eMDyn: 하이브리드 메인 메모리 시스템에서의 동적인 객체 배치 및 이주

Safdar Jamil[1], **김태욱**[2], **박중언**[1], **김영재**[1]
[1]**서강대학교 컴퓨터공학과** [2]**티맥스소프트**
{**safdar, joongeon, youkim**}**@sogang.ac.kr & taeuk_kim@tmax.co.kr**

eMDyn: Dynamic Object Placement and Migration on Hybrid Main Memory System

Safdar Jamil[1], **Taeuk Kim**[2], **Joongeon Park**[1], **Youngjae Kim**[1]
[1]**Sogang University,** [2]**TmaxSoft, South Korea**

요 약

In scientific computing environments, main memory contributes a huge share in power and energy consumption. Recent developments in-memory systems such as Non-Volatile Memory (NVM) devices and Hybrid Main Memory Systems (HMMSs) comprised of DRAM and NVM have the potential for energy efficiency. However, the characteristics of NVM devices in HMMS deteriorate the performance of the system due to higher access latencies than DRAM. In this paper, we extend the methodology proposed in [1] and provide a dynamic module for the runtime of the application to shuffle the object according to the energy constraint. Our methodology also takes into account the access patterns of memory objects and device characteristics. The proposed model for object placement is based on Integer-Linear Programming (ILP). We showed that our methodology optimally migrates the object by reducing the energy efficiency up to 4% with the same performance.

1 Introduction

Scientific applications utilize a large number of resources of data centers, which cause a huge power and energy consumption. It is known that about 30% of this contribution only comes from memory-level (DRAM). To minimize energy consumption, non-volatile memory (NVM) devices such as Spin-Transfer Torque RAM (STT-RAM) and Phase Change Memory (PCM) can be acquired [2]. The attributes of NVMs such as byte-addressability and low energy dissipation than DRAM make them a viable candidate for main memory. However, the access latencies of NVM devices are high to replace DRAM completely yet. So, various Hybrid Main Memory System (HMMS) architectures comprised of DRAM and NVM have been proposed. The placement of memory objects in HMMS becomes challenging as it must retain the performance and reduce energy efficiency. Various works [1, 2] have achieved this goal by placing the memory objects in HMMS. None of the existing studies have considered the change in energy consumption during the execution of the application.

In this paper, we extended the methodology proposed by Taeuk et. al [1] to provide a dynamic object placement by considering the change in energy requirement during the execution of the application and shuffle the objects accordingly to optimize the performance and reduce energy efficiency. Our approach is based on two modules, (i) migration planner which provide a new placement for each object by considering their access patterns and migration cost in terms of energy and time taken to migrate, and (ii) a migration executor which shuffles the memory objects according to their new placements. We evaluated our solution using NAS parallel benchmark and showed that our methodology optimally migrates the memory object and reduces the energy efficiency up to 4% with the same performance.

2 Background & Related Work

2.1 Memory Access Patterns of Scientific Applications

Scientific applications operate for several days and access memory frequently for computation. Heap space is majorly utilized by the scientific applications to dynamically allocate variables. Majorly read variables pf scientific applications change their access patterns as the workload of the application change. Ji et. al [3] analyzes the various access patterns, such as lifetime, accessed volume, localities in page-level and cache line level, of memory objects for various applications grouped in two categories, scientific and general applications. It showed that most of the applications from the scientific group have regular scaling patterns where 126 out of 127 variables were scaling or having fixed access patterns with the scaled workload.

2.2 Related Work

To optimize the performance and reduce the energy efficiency in HMMS, various works have been done [1, 2]. One of the work proposed an optimal placement of memory object based on object profiling, an ILP based placement planner, and a runtime object allocator [1]. Object profiling profiles the application and extracts the access patterns before the runtime of the application. The placement planner is based on 3 ILP constraints, (i) Decision Constraint, (ii) Capacity Constraint, and (iii) Energy Constraint. Taeuk et. al proposed solution only provides the static placement for a specific energy limiting constraint. Once the energy requirement change, we have to terminate the application and obtain the placement of an object according to the new energy constraint and then re-run the application.

3 Design and Implementation

In this section, we will provide the design and implementation overview of our proposed methodology. Inspired from the need to consider the change in energy requirement during the application execution on the HMMS environment, we propose eMDyn which re-evaluates the placement of memory objects placed on HMMS and shuffles them to meet the energy constraint and optimize the performance. Figure 1 shows the overall flow of our approach. The right side of Figure 1 shows that an application placed various memory objects in both of the memory modules respectively. While the left side of the Figure 1 is consist of 3 phases, (i) Profiling, (ii) Planning, and (iii) Runtime Phase. At the bottom of Figure 1, it shows the overall scenario of the application execution where the change of energy constraint is requested and the runtime phase of our proposed solution is triggered, it obtains the new placement policies of objects and migrates the objects accordingly.

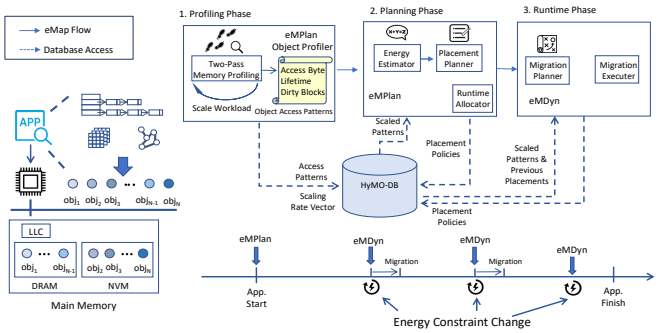


그림 1: System Flow of eMDyn Framework

3.1 Profiling Phase

The profiling phase is the first module of the proposed methodology by [1]. In this work, we adopted as it is designed and implemented in [1] and profile the applications before the execution of the application and extract the access patterns of memory objects and store them at Hybrid Memory Object Database(HyMO-DB) in Figure 1.

3.2 Planning Phase

The Planning phase shown in Figure 1 is adopted from the proposed solution [1]. This phase is an ILP-based object placement planner which calculates the placement of objects by considering 3 constraints, (i) Decision, (ii) Capacity, and (iii) Energy Constraint. We also take into account the per-bit energy model for DRAM and STT-RAM based on prior work [4].

3.3 Runtime Phase

After obtaining the object placements, the runtime phase allocates the memory objects with specific energy limiting constraints. During the application execution, a request from the user can be placed to optimize more performance or reduce more energy efficiency. To consider that request, we proposed eMDyn which interrupts the application execution and re-evaluates the object placement with a new request. eMDyn is also an ILP-based placement planner and consists of two modules, Migration Planner and Migration Executer.

3.3.1 Migration Planner

Inspired from the Placement Planner proposed in [1], Migration Planner is also based on ILP-based planning constraint and considered 3 constraints, Decision, Capacity and Energy Constraints.

I *Decision Constraint:* The decision constraint of eMDyn is adopted from the decision constraint of the Placement Planner proposed in [1].

II *Capacity Constraint:* When migrating the objects from the previous memory module to the new memory device, we consider the overall capacity of the memory devices. CP_i in Equation 1 represents the previous placement of the object while C_D and C_N are the capacities of DRAM and STT-RAM respectively.

$$\sum_{i=1}^{N} \{X_i \cdot CP_i \cdot S_i + (1 - X_i) \cdot (1 - CP_i) \cdot S_i\} \leq C_N$$
$$\sum_{i=1}^{N} \{X_i \cdot (1 - CP_i) \cdot S_i + (1 - X_i) \cdot CP_i \cdot S_i\} \leq C_D \quad (1)$$

III *Energy Constraint:* To consider the energy change requests during the application execution, we modeled this constraint

표 1: Notations used in Modeling

Notations	Description
dE_{AP}	DRAM activate/precharge energy
AV_i	*i*th object access volume
dE_{RW}	DRAM read/write energy
dE_{REF}	DRAM refresh energy
S_i	*i*th object size
L_i	*i*th object lifetime
nE_{AP}	STT-RAM activate/precharge energy
nE_{RBA}	STT-RAM row buffer access energy
sE_{WB}	STT-RAM writeback energy
N_{dc}	Number of dirty cachelines
V_{CL}	Cache line Size (64 byte)
C_D	DRAM capacity
C_N	STT-RAM capacity
CP_i	Previous Allocation of i-th object
ndE_i	i-th object migration energy from NVM to DRAM
dnE_i	i-th object migration energy from DRAM to NVM
T_i	i-th object lifetime
sT_i	i-th object allocation time
fT_i	i-th object de-allocation time
ndL_i	Total latency of i-th object when migrated from NVM to DRAM
dnL_i	Total latency of i-th object when migrated from DRAM to NVM
$MigTD_i$	Time taken to migration i-th object from DRAM to NVM
$MigTN_i$	Time taken to migration i-th object from NVM to DRAM

which considers each object's energy consumption and migration cost from DRAM to NVM, vice-versa. Equation 2 represents the energy consumption when i-th object is migrated from DRAM to NVM. The $MigCostE1_i$ is the migration cost in terms of the energy and it is shown in Equation 3.

$$dnE_i = DE_i \cdot \frac{t - sT_i}{T_i} + MigCostE1_i + NE_i \cdot \frac{fT_i - t}{T_i} \quad (2)$$

$$MigCostE1_i = (dE_{AP} \cdot S_i + dE_{RW} \cdot S_i + dE_{REF} \cdot S_i \cdot MigCostT_i) + (nE_{AP} \cdot S_i + nE_{RBA} \cdot S_i + nE_{WB} \cdot N_{DC} \cdot V_{CB}) \quad (3)$$

Equation 4 shows the energy consumption of i-th object when it is migration from NVM to DRAM and the migration cost in terms of energy is shown in Equation 5.

$$ndE_i = NE_i \cdot \frac{t - sT_i}{T_i} + MigCostE2_i + DE_i \cdot \frac{fT_i - t}{T_i} \quad (4)$$

$$MigCostE2_i = (nE_{AP} \cdot S_i + nE_{RBA} \cdot S_i) + (dE_{AP} \cdot S_i + dE_{RW} \cdot S_i + dE_{REF} \cdot S_i \cdot MigCostT_i) \quad (5)$$

The total energy consumption of all the memory objects can be calculated as shown in Equation 6. The required energy consumption for a particular energy limiting constraint provided by the user can be calculated as shown in Equation 7. The flag in Equation 7 represents the cases according to the user requirement. The first case is when the user wants to meet the energy requirement regardless of any means and the second case is when the user just required to reduce the energy efficiency. The flag variable represents these two cases in terms of a binary flag where 0 represents the first case while 1 represents the second case.

$$E_t = \sum_{i=1}^{N} \Big[X_i \cdot \{CP_i \cdot dnE_i + (1 - CP_i) \cdot ndE_i\} + (1 - X_i) \cdot \{CP_i \cdot dE_i + (1 - CP_i) \cdot nE_i\} \Big] \quad (6)$$

$$Req = flag \cdot [\sum_{i=1}^{N} dE_i \cdot R_{new}] + (1 - flag) \cdot [\sum_{i=1}^{N} \{CP_i \cdot dE_i + (1 - CP_i) \cdot nE_i\}] \quad (7)$$

Now to fulfills the required energy, the total energy consumption should be less than the required energy consumption which is shown in Equation 8.

$$E_t \leq Req \quad (8)$$

IV *Objective Function:* To optimize the performance, we propose the objective function which considers the access latencies and the remaining lifetime of the memory device and the memory objects respectively. The total access latency of the i-th object which was first placed at DRAM and now is going to migrate to NVM is shown in Equation 9 while an object which was previously placed in NVM and migrating to DRAM is shown in Equation 11. The Migration cost in terms of the access latency for each object should also be considered as it is going to affect the performance of the application. We take into account that cost and represent that in Equation 10.

$$dnL_i = L_D \cdot L3M_i \cdot \frac{t - sT_i}{T_i} + MCT_i + L_N \cdot L3M_i \cdot \frac{fT_i - t}{T_i} \quad (9)$$

$$MCT_i = L_D \cdot S_i + L_N \cdot S_i \quad (10)$$

$$ndL_i = L_N \cdot L3M_i \cdot \frac{t - sT_i}{T_i} + MCT_i + L_D \cdot L3M_i \cdot \frac{fT_i - t}{T_i} \quad (11)$$

Thus the total delay time of the object including the migration cost can be shown Equation 12.

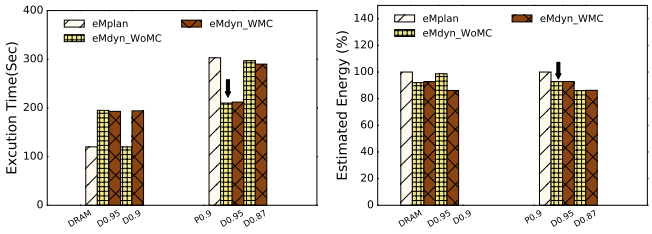
$$f = \sum_{i=1}^{N} \Big[X_i \cdot \{CP_i \cdot dnL_i + (1 - CP_i) \cdot ndL_i\} + (1 - X_i) \cdot L3M_i \cdot \{L_D \cdot CP_i + L_N \cdot (1 - CP_i)\} \Big] \quad (12)$$

3.3.2 Migration Executer

After the migration decision, Migration Executer operates to perform the migration task and shuffle the memory objects. The steps of the migration are: (i) a new object with the same size is allocated in the respective memory device, (ii) current state of the object is copied to the newly allocated object, (iii) the pointer of the candidate object is revised to newly allocated object, and (iii) previous object is de-allocated.

4 Evaluation

We evaluated the efficiency of eMDyn by changing to various energy limiting constraints in terms of execution time and estimated energy consumption. We emulated the HMMS using Quartz [5] in a 16GB memory system and our target application is CG from NAS parallel benchmark [6] with 1.08GB workload. The configuration of HMMS used for the evaluation includes 2GB DRAM and 4GB NVM and the access latencies for DRAM set to 400ns and for NVM read the latency is 840ns while the write latency is 1640ns. We adopted the methodology for energy-measurement the same as [1]. We set the major objects as heap variables greater than 1MB.



(a) Execution Time (b) Expected Energy

그림 2: Performace and energy efficiency of CG application from NPB benchmark with two different energy constraint change requests.

Figure 2 shows the performance in terms of execution time and energy efficiency of our target application. Figure 2(a) shows the execution time of the application when it started the execution with the placement policies obtained from the Planning Phase and the request of change in energy limiting constraint is placed during the application execution. $eMplan$ in Figure 2 represents the methodology of Taeuk et. al [1] while $eMdyn_WMC$ represents when we consider the migration cost in terms of energy and performance during the migration and $eMdyn_WoMC$ shows when we did not consider the migration cost.

The X-axis in the figures shows the change of energy constraint, $P0.x$ is the energy limiting constraint with the application started the execution while $D0.y$ is the energy limiting constraint which was requested during the application execution. The arrows in the figure points out the difference between two configurations $eMdyn_WMC$ and $eMDyn_WoMC$. These arrows show that when the migration cost is not being considered it does not provide an optimal placement which leads to degraded performance and energy efficiency.

5 Conclusion

We proposed a dynamic object placement algorithm which considers the change in energy requirement during the application execution and migrates the memory object accordingly and optimizes the performance and reduces the energy efficiency. The proposed approach efficiently optimizes the performance and reduces the energy efficiency up to 4% without any additional cost.

Acknowledgment

This research was supported by Next-Generation Information Computing Development Program through National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (2017M3C4A7080243).

참고 문헌

[1] S. J. K. Taeuk Kim, "Performance-Energy Mediating Object Placement on Hybrid Main Memory System," *한국정보과학회 학술발표논문집*, 2018.

[2] A. Narayan, T. Zhang, S. Aga, S. Narayanasamy, and A. K. Coskun, "MOCA: memory object classification and allocation in heterogeneous memory systems," in *IPDPS*, 2018.

[3] X. Ji, C. Wang, N. El-Sayed, X. Ma, Y. Kim, S. S. Vazhkudai, W. Xue, and D. Sanchez, "Understanding object-level memory access patterns across the spectrum," SC'17, ACM, 2017.

[4] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating stt-ram as an energy-efficient main memory alternative," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013.

[5] H. Volos, G. Magalhaes, L. Cherkasova, and J. Li, "Quartz: A lightweight performance emulator for persistent memory software," in *Proceedings of the 16th Annual Middleware Conference*, Middleware '15, ACM, 2015.

[6] D. Griebler, J. Loff, G. Mencagli, M. Danelutto, and L. G. Fernandes, "Efficient nas benchmark kernels with c++ parallel programming," in *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, 2018.