Towards Energy Efficient Hybrid On-chip Scratch Pad Memory with Non-Volatile Memory

Jingtong Hu[1], Chun Jason Xue[2], Qingfeng Zhuge[3], Wei-Che Tseng[1], and Edwin H.-M. Sha[1,3]
[1]Dept. of Computer Science, University of Texas at Dallas, Richardson, TX, 75080, USA
[2]Dept. of Computer Science, City University of Hong Kong, Tat Chee Ave, Kowloon, Hong Kong.
[3]Hunan University, Changsha, Hunan, 410082, China.
jthu@utdallas.edu, jasonxue@cityu.edu.hk, qfzhuge@gmail.com, {wxt043000, edsha}@utdallas.edu

***Abstract*—Scratch Pad Memory (SPM), a software-controlled on-chip memory, has been widely adopted in many embedded systems due to its small area and low power consumption. As technology scaling reaches the sub-micron level, leakage energy consumption is surpassing dynamic energy consumption and becoming a critical issue. In this paper, we propose a novel hybrid SPM which consists of non-volatile memory (NVM) and SRAM to take advantage of the ultra-low leakage power consumption and high density of NVM as well as the efficient writes of SRAM. A novel dynamic data allocation algorithm is proposed to make use of the full potential of both NVM and SRAM. According to the experimental results, with the help of the proposed algorithm, the novel hybrid SPM architecture can reduce memory access time by 18.17%, dynamic energy by 24.29%, and leakage power by 37.34% on average compared with a pure SRAM based SPM with the same size area.**

I. Introduction

The use of on-chip memories, including caches and Scratch-pad Memories (SPMs), in computing systems has been a widely-adopted method for addressing long memory access latency for many years. On one hand, cache is probably the mostly widely-used on-chip memory. However, the transition from multi-core (few cores) to many-core (hundreds of cores) architectures has increased the pressure on achieving good cache performance. The power and performance overheads of automatic memory management in hardware, i.e. by caches is becoming prohibitive. Caches consume about half of the processor energy on single-core processor [1], and are expected consume much larger fraction with increase in number of cores.

Most embedded systems are tightly constrained by energy consumption. On-chip cache typically consumes 25%-50% of the processor's area and energy [1]. Therefore, Scratch Pad Memory (SPM), a software-controlled on-chip memory, instead of hardware-controlled cache, has been widely adopted in many embedded systems. However, as the speed of the CMOS transistors keeps increasing along with density, leakage power consumption is becoming a critical issue for memory components with a large number of transistors. In this paper, we propose a novel hybrid on-chip SPM architecture consisting of Non-Volatile Memory (NVM) and SRAM together with an optimal data allocation algorithm. By using the hybrid SPM, we can obtain multiple important benefits, such as ultra-low leakage power consumption, high density, and non-volatility.

Non-volatile memories (NVMs) give us a new way of addressing the memory power consumption problem, because of their attractive characteristics such as low leakage power, high-density, and non-volatility. Previous works have confirmed that using NVMs to build hybrid caches [2], [3], [4], [5], [6] can achieve significant energy saving when configured and used properly. However, to the best of our knowledge, this is the first research on using NVMs to build SPM along with the proper management algorithms.

Even though NVMs have many advantages as described above, there are two challenges we need to answer before we can practically adopt NVMs as on-chip memory. First, a write to NVMs incurs more energy and latency than a read. Second, a majority of NVMs can only sustain limited number of writes compared with SRAM. Consequently, the problem of reducing the number of writes on NVMs has to be solved in order to achieve any practical usage of NVMs as on-chip memory components.

Unlike cache, in which the data management is done in hardware, the data management in SPM is done purely by software. There are many static or dynamic data management algorithms designed for SPM [7], [8], [9], [10], [11]. However, none of them considers the objective of reducing the write activities on NVMs because they are targeting SPMs consisting of pure SRAM. Therefore, it is essential to design a smart SPM memory data management algorithm to reduce the write activities on NVM. Cache replacement policies are proposed in [2], [3], [4], [5], [6] to mitigate the write problems of NVM when it is used as part of hardware-controlled hybrid cache. The policies designed for caches which are very expensive and inappropriate to be implemented in software. Embedded system applications have the benefits of compiler-analyzable data access patterns which could be carefully exploited to produce an effective and efficient data allocation scheme for hybrid SPM architecture.

This work is partially supported by NSF CNS-1015802, Texas NHARP 009741-0020-2009, NSFC 60728206, Changjiang Honorary Chair Professor Scholarship and grants from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 123609][Project No. CityU 123210].

978-3-9810801-7-9/DATE11/©2011 EDAA

In this paper, we first propose a novel hybrid SPM architecture that consists of NVM and SRAM to take advantage of beneficial characteristics from both while still promising an efficient, high endurance, and low leakage on-chip memory solution. Then, we propose a novel optimal dynamic data management algorithm, the Optimal Data Allocation (ODA) Algorithm, which will move the most-written data into SRAM and the most-read data into NVM, to realize the full potential of the hybrid SPM. The ODA algorithm can also be applied in pure SRAM SPM to obtain the optimal data allocation.

According to the experimental results, with the help of the ODA algorithm, the novel hybrid SPM architecture can reduce memory access time by 18.17%, dynamic energy consumption by 24.29% and leakage power consumption by 37.34% when compared with a pure SRAM SPM with the same size area. Thus, the proposed hybrid SPM architecture is a very promising low-power high-performance on-chip memory solution. The major contributions of this paper include:

- a novel low-power and high-performance hybrid SPM architecture that incorporates both NVM and SRAM,
- an optimal dynamic data management algorithm, ODA, for the proposed hybrid SPM architecture which reduces memory access time and dynamic access energy while extending NVM's lifetime, and
- a hybrid SPM simulator to evaluate the proposed hybrid architecture and the optimal data management algorithm.

The rest of this paper is organized as follows. Background and related works are discussed in Section II. Section III presents the hardware and computation model used in this paper. A motivational example is presented in Section IV to illustrate the basic ideas of this paper. The main algorithms are explained in detail in Section V. The experiments are presented in Section VI. Finally, this paper is concluded in Section VII.

II. Background and Related Works

In this section, we introduce the background of hybrid SPM architecture and related works.

Many researchers have proposed SPM data allocation and management techniques to minimize the application execution time. Banakar et al. [1] compared SPM with cache and evaluate simple SPM data management algorithms. Avissar et al. [7] propose a static data allocation scheme for SPMs. Udayakumaran et al. [8], [10] propose a dynamic data allocation method for global and stack data. Dominguez et al. [9] propose dynamic data allocation methods for heap data. Kandemir et al. [11] propose methods to manage data for array-intensive nested loops with regular data access patterns. Ozturk et al. [12] manage data for multiple applications that share the same SPM space. Chen et al. [13] propose dynamic data management for irregular array access patterns. Xue et al. [14] and Ozturk et al. [15] propose loop scheduling algorithms for systems with SPM. Panda et al. [16] propose a technique for efficiently exploiting on-chip SPM by partitioning the applications's scalar and array variables into DRAM and SPM, with the goal of minimizing the total execution time of embedded applications. Takase et al. [17] propose data partitioning and allocation methods for SPM in priority-based preemptive multi-task systems. All of the above research target SPMs which consist of pure SRAM, not the novel SPM architecture proposed in this paper.

Research regarding reducing energy consumption for SPMs are also presented in many existing works. Steinke et al. [18] reduce SPMs' dynamic energy consumption by assigning program and data objects. Kandemir et al. [19] and Chen et al. [20] propose methods to reduce static leakage power consumption. In their works, static leakage power consumption is a big part of power consumption, while in this work, due to the nature of NVMs, very low leakage power exists.

Several previous works [21], [22], [23], [24], [25], [26], [27], [28] confirm that a NVM main memory can achieve significant energy saving with comparable performance to that of a DRAM main memory. Besides using NVMs in main memory, NVMs are also proposed to be used as caches.

The recent emergence of various non-volatile memories (NVMs), such as Magnetic RAM (MRAM) [29], Phase Change Memory (PCM) [30], and embedded Dynamic RAM (eDRAM) [31], has attracted a lot of researchers' interest due to their appealing characteristics, such as low-cost, shock-resistivity, non-volatility, high density and power-economy. [2], [3], [4], [5], [6] have used NVMs to build hybrid caches which can achieve significant energy saving when configured and used properly. All these works focus on integrating NVMs into on-chip cache, whereas our work focuses on SPM. These works imply that it is technically feasible to integrate NVMs with SRAM into on-chip SPM. In these works, cache replacement algorithms are also proposed to reduce the write activities on NVMs in order to reduce the energy consumption and prolong NVMs' endurance. Those cache management polices are neither suitable for SPM management nor can achieve best results for SPMs.

As presented in previous hybrid cache architectures, 3D integration is an essential technique to integrate NVM and SRAM in the same die. 3D hybrid cache with SRAM and NVMs are proposed in [2], [4], [5]. 3D integration can also be adopted to integrate NVM and SRAM for the proposed hybrid on-chip SPM architecture.

III. Hybrid SPM with NVM and SRAM

In this section, we describe the hardware model used in this paper. In the proposed architecture, we use hybrid NVM and SRAM as Scratch Pad Memory. NVM and SRAM share the same address space with the main memory. The CPU can load data from both of them directly and data can be moved between them using special instructions supported by the CPU. Integrating CMOS, which provides the SRAM and logics, with NVM into the same chip can be done with 3D integration [32].

SRAM has higher leakage power. At the same time, writes to SRAM cost less than writes to NVM in terms of time and energy consumption. NVM has very low leakage power and high density. At the same time, the writes to NVM are very expensive. A hybrid architecture offers the low static power consumption and high density of NVMs and the fast and

energy efficient writes of SRAM. Thus, a properly managed hybrid architecture can outperform SPMs consisting of either pure NVM or pure SRAM. In the following sections, we propose the ODA algorithm, which makes full use of the potentials of both the NVM and the SRAM.

IV. Motivational Example

In this section, we use an example to illustrate the main idea of the proposed algorithm. In the example, we show that with proper data allocation, we can take advantage of the benefits of NVM while avoiding the disadvantages of NVM in the hybrid SPM architecture.

Before showing the motivational example, we list notations used in the paper in Table I. Here the cost can be either energy cost or access time cost, depending on the optimization goal.

TABLE I: Notations used in this paper.

Notation	Description
RS	the cost of reading from SRAM.
RN	the cost of reading from NVM.
RM	the cost of reading from main memory.
WS	the cost of writing to SRAM.
WN	the cost of writing to NVM.
WM	the cost of writing to main memory.
$S{\rightarrow}N$	the cost of moving data from SRAM to NVM.
$N{\rightarrow}S$	the cost of moving data from NVM to SRAM.
$S{\rightarrow}M$	the cost of moving data from SRAM to main memory.
$M{\rightarrow}S$	the cost of moving data from main memory to SRAM.
$N{\rightarrow}M$	the cost of moving data from NVM to main memory.
$M{\rightarrow}N$	the cost of moving data from main memory to NVM.

Unlike cache, in which data replacement is purely managed by hardware, data replacement in SPM architectures is managed by software. To efficiently manage data replacement, programs are divided into regions according to the methods from [8], [9], [10], [11]. Basically, we divide the program into regions that are delineated by: (i) the start of each procedure, and (ii) the start of every loop. Before starting the execution of each region, data management code is executed to generate a data allocation which is suitable for this region. Data movement instructions are inserted into the program either by the programmer or the compiler. During the execution of each region, the data allocation remains the same.

In the motivation example, assume we have a region in which 6 data are accessed: A, B, C, D, E, and F. The number of reads and writes is shown in the Table II. Initially, we assume only F is in the SRAM, and all other data is in the main memory. We assume the capacity of the SRAM is 3 and the capacity of the NVM is 2. Accesses to different parts of the memories have different values. The values used in this example are shown in Table III.

Since there is no algorithm designed for a hybrid SPM architecture, we use the algorithm which is designed for pure SRAM SPM architecture in [8] by Udayakumaran et al. for comparison. It is a greedy algorithm which, for each region, the most accessed data is moved into the SPM. In our example, all 6 data have the same number of accesses in this region which is 7. So Udayakumaran's algorithm will choose any 5 of them to put into the NVM and SRAM. One possible solution is: A, B in NVM, C, D, E in SRAM, and F in main memory. With this data allocation, the cost of this program region is 780, which includes the data movement cost and data access cost. Here, the cost could be access time or dynamic energy consumption. With this data allocation, there are 11 writes on the NVM.

TABLE II: Num of access.

Data	Reads	Writes
A	1	6
B	2	5
C	3	4
D	4	3
E	5	2
F	6	1

TABLE III: Cost of each access.

Notation	Costs
RS, WS	1
RN	2.5
WN	7.5
RM, WM	50
$S{\rightarrow}N$	8.5
$N{\rightarrow}S$	3.5
$S{\rightarrow}M$	51
$M{\rightarrow}S$	51
$N{\rightarrow}M$	52.5
$M{\rightarrow}N$	57.5

However, with another data allocation, not only the cost can be reduced but also the number of writes on NVM can be reduced. Instead of the data allocation generated by Udayakumaran's algorithm, if the following data allocation, A, B, C in SRAM, E, F in NVM, D in main memory, is used, the cost of this program region is reduced to 639. And there are only 3 writes on the NVM. Compared with Udayakumaran's algorithm, the cost is reduced by 18.77% and the number of writes on NVM is reduced by 72.73%.

The second data allocation actually is the optimal data allocation which incurs the minimal cost for this region. The ODA algorithm presented in the next section can generate the optimal data allocation. The main reason ODA can reduce the cost is that it differentiates between reads and writes. It then moves the data read the most times into the NVM and moves the data written the most times into the SRAM. In this way, we can take advantage of the NVM while avoiding write activities on the NVM, which reduces the cost and extends the lifetime of the NVM. The details of ODA will be presented in Section V.

V. Dynamic Data Management for Hybrid SPM

In this section, we present the details of the proposed algorithms which generate the data allocation for each program region. First, the problem is formally defined. Then we will illustrate the proposed algorithm with an example. Finally, we present the theorems about the optimality of the ODA algorithm.

A. Problem Definition

Definition 5.1: Given the initial data allocation in the on-chip SPM, size of SRAM, size of NVM, number of accesses to each data in a region (obtained using profiling), the values of RS, RN, RM, WS, WN, WM, $S{\rightarrow}N$, $N{\rightarrow}S$, $S{\rightarrow}M$, $M{\rightarrow}S$, $N{\rightarrow}M$, and $M{\rightarrow}N$, what is the optimal data allocation under which the total cost of the execution of this region is minimized?

The inputs are: initial data allocation in the on-chip SPM, size of SRAM $Size_s$, size of NVM $Size_n$, access sequence in this region, RS, RN, RM, WS, WN, WM, $S{\rightarrow}N$, $N{\rightarrow}S$, $S{\rightarrow}M$, $M{\rightarrow}S$, $N{\rightarrow}M$, $M{\rightarrow}N$.

Output: a data allocation under which the total cost of the execution of this region is minimized. The cost could be either energy or data access time.

B. Optimal Data Allocation for Hybrid SPM

In this section, we introduce the Optimal Data Allocation Algorithm (ODA). The ODA algorithm consists of two steps. In the first step, costs for each data are computed. Then in the second step, the best allocation for the region is determined so that the cost for the execution of this region is minimized. We will use the motivational example alongside the presentation of the ODA algorithm to illustrate the ideas of the algorithm.

Step 1) Computing Costs: In the first step, we will compute costs for each data. We define three costs for each data. The first cost S(D_i) is the total cost for this data if D_i is placed in SRAM. The second cost N(D_i) is the total cost for this data if D_i is in NVM. And the third cost M(D_i) is the total cost for this data if the D_i is in main memory. All costs can be computed as per Equation 1.

$$\sum moving\_cost + (num\_of\_access) * (cost\_of\_each\_access) \quad (1)$$

TABLE IV: Costs for each data.

Data	SRAM S(D_i)	NVM N(D_i)	Main memory M(D_i)
A	58	105	350
B	58	100	350
C	58	95	350
D	58	90	350
E	58	85	350
F	7	30	401

The access information of data is presented in Table II. The three costs of each data is shown in Table IV. The cell corresponding to Column SRAM and Row A is the cost for A if A is in SRAM during the execution of this region. We compute the value for this cell as: $1{\times}RS$ + $6{\times}WS$ + $M{\rightarrow}S$ = 58. Notice that cell F/SRAM is 7 because F is already in SRAM, it does not have the cost of moving data from main memory to SRAM ($M{\rightarrow}S$).

Step 2) Determining Optimal Data Allocation: After we obtain the costs for each data in this region, in the second step, a dynamic programming algorithm is used to compute the best data allocation for the region.

Let C[i, j, k] be the cost of the whole program region where there are j empty spots in the SRAM and k empty spots in the NVM. For $x \leq i$, the x^{th} data has been optimally determined. For $y > i$, the y^{th} data is in the main memory.

The recursive function is shown in Equation 2. According to the recursive function, the Optimal Data Allocation (ODA) Algorithm is presented in Algorithm V.1. In Algorithm V.1, we start with all data placed in the main memory, and iteratively determine the optimal location for each data. The array Action[i,j,k] records the movement instruction for $Data_i$ and

Algorithm V.1 Optimal Data Allocation Algorithm (ODA)

Input: Number of data N_d, $Size_s$, $Size_n$, S(D_i), N(D_i), M(D_i) for each data D_i.
Output: a data allocation under which the total cost of the execution of this region is minimized.

```
1: for m← 1 to N_d do
2:     C[m, Size_s, Size_n] ← Σ_i M(D_i);
3: end for
4: for i ← 1 to N_d do
5:     for j ← Size_s to 0 do
6:         for k ← Size_n to 0 do
7:             C[i, j, k] ← min(C[i−1, j, k], C[i−1, j+1, k] − (M(D_i) −
               S(D_i))), C[i − 1, j, k + 1] − (M(D_i) − N(D_i));
8:             if C[i, j, k] = C[i-1,j,k] then
9:                 Action[i,j,k] = "data D_i in main memory";
10:                Previous[i,j,k] = (i-1,j,k) ;
11:            else if C[i, j, k] = C[i − 1, j + 1, k] − (M(D_i) − S(D_i)))
               then
12:                Action[i,j,k] = "data D_i into SRAM ";
13:                Previous[i,j,k] = (i-1,j+1,k) ;
14:            else if C[i, j, k] = C[i−1, j, k+1]−(M(D_i)−N(D_i)) then
15:                Action[i,j,k] = "data D_i into NVM ";
16:                Previous[i,j,k] = (i-1,j,k+1) ;
17:            end if
18:        end for
19:    end for
20: end for
```

the array Previous[i,j,k] remembers the position of previous optimal solution. After the execution of Algorithm V.1, we can easily construct the optimal data allocation according to the data stored in Action[i,j,k] and Previous[i,j,k]. The time complexity of Algorithm V.1 is O(N_d*$Size_s$*$Size_n$).

According to the ODA algorithm, we construct C[i,j,k] for our example. C[i,j,k] is a 3D array. We show the content of C[i,j,k] in Table V. After the construction of this table, we can extract the optimal solution: putting A, B, C in SRAM, D in main memory, and E, F in NVM. The numbers which are labeled red is the path from which we obtain the optimal solution.

TABLE V: C[i,j,k] for example.

k = 0						
j \ i	1(A)	2(B)	3(C)	4(D)	5(E)	6(F)
0	∞	∞	∞	∞	750	**639**
1	∞	∞	∞	1052	1042	931
2	∞	∞	1354	1344	1334	1223
3	∞	1656	1646	1636	1626	1515

k = 1						
j \ i	1(A)	2(B)	3(C)	4(D)	5(E)	6(F)
0	∞	∞	∞	1015	**1010**	904
1	∞	∞	1312	1307	1302	1196
2	∞	1609	1604	1599	1594	1488
3	1906	1901	1896	1891	1886	1780

k = 2						
j \ i	1(A)	2(B)	3(C)	4(D)	5(E)	6(F)
0	∞	∞	1275	**1275**	1275	1173
1	∞	1567	**1567**	1567	1567	1375
2	1859	**1859**	1859	1859	1859	1757
3	**2151**	2151	2151	2151	2151	2151

$$C[i,j,k] = \begin{cases} \sum_i M(D_i), & \text{if } i = 0, j = Size_s, k = Size_n, \\ \infty & \text{if } j + k < Size_s + Size_n - i \text{ or } j > Size_s \text{ or } k > Size_n, \\ \min(C[i-1,j,k], & \text{if } j + k \geq Size_s + Size_n - i. \\ C[i-1,j+1,k] - (M(D_i) - S(D_i)), \\ C[i-1,j,k+1] - (M(D_i) - N(D_i))) \end{cases} \quad (2)$$

TABLE VI: System Specification.

Component	Baseline System Specification	Target System Specification
CPU Core	Number of cores: 1, frequency: 1.0 GHz	Number of cores: 1, frequency: 1.0 GHz
On-chip SPM SRAM Part	Size: 32 KB, access latency: 5.72 ns, access energy: 0.061 nJ, leakage power: 15.96 mW	Size: 16 KB, access latency: 3.95 ns, access energy: 0.034 nJ, leakage power: 7.99 mW
On-chip SPM NVM Part	None	Size: 64 KB, read latency: 1.55 ns,write latency (SET/RESET): 131.01/61.01 ns, read energy: 0.043 nJ, write energy(SET/RESET): 3.21/3.85 nJ, leakage power: 2.01 mW
Main memory	DDR SDRAM, Size: 512 MB, Access latency: 104.4 ns access energy: 3.26 nJ, leakage power: 200.685 mW	DDR SDRAM, Size: 512 MB, Access latency: 104.4 ns access energy: 3.26 nJ, leakage power: 200.685 mW

VI. Experiments

In this section, first, we present the experimental setup. Then we present the evaluation of the novel hybrid SPM architecture.

A. Experimental Setup

In the experiments, we evaluate a hybrid SPM which consists of SRAM and PCM. We chosen PCM as NVM in this experiment while other types of NVM will also work under the proposed techniques.

We use the NVM simulator *NVsim* [33], which is a PCM-supporting variant of the CACTI tool, to estimate the read/write latencies and energy consumption for a given size of PCM and SRAM. We use 45 *nm* technology with the tool.

We developed a trace-driven hybrid SPM simulator and integrated the NVSim-obtained PCM and SRAM memory model to evaluate the new architecture. The simulator consists of a memory trace processing unit, SPM with SRAM and PCM, and a DDR SDRAM main memory. The target system specifications used for our experiments are shown in Table VI.

We ran benchmarks from Mibench [34] in Simplescalar and collected the memory trace for each region, then fed the memory trace for each region into our simulator. We selected 11 applications from the Mibench benchmark suite: qsort, susan, basicmath, bitcount, dijkstra, patricia, stringsearch, rijndael, sha, CRC32, and FFT.

We implemented our algorithm as a standalone program which takes the memory trace as input and scans through the memory trace to obtain the read and write information of each data. After that, it decides the optimal data allocation for each program region and generates the proper data movement instructions. The data movement instructions for each region are then fed into our SPM simulator. This program can be easily integrated into any compiler.

B. Experimental Results

1) Evaluation of Hybrid SPM: In this section, we compare the hybrid SPM architecture with traditional pure SRAM SPM in terms of memory access time and dynamic energy consumption. The specifications of the hybrid architecture and the baseline architecture are shown in Table VI.

The hybrid SPM has 16KB of SRAM and 64KB of PCM, while the baseline SPM has 32KB of SRAM. Since PCM can be made 4 times denser than SRAM, these two SPMs take up similar amounts of area. We use Udayakumaran's algorithm to manage the data allocation for pure SRAM SPM and use the ODA algorithm to manage the data allocation for hybrid architecture.

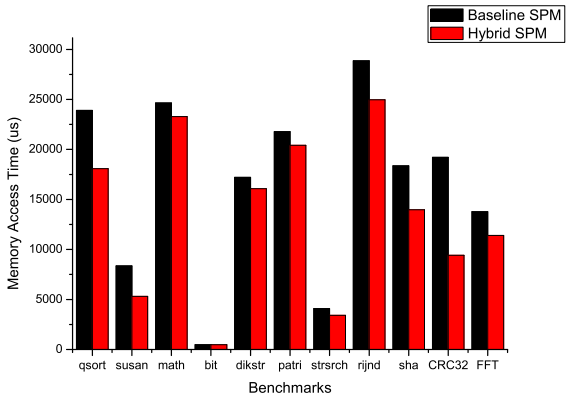

Fig. 1: Memory access time comparison between two SPM architectures.

The comparison of memory access times is shown in Fig. 1. The comparison of dynamic energy costs is shown in Fig. 2. We can see that the hybrid architecture reduces both memory access time and dynamic energy consumption. On average, the hybrid architecture can reduce the memory access time by 18.17% and the dynamic energy by 24.29% in our experiments.

Two main factors contribute to the reduction of memory access time and dynamic energy consumption. First, there are more reads than writes in the data accesses and the ODA algorithm moves the most-read data into NVM and the most-written data into SRAM. Thus, we can take advantage of the inexpensive read of NVM and avoid the expensive writes to the NVM. Second, since NVM is denser than SRAM, for the same area, the hybrid SPM has larger capacity. Therefore, there are fewer off-chip memory access in hybrid SPM than in SRAM

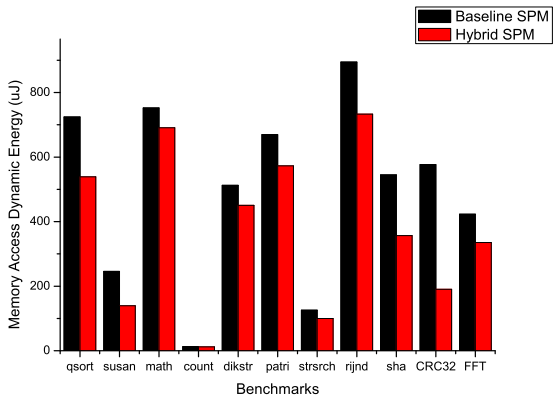

Fig. 2: Memory access dynamic energy comparison between two SPM architectures.

SPM.

The static leakage power consumption for the hybrid architecture is 10 mW as shown in Table VI, and the leakage power for the SRAM architecture is 15.96 mW as shown in Table VI. With the same amount of execution time, the hybrid SPM architecture can reduce the leakage power by 37.34%.

VII. Conclusion

In this paper, we propose a novel hybrid SPM which consists of NVM and SRAM to take advantage of the low leakage power and the high density of NVM and the efficient writes of SRAM. Also, we propose a novel optimal dynamic data management algorithm to realize the full potential of both the SRAM and the NVM. According to our experimental results, with the help of our algorithm, the novel hybrid SPM architecture can reduce memory access time, dynamic energy, leakage power compared to pure SRAM based SPM.

References

[1] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: design alternative for cache on-chip memory in embedded systems," in *CODES '02*, 2002, pp. 73–78.

[2] X. Wu, J. Li, L. Zhang, E. Speight, and Y. Xie, "Power and performance of read-write aware hybrid caches with non-volatile memories," in *DATE '09*, 2009, pp. 737–742.

[3] P. Mangalagiri, K. Sarpatwari, A. Yanamandra, V. Narayanan, Y. Xie, M. J. Irwin, and O. A. Karim, "A low-power phase change memory based hybrid cache architecture," in *GLSVLSI '08*, 2008, pp. 395–398.

[4] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, "Circuit and microarchitecture evaluation of 3d stacking magnetic ram (mram) as a universal memory replacement," in *DAC '08*, 2008, pp. 554–559.

[5] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *ISCA '09*, 2009, pp. 34–45.

[6] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, , and Y. Xie, "Energy- and endurance-aware design of phase change memory caches," in *DATE '10*, 2010, pp. 136–141.

[7] O. Avissar, R. Barua, and D. Stewart, "An optimal memory allocation scheme for scratch-pad-based embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 1, no. 1, pp. 6–26, 2002.

[8] S. Udayakumaran and R. Barua, "Compiler-decided dynamic memory allocation for scratch-pad based embedded systems," in *CASES '03*, 2003, pp. 276–286.

[9] A. Dominguez, S. Udayakumaran, and R. Barua, "Heap data allocation to scratch-pad memory in embedded systems," *J. Embedded Comput.*, vol. 1, no. 4, pp. 521–540, 2005.

[10] S. Udayakumaran, A. Dominguez, and R. Barua, "Dynamic allocation for scratch-pad memory using compile-time decisions," *ACM Trans. Embed. Comput. Syst.*, vol. 5, no. 2, pp. 472–511, 2006.

[11] M. Kandemir, J. Ramanujam, J. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh, "Dynamic management of scratch-pad memory space," in *DAC '01*, 2001, pp. 690–695.

[12] O. Ozturk, M. Kandemir, and I. Kolcu, "Shared scratch-pad memory space management," in *ISQED '06*, 2006, pp. 576–584.

[13] G. Chen, O. Ozturk, M. Kandemir, and M. Karakoy, "Dynamic scratch-pad memory management for irregular array access patterns," in *DATE '06*, 2006, pp. 931–936.

[14] L. Xue, M. Kandemir, G. Chen, and T. Yemliha, "Spm conscious loop scheduling for embedded chip multiprocessors," in *ICPADS '06*, 2006, pp. 391–400.

[15] O. Ozturk, M. Kandemir, and S. H. K. Narayanan, "A scratch-pad memory aware dynamic loop scheduling algorithm," in *ISQED '08*, 2008, pp. 738–743.

[16] P. R. Panda, N. D. Dutt, and A. Nicolau, "Efficient utilization of scratch-pad memory in embedded processor applications," in *EDTC '97*, 1997, p. 7.

[17] H. Takase, H. Tomiyama, and H. Takada, "Partitioning and allocation of scratch-pad memory for priority-based preemptive multi-task systems," in *DATE '10*, 2010, pp. 1124–1129.

[18] S. Steinke, L. Wehmeyer, B. Lee, and P. Marwedel, "Assigning program and data objects to scratchpad for energy reduction," in *DATE '02*, 2002, p. 409.

[19] M. Kandemir, M. J. Irwin, G. Chen, and I. Kolcu, "Banked scratch-pad memory management for reducing leakage energy consumption," in *ICCAD '04*. IEEE Computer Society, 2004, pp. 120–124.

[20] G. Chen and M. Kandemir, "Dataflow analysis for energy-efficient scratch-pad memory management," in *ISLPED '05*, 2005, pp. 327–330.

[21] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ISCA '09*, 2009.

[22] G. Dhiman, R. Ayoub, and T. Rosing, "Pdram: a hybrid pram and dram main memory system," in *DAC '09*, 2009, pp. 664–469.

[23] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," in *ISCA '09*, 2009, pp. 24–33.

[24] J. Hu, C. J. Xue, W.-C. Tseng, Y. He, M. Qiu, and E. H.-M. Sha, "Reducing write activities on non-volatile memories in embedded cmps via data migration and recomputation," in *DAC '10*, 2010, pp. 350–355.

[25] J. Hu, C. J. Xue, W.-C. Tseng, Q. Zhuge, and E. H.-M. Sha, "Minimizing write activities to non-volatile memory via scheduling and recomputation," in *SASP '10*, 2010, pp. 7–12.

[26] J. Hu, W.-C. Tseng, C. J. Xue, Q. Zhuge, Y. Zhao, and E. H.-M. Sha, "Write activity minimization for non-volatile main memory via scheduling and recomputation," *IEEE Transaction on COMPUTER-AIDED DESIGN of Integrated Circuits and Systems*, 2011.

[27] L. Shi, C. J. Xue, J. Hu, W.-C. Tseng, and E. H.-M. Sha, "Write activity reduction on flash main memory via smart victim cache," in *GLVLSI '10*, 2010, pp. 91–94.

[28] W.-C. Tseng, C. J. Xue, Q. Zhuge, J. Hu, and E. H.-M. Sha, "Optimal scheduling to minimize non-volatile memory access time with hardware cache," in *VLSI-SOC '10*, 2010, pp. 131–136.

[29] Y. Chen, H. Li, X. Wang, W. Zhu, W. Xu, and T. Zhang, "A nondestructive self-reference scheme for spin-transfer torque random access memory (stt-ram)," in *DATE '10*, 2010, pp. 148–153.

[30] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé, "Increasing pcm main memory lifetime," in *DATE '10*, 2010, pp. 914–919.

[31] K. C. Chun, P. Jain, and C. H. Kim, "A 0.9v, 65nm logic-compatible embedded dram with >1ms data retention time and 53% less static power than a power-gated sram," in *ISLPED '09*, 2009, pp. 119–120.

[32] Y. Xie, G. H. Loh, B. Black, and K. Bernstein, "Design space exploration for 3d architectures," *J. Emerg. Technol. Comput. Syst.*, vol. 2, no. 2, pp. 65–103, 2006.

[33] X. Dong, N. P. Jouppi, and Y. Xie, "Pcramsim: System-level performance, energy, and area modeling for phase-change ram," in *ICCAD '09*, 2009, pp. 269–275.

[34] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *WWC '01*, 2001, pp. 3–14.