Roofline-based Data Migration Methodology for Hybrid Memories

Jongmin Lee[1], Kwangho Lee[1], Mucheol Kim[2], Geunchul Park[3], Chan Yeol Park[3]

[1] Department of Computer Engineering, Won-Kwang University, Korea
[2] School of Software, Chung-Ang University, Korea
[3] National Institute of Supercomputing and Networking, Korea Institute of Science and Technology Information, Korea
square55@wku.ac.kr, lkh002@wku.ac.kr, mucheol.kim@gmail.com, gcpark@kisti.re.kr, chan@kisti.re.kr

Abstract

High-performance computing (HPC) systems provide huge computational resources and large memories. The hybrid memory is a promising memory technology that contains different types of memory devices, which have different characteristics regarding access time, retention time, and capacity. However, the increasing performance and employing hybrid memories induce more complexity as well. In this paper, we propose a roofline-based data migration methodology called HyDM to effectively use hybrid memories targeting at Intel Knight Landing (KNL) processor. HyDM monitors status of applications running on a system and migrates pages of selected applications to the High Bandwidth Memory (HBM). To select appropriate applications on system runtime, we adopt the roofline performance model, a visually intuitive method. HyDM also employs a feedback mechanism to change the target application dynamically. Experimental results show that our HyDM improves over the baseline execution the execution time by up to 44%.

Keywords: Performance, Data migration, Roofline model

1 Introduction

With the ever-shrinking feature size in the CMOS process technology and increasing performance demands, modern processors typically integrate multiple cores and the number of cores in the same chip area has grown significantly. Continuous technology scaling realizes a many-core processor with hundreds of cores on a single chip [1-3]. These trends necessitate larger DRAMs to accommodate more and bigger programs in the main memory. DRAMs have been popularly used to implement the main memory because of their high densities and low prices. Due to the scaling limitation of DRAMs and the high bandwidth demands, hybrid storage architectures, which contain heterogeneous memories, are likely to be the future memory systems in high-performance computing (HPC) systems [4-7].

Knights Landing (KNL) is the code name for the second-generation Intel Xeon Phi product family [1, 8]. The KNL processor contains tens of cores and it provides the HBM 3D-stacked memory as a Multi-Channel DRAM (MCDRAM). DRAM and MCDRAM differ significantly in terms of access time, bandwidth and capacity. Because of those differences between DRAM and MCDRAM, performance will vary depending on the application characteristics and the usage of memory resources. The efficient use of these systems requires prior application knowledge to determine which data of applications to place in which of the available memories. A common goal is to shorten the execution time of applications, which translates to place applications in the fastest memory. However, fast memory is a limited resource in terms of capacity. As a result, it is important to identify the appropriate data object of applications and host the most profitable applications in fast memory. The switch to multi/many-core processors and hybrid memories means that microprocessors will become more diverse. The growing complexity in HPC environments makes difficult for users to determine the performance of applications quantitatively.

In this paper, we propose a roofline-based data migration strategy for hybrid memories (HyDM). The roofline performance model is a simple and visual model that offers insights for performance analysis [9]. Rather than simply using percent-of-peak estimates, the model can be used to evaluate the quality of attainable performance including locality, bandwidth, and computational throughput.

HyDM periodically monitors the application's behavior using a performance monitoring tool and it selects appropriate applications, which require more memory bandwidth and show memory locality, with a regression-based prediction. By migrating pages of the applications to the high bandwidth memory (i.e. MCDRAM), HyDM improves the memory usages on

*Corresponding Author: Chan Yeol Park; E-mail: chan@kisti.re.kr
DOI: 10.3966/160792642020052103022

hybrid memories. In order to trace performance changes of applications during their executions, HyDM employs a feedback mechanism to change target applications dynamically. Our experimental results demonstrate that HyDM significantly improves the performance of mixed application sets on the Intel KNL processor. HyDM enhances performance by up to 44% compared to the baseline execution time.

The rest of this paper is organized as follows. We provide related works in the next section. Section 3 presents background. Section 4 presents our proposed data migration strategy using the roofline model. Experimental results are given in Section 5. Finally, Section 6 concludes this paper.

2 Related Work

There are some categories of works that are closely related to this paper.

Hybrid memories. Many memory devices have been developed for decades to replace DRAM, which has fast but non-volatile characteristics [17-18]. PRAM is easier to integrate than DRAM, but the number of writable times per cell is limited thus memory life is short. STT-RAM has a fast write speed and good write durability, but it is relatively hard to integrate, and therefore, there is less need to replace DRAM in terms of economy. When examining the new memory technology to date, it is difficult to pursue universal memory, and it is judged to be a non-volatile technology that lacks performance rather than DRAM. To use those memories, many researches have been done on hybrid memories in the form of DRAM and other types of memories together. One of the hybrid memory systems uses DRAM as a cache and PRAM as a main memory [19-20]. They mitigate the durability of PRAM and write delay by filtering the write operations to the main memory using the DRAM cache. In [21], DRAM and PRAM are located at the same level. In order to compensate for the delay in the write operation and the lifetime of the PRAM, a page manager selectively allocates pages among PRAM and DRAM. All of the above techniques are designed to reduce write activities in PRAM, however, this paper addresses the usage of HBM with DRAM.

GPU is the most commonly used hybrid memory to date in HPC [22]. GPU employs GDDR as high-speed memory and relatively slow DRAM as main memory. By storing critical data using prefetch techniques in GDDR, GPU supports fast operation. The GPU operates as an accelerator with respect to DRAM. By comparison, our research explores general processors for HPC environments.

Roofline performance model. The roofline model is used in a number of scientific applications to analyze bottlenecks in the performance of an architecture and to guide software optimizations [9]. Various types of roofline models are proposed in previous works [6, 23-25]. In [23], energy version of the roofline model is proposed to show bounds on performance due to energy limitations. This model focuses on identifying the balance between performance and energy in architectural design. In [24], the roofline model is extended to support the cache hierarchy. Recently, the roofline model is extended for specific applications and platforms such as GPUs [6].

Page migration. A variety of page migration methods using NUMA nodes have been studied [26-29]. A basic methodology to efficiently use memories in a NUMA system is to store the data in the same location as the processor that frequently references the data. In [26], the migration of the pages between nodes is performed by using the characteristic that the memory access pattern repeatedly appears in applications. In [29], a sampling-based approach is used in which pages with excessive remote references are migrated to nodes close to the accessing core. The system continuously samples the excess miss counters to produce a list of candidate pages for migration and replication.

We propose a dynamic memory management methodology using the roofline model, the key contribution of our work is the algorithm that efficiently uses different types of memories in HPC systems without any hardware or software modifications. Although our proposed HyDM targets the Intel KNL processor in this paper, adopting the methodology to the systems employing hybrid memories is possible.

3 Background

3.1 Intel Xeon Phi Processor

Our target processor is the Intel Xeon Phi, which is a series of x86 many-core processors. The Knight Landing (KNL) is the code name for the second-generation Intel Xeon Phi product and it is also a recent example of the hybrid memory systems. The processor is popularly used in HPC systems such as TACC's flagship system, Stampede2, and Berkeley Lab's CORI supercomputer.

In this section, we briefly summarize the main features of the Intel KNL processor, especially we focus on its memory system. Figure 1 illustrates the KNL processor and its connection to the hybrid memories. The KNL processor integrates up to 72 cores together with eight Multi-Channel DRAM (MCDRAM) memories, which support 16GB of memory and they provide the peak bandwidth of 400GB/second. The processor also integrates six DDR4 channels supporting up to 384GB of memory with the peak bandwidth of 100GB/second. The MCDRAMs are positioned on-chip while DRAMs are off-chip.

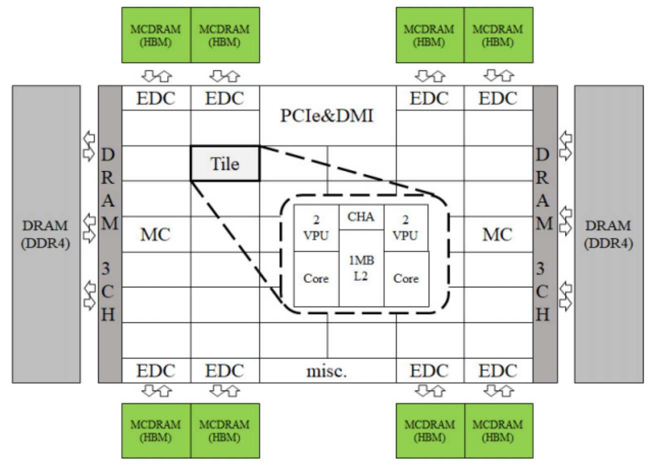

Figure 1. A structure of Intel Knight Landing processor

Figure 1 shows 36 tiles in the KNL processor and each tile consists of the two cores sharing 1MB L2 cache. Tiles are connected through a 2D-mesh network on-chip and they can be clustered in several NUMA configurations. In this paper, we use the Quadrant cluster configuration where the tiles are partitioned in four quadrants as it reduces the latency of L2 cache misses because the worst-case path is shorter. This configuration is the one recommended by Intel as a symmetric multi-processor [10]. MCDRAAM can be configured at boot time in three modes: cache, flat or hybrid mode. The Flat mode configures MCDRAMs to the same address space with DRAMs, Cache mode configures MCDRAMs as a last-level cache. The Hybrid mode separates MCDRAMs as two parts and one is used for an additional addressable memory with DRAMs and another is used for a last-level cache. In this work, we consider the Flat mode. For more details on KNL processor can be found in [1, 11].

Because DRAM and MCDRAM exhibit different memory characteristics, it is necessary to map data objects to appropriate memory in order to run applications efficiently on the systems. However, programming a hybrid memory system and identifying the best object-to-memory mapping is a complex task.

3.2 Roofline Performance Model

The roofline performance model is a visually intuitive method used to bound the performance of floating-point programs running on multi/many-core processors [9]. To evaluate performance, the roofline model ties floating-point performance (GFlops/second), arithmetic intensity (Flops/Byte), and memory bandwidth (GB/second) together. The peak floating-point performance and the peak memory bandwidth represent the attainable performance on a system and the arithmetic intensity shows a ratio of computations to memory accesses. By combining memory usage, computation throughput, and bandwidth, the model assesses the quality of attained performance of applications and provides insights on both the implementation and inherent performance limitations. Although the roofline model does not show precise performance evaluations, it is simple and effective to observe behavior of applications running on systems.

Figure 2 shows the roofline model of the Intel KNL processor with NAS parallel benchmark suites [12]. We executed each benchmark alone and periodically record the position of each benchmark to see performance changes of each benchmark over time. Detailed experimental methodologies will be shown in Section 5. The lines on the top show the peak performance of KNL processor with DRAM, MCDRAM, and floating-point units, respectively. The x-axis shows the arithmetic intensity that is the ratio of total floating-point operations to total data movement. The y-axis represents performance that is the number of floating-point operations completed by the cores. As shown in Figure 2, most benchmarks are changing their positions over time and some benchmarks are located under the memory-bound area with small arithmetic intensities. Arithmetic intensity with a small number means there are more memory requests, and the opposite case means more computations. Thus, one of the straightforward approaches to enhance the performance is moving data of the applications, which require more memory bandwidth, to the high bandwidth memory.

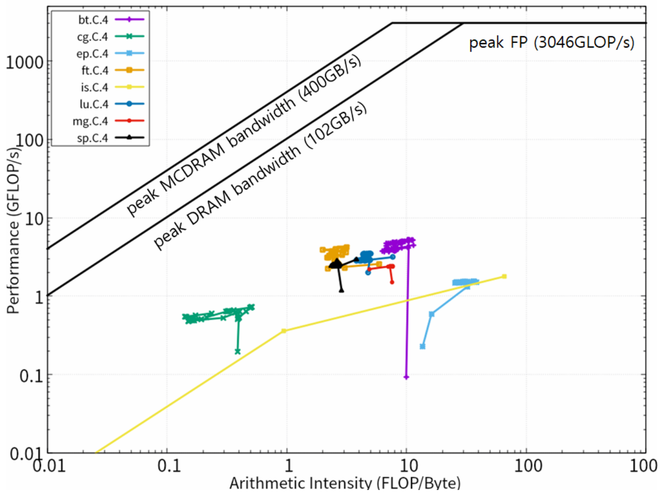

Figure 2. Roofline performance model for NPB applications

4 Proposed Techniques

In this section, we introduce a data migration methodology for hybrid memories called HyDM.

4.1 Overview

Figure 3 shows an overview of HyDM method. HyDM employs three stages to enhance the performance of applications on the KNL processor.

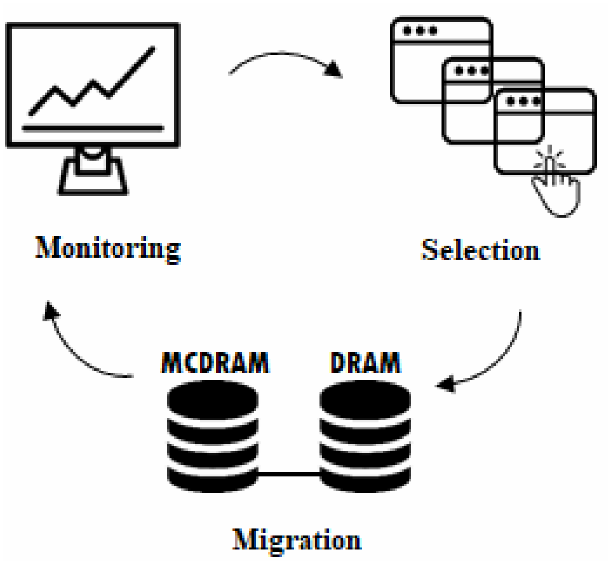

Figure 3. HyDM Methodology Overview

We first monitor the applications during system runtime using hardware monitoring tools. Then, based on the historical data, we select a candidate application, which requires more memory bandwidth, using the roofline model. Next, we migrate data stored in both MCDRAM and DRAM dynamically. By managing application data on MCDRAM and DRAM, HyDM effectively uses hybrid memories.

Algorithm 1 shows the implementation of HyDM. If running applications exist, HyDM makes the list of running application *L* (line 1). *L* stores unique PIDs for each application. The three stages of HyDM are repeatedly performed in a time window *p*. At each time window, historical monitored data from each application are stored in the list $W = \{W_0, W_1, \ldots, W_n\}$ (line 2). Figure 4 shows the structure of list *W*. W_0 is the current time window and the time window W_1 is the previous time window. W_n represents the n-th previous window.

```
Algorithm 1 $HyDMAlgorithm\ (p)$
  Input: $p \leftarrow$ time window period
  /*
  $L$ : the list of running application
  $W$ : the list of windows for monitoring data
  $b$ : the selected application id
  */
  initiate $L$
  while $length(L) > 0$ do
    wait p
    1. $L \leftarrow getCurrentApplication()$
    2. $Monitoring(L, W)$
    3. $b \leftarrow Selection(L, W)$
    4. $Migration(b, L, W)$
  end while
```

```
Algorithm 2 $Monitoring(L, W)$
  Input: $L \leftarrow$ the list of running application
         $W \leftarrow$ the list of windows for monitoring data
  /*
  $M$ : the list of monitoring data for applications
  $i$ : the index of application
  */
  for $i \leftarrow L_0$ to $length(L)$ do
    1. $M_i \leftarrow getMonitoringData(i)$
    2.     /* $M_i.fp$ : # of floating point operations */
    3.     /* $M_i.pref$ : # of page references */
    4.     /* $M_i.pf$ : # of page faults */
  end for
  5. insert $M$ to $W_0$
```

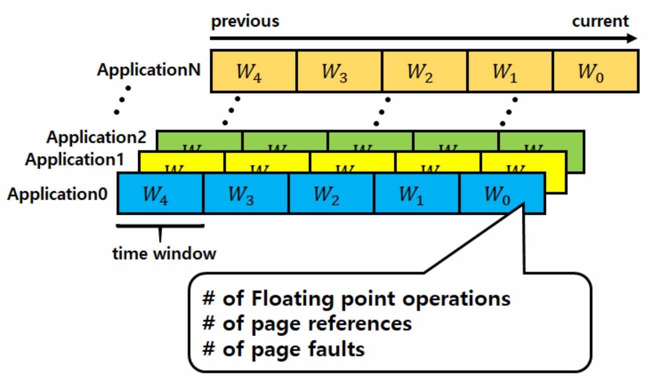

Figure 4. Structure of monitored data

After the Selection procedure with the lists *L* and *W*, HyDM returns the candidate application *b* for migration (line 3). The pages of selected application *b* are migrated by the Migration procedure (line 4). We present the details of our method in the following subsections.

4.2 Monitoring

During the system runs, HyDM monitors applications using hardware monitoring tools. Most processors now include hardware support for performance monitoring such as *perf_event* [13] and *LIKWID* [14]. In this paper, we use *perf_event*. In Algorithm 2, the inputs include the list of running application *L* and the list of windows for monitoring data *W*. Let *M* denote the list of monitored data for running applications in the current time window. Let M_i denote i-th application. The Monitoring procedure collects the number of floating-point operations ($M_i.fp$), page references ($M_i.pref$) and page faults ($M_i.pf$), and it stores those values into the entry corresponding to each type in M_i (line 1-4). $M_i.fp$ and $M_i.pref$ will be used to compute the arithmetic intensity of each application. The *for* loop stops when *i* is equal to the size of *length(L)*. Then, *M* is inserted to the W_0 to prepare the next stage (line 5). Because HyDM only stores a few types of monitoring data, the storage overhead is very small compared to the total memory.

4.3 Selection

Algorithm 3 shows the Selection procedure that chooses an application as a candidate for migration. In order to select an application, which requires more memory bandwidth, HyDM uses the roofline model. When the execution status of applications is mapped to the roofline model, HyDM chooses an application with the lowest arithmetic intensity in the memory-bound area. The strategy in HyDM is to give more chances to the application that shows the highest ratio of memory references to computations.

Algorithm 3 $Selection(L, W)$

```
Input: $L \leftarrow$ the list of running application
       $W \leftarrow$ the list of windows for monitoring data
Output: $b \leftarrow$ the selected application id
/*
$S$ : the sorted list of applications
$i$ : the index of applications
$pf_{avg}$ : the averaged page faults
*/
1. $S \leftarrow regressionAndSort(W)$
2. $pf_{avg} \leftarrow getAvgPageFault(W_0)$
for $i \leftarrow 0$ to $length(S)$ do
  3. if $S_i.pf > pf_{avg}$ then continue
  4. $b \leftarrow getAppId(S_i, L)$
  5. return $b$
end for
```

The *regressionAndSort* procedure first computes the arithmetic intensity of each application using historical floating-point operations and page references stored in W (i.e. $W_{time.appid.fp}$ and $W_{time.appid.pref}$). After that, we perform the linear regression to predict the next arithmetic intensity value for each application, and the results are stored into a list S. Finally, the application list (S) are sorted in ascending order according to the next arithmetic intensity values (line 1). Since all candidate applications are sorted in the list S, the first application of the list is considered for migration. In order to filter applications with low memory locality, HyDM employs a simple technique using a number of page faults monitored in the Monitoring procedure. HyDM compares the number of page faults from the first application with the average number of page faults (pf_{avg}) from all applications (line 3). If the candidate application shows a higher value in page faults, the next application in the list S is considered for migration. If all operations are finished, the *Selection* procedure returns the candidate application b for migration (line 5).

4.4 Migration

Algorithm 4 shows the Migration procedure. Because of the capacity limitation of MCDRAM (16GB), we identify the possibility before performing the data migration. We check that the total memory usage, including the current memory usage of the selected application, does not exceed the threshold parameter t (e.g. 90%). If the usage of MCDRAM is less than the threshold t, HyDM migrates the referenced pages during time window W to MCDRAM (line 1). Note that, the page grouping techniques for selecting the critical pages of the entire page in the application are applicable to our proposed scheme [15-16].

Algorithm 4 $Migration(b, L, W)$

```
Input: $b \leftarrow$ the selected application id
       $L \leftarrow$ the list of running application
       $W \leftarrow$ the list of windows for monitoring data
/*
$r$ : the index of a victim application
$t$ : the threshold ratio of MCDRAM use
*/
if $isMigrationPossible(t, b)$ then
  1. $migrationToMCDRAM(b, W)$
else
  2. $r \leftarrow selectVictim(b, W)$
  3. if $r$ exists then
  4.     $migrationToDRAM(r, W)$
  5. end if
end if
```

When the usage of MCDRAM is larger than the threshold t, the *selectVictim* procedure chooses a victim application to migrate pages from MCDRAM to DRAM (line 2). Since migrating pages back to DRAM frequently induces additional performance overheads, we employ a strict methodology based on a priority. Figure 5 shows priorities of running applications in HyDM and operations in the *selectVictim* procedure. We categorize applications into four priorities according to memory requirements and the arithmetic intensity values. For example, if an application represents an amount of memory request that is higher than the average memory request amount of all running applications in the current time window (i.e. W_0) and the corresponding arithmetic intensity is below the peak MCDRAM bandwidth region of the roofline model (i.e. left side), we assign priority 1. In the opposite case, priority 4 is assigned. After determining the priority for each application, we decide to migrate to DRAM in the order shown in Figure 5. Migration to DRAM is performed only when the candidate application b has a priority value of 1 and an application having priority 4 in the MCDRAM is found (line 3-6). The concept behind the *selectVictim* procedure is to provide more opportunities for applications in MCDRAM. By employing the feedback mechanism, HyDM effectively uses hybrid memories when many applications are running on a system.

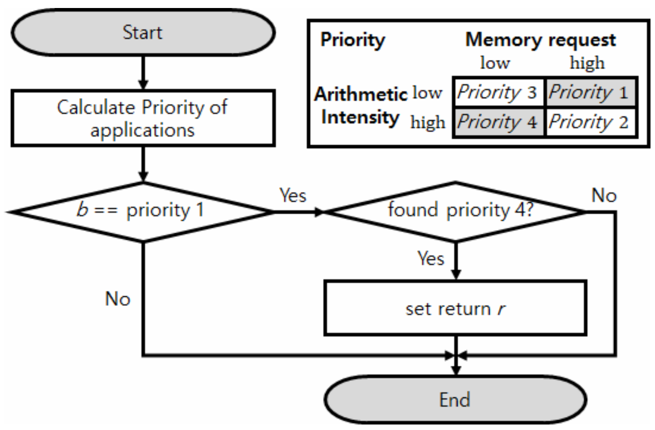

Figure 5. *selectVictim* procedure

5 Experimental Results

In this section, we present the methodologies for evaluations and their results with discussion.

5.1 Methodology

The experimental system is equipped with the Intel Xeon Phi(TM) CPU 7250@1.40GHz, 68 cores per socket, 4 threads per core, and a total of 272 threads available with the hyper-threading technology. The system includes 96GB DDR4 (DRAM) and 16GB HBM (MCDRAM).

We evaluated NAS Parallel Benchmark (NPB) related to computational fluid dynamics [12]. The NPB consists of five kernel benchmarks (IS, EP, CG, MG, FT) and three pseudo benchmarks (BT, SP, LU). For all experiments, we used standard test problems (CLASS-C). Table 1 shows the benchmark execution results when they were run on the system alone including average execution times, floating-point operations, memory accesses, and the amount of peak memory use, respectively.

Table 1. NAS Parallel Benchmarks (NPB) characteristics

Name	Average execution time (sec.)	FP operatioms (GFop)	Memory accesses (read/write) (mil.)	Peak memory use (MB)
IS.C	31	23	758 / 338	1,572
EP.C	462	494	2,739 / 1,028	20
CG.C	319	261	31,174 / 2,940	1,102
MG.C	129	213	7,507 / 2,940	3,536
FT.C	335	837	18,555 / 10,197	7,188
BT.C	920	2,560	45,682 /17, 270	1,676
SP.C	626	1,918	92,526 / 47,727	1,416
LU.C	733	2,504	84,716 / 38,302	760

Table 2 shows the design parameters of HyDM and their values used in evaluations. The minimum time unit of *perf_event* is 1ms. When hardware events for monitoring are executed frequently, however, we observed that performance degradation occurred. We adjusted the numerical values without affecting system performance through a heuristic method. To assume the situation of the system running a number of applications that require much larger capacity than the capacity of MCDRAM, we perform the NPB programs in the number of multiples (e.g. bt0, bt1, bt2, bt3). All experiments were conducted using four threads per application. Therefore, when 32 applications are running in parallel, a total of 128 threads were created. We randomly assigned programs to the cores using the default policy. Because the Intel KNL processor is a tile structure, performance will vary depending on how threads are allocated. The study of how to allocate threads in the proper place is beyond the scope of this paper and is not covered.

Table 2. HyDM design parameters

Descriptions	Values
The time window period: p	100 (ms)
The number of applications: *Length (L)*	32
Size of monitoring windows: $W = \{W_0, W_1, ..., W_n\}$	5, 10
The threshold ratio of MCDRAM use: t	90 (%)

5.2 Performance Evaluation Results

To evaluate the performance impact of the proposed HyDM, we calculated the average execution times. Figure 6 illustrates the average execution times for 100 runs with 5 monitoring windows. In the legend, Default indicates that HyDM is not applied, which uses MCDRAM and DRAM in interleaving mode, and Random indicates that running applications were randomly selected for migration. HyDM indicates that the proposed technique is applied. We added the Random method to compare the effects of the proposed HyDM. We normalized the results to Default. Data migration using HyDM and Random methods reduced execution times on most benchmarks. This performance improvement is because memory-intensive applications, which have many memory accesses as shown in Table 2, were executed on MCDRAMs after migration. Especially, several benchmarks showed large reductions with HyDM in the execution time such as *cg* (42%~45%), *ft* (13%~16%), and *sp* (14%~17%). HyDM is superior to Random on most benchmarks because the proposed technique provides a better choice of selecting applications with more memory requests. However, when HyDM and Random schemes were applied, *is* benchmark showed increase in execution times (2%~7%). This is because *is* benchmark has random memory access patterns to perform integer sort operations and its short execution time. If the execution time of an application is short, the overhead of stopping the execution and performing the migration may seem relatively large. In addition, due to the nature of the memory pattern of *is*, many page faults occurred and MCDRAM supporting high memory bandwidth was not used effectively. On average, the reduced execution times are 18% and 12% with HyDM and Random, respectively. The average execution times of applications are shown in Table 3. Figure 7 shows average execution times of the proposed HyDM when the number of monitoring windows varied from five to ten. We normalized the results to Default. When we applied 10 monitoring windows, the overall execution time was 2% higher than 5 monitoring windows. As the size of the monitoring windows increases, HyDM collects more historical data. HyDM uses the past information to predict future, but too old records do not reflect the characteristics of the current working sets. Therefore, it is more effective to make a prediction using five monitoring windows.

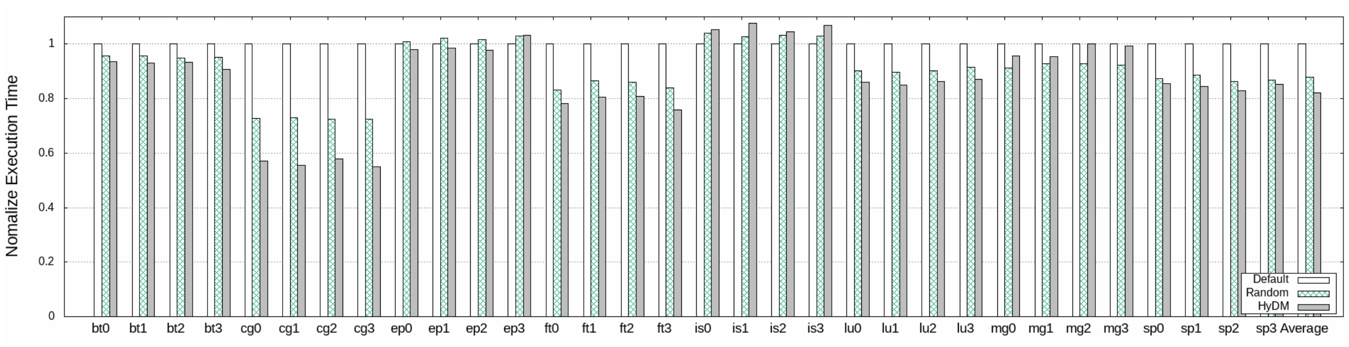

Figure 6. Average execution time results normalized to that of the baseline

Table 3. Average execution times

		Application							
		bto	bt1	bt2	bt3	cg0	cg1	cg2	cg3
Execution Time (sec.)	Default	145.6705	1437.0035	1455.0347	1456.4611	1211.8204	1226.5472	1225.0381	1210.2273
	Random	1397.8919	1374.6753	1365.6279	1379.9398	862.26799	904.2012	877.38188	875.9205
	HyDM	1354.1377	1343.1216	1349.0682	1319.5702	693.5221	679.97567	704.3997.	666.17277
		ep0	ep1	ep2	ep3	ft0	ft1	ft2	ft3
	Default	368.533	362.45586	372.86993	362.03356	566.74406	532.34749	540.34991	577.36135
	Random	375.89024	365.28827	370.48978	374.02926	464.2835	452.9071	467.89133	474.9293
	HyDM	360.71978	358.63339	362.7888	371.4111	440.01775	428.69139	441.0862	430.21907
		is0	is1	is2	is3	lu0	lu1	lu2	lu3
	Default	45.547779	45.090701	45.380379	45.105202	1227.4246	1241.1089	1227.9619	1216.829
	Random	47.249491	46.956435	47.844231	47.406987	1113.3445	1107.7117	1101.0334	1112.3716
	HyDM	47.842015	48.495822	47.416234	48.158338	1056.3436	1051.4723	1059.6893	1058.1215
		mg0	mg1	mg2	mg3	sp0	sp1	sp2	sp3
	Default	164.1152	159.81184	157.41794	162.38472	1159.6789	1150.2174	1172.766	1159.743
	Random	149.60807	147.07064	146.96972	149.85035	1005.484	1021.416	996.74817	1009.4771
	HyDM	156.87207	152.70091	159.9955	159.44007	989.12922	971.90131	964.4554	993.0762

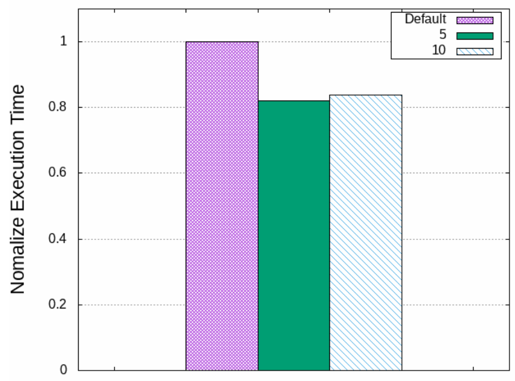

Figure 7. Sensitivity analysis of the number of monitoring windows

5.3 Case Analysis Results

In this subsection, we analyze the proposed HyDM in several cases. Figure 8 and Figure 9 shows the roofline performance models and the bandwidth results respectively for one evaluation case. Due to the limitation of space, we select several benchmarks which show notable features.

In Figure 8, benchmarks show different shapes in the roofline models during execution. We adjusted the number of points in each item to twenty to improve the readability of graphs. While *bt* and *mg* show small changes over time in the roofline models, *cg* and *is* show large changes. This indicates how the characteristics of the working sets change over time. In

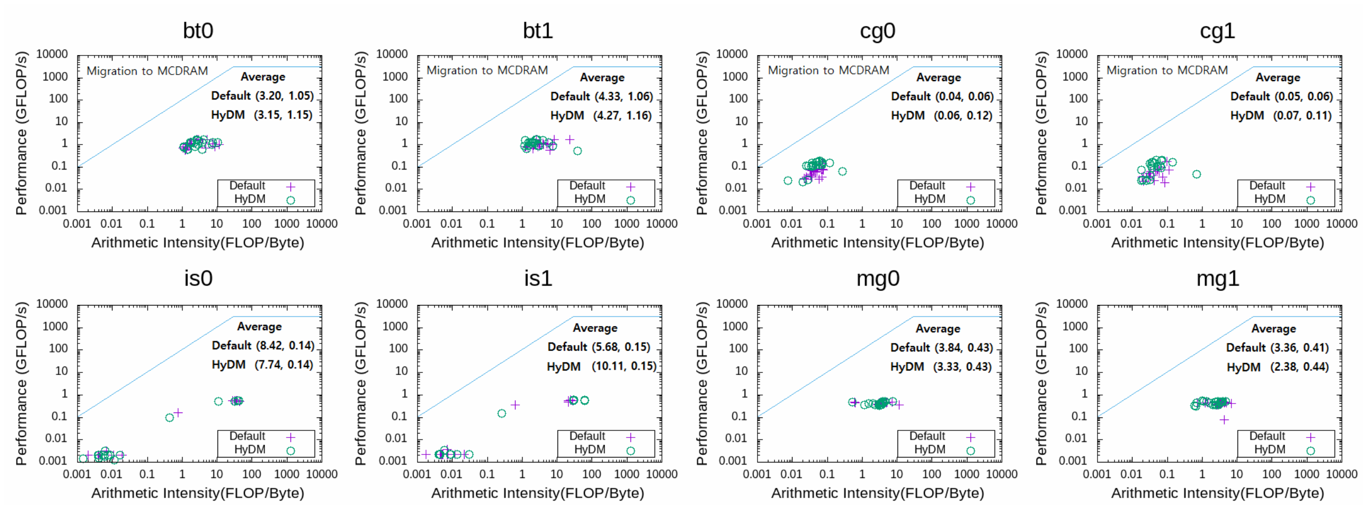

Figure 8. Roofline performance model

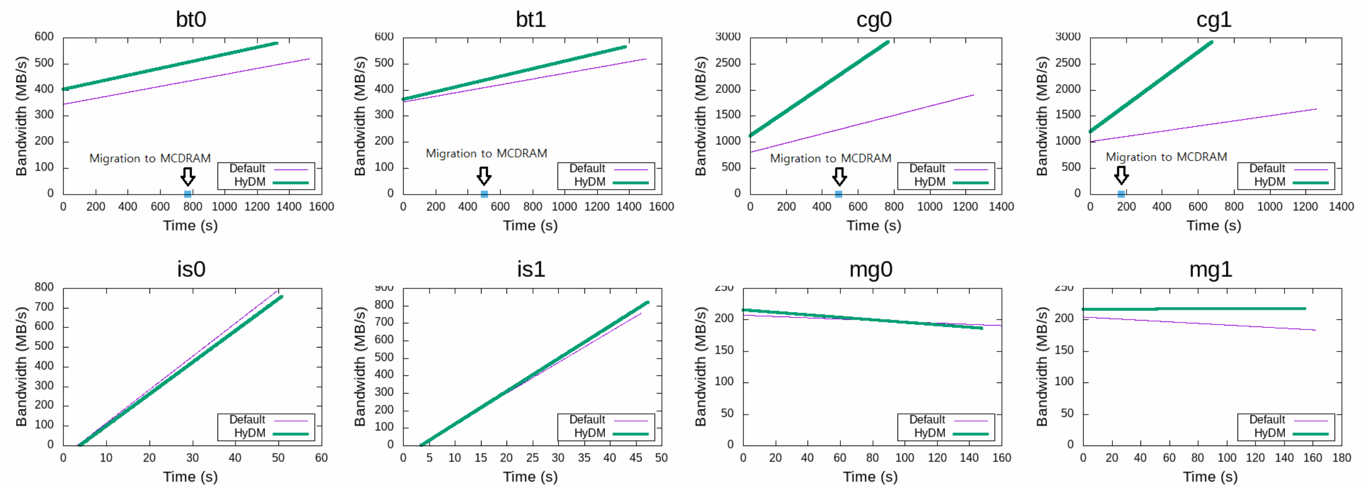

Figure 9. Bandwidth results with linear regression

order to indicate the benchmarks that migrated to MCDRAM with HyDM, “Migration to MCDRAM” is shown on the figures. In this case, *bt* and *cg* were migrated to MCDRAM and *is* and *mg* were not migrated. In *bt* benchmarks, the average floating point operations increased by about 9% and the *cg* benchmarks showed about 80% increases in floating point operations. In *cg* benchmarks, when HyDM was applied, floating point operations are noticeably increased (0.06 to 0.12). It can be interpreted as a lot of memory requests being used for floating point operations.

Figure 9 shows the bandwidth results of applications that performed linear regression for the same case as described above. The x-axis represents time and the y-axis represents bandwidth. We graphically show when the data migration happened. In *bt* and *cg*, the bandwidth was increased with HyDM. However, *is* and *mg*, the changes of bandwidth are small because migration is not performed with HyDM.

Finally, Figure 10 provides a visual representation of the case where migration to DRAM occurs. The left figure shows the execution time of each application and the time of migrations, and the right figure shows the accumulated number of applications on MCDRAM. In left figure, the occurrence of the migration to MCDRAM is indicated by a cross (x), and the occurrence of migration to DRAM is indicated by a star (*). In particular, the candidate application that caused the migration to DRAM is shown in a small square. Because migration back to DRAM may cause performance degradation, HyDM adopted a strict policy. In this case, migration to DRAM occurred twice in total. When 32 applications were executed, up to 15 applications were executed on MCDRAM as shown in the right figure.

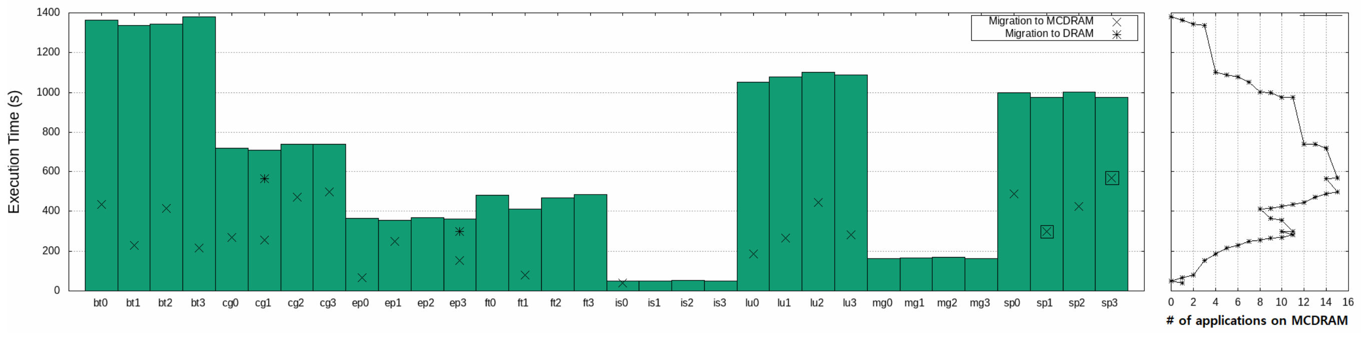

Figure 10. Data migration case analysis

6 Conclusion

The hybrid memory is a promising memory technology for future HPC systems. However, effective use of the system is becoming increasingly difficult as the HPC environment is diversifying. In this paper, we proposed a dynamic data migration strategy using the roofline performance model called HyDM. HyDM uses a hardware monitoring tool to observe the status of programs running on the system and perform migration based on the collected data. Also, a feedback mechanism is implemented for the case where the total memory usage used for the programs is larger than the size of the high bandwidth memory. We demonstrated that the proposed HyDM significantly improves the performance of mixed application sets on the Intel KNL processor. HyDM enhances performance by up to 44% compared to the baseline execution time. In the future, we plan to change our target system from a single node system to a multi-node system, which includes multiple HPC processors connected by network. We are also planning to adapt the proposed scheme to various computing environments such as cache memory, big data, and network systems [30-34].

Acknowledgments

This Research was supported by the Korea Institute of Science and Technology Information (KISTI).

References

[1] A. Sodani, Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor, *IEEE Hot Chips 27 Symposium*, Cupertino, CA, USA, 2015, pp. 1-24.

[2] R. R. Ronen, Larrabee - A Many-Core Intel Architecture for Visual Computing, *the 6th ACM conference on Computing frontiers (ACM CF)*, Ischia, Italy, 2009, pp. 225-225.

[3] A. K. Singh, M. Shafique, A. Kumar, J. Henkel, Mapping on Multi/Many-core Systems - Survey of Current and Emerging Trends, *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, USA, 2013, pp. 1-10.

[4] I. B. Peng, R. Gioiosa, G. Kestor, P. Cicotti, E. Laure, S. Markidis, Exploring the Performance Benefit of Hybrid Memory System on HPC Environments, *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Lake Buena Vista, FL, USA, 2017, pp. 683-692.

[5] I. B. Peng, R. Gioiosa, G. Kestor, J. S. Vetter, P. Cicotti, E. Laure, S. Markidis, Characterizing the Performance Benefit of Hybrid Memory System for HPC Applications, *Parallel Computing*, Vol. 76, pp. 57-69, August, 2018.

[6] A. Lopes, F. Pratas, L. Sousa, A. Ilic, Exploring GPU Performance, Power and Energy-efficiency Bounds with Cache-aware Roofline Modeling, *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Santa Rosa, CA, USA, 2017, pp. 259-268.

[7] O. Mutlu, Memory Scaling: A Systems Architecture Perspective, *2013 5th IEEE International Memory Workshop (IMW)*, Monterey, CA, USA, 2013, pp. 21-25.

[8] I. Jabbie, G. Owen, B. Whiteley, Performance Comparison of Intel Xeon Phi Knights Landing, *SIAM Undergraduate Research Online*, Vol. 10, pp. 268-281, December, 2017.

[9] S. W. Williams, A. Waterman, D. A. Patterson, *Roofline: An Insightful Visual Performance Model for Floating-point Programs and Multicore Architectures*, Technical Report No. UCB/EECS-2008-134, October, 2008.

[10] Colfax, *Clustering Modes in Knights Landing Processors*, 2016.

[11] J. Jeffers, J. Reinders, A. Sodani, Knights Landing architecture, *Intel Xeon Phi Processor High Performance Programming*, 2nd Edition, Morgan Kaufmann, 2016.

[12] D. Bailey, J. Bartion, T. Lasinski, H. Simon, *The NAS Parallel Benchmarks*, Technical Report RNR-91-002, NASA Ames Research Center, August 1991.

[13] V. M. Weaver, Linux perf_event Features and Overhead, *Second International Workshop on Performance Analysis of Workload Optimized Systems (FastPath Workshop)*, Austin, TX, USA, 2013, pp. 1-7.

[14] J. Treibig, G. Hager, G. Wellein, LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments, *2010 39th International Conference on Parallel Processing Workshops (ICPPW)*, San Diego, CA, USA, 2010, pp. 207-216.

[15] L. E. Ramos, E. Gorbatov, R. Bianchini, Page Placement in Hybrid Memory Systems, *International Conference on Supercomputing (ICS)*, Tucson, Arizona, USA, 2011, pp. 85-95.

[16] D. Shin, S. Park, S. Kim, K. Park, Adaptive Page Grouping for Energy Efficiency in Hybrid PRAM-DRAM Main Memory, *ACM Research in Applied Computation Symposium*, San Antonio, Texas, USA, 2012, pp. 395-402.

[17] J. Meena, S. Sze, U. Chand, T.-Y. Tseng, Overview of Emerging Nonvolatile Memory Technologies, *Nanoscale Research Letters*, vol. 9, no. 1, p. 526, September, 2014.

[18] R. F. Freitas, W. W. Wilcke, Storage-class Memory: The Next Storage System Technology, *IBM Journal of Research and Development*, vol. 52, no. 4.5, pp. 439-447, July, 2008.

[19] M. K. Qureshi, V. Srinivasan, J. A. Rivers, Scalable High Performance Main Memory System Using Phase-change Memory Technology, *ACM SIGARCH Computer Architecture News*, Vol. 37, No. 3, pp. 24-33, June, 2009.

[20] Y. Ro, M. Sung, Y. Park, J. H. Ahn, Selective DRAM Cache Bypassing for Improving Bandwidth on DRAM/NVM Hybrid Main Memory Systems, *IEICE Electronics Express*, Vol. 14, No. 11, pp. 20170437-20170437, May, 2017.

[21] G. Dhiman, R. Z. Ayoub, T. Rosing, PDRAM - A Hybrid PRAM and DRAM Main Memory System, *2009 46th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2009, pp. 664-669.

[22] K. Nakano, The Hierarchical Memory Machine Model for GPUs, in *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW)*, Cambridge, MA, USA, 2013, pp. 591-600.

[23] J. Choi, D. Bedard, R. J. Fowler, R. W. Vuduc, A Roofline Model of Energy, *2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS)*, Boston, MA, USA, 2013, pp. 661-672.

[24] A. Ilic, F. Pratas, L. Sousa, Cache-aware Roofline Model - Upgrading the Loft, *Computer Architecture Letters*, Vol. 13, No. 1, pp. 21-24, January-June, 2014.

[25] D. Doerfler, J. Deslippe, S. Williams, L. Oliker, B. Cook, T. Kurth, M. Lobet, T. M. Malas, J.-L. Vay, H. Vincenti, Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor, *International Conference on High Performance Computing (ISC Workshops)*, Frankfurt, Germany, 2016, pp. 339-353.

[26] W. J. Bolosky, R. P. Fitzgerald, M. L. Scott, Simple But Effective Techniques for NUMA Memory Management, *The twelfth ACM symposium on Operating systems principles (SOSP)*, Litchfield Pk., AZ, USA,1989, pp. 19-31.

[27] R. P. LaRowe, C. S. Ellis, M. A. Holliday, Evaluation of NUMA Memory Management through Modeling and Measurements, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 6, pp. 686-701, November, 1992.

[28] Z. Majo and T. R. Gross, Memory Management in NUMA Multicore Systems: Trapped between Cache Contention and Interconnect Overhead, *The International Symposium on Memory Management (ISMM)*, San Jose, California, USA, 2011, p. 11-20.

[29] L. Noordergraaf, R. van der Pas, Performance Experiences on Sun's Wildfire Prototype, *ACM/IEEE Conference on Supercomputing*, Portland, OR, USA, 1999, pp. 1-16.

[30] W. Zhang, Z. Zhang, H.-C. Chao, Cooperative Fog Computing for Dealing with Big Data in the Internet of Vehicles: Architecture and Hierarchical Resource Management, *IEEE Communications Magazine*, Vol. 55, No. 12, pp. 60-67, December, 2017.

[31] H. F. Rashvand, H. C. Chao, *Dynamic Ad-Hoc Networks*, The Institution of Engineering and Technology, 2013.

[32] F.-H. Tseng, W.-C. Chien, S.-J. Wang, C. F. Lai, H.-C. Chao, A Novel Cache Scheme based on Content Popularity and User Locality for Future Internet, *2018 27th Wireless and Optical Communication Conference (WOCC)*, Hualien, Taiwan, 2018, pp.1-5.

[33] A. Pollard, *Flow in Tee Junction*, Ph.D. Thesis, University of London, London, UK, 1978.

[34] R. Agrawal, R. Rajan, Performance Bounds for Guaranteed and Adaptive Services, *IBM Research Report RC 20649*, December, 1996.

Biographies

Jongmin Lee received the integrated master's Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology, Korea, in 2015. Dr. Lee joined the faculty of the Department of Computer Engineering at Wonkwang University, Korea, in 2018. He is interested in computer architectures, memory systems, and embedded systems/software.

Kwangho Lee received the B.S. degree in computer engineering from Wonkwang University, Korea, in 2019. He is currently a graduate student at Wonkwang University, Korea. He is interested in embedded systems/software, operating systems, computer architectures.

Mucheol Kimm is a faculty in the School of Computer Science and Engineering at Chung-Ang University. He received the BS, MS, Ph.D. degrees from the school of Computer Science and Engineering at Chung-Ang University, Seoul, Korea in 2005, 2007 and 2012, respectively. He was an assistant professor in a department of computer & software engineering at Wonkwang University (2017-2018). In 2014-2016, he was an assistant professor of Department of Media Software at Sungkyul University, Korea. In 2011-2014, he had been working as a Senior

Researcher in Korea Institute of Science and Technology Information (KISTI), Daejeon, Korea. His research interests include Data Mining, Information Retrieval, Web Technology, Social Networks and High Performance Computing

Geunchul Park is currently a researcher at center for development of supercomputing system in the National Institute of Supercomputing and Networking at Korea Institute of Science and Technology Information (KISTI). His research interests are in high performance and distributed computing, parallel program optimization, system software in HPC, etc.

Chan Yeol Park is currently a principal researcher at The Center for Development of Supercomputing System in Korea Institute of Science and Technology Information (KISTI). His research interests are in the integrated implementation of HPC system with fault tolerance and performance optimization, etc.