. Prodromou, and L.K. John, "Silc-fm: Subblocked interleaved cache-like flat memory organization," 2017 IEEE Int. Symp. High Performance Computer Architecture (HPCA), pp.349–360, IEEE, 2017.

[25] S. Lee, H. Bahn, and S.H. Noh, "Clock-dwf: A write-history-aware page replacement algorithm for hybrid pcm and dram memory architectures," IEEE Trans. Comput., vol.63, no.9, pp.2187–2200, Sept. 2013.

[26] R. Salkhordeh and H. Asadi, "An operating system level data migration scheme in hybrid dram-nvm memory architecture," 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp.936–941, IEEE, 2016.

[27] A. Kokolis, D. Skarlatos, and J. Torrellas, "Pageseer: Using page walks to trigger page swaps in hybrid memory systems," 2019 IEEE Int. Symp. High Performance Computer Architecture (HPCA), pp.596–608, IEEE, 2019.

[28] E. Vasilakis, V. Papaefstathiou, P. Trancoso, and I. Sourdis, "Hybrid2: Combining caching and migration in hybrid memory systems," 2020 IEEE Int. Symp. High Performance Computer Architecture (HPCA), pp.649–662, IEEE, 2020.

[29] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M.D. Hill, D.A. Wood, "The gem5

simulator," ACM SIGARCH computer architecture news, vol.39, no.2, pp.1–7, May 2011.

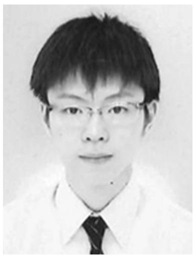
[30] S. Rashidi, M. Jalili, and H. Sarbazi-Azad, "A survey on pcm lifetime enhancement schemes," ACM Comput. Surv. (CSUR), vol.52, no.4, pp.1–38, July 2019.

[31] J.L. Henning, "Spec cpu2006 benchmark descriptions," ACM SIGARCH Computer Architecture News, vol.34, no.4, pp.1–17, Sept. 2006.

[32] R. Panda, S. Song, J. Dean, and L.K. John, "Wait of a decade: Did spec cpu 2017 broaden the performance horizon?," 2018 IEEE Int. Symp. High Performance Computer Architecture (HPCA), pp.271–282, IEEE, 2018.

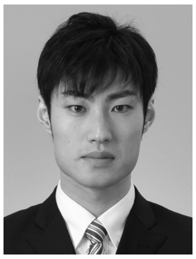
[33] Advanced Micro Devices, AMD64 Architecture Programmer's Manual Volume 2: System Programming, March 2021.

Shunsuke Tsukada received the B.E. degree from Tohoku University in 2020. He is currently a master course student of Graduate School of Information Sciences, Tohoku University.



Hikaru Takayashiki received the B.E. degree and the Master Degree of information sciences from Tohoku University in 2018 and 2020, respectively. He is currently a doctor course student of Graduate School of Information Sciences, Tohoku University.

Masayuki Sato received the B.E. degree from Tohoku University in 2007. He also received the M.S. and Ph.D. degrees of information sciences from Tohoku University in 2009 and 2012, respectively. He was an assistant professor in Graduate School of Information Sciences, Tohoku University, from April 2016 to December 2019. He is currently an associate professor since January 2020. His research interests include high-performance and low-power computer architectures and its applications.



Kazuhiko Komatsu is an Associate Professor at Cyberscience Center, Tohoku University. His research interests include high performance computing. He received the B.E. Degree in Mechanical Engineering and the M.S. and Ph.D. Degrees in Information Sciences from Tohoku University in 2002, 2004, and 2008, respectively.

Hiroaki Kobayashi is Professor in Graduate School of Information Sciences, Special Adviser to President for ICT innovation, and Special Adviser to Director of Cyberscience Center, Tohoku University. In 1995, 1997–1998 and 2001–2002, he was Visiting Associate Professor of Stanford University. His research interests include high-performance computer architectures, supercomputer systems, and their applications. He received the B.E. Degree in Communication Engineering, and the M.E. and D.E. Degrees in Information Engineering from Tohoku University. He is a senior member of IEEE CS, and a member of ACM, IEICE and IPSJ. He is also an associate member of Science Council of Japan. He received 2017 Minister of Education Award in the Field of Computer Science and 2018 Minister of Education Award in the field of Science and Engineering.