HyDM: Data Migration Methodology for Hybrid Memories targeting Intel Knights Landing Processor

Jongmin Lee, Kwangho Lee, Kidong Yun
Department of Computer Engineering
Won-Kwang University
Iksan, Korea
{square55, lkh002, s2mos2}@wku.ac.kr

Mucheol Kim
School of Software
Chung-Ang University
Seoul, Korea
Mucheol.kim@gmail.com

Geunchul Park, Chan Yeol Park
National Institute of Supercomputing and Networking, KISTI
Daejeon, Korea
{gcpark, 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 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 22%.**

***Index Terms*—performance, data migration, roofline model**

I. 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 Muti-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 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. The roofline performance model is a simple and visual model that offers insights for performance analysis [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.

In this paper, we propose a data migration strategy for hybrid memories (HyDM). HyDM periodically monitors the application's execution status and it selects appropriate applications, which require more memory bandwidth. By migrating pages of the applications to the high bandwidth memory (i.e. MCDRAM), HyDM improves the memory usages on 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 Intel KNL processor. HyDM enhances performance by up to 22% compared to the baseline execution time.

The rest of this paper is organized as follows. We provide background in the next section. Section III presents our proposed data migration strategy using the roofline model. Experimental results are given in Section IV. Section V presents related prior works. Section VI concludes this paper.

II. Background

A. Intel Knight Landing (KNL) Processor

In this section, we briefly summarize the main features of the Intel KNL processor, especially we focus on its memory system.

Fig. 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

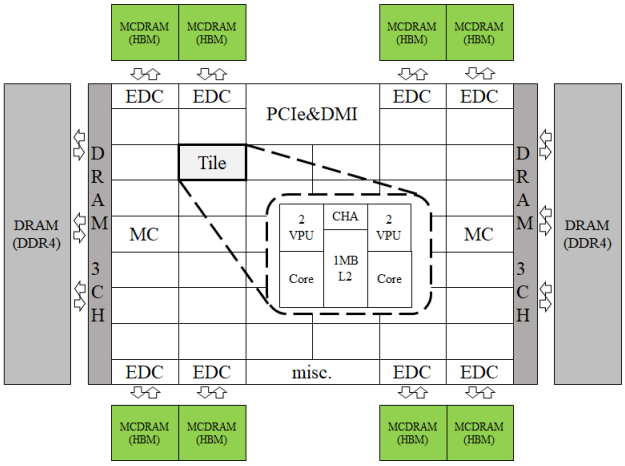

Fig. 1. A structure of Intel Knight Landing processor

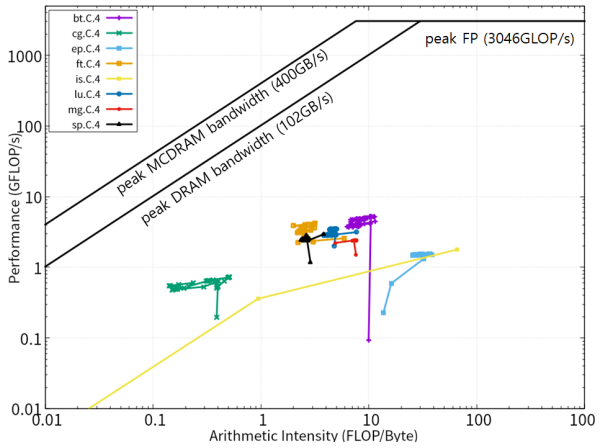

Fig. 2. Roofline performance model

MCDRAMs are positioned on-chip while DRAMs are off-chip. Fig. 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 only 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]. MCDRAM 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].

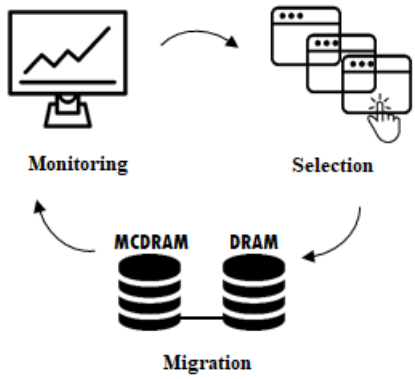

Fig. 3. HyDM Methodology Overview

B. 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]. 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.

Fig. 2 shows the roofline model of Intel KNL processor with NAS parallel benchmark suites [12]. We periodically record the position of each benchmark to see performance changes of each benchmark over time. Detailed experimental methodologies will be shown in Section IV-A. The black-colored lines 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 Fig. 2, most benchmarks are changing their positions over time and they 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.

III. Proposed Techniques

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

A. Overview

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

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

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 point, historical monitored data from each application are stored in the list $W = \{W_0, W_1, ..., W_n\}$ (line 2). Note that 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. 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.

B. 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 the ith 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.

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
V : the miss predicted 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 in V then continue
  4. if $S_i.pf > pf_{avg}$ then continue
  5. $b \leftarrow getAppId(S_i, L)$
  6. return b
end for

C. 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.

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. 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. We first check that the candidate application has a historical record of miss prediction (V). The list V generated in *Migration* procedure stores applications that did not show performance improvement after migration (line 3). In order to filter applications with low memory locality, HyDM employs

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
/*
V : the list of miss predicted applications
r : the index of application for rollback
t : the threshold ratio of MCDRAM use
*/
if $isMigrationPossible(t, b)$ **then**
1. $migrationToMCDRAM(b, W)$
else
2. $r \leftarrow checkFlops(W)$
3. if r exists then
4. insert r to V
5. $migrationToDRAM(r, W)$
6. end if
end if

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 averaged number of page faults (line 4). If all operations are finished, the *Selection* procedure returns the candidate application b for migration (line 6).

D. Migration

Algorithm 4 shows the *Migration* procedure. Because of the limited capacity 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 t, the entire page of the selected application is migrated to MCDRAM (line 1). Although we migrate the entire page to maintain the simplicity of the HyDM, the page grouping techniques for selecting the critical pages of the entire page in the application are applicable to our proposed scheme [15], [16].

When the usage of MCDRAM is larger than t, we check the changes in flops for applications, which have migrated to MCDRAM (line 2). If the flops of an application have improved at least once during the time windows compared to the previous flops, we give more chances to the application to be in MCDRAM. Since migrating pages frequently induces additional overheads in terms of performance and energy, we employ a strict methodology for rollback. If we found the application that flops does not change, the *checkFlops* returns the application id r. After that, the application is inserted to the list V to record the miss prediction and pages of the application are migrated to the DRAM again (line 3-6). By employing the feedback mechanism above, HyDM effectively uses hybrid memories when many applications are running on a system.

IV. Experimental Results

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

A. 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 I shows the benchmark execution results when they are run on the system alone including averaged execution times (twenty times), floating-point operations, memory accesses, and the amount of peak memory use, respectively.

Table II shows the design parameters of HyDM and their values set in evaluations. The minimum time unit that can be monitored through *perf_event* is 1ms. When hardware events for monitoring are executed frequently, however, performance degradation occurs. We adjusted the numerical values without affecting performance through a heuristic method. To assume the system situation in which applications that require much larger capacity than the capacity of the MCDRAM are running, we perform the NPB programs in the number of multiples.

B. Performance Evaluation

Fig. 4 shows the changes in the roofline model when we run 16 NPB programs in parallel. We have selected several programs due to page limitation. On the roofline model, we

TABLE I
NAS Parallel Benchmark (NPB) characteristics

Name	Average execution time (sec.)	FP operations (GFlop)	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 / 810	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 II
HyDM parameters

Descriptions	Values
The time window period: p	100 (ms)
The number of applications: $length(L)$	16, 32
Size of monitoring windows: $W = \{W_0, W_1, ..., W_n\}$	5, 10
The threshold ratio of MCDRAM use: t	90 (%)

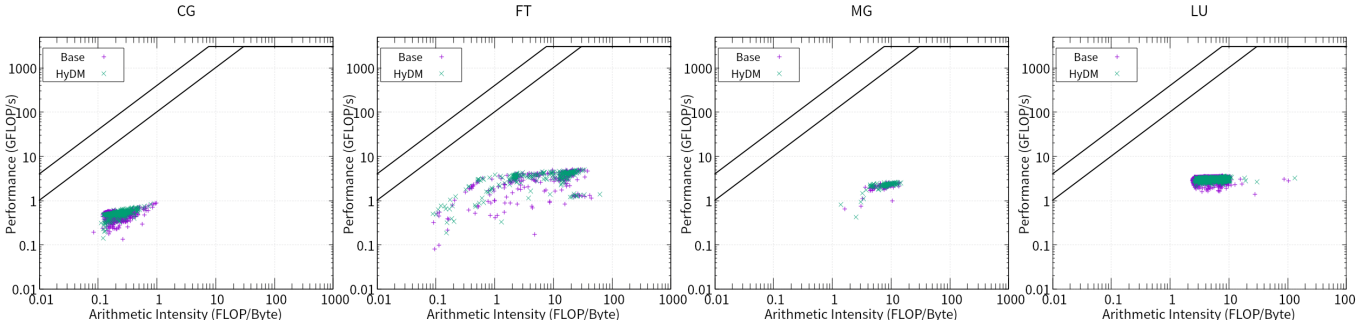

Fig. 4. Roofline performance models

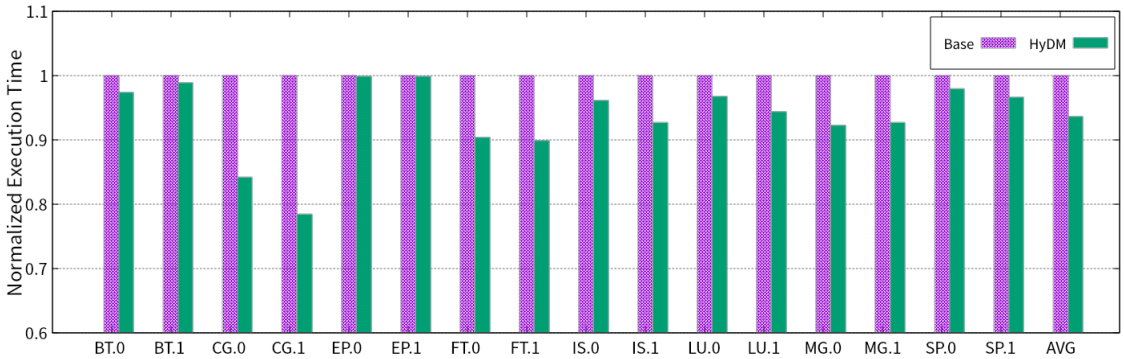

Fig. 5. Averaged execution time results

can see that HyDM works effectively through the rise of the overall flops values.

To evaluate the performance impact of the proposed HyDM, we randomly assigned programs to the cores and calculated the average execution times. Fig. 5 show the averaged execution times for 100 runs. The results are normalized to the baseline. Several benchmarks show large increases in the execution time such as CG.0(15%), CG.1(22%), and FT.1(10%). On average, the reduced execution time is 6.5% with HyDM. This performance improvement of HyDM is due to memory-intensive applications are effectively migrated to MCDRAM.

V. 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]–[28]. 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.

VI. 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. Experiments using a real system, including the Intel KNL processor, we demonstrate that the proposed HyDM improves system performance.

Acknowledgment

This work was supported by Korea Institute of Science and Technology Information (KISTI) grant funded by the Korea government (MSIT) (No.K-18-L12-C07-S01).

References

[1] A. Sodani, "Knights landing (KNL): 2nd Generation Intel® Xeon Phi processor," in *IEEE Hot Chips 27 Symposium*. IEEE, 2015, pp. 1–24.

[2] R. Espasa, "Larrabee - A Many-Core Intel Architecture for Visual Computing." *HiPEAC*, vol. 5952, no. Chapter 2, pp. 2–2, 2010.

[3] A. K. Singh, M. S. 0001, A. K. 0001, and J. Henkel, "Mapping on multi/many-core systems - survey of current and emerging trends." *DAC*, p. 1, 2013.

[4] I. B. Peng, R. Gioiosa, G. Kestor, P. Cicotti, E. Laure, and S. Markidis, "Exploring the Performance Benefit of Hybrid Memory System on HPC Environments," in *IPDPSW*. IEEE, 2017, pp. 683–692.

[5] I. B. Peng, R. Gioiosa, G. Kestor, J. S. Vetter, P. Cicotti, E. Laure, and S. Markidis, "Characterizing the performance benefit of hybrid memory system for HPC applications," *Parallel Computing*, vol. 76, pp. 57–69, 2018.

[6] A. Lopes, F. Pratas, L. Sousa, and A. Ilic, "Exploring GPU performance, power and energy-efficiency bounds with Cache-aware Roofline Modeling," in *ISPASS*. IEEE, 2017, pp. 259–268.

[7] O. Mutlu, "Memory scaling: A systems architecture perspective," in *IMW*. IEEE, 2013, pp. 21–25.

[8] I. Jabbie, "Performance Comparison of Intel Xeon Phi Knights Landing," *SIAM Undergraduate Research Online*, vol. 10, 2017.

[9] S. W. Williams, A. Waterman, and D. A. Patterson, "Roofline: An insightful visual performance model for floating-point programs and multicore architectures," EECS Department, University of California, Berkeley, Tech. Rep., 2008.

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

[11] J. Jeffers, J. Reinders, and A. Sodani, *Knights Landing architecture*. Morgan Kaufmann, Jan. 2016.

[12] D. H. Bailey, "NAS Parallel Benchmarks." *Encyclopedia of Parallel Computing*, no. Chapter 133, pp. 1254–1259, 2011.

[13] V. M. Weaver, "Linux perf_event Features and Overhead," in *FastPath Workshop*, 2013.

[14] J. Treibig, G. Hager, and G. Wellein, "LIKWID: A Lightweight Performance-Oriented Tool Suite for x86 Multicore Environments," in *ICPPW*. ACM, 2010, pp. 207–216.

[15] L. E. Ramos, E. Gorbatory, and R. Bianchini, "Page placement in hybrid memory systems," in *ICS*. ACM, 2011, pp. 85–95.

[16] D. Shin, S. Park, S. Kim, and K. Park, "Adaptive page grouping for energy efficiency in hybrid PRAM-DRAM main memory," in *ACM Research in Applied Computation Symposium*. ACM, 2012, pp. 395–402.

[17] J. Meena, S. Sze, U. Chand, and T.-Y. Tseng, "Overview of emerging nonvolatile memory technologies," *Nanoscale Research Letters*, vol. 9, no. 1, p. 526, 2014.

[18] R. F. Freitas and 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, 2008.

[19] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, p. 24, Jun. 2009.

[20] Y. Ro, M. Sung, Y. Park, and 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. 20 170 437–20 170 437, 2017.

[21] G. Dhiman, R. Z. Ayoub, and T. Rosing, "PDRAM - a hybrid PRAM and DRAM main memory system." *DAC*, p. 664, 2009.

[22] K. Nakano, "The Hierarchical Memory Machine Model for GPUs," in *IPDPSW*. IEEE, 2013, pp. 591–600.

[23] J. Choi, D. Bedard, R. J. Fowler, and R. W. Vuduc, "A Roofline Model of Energy." *IPDPS*, pp. 661–672, 2013.

[24] A. Ilic, F. Pratas, and L. Sousa, "Cache-aware Roofline model - Upgrading the loft." *Computer Architecture Letters*, vol. 13, no. 1, pp. 21–24, 2014.

[25] D. Doerfler, J. Deslippe, S. Williams, L. Oliker, B. Cook, T. Kurth, M. Lobet, T. M. Malas, J.-L. Vay, and H. Vincenti, "Applying the Roofline Performance Model to the Intel Xeon Phi Knights Landing Processor." *ISC Workshops*, vol. 9945, no. 16, pp. 339–353, 2016.

[26] W. J. Bolosky, R. P. Fitzgerald, and M. L. Scott, "Simple But Effective Techniques for NUMA Memory Management." *SOSP*, pp. 19–31, 1989.

[27] R. P. LaRowe, C. S. Ellis, and 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, 1992.

[28] Z. Majo and T. R. Gross, "Memory management in NUMA multicore systems," in *ISMM*. New York, New York, USA: ACM Press, 2011, p. 11.

[29] L. Noordergraaf and R. van der Pas, "Performance experiences on Sun's Wildfire prototype," in *ACM/IEEE conference on Supercomputing*. New York, New York, USA: ACM Press, 1999, pp. 38–es.