Memory Management Methodology for Application Data Structure Refinement and Placement on Heterogeneous DRAM/NVM Systems

Manolis Katsaragakis[⋆†], Lazaros Papadopoulos[⋆], Christos Baloukas[⋆], Dimitrios Soudris[⋆]
[⋆]*Microprocessors and Digital Systems Laboratory, ECE , National Technical University of Athens, Greece*
[†]*Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001 Heverlee, Belgium*
[⋆]{mkatsaragakis, lpapadop, cmpalouk, dsoudris}@microlab.ntua.gr

***Abstract*—The emergence of memory systems that combine multiple memory technologies with alternative performance and energy characteristics are becoming mainstream. Existing data placement strategies evolve to map application requirements to the underlying heterogeneous memory systems. In this work, we propose a memory management methodology that leverages a data structure refinement approach to improve data placement results, in terms of execution time and energy consumption. The methodology is evaluated on three machine learning algorithms deployed on various NVM technologies, both on emulated and on real DRAM/NVM systems. Results show execution time improvement up to 57% and energy consumption gains up to 41%.**

I. Introduction

Nowadays, the growth of applications that handle large data sets, while following the in-memory processing paradigm, is rapidly increasing. Machine Learning(ML)-enabled applications deployed at the edge of Internet of Things (IoT) networks to support near-sensor processing [1], online analytical processing (OLAP) applications managing ever-growing data sets [2], as well as scientific and big-data applications [3] that require high processing throughput without I/O bottlenecks, lead to severe pressure on the main memory of computing systems.

However, DRAM hits its physical limits [4], as well as scaling limitations, as leakage and refresh power increase with DRAM size requirements. In order to overcome these limitations, recent years have witnessed the emergence of various technologies, such as Phase-Change Memory (PCM), Resistive RAM (ReRAM), Spin Transfer Torque RAM (STT-RAM) and 3D-XPoint [5], which have distinct advantages and disadvantages over traditional memory (DRAM). Modern commercial platforms often package together conventional DRAM DIMMs with non-volatile memory devices (NVMs), such as Intel Optane DC Persistent memory [6]. Such heterogeneous memory systems trade uniform access and simplicity for developers, to achieve benefits by integrating multiple memory technologies on each computing node. A typical heterogeneous memory system consists of a large, energy efficient, but high latency (and/or low bandwidth) NVM and a fast DRAM of limited capacity and increased energy requirements.

As heterogeneous memory systems emerge, a lot of effort is dedicated by the research community to develop efficient data placement algorithms [7], [8]. These sophisticated approaches guide developers, or Operating Systems, to the efficient placement of data structures, memory objects or pages over the alternative types of memory, focusing on performance and/or energy consumption optimization. Typically, data placement algorithms are utilized as black boxes by application developers, as they receive as input the application data in their existing state, without considering any data-related optimization that would improve the execution time and the energy consumption of applications deployed on heterogeneous memory systems. Moreover, traditionally applications are developed to target conventional single DRAM systems, while the modification of them to target hybrid DRAM/NVM systems requires significant effort. Thus, the majority of application developers tend to utilize the applications as they are initially implemented. In this work, we investigate how application-level data optimizations can improve the results of existing data placement algorithms.

To motivate the need for application-level data optimizations we developed a synthetic benchmark with four data structures and various input workloads triggering alternative access patterns. Each data structure implementation is accessed either sequentially or randomly, while they behave different in terms of read and write accesses, thus not all of them are NVM-friendly. This makes finding the best implementation for each data structure quite tricky, without proper profiling and optimization tools [9]. In our scenario, all initial implementation are vectors (dynamically sized arrays), which is the default implementation in many algorithms. Our motivational example is deployed on a DRAM/PCM memory system, emulated by the HP Quartz [10]. The data placement is managed by the algorithm described in [8]. As demonstrated in Table I, changing the data structure organization of the application, the execution time, as well as the total energy consumption can be reduced. More specifically, by replacing the first and the third vectors of the original application version ($P1$) with single linked list (SLL) and the last with double linked list (DLL) ($P2$) the execution time is reduced by 57%, while the energy consumed on memory is also reduced by 5%. These results motivate a thorough investigation of the impact of high-level data organization optimizations to placement approaches for heterogeneous memories.

In this work, we propose a memory management methodology that consists of a dynamic data structure refinement step, based on approaches such as [11][12] and [9] that improves the

This work has been partially funded by EU Horizon 2020 program under grant agreement No 101015922 AI@EDGE (https://aiatedge.eu/).

TABLE I: Execution time and energy consumption results for a synthetic application deployed on a DRAM/NVM system.

Application version	Data structure configuration	Execution time (sec)	Memory energy (J)
P1	Vector, Vector, Vector, Vector	90	21.9
P2	SLL, Vector, SLL, DLL	38.4	20.8

application data organization at data structure level, and followed by sophisticated data placement algorithms that handle the assignment of data to heterogeneous memory systems. We investigate the impact of improving the data organization to the placement algorithms and provide a methodology supported by an open source tool-flow targeting application developers that enables performance and energy consumption improvements for applications deployed on heterogeneous memories.

The rest of the paper is organized as follows: The proposed methodology is described in Section II. Section III describes the experimental evaluation results and discusses the main observations. Finally, in Section IV we draw conclusions.

II. Methodology

The proposed methodology (Fig. 1) consists of three steps:

1) The *optimization of data organization*, which leverages design space exploration techniques for performing source-to-source data-flow transformations, to select efficient data structures by minimizing the application data accesses and memory footprint. Aiming to optimize the performance and the energy of an application executed in hybrid DRAM/NVM systems, the number of memory accesses is essential to tackle the high write latency of NVMs and respect its limited write endurance, while the reduced memory footprint plays significant role to limit the inherent DRAM high leakage power for data storage. It receives as input an application source code and generates a number of Pareto optimal solutions, with different memory requirements and same functionality.
2) The *memory object analysis and profiling*, which, based on memory-trace profiling tools, analyzes the application versions produced by the previous step and generates profiling information to be provided to data placement algorithms for heterogeneous memories.
3) The *placement on heterogeneous memory systems and evaluation*, in which data placement decisions for each application version are made, based on the profiling information generated. Application versions with different data organization are deployed on heterogeneous memory systems, so that gains and trade-offs are identified, between performance and/or energy consumption.

The proposed methodology aims at improving the results of existing data placement approaches for heterogeneous memories. Typical data placement approaches rely mostly on the third (and partially in some cases the second) step, resulting on a single data placement solution for each memory technology [8], [7], [13]. This work proposes the optimization of data organization as first step to further improve the results of existing data placement approaches. By refining the application data structures, more and different placement solutions are identified compared to the original application solution, thus providing (i) trade-offs between metrics such as performance and energy consumption for applications deployed on heterogeneous memory system(s) and (ii) improvements in terms of execution time and/or energy compared to the original application.

A. Optimization of Data Organization

The data organization optimization approach, which is the first step of the proposed methodology derives from data structure refinement approaches such as Brainy [11] and Dynamic Data Type Refinement (DDTR) [12]. In general, these works define a design space of customized data structure implementations. Then, by leveraging design-space exploration techniques, each implementation is evaluated for the application under optimization in terms of platform-independent metrics (e.g. data accesses and memory footprint) or platform-specific metrics (e.g. performance, cache miss ratio and energy consumption). Finally, Pareto optimal data structure implementations are identified to enable trade-offs between the aforementioned metrics.

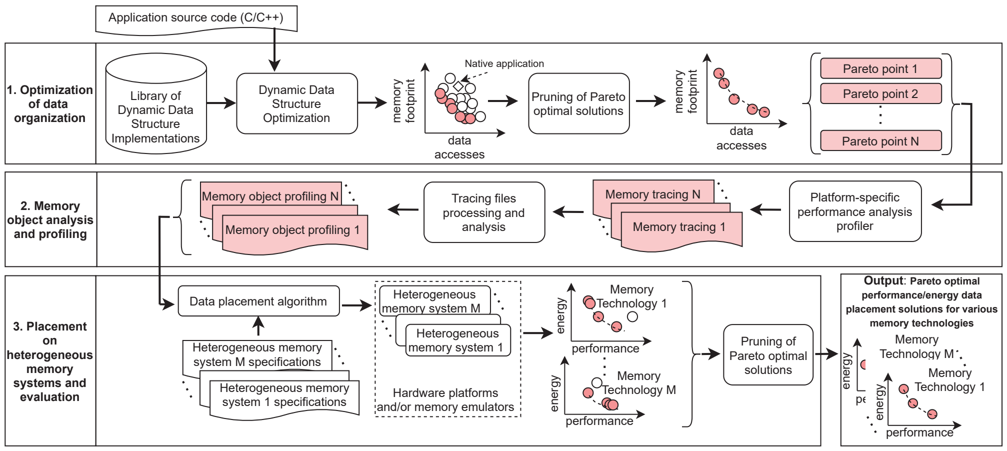
The data organization optimization approach integrated in the proposed methodology is partially based on the Dynamic Data Type Refinement (DDTR)[12], which, in this work, is substantially extended with new features, aiming to provide NVM-friendly data structure implementations and is detailed in the rest of this section.

1) Extended Design Space: Fig. 2 illustrates a simple integration example of the DDTR library interface. The extended DDTR is based on a modular library of Standard Template Library-like (STL) data structure implementations that defines a design space (Table II). However, unlike the previous approaches (e.g. [12]), a set of STL-like tree-based data structures are integrated in the design space used in this work, as shown in Table II (MAP-RB and MAP-AVL). Tree-based data structures reflect structural relationships in data, represent hierarchies and provide an efficient insertion and searching, thus limiting the number of read and write accesses, i.e. making them a strong data type candidate for NVM technologies. Therefore, we extend the design space towards tree-based data structures.

2) Extended Profiling: To apply the data organization optimization approach, the application's STL data structures are replaced with the corresponding library implementations, as shown in the code snippet of Fig. 2. The application is then executed for each different data structure implementation. The alternative implementations are shown in Table II. For every combination of data structure implementations, we collect read and write data accesses and memory footprint per data structure, as platform independent metrics. Profiling techniques used in existing data structure refinement approaches rely on instrumentation [11][12], which often results to an extremely time consuming exploration procedure, especially in cases in which the number of application data structures is relatively high. In this work, the internal variables of each data structure

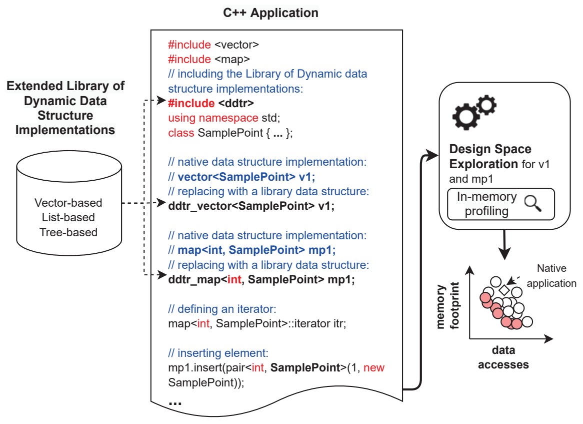
Data Structure	Short Description
VEC	An array that can grow or shrink dynamically (Vector)
SLL	Single Linked List
SLLR	SLL with a pointer to the last accessed element
DLL	Double Linked List
DLLR	DLL with a pointer to the last accessed element
MAP-RB	An associative container (map in STL) that stores pairs of key-value in a Red-Black tree
MAP-AVL	A map that stores key-value pairs in an AVL tree

TABLE II: The design space of data structure implementations used for the data organization optimization (extended DDTR).



Application source code (C/C++)
1. Optimization of data organization
Library of Dynamic Data Structure Implementations
Dynamic Data Structure Optimization
Native application
memory footprint
data accesses
Pruning of Pareto optimal solutions
memory footprint
data accesses
Pareto point 1
Pareto point 2
Pareto point N
2. Memory object analysis and profiling
Memory object profiling N
Memory object profiling 1
Tracing files processing and analysis
Memory tracing N
Memory tracing 1
Platform-specific performance analysis profiler
3. Placement on heterogeneous memory systems and evaluation
Data placement algorithm
Heterogeneous memory system M
Heterogeneous memory system 1
Heterogeneous memory system M specifications
Heterogeneous memory system 1 specifications
Hardware platforms and/or memory emulators
energy
Memory Technology 1
performance
energy
Memory Technology M
performance
Pruning of Pareto optimal solutions
Output: Pareto optimal performance/energy data placement solutions for various memory technologies
Memory Technology M
Memory Technology 1
energy
energy
performance

Fig. 1: Overview of the data structure refinement, data placement and evaluation methodology.



```
#include <vector>
#include <map>
// including the Library of Dynamic data
structure implementations:
#include <ddtr>
using namespace std;
class SamplePoint { ... };

// native data structure implementation:
// vector<SamplePoint> v1;
// replacing with a library data structure:
ddtr_vector<SamplePoint> v1;

// native data structure implementation:
// map<int, SamplePoint> mp1;
// replacing with a library data structure:
ddtr_map<int, SamplePoint> mp1;

// defining an iterator:
map<int, SamplePoint>::iterator itr;

// inserting element:
mp1.insert(pair<int, SamplePoint>(1, new
SamplePoint));
...
```

Fig. 2: Code snippet of the extended DDTR approach

are encapsulated in order to accurately capture each individual data access for each operation like insert, update or delete and also record every memory allocation thus providing an accurate depiction of each implementation's performance and memory consumption. All logging information are kept in memory to avoid execution time overhead when profiling the application.

3) Design Space Pruning: The profiling results for each configuration of data structures are processed to provide a Pareto curve of data accesses vs. memory footprint (Fig. 1). Each different point of the curve corresponds to the application implemented with alternative data structure combinations, but the same functionality. As every possible combination is considered a different case, for applications with a relatively large number of data structures, the number of Pareto points can skyrocket, with several points being, however, having negligible difference in data accesses and memory footprint.

Unlike existing DDTR approaches [12], a simple heuristic is used to prune the Pareto points that are too close to each other(neighbor points) and to provide to the next steps of the proposed methodology a relatively small number of solutions. In order to detect the neighbor points, we count the Euclidean distance among all points in the Design Space and for those that their distance is less than a user-specified threshold, we keep a single point, as the alternative solutions provide almost the similar memory accesses and footprint. Threshold variable is defined through extensive experimentation. The functionality for the pruning heuristic is summarized in Algorithm 1.

By designing an exploration-based data structure refinement approach, as an extended DDTR, advanced profiling and a heuristic for design space pruning, we were able to integrate it as a first step in the proposed methodology and to provide input to data placement algorithms for applications deployed on heterogeneous memory systems.

B. Memory Object Analysis and Profiling

The *Optimization of data organization* step identifies a number of Pareto optimal data structure implementations, which correspond to alternative application "versions", each one implemented with a different Pareto optimal data structure implementation. To apply data placement on heterogeneous memories, metrics such as data accesses and memory footprint are not enough for typical placement algorithms, since more fine-grained and platform-specific metrics are required. Therefore each application version is analyzed using performance analysis tools for memory tracing information at memory object granularity, including load, stores and object size. Intel PCM and VTune Profiler can be used for this purpose. This low-level information is propagated as input to the data placement algorithms of the third step of the proposed methodology.

C. Placement on Heterogeneous Memory Systems and Evaluation

The profiling results for each application version are fed to data placement algorithms such as [8] and [7], which provide memory object placement solutions for heterogeneous memory systems. Memory systems can be either emulated (e.g. for evaluating different memory technologies) or real hardware can be used, such as DRAM paired with Intel Optane DCPM. A data placement algorithm is applied to each application version and for each different memory system. Execution on an emulator or on actual hardware is used to generate a set of performance/energy Pareto curves, one for each memory

Algorithm 1 Heuristic for Pareto space pruning

```
1: for pX in Pareto Space do
2:     for pY in Pareto Space do
3:         // Count Euclidean Distance(ED)
4:         ED=(pX.footprint - pY.footprint)$^2$ + (pX.accesses - pY.accesses)$^2$
5:         if ED $<$ Threshold then
6:             delete(pY) // Remove neighbour points
7:         end if
8:     end for
9: end for
```

technology. Each Pareto point corresponds to a different application version (i.e. application with different data structure implementation), as shown in Fig. 1. The alternative data organization results to different placement decisions, which naturally affect the performance (i.e. execution time) and the energy consumption of an application executed on a system with heterogeneous memory.

III. Evaluation

A. Experimental Setup

The proposed methodology is evaluated based on the following criteria: (i) The extent by which it can provide improved performance and/or energy consumption compared to native applications (i.e. applications with the native data structure implementations), when deployed on heterogeneous memory systems. (ii) The extent by which it can be used to identify trade-offs between performance and energy consumption. (iii) The contribution of DDTR extensions to supporting the proposed methodology. (iv) Whether it can be applied successfully to multiple data placement algorithms and multiple memory technologies using emulation and real hardware.

To evaluate the methodology based on the aforementioned criteria we utilize three representative algorithms, which make heavy use of C++ STL data structures, were selected from the Shark ML Library [14], as presented in Table III. For the memory object analysis and profiling, Intel Vtune Profiler was used with 0.1ms sampling rate. A state-of-the-art data placement algorithm was implemented from scratch and integrated in our tool, as a candidate for the third step of the methodology [8]. It utilizes first-order analytical models of memory access time and energy consumption to provide energy savings. PCM, ReRAM and STT-RAM memory technologies were emulated using HP Quartz emulator [10]. The latency and power specifications of each technology are reported in related works [8], [15]. Evaluation on real hardware was conducted on a 2x20 core Intel Xeon Gold 5218R CPU @2.10GHz with 4x32GB DDR4 DIMMs and 2x256GB Optane DC NVDIMMs configured in App Direct mode. Optane DCPM latency was measured using Intel Memory Latency Checker. Intel PCM was used to monitor DRAM and Optane DCPM energy consumption.

B. Evaluation Results

The experimental evaluation of our proposed methodology is summarized in Fig. 3. Fig. 3a demonstrates the execution

ML Algorithm	Data Structures	Short Description
SMS-EMOA	2 vectors, 1 map	Multi-objective selection based on dominated hypervolume
NSGA-II	2 vectors, 1 map	Fast and elitist multi-objective genetic algorithm
SteadyState MO-CMA	4 vectors, 1 map	Steady-State Multi-Objective Selection in Covariance Matrix Adaptation

TABLE III: Short Description of Shark-ML algorithms

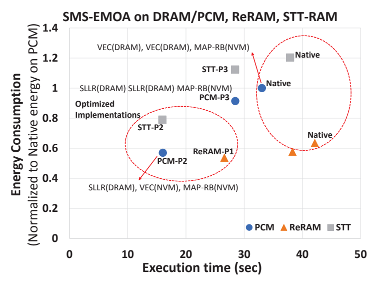
time vs. energy consumption results for the SMS-EMOA algorithm, deployed systems that consist of DRAM/PCM, DRAM/ReRAM and DRAM/STT-RAM. Each point represents a version of SMS-EMOA with alternative data organization deployed on one of the aforementioned heterogeneous memory systems. The results of SMS-EMOA with the original data structure implementations (*VEC*, *VEC*, *MAP-RB*) deployed on each system are also shown in the figure. For the native implementation deployed in DRAM/PCM, the data placement algorithm applied at the third step of the methodology recommends the placement of the memory objects of the first and the second data structure (*VEC*) on DRAM, while the third (*MAP-RB*) on NVM, resulting to execution time of 33sec. The energy consumption of the native version is used as a baseline.

Initially, we applied the DDTR interface, as shown in Fig.2 and performed the design space exploration (Fig. 2). The output of the optimization of data organization is shown in Fig. 4. After the Pareto space pruning, three Pareto optimal application versions were identified: (i) The native implementation, (ii) one in which the data structures are implemented as *SLLR*, *VEC* and *MAP-RB*, respectively. (iii) Another one with *SLLR*, *SLLR*, *MAP-RB* data structures. By applying the second and the third step of the methodology, for the DRAM/PCM memory system, solutions PCM-P2 and PCM-P3 are identified, as shown in Fig 3a. PCM-P2, in which the first data structure implemented as *SLLR* is placed on DRAM, the second (*VEC*) and the third *MAP-RB* are placed on NVM, provides 51% lower execution time compared to the native, as well as 54% lower energy consumption. Similar results are observed for the SMS-EMOA deployed on the DRAM/ReRAM (37.5% and 15%) and DRAM/STT-RAM systems (57% and 41%, respectively).

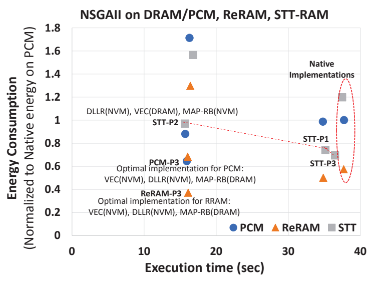
The output of the methodology applied to NSGAII is illustrated in Fig. 3b. For NSGAII with optimized data structure implementations deployed on DRAM/PCM (PCM-P3), the execution time is reduced by 57% and energy consumption by 36% compared to the native. For DRAM/ReRAM, by 54% and 35%, respectively (ReRAM-P3). Performance/energy trade-offs are identified for the NSGAII deployed on a DRAM/STT-RAM (implementations STT-P2, STT-P1 and STT-P3).

MOCMA is a more complex ML algorithm, in which the DDTR library interface was applied to five data structures, including one tree-based. The results are presented in Fig. 3c. Significant execution time gains are observed compared to the MOCMA with original data structure implementations, when deployed on DRAM/PCM or DRAM/ReRAM (about 50% for execution time and 37% for energy consumption), while trade-offs between the two metrics are demonstrated for DRAM/STT-RAM. The native implementation deployed on DRAM/STT-RAM provides the lowest execution time (10.67sec), while STT-P3 version (*SLLR(NVM)*, *SLL(NVM)*, *SLL(DRAM)*, *VEC(DRAM)*, *MAP-AVL(DRAM)*) can be selected for trading execution time for 41% energy reduction. We observe that significant performance gains are obtained when the second data structure is implemented as *VEC*, especially for the DRAM/PCM and DRAM/ReRAM experiments.

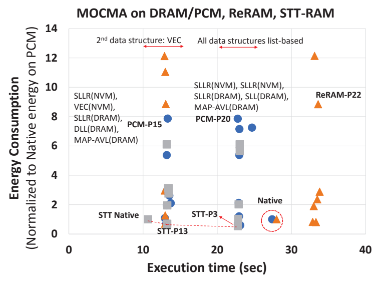
Evaluation results of SMS-EMOA, NSGAII and MOCMA on DRAM/Optane DCPM are shown on Fig. 3d, Fig. 3e and Fig. 3f, respectively. It is noticed that execution time reductions ranging from 6% to 8.3% can be obtained by improving



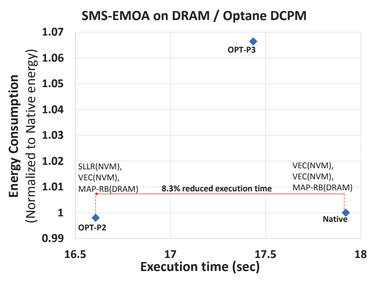
(a) SMS-EMOA on PCM, ReRAM, STT



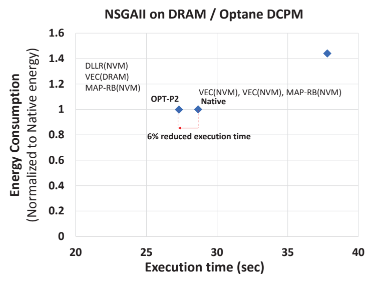
(b) NSGAII on PCM, ReRAM, STT



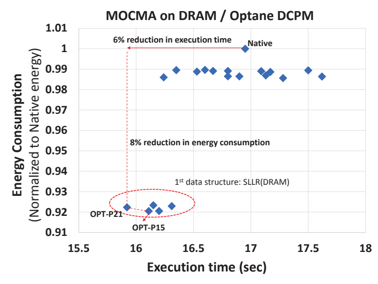
(c) MOCMA on PCM, ReRAM, STT



(d) SMS-EMOA on Optane DCPM



(e) NGSAII on Optane DCPM



(f) MOCMA on Optane DCPM

Fig. 3: Evaluation results of three machine learning applications on a variety heterogeneous memory architectures.

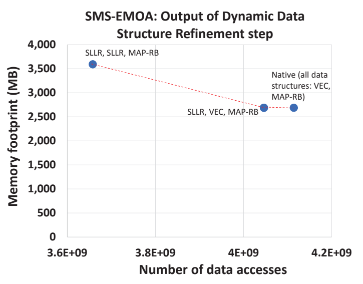


Fig. 4: Optimization of data organization for the SMS-EMOA.

the data organization of each application. For the MOCMA, implementing the second data structure as SLLR placed on DRAM, results in 6% execution time reduction, as well as in 8% lower energy consumption. Performance/energy trade-off is also identified (OPT-P21, OPT-P15).

C. Discussion

The data structure refinement step can significantly contribute to improved data placement results in terms of execution time and energy consumption: As shown in Fig. 3, execution time and energy consumption can be reduced up to 57% and 41% respectively. Also, trade-offs between the aforementioned metrics are observed (e.g. Fig.3b and Fig.3c).

To further investigate gains observed, we identified **the most important data structure-level parameters that affect the placement**, thus the execution time and the energy consumption. (i) The number of write and read accesses in data structures affect the number memory accesses, which affect placement. In the performance and energy models of the data placement algorithm used for evaluation, memory read and write accesses are parameters to the average memory access of memory objects [8]. (ii) The memory size of each data structure, as well as the lifetime of each element has significant impact on the energy consumption, e.g. object size is a parameter in the energy model of the placement algorithm used for evaluation [8]. (iii) The write/read (W/R) ratio of data structures indirectly affects placement, e.g. for the SMS-EMOA Native, STT-P2 and STT-P3 versions are shown in Fig.3a. The corresponding results are shown on Table IV. The native version of the application, which has 0.97 W/R ratio is executed in 37.8sec when deployed on a DRAM/STT-RAM system, with the first two data structures placed on NVM. When the first data structure (DS1) is implemented as SLLR instead of VEC (STT-P2), the number of memory writes and the W/R ratio drop by more than 3x. DS1 is still placed on NVM and since the write latency is increased 1.4x compared to read latency for STT-RAM technology, the W/R ratio reduction of DS1 on NVM has significant impact on the execution time, which drops to 16sec. For STT-P3, the implementation of DS2 as SLLR results in memory writes increase and, since all data structures are placed in NVM, execution time is higher than STT-P3 and lower than the native. Similar explanations can be provided for the energy consumption. Finally, W/R ratio of data structures naturally affects NVM endurance. Although endurance is beyond the scope of this work it can be integrated as future metric.

From another perspective, **poor data organization has negative impact on placement results on heterogeneous memories:** The straightforward selection of data structures by application developers is not always the most efficient solution, as shown in Fig.3. Developers do not normally consider data placement or other low-level details when developing applications. Thus, a semi-automated tool-flow that combines a data structure refinement approach with placement on heterogeneous memory systems is useful for application developers by allow-

TABLE IV: W/R ratio of SMS-EMOA on DRAM/STT-RAM

	DS1	DS2	DS3	#reads	#writes	W/R ratio	exec. time
Native	VEC (NVM)	VEC (NVM)	MAP-RB (DRAM)	7.4M	7.2M	0.97	37.8sec
STT-P2	SLLR (NVM)	VEC (NVM)	MAP-RB (DRAM)	8.8M	2.8M	0.31	16sec
STT-P3	SLLR (NVM)	SLLR (NVM)	MAP-RB (NVM)	7.6M	7.5M	0.98	28.4sec

ing them to focus on application algorithmic aspects, entirely.

The proposed methodology can be applied to a variety of memory technologies and integrate various data placement algorithms: It was extensively evaluated both on emulated and on a real heterogeneous memory system. However, similar results are expected for other memory technologies. The data placement for the evaluation was based on an algorithm that focuses on energy improvements [8]. However, alternative algorithms, can be used instead. Fig. 5 shows the SMS-EMOA deployed on a variety of heterogeneous systems using a data placement algorithm that proposes a utility model that takes into account memory access performance and energy consumption to direct object placement on NVM or DRAM in order to minimize execution time [7]. A notable difference between Fig. 5 and Fig. 3b is that more data structures are placed on DRAM when applying the alternative placement algorithm, due the fact that it is oriented towards execution time improvement.

With respect to the **effort required to apply the methodology** it is minimal for applications that use C++ STL. Applying the library of data structures, as shown on Fig. 2 is straightforward. More effort should be required for C applications, especially if the data structure implementations have substantially different interface than the corresponding STL.

Apart from the implementation of the data structure library interface to applications, all other steps of the methodology are fully automated: The exploration time depends mainly on the application's execution time. As an example, for an application that is executed in 10sec, the methodology generates results between 30min and 2-3 hours, depending on the number of Pareto solutions provided after the pruning of the first step. Due to the advanced profiling used in the extended DDTR, as explained in Section II-A, profiling overhead of the first step of the methodology is negligible. The most time consuming step is the memory object analysis and profiling that relies on instrumentation (second step in Fig.1). However, the pruning of the Pareto space, as explained in Section II-A significantly reduces analysis time. More specifically, in the evaluation experiments of Section III, between 25% and 30% of the Pareto points identified by the

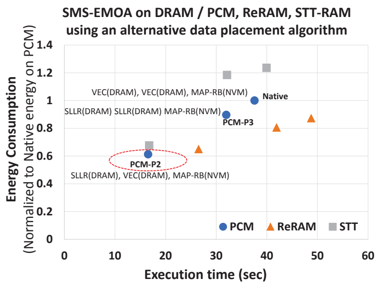


Fig. 5: SMS-EMOA on DRAM/PCM, ReRAM, STT-RAM using an alternative data placement algorithm.

first step of the methodology for each application were pruned. The tool-flow is publicly available with open source licence[1].

Data migration between memory types is not considered in this work, however migration can further improve the results of the data placement. Additionally, migration can be a response to unforeseen dynamic situations (e.g. when a dynamic data structure grows so much that needs to be placed in a larger memory). However, in this work we do not consider such cases. The size of data structures of the applications used for evaluation in section III-B does not vary significantly.

IV. Conclusion

This work investigates the impact of high-level data organization optimizations on data placement on heterogeneous memories and proposes a memory management methodology that supports developers to refine application data structures, before the placement on heterogeneous memory systems. The methodology is extensively evaluated for various technologies on emulated and real memory systems. Analysis of the results shows that the dynamic data structure refinement has significant positive impact on placement outcome and particularly on application execution time and energy consumption.

References

[1] A. Xygkis, D. Soudris, L. Papadopoulos, S. Yous, and D. Moloney, "Efficient winograd-based convolution kernel implementation on edge devices," in *DAC*. IEEE, 2018, pp. 1–6.

[2] A. Hassan, D. S. Nikolopoulos, and H. Vandierendonck, "Fast and energy-efficient olap data management on hybrid main memory systems," *IEEE Transactions on Computers*, vol. 68, no. 11, 2019.

[3] C. Wang, H. Cui, T. Cao, J. Zigman, H. Volos, O. Mutlu, F. Lv, X. Feng, and G. H. Xu, "Panthera: holistic memory management for big data processing over hybrid memories," in *PLDI*, 2019, pp. 347–362.

[4] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "Archshield: Architectural framework for assisting dram scaling by tolerating high error rates," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, 2013.

[5] Y. Ho, G. M. Huang, and P. Li, "Nonvolatile memristor memory: Device characteristics and design implications," in *ICCAD*, 2009, pp. 485–490.

[6] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor *et al.*, "Basic performance measurements of the intel optane dc persistent memory module," *arXiv preprint arXiv:1903.05714*, 2019.

[7] H. Liu, R. Liu, X. Liao, H. Jin, B. He, and Y. Zhang, "Object-level memory allocation and migration in hybrid memory systems," *IEEE Transactions on Computers*, vol. 69, no. 9, pp. 1401–1413, 2020.

[8] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos, "Software-managed energy-efficient hybrid dram/nvm main memory," in *ACM CF*, 2015.

[9] D. Atienza, C. Baloukas, L. Papadopoulos, C. Poucet, S. Mamagkakis, J. I. Hidalgo, F. Catthoor, D. Soudris, and J. Lanchares, "Optimization of dynamic data structures in multimedia embedded systems using evolutionary computation," in *SCOPES*, 2007, pp. 31–40.

[10] 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*, 2015.

[11] C. Jung, S. Rus, B. P. Railing, N. Clark, and S. Pande, "Brainy: Effective selection of data structures," *ACM SIGPLAN Notices*, vol. 46, no. 6, pp. 86–97, 2011.

[12] D. A. Alonso, S. Mamagkakis, C. Poucet, M. Peón-Quirós, A. Bartzas, F. Catthoor, and D. Soudris, *Dynamic memory management for embedded systems*. Springer, 2015.

[13] M. B. Olson, T. Zhou, M. R. Jantz, K. A. Doshi, M. G. Lopez, and O. Hernandez, "Membrain: Automated application guidance for hybrid memory systems," in *NAS*. IEEE, 2018, pp. 1–10.

[14] C. Igel, V. Heidrich-Meisner, and T. Glasmachers, "Shark," *Journal of Machine Learning Research*, vol. 9, pp. 993–996, 2008.

[15] W. Wei, D. Jiang, S. A. McKee, J. Xiong, and M. Chen, "Exploiting program semantics to place data in hybrid memory," in *PACT*. IEEE, 2015, pp. 163–173.

[1]https://github.com/mkatsa/DDTR-DRAM-NVM