Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2017) 25: 3355 – 3366
© TÜBİTAK
doi:10.3906/elk-1608-22

Access pattern-aware data placement for hybrid DRAM/NVM

Didem UNAT ERTEN[*]
Department of Computer Engineering, Koç University, İstanbul, Turkey

Received: 02.08.2016 • **Accepted/Published Online:** 08.01.2017 • **Final Version:** 30.07.2017

Abstract: In recent years, increased interest in data-centric applications has led to an increasing demand for large-capacity memory systems. Nonvolatile memory (NVM) technologies enable new opportunities in terms of process-scaling and energy consumption, and have become an attractive memory technology that serves as a secondary memory at low cost. However, NVM has certain disadvantages for write references, due to its high dynamic energy consumption for writes and low bandwidth compared to DRAM writes. In this paper, we propose an access-aware placement of objects in the application code for two types of memories. Given the desired power savings and acceptable performance loss, our placement algorithm suggests candidate variables for NVM. We present an evaluation of the proposed technique on two applications and study the energy and performance consequences of different placements.

Key words: Data placement, nonvolatile memory, static analysis

1. Introduction

DRAM, generally used for main memory, is an important component of computing systems. Although it provides fast and durable storage, it shares up to 41% of system power consumption [1,2]. Additionally, because it must be periodically refreshed, it consumes power even when not in use. Moreover, the scaling of DRAM technology is problematic [3], which implies that it will be difficult for DRAM to provide low-cost and high-capacity memory for future high-performance computing systems, since the cost of memory is directly proportional to memory cell size. Scaling the feature size leads to higher density and lower cost. Byte-addressable nonvolatile memory (NVM) technologies, such as phase change memory (PCM), have been studied to replace or augment DRAM in future memory systems due to their better process scaling characteristics. PCM is expected to scale up to 9 nm and prototypes in 3 nm have already been fabricated [4]. As a result, NVM is considered as an alternative scalable memory solution for main memory. On the other hand, the write bandwidth and write power of NVM still limit its wide adoption. Compared to DRAM, the dynamic write energy is 4–40 times worse and write bandwidth is 3 times slower [5–8]. Computer architects rigorously explore techniques to overcome these drawbacks and make this memory technology competitive with DRAM in terms of performance and dynamic energy cost.

Recent research combines NVM and DRAM to take advantage of both storage types, and proposes PCM/DRAM hybrid main memory system architectures. In this study, we assume a hybrid memory design, where NVM (e.g., PCM) and DRAM coexist in a compute node, as envisioned by Ang et al. in abstract machine models for HPC systems [9]. NVM might have a particular address range assigned to it or it might act

[*]Correspondence: dunat@ku.edu.tr

as a temporary buffer for the primary memory. In such a scenario, to gain the full advantage of the memory subsystem, a programmer or compiler would need to be explicit about the initial placement of the data on different types of memories.

This paper investigates whether or not an application can benefit from a hybrid memory system, and proposes placement algorithms with different objectives to decide what objects in an application should be placed on what type of memory. We automatically analyze an application's read/write access pattern using a compiler-based loop analysis tool. The extracted access information is then fed into our object placement algorithms, which suggest data structures that are suitable for NVM. Our study also evaluates whether applications have sufficient low-write memory traffic for NVM or not. Since read and write voltages are highly dependent on the technology, we suggest that the minimum values for NVM be energy-effective alternatives to DRAM.

This paper is organized as follows. In Section 2, we provide a brief background and motivation for using NVM technologies in future high-performance systems. Section 3 details our methodology, including the analysis tool, metrics used for the analytical model, our energy model, and the data placement algorithms with different objectives. Section 4 evaluates our tool using two motivating applications. Section 5 discusses the related work, and, finally, Section 6 concludes the paper.

2. Background and motivation

Emerging NVM technologies are word-addressable, as opposed to block-addressable, which make them an attractive main memory alternative to DRAM. PCM is one of the emerging nonvolatile random-access memories. It is based on reversible phase conversion between the amorphous and the crystalline state of a chalcogenide glass as a result of heat change. The heat produced by the electric current sets the chalcogenide between the two states [10].

For PCM to be a cost-effective alternative to DRAM, it should provide better or comparable memory cell size and scalability, since the cost of memory is directly proportional to memory cell size. DRAM cells occupy 6–8 F^2, where F is the smallest lithographic feature size. On the other hand, 4–6 F^2 of PCM has already been demonstrated [10]. A smaller device footprint leads to a higher density and, thus, lowers the cost per bit. PCM is expected to scale 9 nm and higher density than DRAM, whereas scaling DRAM beyond 18 nm is challenging, as indicated by the International Technology Roadmap for Semiconductors (ITRS 2009) and other researchers [7,11,12]. Recent research shows that more than one bit of data per physical cell can be represented without decreasing feature size [13] in PCM. In addition, the physical structure of PCM is three-dimensional, which allows for many transistors in a fixed area. This means that NVM can provide more GB in the same size package.

NVM favors read references over write references due to the high dynamic energy consumption for writes and low bandwidth rates compared to DRAM writes, as shown in the Table. As a result, write references to NVM have an adverse effect on both dynamic power consumption and application performance. However, NVM has a comparable read performance with DRAM and much better leakage (4.23 mW/GB vs. 451 mW/GB). Although PCM challenges reliable DRAM technology in the long term, in this work we would like to address the following questions: 1) which data structure(s) should be placed in DRAM and NVM to avoid performance loss when NVM is used to augment DRAM to increase system storage capacity? 2) What is the change in energy consumption as a result of such data placement? In order to answer these questions, we need to collect a memory access pattern in an application. We utilize the ExaSAT tool, previously developed by Unat et al. [14], and build our analysis on ExaSAT.

Table. Characteristics of PCM and DRAM technologies.

	DRAM	PCM	DRAM/PCM
Latency read (ns)	39 [19]	40 [19]	0.98
Latency write (ns)	48 [19]	139 [19]	0.35
Bandwidth read (MB/s)	775 [19]	760 [19]	1.02
Bandwidth write (MB/s)	632 [19]	220 [19]	2.87
Dynamic energy (nJ)	5.9 [18]	80.4 [18]	0.07
Read dynamic energy (nJ)	12.7 [18]	418.6 [18]	0.03
Write leakage (mW/GB)	451 [18]	4.23 [18]	106.62

3. Methodology

To find the appropriate data structures for NVM, the community standard is to use the read/write ratio of the variables [15]. However, write access rate is as important as the read/write ratio, because a program may have high rates of both read and write. Instead, we use the metric write access rate for evaluating the candidate data structures for DRAM and NVM. Variables with a low write access rate do not greatly increase dynamic energy and do not have significant impact on performance; thus, they are good candidates for NVM. If the dynamic energy consumption for reads to NVM is not comparable to that of DRAM, the write access rate should be complemented with the read/write ratio or read access rate.

3.1. Application-specific metrics

Let w_v be the write references for a variable v and R be all references to memory by all variables V. Then the write access rate for v is w_v/R. Similarly, let r_v be the read references for a variable v; then the read access rate is r_v/R. Note that $R = \Sigma(w_v + r_v)$ for all $v \in V$. To characterize NVM opportunities, we look at three metrics in an application:

1. Write access rate: low access rate is better, because dynamic power consumption for writes is higher for NVM, and NVM has shorter write endurance.
2. Read access rate: lower is better due to dynamic energy consumption. On the other hand, the read performance of NVM is comparable to that of DRAM, although NVM reads have static power advantage.
3. Memory footprint: higher is better, since leakage power saving is proportional to the size of the data structure.

A data structure may be a struct containing multiple fields (or components). We compute the read and write the counts for each field, because each one may have a different access pattern. This is possible when struct of array is used, as opposed to array of struct, to represent the data structures in the application. On the other hand, access to a field is taken as a whole; when a single element of a field is read/written, we assume that the entire (field) array is accessed. This assumption typically holds true, because many applications loop over the entire array, instead of accessing a few elements of the array.

3.2. Architecture-specific metrics

We need to identify a set of important architectural characteristics for different memory types. These characteristics can be categorized as performance-related and power-related metrics. Performance-related metrics include random read/write latencies and read/write bandwidth. Power-related metrics include dynamic energy and leakage power, because memory consumes energy both when it is active and when it is idle. The dynamic energy can be different for read and write references.

The Table shows values for architecture-specific metrics for PCM and DRAM technologies. The values of these metrics are presented in several works in the literature [16–19], including the International Technology Roadmap for Semiconductors (http://www.itrs. net/Links/2013ITRS/2013Tables/ERD_2013Tables.xlsx). Suzuki and Swanson conducted a comprehensive study of the evolution of NVM technologies, and gathered information from over 300 technical papers published since 2000 in prestigious journals and conferences [19]. In this study, we use their survey data to model the performance, energy, and power characteristics of hybrid memory architecture.

As seen in the Table, the performance characteristics of both PCM and DRAM are comparable for reads; both read latency and bandwidth of PCM are very close to those of DRAM. As a result, there is no significant performance benefit in placing read-intensive data to DRAM versus NVM. On the other hand, writes to PCM have higher latency and lower bandwidth rates.

3.3. Energy model

In this section, we analytically estimate the energy impact of placing data in different types of memory. We adopt the energy model suggested in [18]. Static energy consumption is based on the memory footprint and the lifetime of the variable. Dynamic energy, however, depends on memory access frequency and type of access. Energy consumed by a variable v residing a memory type $t$is then

$$E_v(t) = r_v \times D_r(t) + w_v \times D_w(t) + S_v \times T_v \times L(t).$$

Read references, r_v, and write references, w_v, are computed with our analysis tool, discussed in Section 3.4. D_r and D_w are the dynamic energy for reading/writing, respectively, from/to the memory type, $t.L$ represents the leakage power for t, S_v is the footprint of the variable, and T_v is the lifetime of a variable. From the Table, we expect that *L(nvram)* $<$ *L(dram)*; however, *D(nvram)* $>$ *D(dram).* As a result, the values used for architectural metrics are important for our study; the system will either save energy or lose it for placing data in NVM.

3.4. Access pattern analysis

To collect memory access references, type, and object information, we employ the ExaSAT framework [14]. ExaSAT is a compiler-based framework, used for studying software optimizations on current and future architectures [20]. The framework statically analyzes an application and automatically gathers information about data accesses, static variable references, and arithmetic operations. It outputs an XML that represents the application workload on a loop-by-loop basis. ExaSAT uses the XML file, and feeds into a performance model that can combine hardware and software characteristics to estimate memory bandwidth usage or memory traffic between DRAM and last level cache. It also computes the arithmetic intensity of an application, and decides whether the application is compute or memory-bound. The framework is tested on production combustion simulations and is used to answer key design questions for future node architectures. In this project, we focus on the memory access information that the framework provides for read/write references and their frequencies.

ExaSAT creates read and write counts for each variable for every loop in every function. However, moving an object between two successive loops, or even functions, is not practical, because it will lead to excessive memory traffic between NVM and DRAM. As a result, the initial placement plays an important role in maximizing application performance. Instead of driving our model from the loop-by-loop basis counts, we calculate the total reads and writes for each data structure.

One of the limitations of our approach is that ExaSAT does not take into account any memory access optimizations that might be performed by the compiler, as it analyzes the code as it is written. However, the commercial compilers do not drastically affect the lower bound of memory access, because they cannot safely generate cache-optimized codes, except for simple loops. It is important to note that ExaSAT only estimates compulsory memory access rather than actual memory access, because the analysis is statically performed at compile-time. As a result, the placement algorithms may produce a suboptimal placement, because they do not have the runtime information.

3.5. Placement algorithms

We can categorize the placement algorithms based on their objectives. Algorithm 1 in Figure 1 shows the data placement algorithm with a performance objective. For each variable v, we compute its read and write references in program P. These values are computed with the help of the ExaSAT tool. Then we sort the variables with a decreasing write reference. If two variables have the same write reference counts, then the higher read reference precedes the lower read reference. In the case of equality, the variable with the lower footprint precedes the one with the higher footprint. Starting from the beginning of the sorted list, the second loop in Line 13 appends the variables to the DRAM candidate list until DRAM is full, since the objective is to take advantage of DRAM for write- and read-intensive objects. Line 19 takes the set difference; any variable that cannot be placed in DRAM is allocated to NVM.

Algorithm 1 Data Placement with Performance Objective

```
 1: procedure FindNvmCandidates
 2:     $P \leftarrow$ program
 3:     $V \leftarrow$ variable list in P
 4:     $V_{nvm} \leftarrow \{\ \}$
 5:     $V_{dram} \leftarrow \{\ \}$
 6:     $S \leftarrow 0$
 7:     for each $v \in V$ do
 8:         $r_v \leftarrow$ compute read references of $v$ in P
 9:         $w_v \leftarrow$ compute write references of $v$ in P
10:     end for
11:     $V' \leftarrow$ sort variable list $V$ based on decreasing $w_v$,
12:             then decreasing $r_v$, then increasing $S_v$
13:     for each $v \in V'$ do
14:         if $S + S_v \leq S_{dram}$ then
15:             $V_{dram} \leftarrow V_{dram} + \{v\}$
16:             $S \leftarrow S + S_v$
17:         end if
18:     end for
19:     $V_{nvm} \leftarrow V' \setminus V_{dram}$
20: end procedure
```

Figure 1. Data placement algorithm with performance objective.

The second algorithm in Figure 2 performs the data placement with the objective of saving energy. After computing the r_v and w_v values of a variable, we compute its dynamic and static energy consumption as if it were to be placed on NVM (v_{nvm}) and on DRAM (v_{dram}). Line 13 adds the NVM and DRAM pairs to the energy list, and then sorts them based on the increasing DRAM energy consumption. Line 18 compares the energy consumption for two memory types. If DRAM consumption is less than that of NVM and there is still space on DRAM, then variable v is placed on DRAM, otherwise it is placed on NVM. Sorting the list ensures that no object is placed onto NVM due to its capacity limitation, as it has lower energy consumption than that of an object, which is already placed on DRAM.

Algorithm 2 Data Placement with Energy Saving Objective

```
 1: procedure FINDNVMCANDIDATES
 2:     $P \leftarrow$ program
 3:     $V \leftarrow$ variable list in P
 4:     $V_{nvm} \leftarrow \{\ \}$
 5:     $V_{dram} \leftarrow \{\ \}$
 6:     $E \leftarrow \{(\ ,\ )\}$
 7:     $S \leftarrow 0$
 8:     for each $v \in V$ do
 9:         $r_v \leftarrow$ compute read references of $v$ in P
10:         $w_v \leftarrow$ compute write references of $v$ in P
11:         $v_{nvm} \leftarrow r_v * D_w(nvm) + w_v * D_w(nvm) + S_v * T_v * L(nvm)$
12:         $v_{dram} \leftarrow r_v * D_w(dram) + w_v * D_w(dram) + S_v * T_v * L(dram)$
13:         $E \leftarrow E \cup \{(v_{dram}, v_{nvm})\}$
14:     end for
15:     $E' \leftarrow$ sort $E$ based on increasing $v_{dram}$
16:     for each $(v_{dram}, v_{nvm}) \in E'$ do
17:         if $v_{dram} \leq v_{nvm}$ && $S + S_v \leq S_{dram}$ then
18:             $V_{dram} \leftarrow V_{dram} \cup \{v\}$
19:             $S \leftarrow S + S_v$
20:         end if
21:     end for
22:     $V_{nvm} \leftarrow V' \setminus V_{dram}$
23: end procedure
```

Figure 2. Data placement algorithm with energy-saving objective.

The third algorithm in Figure 3 finds the middle ground for the performance and power in data placement. Similarly to Algorithm 1, Algorithm 3 computes the read/write references of the variables and then sorts them based on their write counts. Then Line 13 traverses the sorted variable list and computes the dynamic and static energy consumption of each variable for two memory types. Line 16 checks if DRAM has space to hold any further objects. Line 17 examines whether the energy consumption of variable v on DRAM is less than that of NVM or not. If NVM is more advantageous in terms of energy consumption, then the algorithm also checks the write access rate of the variable, because placing this variable on NVM would lead to performance degradation. Based on the desired access rate threshold, the variable might still be placed on DRAM if there are too many accesses to it. The threshold value can be experimentally set, or can be set by the desired fraction of memory objects on one type of memory.

4. Evaluation

4.1. Motivating applications

To evaluate our analysis and investigate the benefits of hybrid NVM-DRAM technologies, we study two applications: CNS and SMC [21]. Both applications are direct numerical combustion simulations and are representative of real-life scientific applications. CNS and SMC play an important role in stencil-based partial differential equation solvers. While CNS consists of 1347 lines of Fortran code and contains 35 three-dimensional

```
Algorithm 3 Data Placement with Performance-Power Objective
1: procedure FindNvmCandidates
2:     $P \leftarrow$ program
3:     $V \leftarrow$ variable list in P
4:     $V_{nvm} \leftarrow \{\ \}$
5:     $V_{dram} \leftarrow \{\ \}$
6:     $S \leftarrow 0$
7:     for each $v \in V$ do
8:         $r_v \leftarrow$ compute read references of $v$ in P
9:         $w_v \leftarrow$ compute write references of $v$ in P
10:    end for
11:    $V' \leftarrow$ sort variable list $V$ based on decreasing $w_v$,
12:        then decreasing $r_v$, then increasing $S_v$
13:    for each $v \in V'$ do
14:        $v_{nvm} \leftarrow r_v * D_w(nvm) + w_v * D_w(nvm) + S_v * T_v * L(nvm)$
15:        $v_{dram} \leftarrow r_v * D_w(dram) + w_v * D_w(dram) + S_v * T_v * L(dram)$
16:        if $S + S_v \leq S_{dram}$ then
17:            if $v_{dram} \leq v_{nvm} \parallel w_v > W_{threshold}$ then
18:                $V_{dram} \leftarrow V_{dram} \cup \{v\}$
19:                $S \leftarrow S + S_v$
20:            end if
21:        end if
22:    end for
23:    $V_{nvm} \leftarrow V' \setminus V_{dram}$
24: end procedure
```

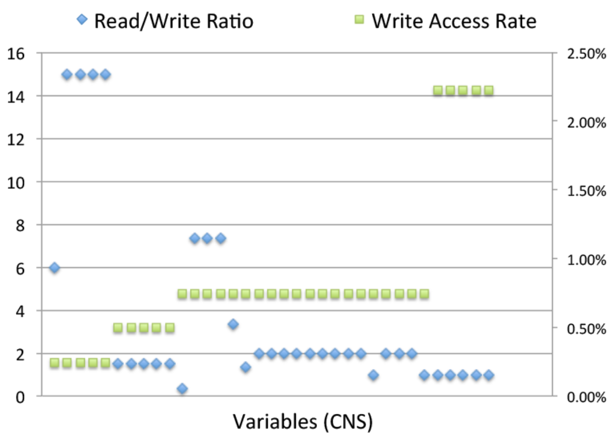
Figure 3. Data placement algorithm with performance power objective.

arrays, the SMC code contains 3830 lines of code and 159 three-dimensional arrays. Due to the length of the applications and the high number of arrays, it is difficult for a programmer to manually analyze these codes to identify candidate arrays for NVM and DRAM. Our compiler-based approach automatically analyzes the program and gathers write and read frequencies of arrays.

4.2. Results

For the experimental results, we assume there is 128 GB of DRAM and 2 TB of NVM, which is a realistic configuration for a high-performance computing system. We use a relatively large problem size of 1024^3 so that most data can fit into the existing DRAM. We use one timestep of the solver as execution time for the applications. Firstly, we evaluate the read/write ratio and write the access rates of the motivating applications. Figure 4 shows the application metrics for the CNS (left) and SMC code (right). The x-axis represents different variables in the applications, sorted by their write access rates. Because SMC has more three-dimensional variables (159 versus 35), there are more data points in the SMC figure. We are primarily interested in arrays rather than scalar variables, because the idle power consumption is proportional to the memory footprint. It is important to note that the low write access rate does not exclude variables with high read/write ratio. However, the read/write ratio omits variables that have a low ratio and low write access rate, which could have benefitted from NVM.

Figure 5 shows the fraction of data as a function of the (sorted) write access rates. Arrays with a large write access rate reside in DRAM, and the rest reside in NVM. The programmer can select a threshold value to serve as a cut-off value for data placement, or can completely fill DRAM first and then move to NVM, using the proposed Algorithm 1. Because Algorithm 1 sorts the arrays in terms of their write access rates, the high access rates will be placed on DRAM first. Assuming a problem size of 1024^3 for the SMC code, the total memory



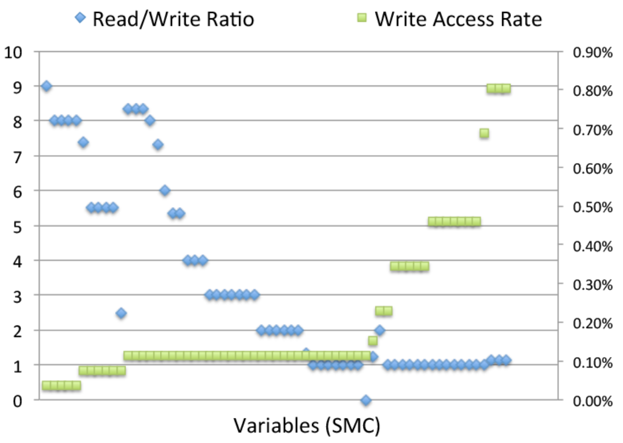
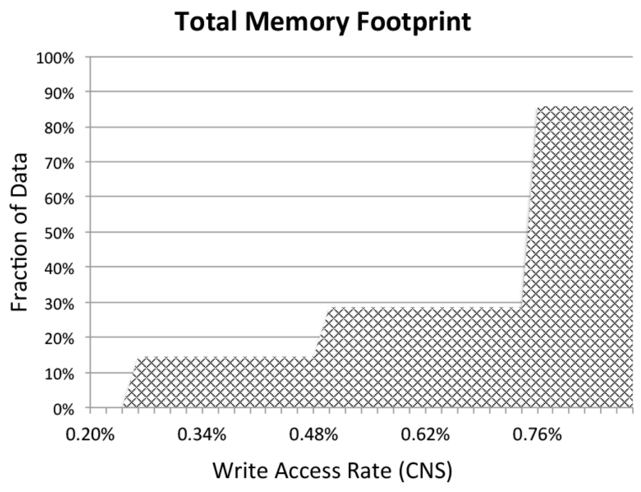


Figure 4. Read/write ratio (left y axis) and write access rates (right y axis) are shown for CNS and SMC. The x axis indicates a different variable in the applications.

requirement for holding all the data structures is 1272 GB (1.2 TB). In that case, only 10% of data can reside in DRAM; the rest have to go to NVM. Algorithm 1, with the objective of performance, helps select 10% of data as candidates for DRAM, so that performance penalties are minimized due to the write access latencies.



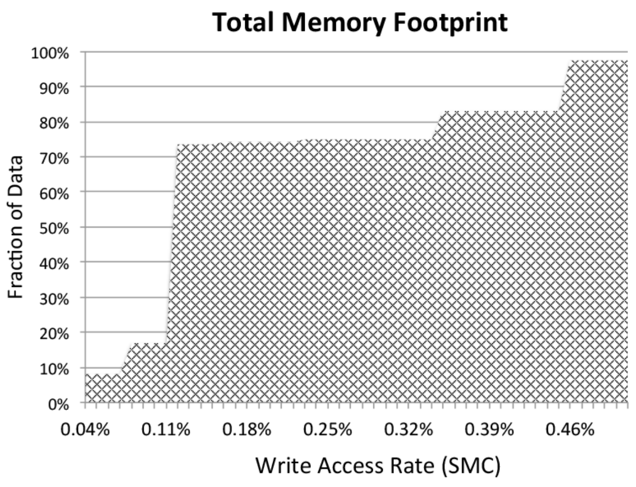


Figure 5. Fraction of data as a function of write access rates. High write access rates are better candidates for DRAM.

An application-specific threshold value can be computed based on the desired performance or power budget. Our analysis helps choose an application-specific value. In SMC, there are several arrays with low read and write access rates. If a write access rate of $\leq 0.11\%$ is chosen, then a large fraction of data (77%) qualifies for NVM, which translates into roughly 77% idle power savings. Nevertheless, the dynamic energy for these arrays will increase. For the CNS code, a higher threshold value is needed compared to SMC. The threshold line can be moved to the left or right depending on the desired power savings, acceptable performance loss, or the size of DRAM.

Figure 6 shows both dynamic and static energy consumption for DRAM (left) and NVM (right), using the energy values in the Table. Each data point on the x-axis represents a different variable in the SMC code. As seen from the figure, DRAM's dynamic and static energy consumptions are very similar. However, on NVM, energy consumption due to leakage is almost negligible. The main energy consumption is due to the dynamic energy. Overall, NVM has much higher total energy consumption for each variable than DRAM, using the hardware metrics in the Table. Based on these energy values, Algorithms 2 and 3 will fill DRAM until it is full and then move to NVM, because there is no energy gain for placing data in NVM. Furthermore, this means that Algorithm 3 converges with Algorithm 1, according to the current technology parameters. For NVM to become more competitive, its dynamic energy consumption should be improved with new advances in technology.

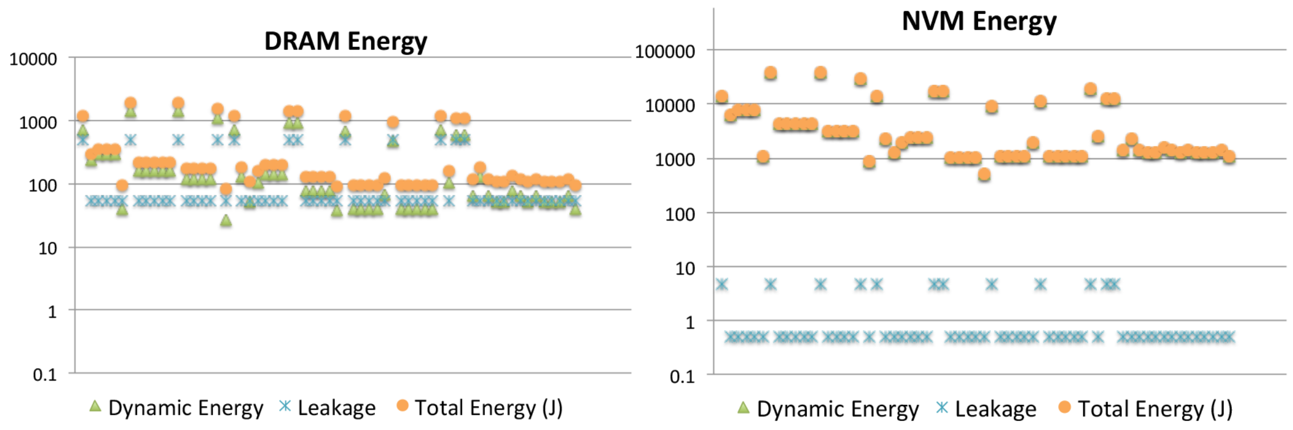


Figure 6. Total energy consumption in joules for each variable in SMC code when a variable is placed on DRAM or NVM.

Next, we investigate the effect of further improved read/write dynamic energy values on NVM. In particular, we choose hypothetical values and set the read and write energy consumption to only 50% worse than that of DRAM. All other parameters remain unchanged. Figure 7 compares the DRAM and NVM total energy consumption for each variable in the SMC code, based on the improved hardware metrics. The x-axis includes the variable names, sorted according to their qualification for DRAM, and uses Algorithm 3, which takes both the performance and energy consumption into account. Starting from the left of the x-axis, the variables are placed into DRAM until it is full, and then the remaining variables go to NVM. As seen from the figure, some variables may have lower total energy consumption in NVM than DRAM. However, because of their high access rates, Algorithm 3 places them in DRAM due to performance concerns.

In summary, our results suggest that write-access rate threshold is an important metric for deciding the data placement between NVM and DRAM. In particular, an application-specific write-access rate threshold should be chosen to avoid performance penalties. NVM technology still lags far behind in terms of dynamic energy consumption, compared to DRAM. Unless dynamic energy consumption improves in future systems, NVM cannot compete with DRAM. Despite these disadvantages, NVM has cost and capacity advantages, which allow programmers to utilize a large amount of memory at low cost for their applications.

4.3. Related work

Several researchers have proposed hybrid NVM and DRAM memory systems [22]. Proposed solutions in the literature on data placement on hybrid NVM and DRAM are predominantly based on operating systems or hardware support. The OS migrates data over the course of the execution to the most appropriate type of memory with the help of hardware extensions. Ramos et al. proposed a dynamic page placement solution through OS support in hybrid memory systems [23], which is agnostic about applications. Another group developed a memory management method for hybrid memory that keeps the write history to hide the slow

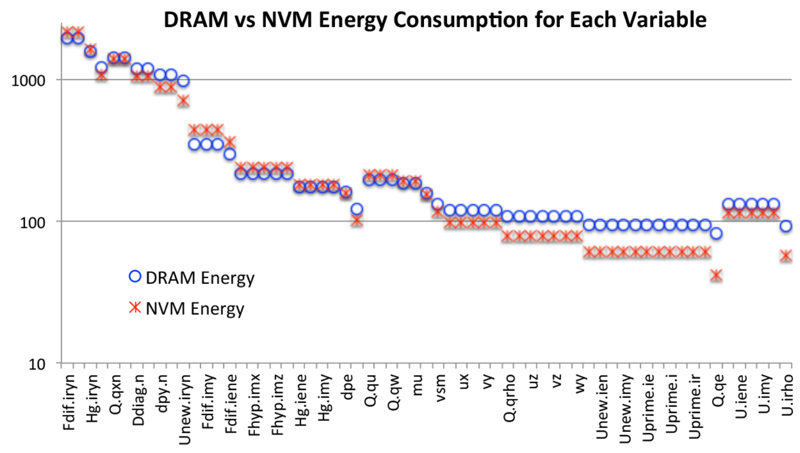


Figure 7. Data placement for variables using Algorithm 3. Starting from the left of x axis, the variables are placed onto DRAM until it is full.

writes to PCM [24]. Seok et al. [25] developed a page monitoring system for page migration between PCM and DRAM to reduce access latencies to PCM. Application agnostic data placement techniques for hybrid memories supported in the hardware, such as [7], have the advantage of hiding complexities from the user. However, they require extensions to the hardware to assist the OS. Our solution makes the application portable, because it does not depend on the OS or hardware extension used in the target platform.

Qureshi et al. [8] propose using DRAM as a page cache for PCM memory. The cache buffers frequently accessed pages, improves the endurance, and reduces the number of writes to the PCM. Zhang and Li propose a hybrid memory design, where PCM is used as a primary memory and a small portion of DRAM is used as a write-buffer [22]. Similarly, in [23], the authors propose a page replacement algorithm that leverages the memory controller to monitor programs. Then the operating system uses this information to make decisions about page migrations. The common idea behind these proposed approaches is that they use a global view of the programs. This work brings application-specific knowledge into the picture, when a decision is made about which objects are placed on NVM. We rely on an analysis tool that statically analyzes an application by studying its data structures and places data accordingly on DRAM or NVM. Since we rely on compiler analysis, which is performed statically at compile time, this tool introduces lower overhead than those at runtime. Moreover, our approach is orthogonal to the previously proposed approaches and can be combined with them to obtain better results.

Closest works to this are by Hassan et al. [18] and Li et al. [15]. Hassan et al. [18] proposed a programming interface to place data within the user application. Similarly to our work, they argue that object-based placement is better than page placement, since program objects are strongly biased towards one particular memory type. Unlike our compiler-based approach, they use a profiling tool and instrument the code to collect memory access information. This requires extra effort by the programmer. Li et al. [15] studied scientific workload and followed a dynamic approach for data placement. They developed a PIN-based binary instrumentation tool, which adds the execution time overhead to the application. On the other hand, our approach is static and lightweight, because it is compiler-based and the placement relies on an analytical model. Our analytical model sheds light on the usability of NVM, but has limitations. To assess whether dynamic energy consumption overshadows idle energy savings or not, our future work will employ memory simulators. Architectural simulation can provide more accurate power-saving numbers. Conversely, our analysis is practical and can be used by a memory manager with a low overhead.

Wang et al. [26] studied the placement for embedded systems. Because real-time tasks have a hard deadline, they favor the placement of data on only one type of memory. This is very restrictive for scientific workload, because a task may have multiple data objects; additionally, in our case the objects can reside on two different memory types. Another approach to collecting object information is to use a tool that captures data movement by tracking data objects in an application at runtime, such as ADAMANT [27]. Such a tool can be combined with our placement algorithms to place data not only on NVM but also on newly introduced 3D stack memories such as MCDRAM [28].

5. Conclusion

Nonvolatile memories have emerged as an alternative to DRAM, because they use extremely low power when in standby mode. On the other hand, they have low bandwidth rates, especially for writes, whereas reads are more comparable to DRAM. They also consume more dynamic power for writes and have limited write endurance. Hybrid DRAM-NVM memory architectures combine the best of both worlds; however, the technology is at its early stage. This paper investigated whether there is any opportunity or not to store variables in the NVM instead of DRAM. We used an analysis tool and categorized variables as good or bad candidates for NVM. Our approach is compiler-based and uses application information for the best initial placement of data. Future research will employ power simulations to tune a write access rate threshold for given power and performance goals.

References

[1] Lefurgy C, Rajamani K, Rawson F, Felter W, Kistler M, Keller TW. Energy management for commercial servers. Computer 2003; 36: 39-48.

[2] Mittal S. A survey of power management techniques for phase change memory. Int J Comput-Aid Eng 2014; 1223: 41-54.

[3] Hoefflinger B. ITRS: International technology roadmap for semiconductors. In: Hoefflinger B, editor. Chips 2020: A Guide to the Future of Nanoelectronics. Berlin, Germany: Springer-Verlag, 2012. pp. 161-174.

[4] Raoux S, Burr G, Breitwisch M, Rettner C, Chen Y, Shelby RM, Salinga M, Krebs D, Chen SH, Lung HL et al. Phase-change random access memory: a scalable technology. IBM J Res Dev 2008; 52: 465-479.

[5] Zhou P, Zhao B, Yang J, Zhang Y. A durable and energy efficient main memory using phase change memory technology. SIGARCH Comput Archit News 2009; 37: 14-23.

[6] Caulfield AM, Coburn J, Mollov T, De A, Akel A, He J, Jagatheesan A, Gupta R, Snavely A, Swanson S. Understanding the impact of emerging non-volatile memories on high-performance, IO-intensive computing. In: ACM/IEEE 2010 High Performance Computing, Networking, Storage and Analysis Conference; 13–19 November 2010; Washington, DC, USA. New York, NY, USA: IEEE. pp 1-11.

[7] Lee BC, Ipek E, Mutlu O, Burger D. Architecting phase change memory as a scalable dram alternative. SIGARCH Comput Archit News 2009; 37: 2-13.

[8] Qureshi MK, Srinivasan V, Rivers JA. Scalable high performance main memory system using phase-change memory technology. SIGARCH Comput Archit News 2009; 37: 24-33.

[9] Ang J, Barrett R, Benner R, Burke D, Chan C, Cook J, Donofrio D, Hammond S, Hemmert K, Kelly SM et al. Abstract machine models and proxy architectures for exascale computing. In: IEEE 2014 Hardware–Software Co-Design for High Performance Computing Workshop; 17 November 2014; Piscataway, NJ, USA. New York, NY, USA: IEEE. pp. 25-32.

[10] Burr G, Breitwisch M, Franceschini M, Garetto D, Gopalakrishnan K, Jackson B, Kurdi B, Lam C, Lastras A, Padilla A et al. Phase change memory technology. J Vac Sci Technol B 28: 223-262.

[11] Burr G, Kurdi B, Scott J, Lam C, Gopalakrishnan R, Shenoy R. Overview of candidate device technologies of storage-class memory. IBM Res Dev 2008; 52: 449-464.

[12] Mandelman J, Dennard H, Bronner G, DeBrosse J, Divakaruni R, Li Y, Radens C. Challenges and future directions for the scaling of dynamic random-access memory (DRAM). IBM J Res Dev 2002; 46: 187-212.

[13] Meena J, Sze S, Chand U, Tseng T. Overview of emerging nonvolatile memory technologies. Nanoscale Res Lett 2014; 9: 526-559.

[14] Unat D, Chan C, Zhang W, Williams S, Bachan J, Bell J, Shalf J. ExaSAT: An exascale co-design tool for performance modeling. Int J High Perform C 2015; 29: 209-232.

[15] Li D, Vetter J, Marin G, McCurdy C, Cira C, Liu Z, Yu W. Identifying opportunities for byte-addressable non-volatile memory in extreme-scale scientific applications. In: IEEE 2012 International Parallel and Distributed Processing Symposium; 21–25 May 2012; Washington, DC, USA. New York, NY, USA: IEEE. pp. 945-956.

[16] Doller E. Forging a future in memory—new technologies, new markets, new applications. In: IEEE 2010 Hot Chips Symposium; 22–24 August 2010; Stanford, CA, USA. New York, NY, USA: IEEE. pp. 1-28.

[17] Dong X, Xu C, Xie Y, Jouppi NP. NVSim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. IEEE T Comput Aid D 2012; 31: 994-1007.

[18] Hassan A, Vandierendonck H, Nikolopoulos DS. Software-managed energy-efficient hybrid DRAM/NVM main memory. In: ACM 2015 Computing Frontiers Conference; 18–21 May 2015; Ischia, Italy. New York, NY, USA: ACM. pp. 23:1-23:8.

[19] Suzuki K, Swanson S. The non-volatile memory technology database (nvmdb). Technical Report CS2015-1011, Department of Computer Science & Engineering, San Diego, CA, USA: University of California, May 2015. http://nvmdb.ucsd.edu.

[20] Chan C, Unat D, Lijewski M, Zhang W, Bell J, Shalf J. Software design space exploration for exascale combustion co-design. In: Kunkel JM, Ludwig T, Meuer H, editors. Supercomputing. Berlin, Germany: Springer-Verlag, 2013. pp. 196-212.

[21] Emmett M, Zhang W, Bell JB. High-order algorithms for compressible reacting flow with complex chemistry. Combust Theor Model 2014; 18: 361-387.

[22] Zhang W, Li T. Exploring phase change memory and 3D die-stacking for power/thermal friendly, fast and durable memory architectures. In: IEEE 2009 Parallel Architectures and Compilation Techniques Conference; 12–16 May 2009; Raleigh, NC, USA. New York, NY, USA: IEEE. pp. 101-112.

[23] Ramos LE, Gorbatov E, Bianchini R. Page placement in hybrid memory systems. In: ICM 2011 Supercomputing Conference; 31 May–4 June 2011; Tucson, AZ, USA. New York, NY, USA: ACM. pp. 85-95.

[24] Lee S, Bahn H, Noh SH. Characterizing memory write references for efficient management of hybrid PCM and DRAM memory. In: IEEE 2011 International Symposium on Modeling, Analysis Simulation of Computer and Telecommunication Systems; 25–27 July 2011; Singapore, Singapore. New York, NY, USA: IEEE. pp 168-175.

[25] Seok H, Park Y, Park KH. Migration based page-caching algorithm for a hybrid main memory of dram and pram. In: ACM 2011 Applied Computing Symposium; 21–24 March 2011; Taichung, Taiwan. New York, NY, USA: ACM. pp. 595-599.

[26] Wang Z, Gu Z, Shao Z. Optimized allocation of data variables to PCM/DRAM-based hybrid main memory for real-time embedded systems. IEEE Embedded Syst Lett 2014; 6: 61-64.

[27] Cicotti P, Carrington L. ADAMANT: Tools to capture, analyze and manage data movement. Procedia Comput Sci 2016; 80: 450-460.

[28] Borkar S. 3D integration for energy efficient system design. In: IEEE 2011 Design Automation Conference; 5–9 June 2011; New York, NY, USA. New York, NY, USA: IEEE. pp. 214-219.