Thermal-aware Dynamic Page Allocation Policy by Future Access Patterns for Hybrid Memory Cube (HMC)

Wei-Hen Lo, Kai-zen Liang, TingTing Hwang
Department of Computer Science, National Tsing Hua University, R.O.C
turtleevil_1@hotmail.com, tingting@cs.nthu.edu.tw

ABSTRACT

The Hybrid Memory Cube (HMC) is a promising solution to overcome memory wall by stacking DRAM chips on top of a logic die and connecting them with dense and fast Through Silicon Vias (TSVs). However, 3D stacking technique brings another problem: high temperature and temperature variations between the DRAM dies. The thermal problem may lead to chip failure of 3D stacked DRAMs since the temperature may exceed the maximum operating temperature. Dynamic thermal management (DTM) scheme such as bandwidth throttling can effectively decrease the temperature. However, it results in the loss of the performance. To maximize the performance of the system with HMC, the appropriate memory mapping should consider the thermal characteristics of HMC, memory interference and bandwidth variations among processes, and current temperature conditions of each memory channel. This paper proposes a thermal-aware dynamic OS page allocation using future access pattern to find a best performance-oriented setting of the above factors. An analytical model has been proposed to estimate the system performance considering the memory interference, the bandwidth variation, and the throttling impact. Our method can improve the system performance by 12.7% compared to best performance-oriented allocation method (MCP) [1]. The average error rate of our analytical model to predict the trend of performance variations is only 0.86%.

I. INTRODUCTION

Main memory performance becomes a bottleneck in system performance, due to memory wall problem. To solve this problem, three-dimensional integrated circuits (3D ICs) technology may be used where multiple DRAM dies are stacked and linked by Through-Silicon Vias (TSVs). 3D-ICs provide great bandwidth benefit and energy reduction because of shorter vertical interconnect. Recently, a heterogenous 3D-stacking DRAM package named Hybrid Memory Cube (HMC) , implemented by Micron [5], has come to the market. It contains several DRAM dies stacked on top of a CMOS logic layer. The logic layer at the base of each stack contains several DRAM memory controllers with host processors over high speed serial links using an abstracted packet interface. Each memory controller is connected to a vault (memory channel) which contains several memory ranks. The TSV connections combined with the usage of multiple memory controllers near the memory arrays form a device that exposes significant memory-level parallelism and is capable of providing bandwidth an order of magnitude more than current DDRx solutions do.

In spite of the bandwidth benefit and the power reduction of 3D memory, higher memory bandwidth usage and increased power density make heat dissipation a problem. The increased temperature may cause the DRAM chips to exceed their maximum operation temperatures (85 °C) and result in chip damage during run time. In order to solve the thermal problem of memory, many thermal aware studies have been proposed [6]–[14]. They can be classified into two categories: dynamic and static thermal managements. The dynamic thermal management detects the thermal conditions during run time and stops the hot units from operating until their temperatures drop down. Throttling [6]–[8], dynamic voltage frequency scaling [9] belong to this category. Dynamic thermal management methods can precisely monitor temperature variation and guarantee that the system temperature will never be higher than a predefined constraint, though, at the cost of slowdown of the processor execution. As for the static thermal management, the profiling data of programs is generated first. Then, the data is used to analyze the temperature distribution during run time. For example, Sankaranarayanan et al. [10] have proposed a floorplan technique which distributes hot functional units uniformly and surrounds them with cold functional units to reduce hotspot temperature. However, this technique is less flexible because the same floorplan is used for all applications. Once the behavior of some workload is different from the profiling history, this method may not work well. Another example is performed at compiler-level where Mutyam et al. [11] proposed a static thermal management which distributes computations to different functional units so that the hotspots are prevented.

The above techniques are mainly targeting on 2D memory. Recently, there are also researches addressing the thermal and energy problems of 3D memory. Tsai et al. [12] discuss the energy and delay saving of 3D cache memory. Although their method is suitable for custom cache design, it cannot be applied to 3D-stacking DRAM memory. Khurshid et al. [13] proposed a data compression/decompression method to compress the data in hot regions (lower layers of HMC). Since there are lesser bursts of read/write compressed cache blocks, the power consumption is reduced leading to slower temperature rise. However, this method did not consider the memory mapping issue. Mapping frequently accessed blocks to cooler layers can further improve the efficiency of this method. Hsieh et al. proposed a memory mapping algorithm for 3D memory [14]. They design a new hardware to remap memory banks so that each memory request will access different vertical location in different tier of 3D memory. Therefore, the temperature rise will be alleviated since the same vertical region will not be accessed at the same time. Nevertheless, this approach is hard to apply to HMC because the ranks or the banks in their work are placed vertically while in HMC, the memory ranks and the memory banks are placed horizontally in each tier. Assigning multiple memory requests to different memory banks may still result in accessing the same region stacking vertically. Hence, to solve the memory mapping problem in HMC, we need a new approach to consider the performance and thermal relation between multiple memory requests at page-level instead of block-level.

To derive an appropriate memory mapping during run time, a thermal-aware memory allocator must consider the power behavior of each page and the physical location of HMC, that is, to map those frequently accessed pages to the physical DRAM dies with higher heat dissipation ability. However, simply allocating pages according to their access frequencies may lead to severe memory interference among different processes, because memory requests from different processes accessing the same memory channel may contend with each other. This will cause performance degradation. Moreover, the bandwidth assigned to each process may affect the IPC of that process. Hence, it is very challenging to find the balance between memory interference, bandwidth variation, and temperature rise.

This paper brings the following novel contributions:

1) To the best of our knowledge, this is the first paper targeting thermal-aware memory mapping problem of HMC. We propose a thermal and access patterns aware OS page allocation method to effectively improve the overall system performance.
2) Since it is hard to find the balance among memory interference, bandwidth variations and throttling impact, an analytical model is developed to estimate the system performance.

The rest of this paper is organized as follows. In Section II, we will show our motivation by demonstrating the performance difference between different dynamic OS page allocation policies. Next, in Section III, the details of our thermal and access patterns aware OS page allocation policy and the off-line analytical learning model are proposed. Section IV presents the experimental results of our approach. Finally, the conclusions are given in Section V.

II. MOTIVATION

We first review the thermal characteristics of Hybrid Memory Cube (HMC) [1] in Section II-A. To maintain high reliability, dynamic bandwidth

978-3-9815370-6-2/DATE16/

throttling is one effective way to prevent memory damaged by extremely high temperature. However, this scheme may hurt system performance because of the loss of memory bandwidth. Considering the performance degradation caused by the thermal throttling of Hybrid Memory Cube, we demonstrate how to allocate memory properly to avoid thermal throttling during run time and improve system performance in Section II-B.

A. Thermal Characteristics of Hybrid Memory Cube

The Hybrid Memory Cube (HMC) is a heterogenous 3D stack of DRAM dies on top of a logic die. Each layer of DRAM dies are of size *1* Gb, divided into *16* partitions. Each partition contains *2* memory banks. A vertical stack of partitions is called a vault, which is a memory channel. Through Silicon Vias (TSVs) are used to connect control and data signals between different DRAM dies. Each DRAM die has different heat dissipation abilities. For example, the bottom tiers have worse heat dissipation abilities than top tiers if the heat sink is placed on the top of HMC. Hence, the temperature of bottom layers is often very high. Figure 1 shows an overall structure and the thermal status of the Hybrid Memory Cube (HMC). As we can see, the higher dies are cooler than the lower dies since it is more difficult to dissipate the power of physical blocks in the lower dies. Therefore, those pages which are accessed more frequently should be allocated in top tiers. Wrong page allocation policy may lead to temperature rise and the chip may eventually exceed its maximum operating temperature and enter throttling state.

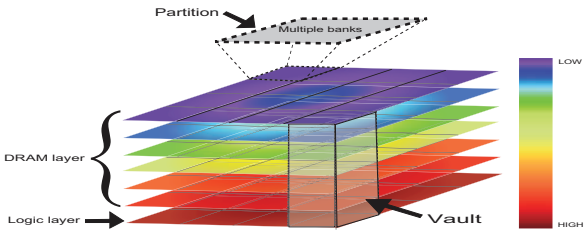

Fig. 1. HMC structure and its temperature distribution under maximum power dissipation conditions

B. Thermal and Access Patterns Aware Dynamic OS Page Allocation

Since the Hybrid Memory Cube (HMC) stacks multiple DRAM dies vertically, its thermal impact is critical for the performance and the reliability of system. To prevent the maximum temperature from reaching a critical level, the simplest way is to throttle memory traffic. Nevertheless, throttling may reduce the total bandwidth and exacerbate the memory wall problem.

To prevent the performance degradation under thermal throttling, the number of throttling commands executed during run time on a Hybrid Memory Cube (HMC) must be reduced. To achieve this goal, we will propose a thermal aware dynamic OS page allocation policy. An intuitive method works as follows: when OS decides to allocate a page to a memory channel (or called a vault) whose temperature exceeds a pre-defined threshold, it reallocates this page to another coolest memory channel instead. That is, the *current* temperatures of channels are utilized to allocate pages. Since the working pages now are distributed more evenly between memory channels, the speed of temperature rise in memory channels will be alleviated. However, one important characteristic of HMC, that is, the heat dissipation ability of top tiers being better, is not utilized.

Based on the characteristics of 3D structure of HMC, the hot pages or hot memory segments need to be allocated to the top layers of DRAM dies since they have better heat dissipation. However, the above straightforward method is not able to predict whether a page is hot or cold during run time. To solve this problem, we propose a prediction method by utilizing future access patterns of each memory segment for allocation. By profiling the memory access history from the trace files, the access frequency of each page or memory segments can be collected [3]. Then, according to the future access patterns of pages and memory segments, the OS can allocate hotter pages or memory segments that have been predicted to the top tiers of DRAM dies more precisely.

Figure 8 shows a comparison of temperature variations among the channel partition based page allocation method [1] denoted as *MCP*, the straightforward dynamic page allocation method denoted as *nofuture*, and the thermal and access patterns aware dynamic page allocation method denoted as *future*. The channel partitioning based method [1] (also named as MCP) is a performance-driven memory allocation method for 2D memory. It can effectively reduce the memory interference among processes and achieve high system performance. However, allocating pages of a memory intensive application to few channels may speed up the temperature rise of those channels. As a result, the maximum temperature of the MCP rises very fast and finally stays around *85* to *90* °C, which already exceeds the maximum operating temperature *85* °C. On the other hand, the second dynamic page allocation method with no future information keeps reallocating some pages to cooler channels when their original target channels exceed a static temperature threshold close to the throttling temperature threshold. It can effectively reduce the maximum temperature to averagely *80* °C, but the maximum temperature of memory channels still exceeds the throttling threshold (*75* to *80* °C). Finally, the last method, the thermal and access patterns aware dynamic page allocation method with future information successfully reduces the maximum temperature below the lowest throttling threshold *75* °C during most of the execution time. This means the dynamic page allocation method with future information can effectively reduce the probability of throttling and improve the system performance.

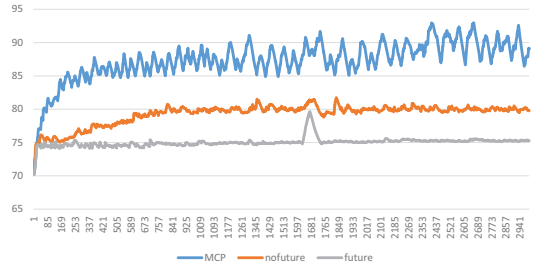

Fig. 2. Temperature variations among three different page allocation policies

III. Methodology

In this section, we first review the previous work channel partition based page allocation method [1] in Section III-A. Next, the details of our thermal and access patterns aware dynamic OS page allocation will be introduced in Section III-B. Finally, in Section III-C, we will introduce the performance trade off between the memory interference, the memory bandwidth variation and the thermal throttling of HMC. Then, a performance prediction model will be introduced to decide the optimal pre-defined temperature threshold for data reallocation.

A. *Memory Channel Partitioning*

In multi-cores system, memory interference among the processes can greatly degrade the system performance. To reduce the memory interference among all the applications, the channel partition based page allocation method (also named as MCP) [1] was proposed. Figure 3 shows the overall flow of channel partitioning based method. OS will assign some new preferred memory channels to each application at the end of each execution interval. MCP categorizes applications into two groups based on their miss per kilo instructions (MPKI), which are memory intensive group and memory non-intensive group. Next, it will further categorize the memory intensive applications based on their row buffer hit rate (RBH) values into low and high row buffer hit rate groups. Then, the memory channels are first partitioned between the memory intensive and memory non-intensive groups. Their algorithm partitions the memory channels based on the number of applications in that group to improve performance by memory isolation. Finally, MCP determines which applications within a group are mapped to which channels, when more than one channel is allocated to a group. For each group, MCP prioritizes applications according to memory-intensity. That is, MCP starts from the least intensive application in the group and maps the applications to the first allocated channel of the group till the bandwidth demand is $\frac{Sum\ of\ MPKIs\ of\ applications\ in\ the\ group}{Number\ of\ channels\ allocated\ to\ the\ group}$. Then, MCP moves on to the next channel and allocate applications to it. This procedure is repeated for every group till all applications are assigned some preferred channels. MCP [1] is a very efficient method to reduce memory interference and outperforms all other

OS page allocation methods for multi-cores system. It is the state-of-the-art OS page allocation policy.

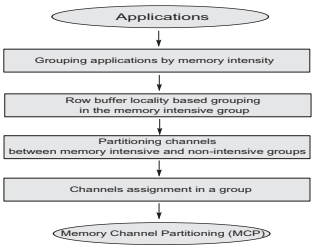

Fig. 3. Overall flow of channel partition based method [1]

B. Our Thermal and Access Patterns Aware Dynamic OS Page Allocation Method

Although MCP [1] can reduce memory interference and get high system performance, it results in serious bandwidth throttling in HMC. Hence, we propose our thermal and access patterns aware dynamic OS page allocation method to reduce the number of thermal throttling commands during execution time. In this section, we first describe how the most intuitive dynamic OS page allocation method works. Then, we show how to extract the future access frequency as a feature for prediction of thermal effect. After the future access patterns are collected, our thermal and access patterns aware dynamic OS page allocation method will utilize the temperature of the memory channels and the future access patterns to reallocate the pages to new preferred channels. We will show the detailed flow of our thermal and access patterns aware dynamic OS page allocation method in the following paragraphs.

We introduce our thermal aware dynamic OS page allocation policy. The method works as follows. First, the default OS page allocation policy is MCP [1]. This allocation will result in the least memory interference among processes and the best performance. Then, to consider thermal effect, when the temperatures of a hot channel reaches a temperature threshold T_t, OS starts to change its allocation policy. Assume the page P_A is waiting for allocating and its preferred memory channel assigned by MCP [1] is the hottest memory channel C_{MCP}. If the temperature of C_{MCP} exceeds the pre-defined temperature threshold T_t, OS re-assigns the coolest memory channel C_{cold} at that moment as the new preferred channel C_{new} of page P_A. Note that, if the temperature of C_{MCP} does not exceed the threshold T_t, OS will adopt the best performance-oriented decision made by MCP [1] as shown in Figure 4.

Algorithm 1 : Thermal and Access Patterns Aware Dynamic OS Page Allocation Method

Inputs : The page P_A which is waiting for allocating and its preferred memory channel assigned by MCP is the hottest memory channel C_{MCP}; The preferred bank $B_{FutureFreq}$ of P_A; Current coolest memory channel C_{cold}; A pre-defined temperature threshold T_t

Outputs : New preferred memory bank B_{new} of P_A and new preferred memory channel C_{new} of P_A

```
1  : if (The temperature of C_MCP > T_t) then
2  :     if (The capacity of memory channel C_cold is not enough for P_A) then
3  :         Find another coolest channel C_another and redo this algorithm;
4  :     end if
5  :     C_new = C_cold;
6  :     B_new = ExtractPreferredBank(P_A, C_cold);
7  : else
8  :     C_new = C_MCP;
9  :     B_new = B_FutureFreq;
10 : end if
11 : return C_new and B_new;
```

Fig. 4. Our algorithm of thermal and access patterns aware OS page allocation method

However, as shown in Section II, we can further reduce the temperature rise of HMC if the thermal characteristics of HMC and the future access patterns of applications are also considered. Since the top layers of DRAM dies have higher heat dissipation abilities, the hottest pages should be allocated to the most top layers. To predict whether a page is a hot page or not, future access patterns of that page are necessary. In our method, we simply collect the average access frequency from the trace files of applications to predict the future access patterns of pages [3] as shown in Equation (1). The symbols N and $Access_i$ represent the total number of pages of the application and the number of accesses to page i, respectively. The average access frequency of page i is denoted as $FutureFreq_i$. It is the ratio of the number of accesses to page i and the sum of the accesses to all pages. After the future access patterns of all applications are collected, we sort the pages by their future access frequency values and then assign them to memory banks from the most top tier to the most bottom tier of HMC. Figure 5 shows a simple illustrative example to decide the preferred memory bank for a page or a memory segment assuming that no other process competes for the same bank. In Figure 5, pages P_1 to P_N are sorted by their $FutureFreq$ values. Since the heat sink is placed on the top of the HMC, the top layers of the memory dies are cooler. Assume pages P_1 to P_a have the highest $FutureFreq$ values, the preferred memory banks of these pages will be assigned to the top tier of HMC. Then, the pages P_{a+1} to P_b whose $FutureFreq$ values are second higher will be assigned to the next top tier of HMC, and so on. In the end, pages P_{c+1} to P_n will be assigned to the bottom tier of HMC.

$$FutureFreq_i = \frac{Access_i}{\sum_{i=0}^{N} Access_i} \tag{1}$$

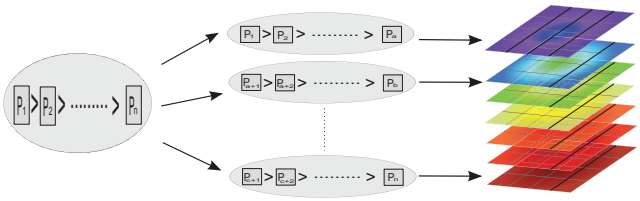

Fig. 5. An example of preferred memory banks assignment

That is, if the temperature of C_{MCP} exceeds the threshold T_t, OS not only re-assigns the current coolest channel C_{cold} to new preferred memory channel C_{new} of P_A, but also re-assigns the most suitable memory bank to new preferred memory bank B_{new} of P_A by $FutureFreq_{P_A}$. However, in real cases, there may be more than one processes competes for the same cool memory location. The Function *PreferredBanksExtract* in line *5* determines which memory bank is the most suitable candidate.

The details of Function *ExtractPreferredBank* are shown in Figure 6. This function is to determine which bank is the best choice of new preferred memory bank of page *p*. Its first step is to scan every memory bank b_i from the most top tier to the most bottom tier in channel *C*. It will check if there is still enough capacity left for p and all other non-allocated pages whose preferred banks are also in the same tier *i* of channel *C*. If the capacity is enough, a free bank b_i will be assigned to the new preferred memory bank B_{new} of *p*. If there is no room for *p* in the same tier, the function first finds a candidate page p_n and compares $FutureFreq_p$ with $FutureFreq_{p_n}$ to see which page is hotter and is deserved to occupy the space in the preferred bank of p_n. Page p_n is another non-allocated page whose preferred channel is *C* and preferred bank is in the same tier. In addition, $FutureFreq_{p_n}$ is the minimum future access frequency value among all non-allocated pages in the same tier. If $FutureFreq_p$ is larger than $FutureFreq_{p_n}$, p takes over the place of p_n and kicks out p_n from the current tier *i*. Hence, p_n needs to search for a new preferred memory space in the next tier of the HMC. Then, this function will call Function *UpdatePreferredBanks* to update new preferred memory bank of p_n. In Function *UpdatePreferredBanks*, p_n will check if there is any free bank for allocating. If not, another page may be squeezed out by p_n and search for its new preferred memory bank in the next next tier, and so on.

Figure 7 illustrates an example. Assume that the memory channel *A* is the preferred channel of process *1* assigned by MCP [1] and the memory channel *B* is the preferred memory channel of process *2* assigned by MCP [1]. The channel *A* is a hot channel whose temperature is about to exceed the pre-defined temperature threshold and the channel *B* is the coolest memory channel at this moment. Page P_A is the page of process *1* which is about

Function 1 : Function ExtractPreferredBank(page, channel)

Inputs : Page p; memory channel C
Outputs : New preferred memory bank B_{new} of p
1 : **for** Each tier i of HMC **do**
2 : **If** there is enough capacity in bank b_i **do**
3 : $B_{new} = b_i$;
4 : **return** B_{new}
5 : **end if**
6 : Find a non-allocated page p_n with minimum FutureFreq_{p_n} whose preferred bank is located in tier i
7 : Compare FutureFreq_p and FutureFreq_{p_n}
8 : **if** $\text{FutureFreq}_p > \text{FutureFreq}_{p_n}$ **then**
9 : B_{new} = the bank of p_n;
10 : UpdatePreferredBanks(p_n, $i+1$, C);
11 : Break;
12 : **end if**
13 : **end for**
14 : **return** B_{new}

Fig. 6. Details of function ExtractPreferredBank(page, channel)

to be allocated. The pre-defined preferred memory bank of P_A is at the top tier of memory channel *A* based on the future access frequency value of P_A. Let page P_B is another page of process *2* and its preferred memory bank is at the top tier of memory channel *B*. Among all non-allocated pages whose preferred memory bank is the top tier of memory channel *B*, the future access frequency value of P_B is minimum. In Figure 7 (a), assume there is no enough room for P_A, we need to compare the future access frequency values of P_A and P_B to see which page is deserved to stay in this memory bank. Figure 7 (b) shows that the future frequency value of P_A is larger than that of P_B. In this case, the preferred memory bank and preferred memory channel of P_A will be updated to *i* and *B*. Page P_B will need to find its new preferred memory bank in the next bottom tier of memory channel *B*. If it squeezes out another page in the next bottom layer, that page will need to find another preferred memory bank in the next next bottom tier of HMC.

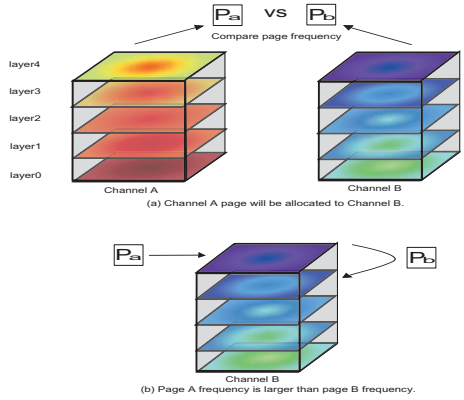

Fig. 7. An example of our thermal and access patterns aware OS page allocation method

C. *Trade Off between Interference, Bandwidth Variation, and Throttling*

Although our thermal and access patterns aware OS page allocation method can effectively reduce the probability of throttling, there is still another problem. The default assignment by MCP is best for performance, we should keep using MCP as long as possible. Our method begins to work only if the temperature of a memory channel is about to exceed the pre-defined temperature threshold. However, the optimal temperature threshold for system performance is very hard to define. Figure 8 shows an experimental results of the overall system weighted speedup of three different workloads (workload *A, B, C*) with four different temperature thresholds (*71, 72, 73, 74*°C) when OS performs our thermal and access patterns aware OS page allocation method. The x-axis represents the different temperature thresholds and the y-axis represents the weighted speedup of the system. The optimal temperature thresholds are different for different workloads. Workloads *A, B, C* have optimal temperature thresholds *74*°C, *72*°C, *73*°C, respectively. There are three reasons for this phenomenon. First, the reallocation of pages may degrade system performance because it increases the interference between processes inside the same memory channel. Second, the page reallocation may increase the overall bandwidth of memory non-intensive processes. This is because MCP tends to give more bandwidth to memory intensive applications, but the page reallocation may increase the number of channels accessed concurrently by memory non-intensive applications. Third, the throttling of HMC may lead to severe bandwidth degradation of the whole system. Hence, the system performance degrades if the system is too late to reallocate pages. To predict optimal temperature threshold, we need an analytical method to model memory interference, bandwidth variation, and throttling effect.

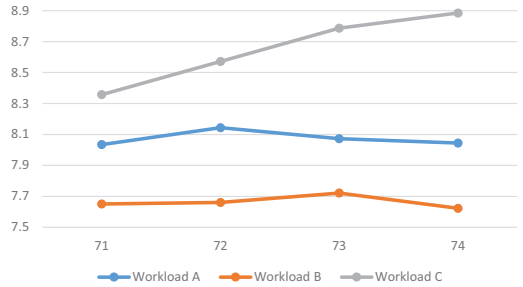

Fig. 8. Performance comparison of multitple workloads between different pre-defined temperature thresholds

In order to estimate the system performance of different temperature thresholds, we propose an analytical performance model. It considers memory interference, bandwidth variation, and throttling impact. The analytical model is shown as follows:

$$WS_E = P_T \times WS_T + (1 - P_T) \times WS_{NT}. \quad (2)$$

The symbols WS_E in Equation (2) represents the overall estimated weighted speedup of the system. It consists of two parts: the estimated weighted speedup WS_T which models the weighted speedup when the HMC throttles and the estimated weighted speedup WS_{NT} which models the weighted speedup when the HMC never throttles. The symbol P_T represents the estimated portion of execution time that the HMC keeps throttling. By utilizing this model, we can estimate the weighted speedup for different temperature thresholds and choose the optimal one for best performance.

The learning-based analytical model WS_E is designed to estimate the performance for a group of workloads whose memory behaviors are similar. Therefore, the first step is to collect the data for training. For example, assume workload *A, B* have similar memory behaviors, where workload *A* consists of *4* benchmarks *a, b, c, d* and workload *B* consists of *4* benchmarks *e, f, g, h*. We run simulations of A and B by applying our thermal and access patterns aware dynamic OS page allocation method under different pre-defined temperature thresholds (such as 71, 72, 73, 74°C). That is, for each benchmark, we can collect *4* traces files. In this example, the total number of the trace files for training is $2 \times 4 = 8$. By adding new workloads whose memory behaviors are also similar, the training model may become more accurate.

Equation (3) shows how we derive the estimated weighted speedup WS_{NT}. The $IPC(E)_i^{shared}$ represents the estimated IPC of process *i* under shared execution condition. To derive $IPC(E)_i^{shared}$, the bandwidth variation and the memory interference must be considered together. Hence, in Equation (4), we define $IPC(BW)_i$ as shared IPC of process *i* considering bandwidth variation and $\triangle IPC_i$ as the IPC degradation because of memory interference.

$$WS_{NT} = \sum_{i=1}^{n} \frac{IPC(E)_i^{shared}}{IPC_i^{alone}}. \quad (3)$$

$$IPC(E)_i^{shared} = IPC(BW)_i - \triangle IPC_i. \quad (4)$$

$$IPC(BW)_i = MaxFreq_i \times \alpha_i + \beta_i. \quad (5)$$

$$\triangle IPC_i = (\sum_{j=1}^{p} FutureFreq_j^C) \times \gamma_i + MPKI_i \times \epsilon_i + \delta_i. \quad (6)$$

To derive the model of $IPC(BW)_i$, we need to extract the key input features, which is the maximum memory bank access frequency among all accessed memory banks. By extracting this information from the trace files we mentioned above, we can gather the feature of each process. Then, for each process *i*, the maximum memory bank access frequency (denoted as $MaxFreq_i$) is used as the input of learning-based polynomial regression model [4] whose order is set to *1*. The rational of $MaxFreq_i$ is that if the memory requests of process *i* are distributed to many memory channels, $MaxFreq_i$ is decreased. This means that the process gets more memory bandwidth and its IPC may increase. After training process is done, the parameters α_i and β_i will be derived in Equation (5).

On the other hand, to derive the model of $\triangle IPC_i$, we also extract two features from the trace files. The first feature is $\sum_{j=1}^{p} FutureFreq_j^C$. The symbol *p* and *C* represent the total number of non-allocated pages of other processes and the preferred memory channels of process *i*. The feature means that we will sum up all future frequency values of pages of other processes whose preferred memory channels are also *C*. This means that these non-allocated pages are going to contend with the pages of process *i* in the same memory channels in the future. The second feature $MPKI_i$ is the miss per kilo instruction (MPKI) in the last level cache of the process *i*. This feature decides if the process is a memory intensive process or memory non-intensive process. The rational behind the second feature is that the performance of memory intensive processes is hard to be interfered by other processes. On the other hand, if the scheduling policy of memory controller does not support prioritizing memory requests of non-intensive applications, the performance of memory non-intensive processes is easily affected by other processes, especially memory intensive processes. Therefore, we apply multivariate regression model [4] to model these two features and predict the value of $\triangle IPC_i$. The parameters γ_i, ϵ_i and δ_i in Equation (6) will be known after training process is done.

As for the weighted speedup WS_T, we formulate it as the Equation (7). The symbol *p* represents the total port number of the HMC. The symbols $P_T(i)$ and $WS_T(i)$ represent the probabilities and the weighted speedup when there are *i* open ports in the HMC. To estimate $WS_T(i)$, we run a series of simulations without modelling the throttling effect. In these simulations, the number of open ports is changed from *1* to *p* in the HMC. After collecting the weighted speedups in these simulations, the learning-based method polynomial regression is used again to estimate $WS_T(i)$. The feature we adopt in this model is the weighted speedups of the cases where all ports are open, which is denoted as $WS(p)$. After the training processes, the values α_i and β_i will be derived to form the estimation model $WS_T(i)$ as shown in Equation (8). That is, if we know the estimated weighted speedup of a workload with all open ports ($WS(p)$), we can infer the weighted speed of that workload when the number of open ports changes. To estimate $P_T(i)$, the simulations of dynamic allocation with different pre-defined temperature thresholds are utilized again. The portion of execution time that the system only open *i* ports is collected to estimate the corresponding probabilities. After all these models are built up, we can estimate WS_T for any workload.

$$WS_T = \sum_{i=1}^{p} P_T(i) \times WS_T(i). \quad (7)$$

$$WS_T(i) = \alpha_i \times WS(p) + \beta_i. \quad (8)$$

To apply the analytical model of a group of workloads, we run the simulation of a new workload by using MCP method [1]. The corresponding features are fed into the off-line training model of the group whose memory behaviors are similar and we can derive the estimated performances of different pre-defined thresholds. Then, the estimated optimal pre-defined threshold is chosen and is set in the system. During run time, the system will run our thermal and access patterns aware OS page allocation method shown in Algorithm 1 according to this pre-defined temperature threshold.

TABLE I
ARCHITECTURAL PARAMETERS

Numer of Cores	16
ISA	x86
Core Model	2GHz processors, single pipeline
L1 Cache	16KB, 64B lines, 4-way assoc., Access latency 1 cycles
L2 Cache	512KB, 64B lines, 8-way assoc., Access latency 10 cycles
Main Memory	1GB (16 channels, 8 ranks per channel, 2 banks per rank)
Interconnect	MESH, NOC routers
OS	Linux 2.6.15

TABLE II
THERMAL PARAMETERS IN HOTSPOT

Chip thickness	0.1 mm
Silicon thermal conductivity	100 W/mK
Silicon specific heat	1750 kJ/m^3K
Spreader thickness	1 mm
Spreader thermal conductivity	400 W/mK
DRAM thickness	0.02 mm
DRAM thermal conductivity	100 W/mK
Interface material thickness	0.02 mm
Interface material conductivity	4 W/mK
Heatsink thickness	6.9 mm
Heatsink resistance	0.1 K/W

IV. EXPERIMENTAL RESULTS

In this section, we first describe our simulation environment and workloads selected from SPEC CPU 2006 in Section IV-A. Section IV-B compares the performance among different memory allocation policies and evaluate the benefits of our analytical model. In Section IV-C, we evaluate the accuracy of our training based model.

A. Simulation Environment

Environment: In our experiments, we use SIMICS [15] + GEMS [16] full system simulator to generate a cycle-accurate x86 system running Linux 2.6.15 with *16* cores. The Buffer-on-Board Memory System Simulator [**?**] is used to simulate HMC behavior. Memory is divided into *16* channels (vaults) and each channel has *8* ranks and *16* banks. The size of memory is *1* GB. HotSpot-5.02 [18] is used as our thermal simulation tool. The parameters in HotSpot-5.02 [19] for 3D architecture are shown in Table II.

Workloads: We classified SPEC CPU 2006 benchmarks into two categories: memory intensive applications and memory non-intensive applications. The memory intensive applications include mcf, libquantum, leslie3d, soplex, lbm, GemsFDTD, xalancbmk, omnetpp, cactusADM. The memory non-intensive applications include gcc, tonto, povray, calculix, gromacs. In our experiments, *5* workloads are from all memory intensive benchmarks and *5* workloads mix of half memory intensive benchmarks and half memory non-intensive benchmarks.

B. Results of Thermal and Access Patterns Aware Dynamic OS Page Allocation

In the first experiment, we compare the performance of three different schemes: MCP [1], *nofuture*, and *future*. The first scheme is the default MCP [1] method. The second scheme *nofuture* is that when the temperature of the hottest channel exceeds the pre-defined temperature threshold, OS will start to reassign pages to the current coolest channel without considering the future access frequencies. The third scheme *future* is our thermal and access patterns aware OS page allocation method. Figure 9 shows the comparison results of all workloads. In average, our method can improve the system performance by *12.7%* and *9.3%* compared to MCP scheme and *nofuture* scheme.

Figure 10 shows the performance comparison among different pre-defined temperature thresholds. The bars *71, 72, 73, 74* °C represent different statically fixed temperature thresholds. The bar named *Analytical* represents that the pre-defined threshold is calculated by our analytical model. All weighted speedups are normalized to the weighted speedup of MCP scheme. For each workload,

our analytical model can choose the optimal or near optimal weighted speedups from the *4* different statically defined thresholds. In average, our dynamic allocation method with the threshold defined from our analytical model improves *3.5%*, *2%*, *0.5%*, and *0.4%* compared to the performance of our dynamic allocation method with the statically fixed thresholds *71, 72, 73 ,74* °C.

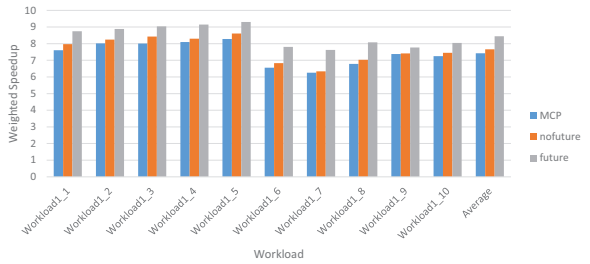

Fig. 9. Performance comparison among different schemes

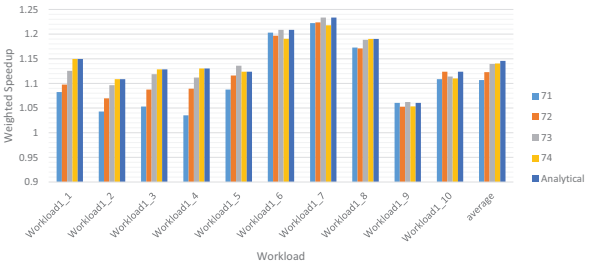

Fig. 10. Performance comparison among different pre-defined temperature thresholds

C. Accuracy of the Analytical Performance Model

In this section, we will evaluate the accuracy of our learning-based model. To examine the accuracy of the model, the estimated performance slopes $Slope_E(T)$ and the real performance slopes $Slope_R(T)$ between two pre-defined temperature thresholds *T* and *(T-1)* need to be tracked. The definitions of $Slope_E(T)$ and $Slope_R(T)$ are shown in Equations (9) and (10). We pick the real weighted speedup when the threshold is the lowest temperature as a normalized basis and denote it as $WS_R(T_{lowest})$. $WS_E(T)$ represents the estimated weighted speedup when the pre-defined temperature threshold is set to T °C. The difference between two weighted speedups $WS_E(T)$ and $WS_E(T-1)$ are calculated and normalized to $WS_R(T_{lowest})$. $Slope_R(T)$ works similarly as $Slope_E$(T). It calculates the difference of weighted speedups of $WS_R(T)$ and $WS_R(T-1)$. Therefore, if the difference between $WS_E(T)$ and $WS_R(T)$ is *0*, we know that our model can exactly predict the trend of performance variation between different temperature thresholds. For example, if the temperature threshold changes from *71* to *72* °C and the real weighted speedup also changes from *8* to *8.2*, $Slope_R(72) = \frac{8.2-8}{8} = 2.5\%$. Assume $WS_E(71)$ and $WS_E(72)$ be *7.8* and *8.0* and the lowest temperature threshold be *71* °C. $Slope_E(72)$ is $\frac{8-7.8}{8} = 2.5\%$. It means that the slope of the performance is exactly the same between the real value and the estimated value. Table III shows the results of $Slope_R(T) - Slope_E(T)$, where *T* varies from *72* to *74* °C. In average, the overall difference between the real slope of weighted speedups and the estimated weight speedups is *0.86%*.

$$Slope_E(T) = \frac{WS_E(T) - WS_E(T-1)}{WS_R(T_{lowest})} \quad (9)$$

$$Slope_R(T) = \frac{WS_R(T) - WS_R(T-1)}{WS_R(T_{lowest})} \quad (10)$$

V. Conclusions

In this article, we have proposed our thermal and access patterns aware dynamic OS page allocation method to appropriately map pages in HMC for best performance. Our method considers all possible factors such as thermal characteristics of HMC, access patterns of each process, memory interference between all processes, bandwidth variations of all processes and current thermal conditions of memory channels. We first develop an analytical model to estimate the system performance for different pre-defined temperature thresholds. After the model is trained, we use the analytical model to define the optimal temperature threshold for data reallocation. During run time, the system will reallocate the pages by our dynamic allocation method when the temperature of a hot memory channel is about to exceed the pre-defined temperature threshold. In average, our method can improve the system performance by *12.7%* compared to best performance-oriented allocation method (MCP) [1] for 2D memory. The average error rate of our analytical model to predict the trend of performance variation is only *0.86%*.

TABLE III
Results of $Slope_R(T) - Slope_E(T)$ (%)

Workload	72	73	74	Average
1_1	0.65	1.73	1.58	0.99
1_2	2.23	0.05	1.04	0.83
1_3	3.02	2.35	0.65	1.52
1_4	4.97	1.59	1.45	2.00
1_5	2.47	1.51	1.36	1.35
1_6	0.71	0.48	0.34	0.38
1_7	1.06	0.24	0.56	0.47
1_8	0.63	0.13	0.07	0.20
1_9	0.17	0.60	0.53	0.33
1_10	1.13	0.43	0.70	0.57
Avg.	1.70	0.91	0.83	0.86

References

[1] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, Thomas Moscibroda, "Reducing memory interference in multicore systems via application-aware memory channel partitioning," *MICRO'11*, pp. 374-385, December 2011.

[2] K. Man, "Bensley fb-dimm performance/thermal management," in Intel Developer Forum, 2006.

[3] A. Hsieh, TingTing Hwang, "Thermal-aware memory mapping in 3D designs," *ACM Trans. Embedded Comput. Syst. 13(1): 4* (2013)

[4] E. ALPAYDIN, 2010. Introduction to Machine Learning, 2nd Ed. MIT Press

[5] J. Jeddeloh and B. Keeth, "Hybrid memory cube new dram architecture increases density and performance," *in VLSI Technology (VLSIT), 2012 Symposium,* pp. 8788 June 2012.

[6] K. Skadron, T. Abdelizaher, and M. R. Stan, "Control theoretic techniques and thermal-rc modeling for accurate and localized dynamic thermal management," *In HPCA'02*. pp. 1728 March 2002.

[7] K. Man, "Bensley fb-dimm performance/thermal management," in Intel Developer Forum, 2006.

[8] J. Iyer, C. L. Hall, J. Shi, and Y. Huang, "System memory power and thermal management in platforms built on Intel Centrino Duo mobile technology," vol. 10, no. 2, pp. 123132, May 2006.

[9] J. Lin, H. Zheng, Z. Zhu, H. David, and Z. Zhang, "Thermal modeling and management of dram memory systems," in Proceedings of the 34th annual international symposium on Computer architecture, ser. *ISCA 07*. New York, NY, USA: ACM, 2007, pp. 312322. [Online]. Available: http://doi.acm.org/10.1145/1250662.1250701

[10] K. Sankaranarayanan, S. Velusamy, M. Stan, and K. Skadron, "A case for thermal-aware floorplanning at the microarchitectural level," J. Instruction-Level Parall. 7.

[11] M. Mutyam, F. Li, V. Narayanan, M. Kandemir, and M. J. Irwin. "Compiler-directed thermal management for VLIW functional units," *In LCTES'06,* pp. 163172 July 2006.

[12] Y.-F. Tsai, Y. Xie, N. Vijaykrishnan, and J. M. Irwin, "Three-dimensional cache design exploration using 3Dcacti," *In Proceedings of the International Conference on Computer Design,* pp 519524.

[13] M. J. Khurshid, and M. Lipasti, "Data Compression for thermal mitigation in the hybrid memory cube," *In ICCD'13,* pp. 185-192 October 2013.

[14] A.-C. Hsieh, and T.-T. Hwang, "Thermal-aware memory mapping in 3D designs," *In DATE'09,* pp 1361-1366 April 2009.

[15] P. Magnusson et al. "Simics: A full system simulation platform." Computer, 35(2), Feb 2002.

[16] M. M. K. Martin et al. "Multifacets general execution-driven multiprocessor simulator (GEMS)"

[17] E. Cooper-Balis, P. Rosenfeld, B. Jacob, "Buffer-On-Board Memory Systems," *In ISCA'12,* pp. 392-403 June 2012.

[18] W. Huang , K. Sankaranarayanan , R. J. Rib , M. R. Stan , K. Skadron, "An Improved Block-Based Thermal Model in HotSpot 4.0 with Granularity Considerations," *In Proceedings of the Workshop on Duplicating, Deconstructing, and Debunking,* 2007

[19] J. Meng, D. Rossell and A. K. Coskun, "Exploring Performance, Power, and Temperature Characteristics of 3D Systems with On-Chip DRAM," *IGCC'11,* pp. 1-6 July 2011.