Data Allocation Optimization for Hybrid Scratch Pad Memory With SRAM and Nonvolatile Memory

Jingtong Hu, *Student Member, IEEE*, Chun Jason Xue, *Member, IEEE*, Qingfeng Zhuge, Wei-Che Tseng, and Edwin H.-M. Sha, *Senior Member, IEEE*

***Abstract*— Embedded systems normally have a tight energy budget. Since the on-chip cache typically consumes 25%–50% of the processor's area and energy consumption, scratch pad memory (SPM), which is a software-controlled on-chip memory, has been widely adopted in many embedded systems due to its smaller area and lower power consumption. However, as the speed of the CMOS transistors increases 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 SPM which consists of static random-access memory (SRAM) and nonvolatile memory (NVM) to take advantage of the ultralow leakage power and high density of latter. A novel dynamic data management algorithm is also proposed to make use of the full potential of NVM. According to the experimental results, with the help of the proposed algorithm, the novel hybrid SPM architecture can reduce the memory access time by 18.17%, the dynamic energy by 24.29%, and the leakage power by 37.34% compared with a baseline pure SRAM SPM with the same area.**

***Index Terms*— Cache, energy, magnetic random access memory (MRAM), nonvolatile memory (NVM), on-chip memory, phase change memory, scratch pad memory (SPM).**

I. Introduction

HARDWARE-CONTROLLED on-chip caches are equipped in most modern general-purpose computer systems. Caches can capture program behavior and dynamically adjust content to provide efficient data accesses for CPU. On-chip cache typically consumes 25%–50% of the processor's area and energy [1]. This is not a large concern for general-purpose systems; however, most embedded systems are tightly constrained by area and energy. Thus, instead of hardware-controlled cache, scratch pad memory (SPM), which is a software-controlled on-chip memory, has been widely adopted in many embedded systems as on-chip memory. ARM10E, Analog Devices ADSPTS201S, Motorola M-core MMC221, Renesas SH-X3, and TI's TMX320C6xxx are examples of such embedded processors.

SPM already has a large advantage when compared with cache in terms of area and power consumption. However, as the speed of the CMOS transistor keeps increasing along with density, leakage power consumption is becoming a critical issue for memory components with a large number of transistors. As shown in [2], on average, 33.7% of the total memory energy consumed by SPM is the leakage power. In [2], a compiler-guided leakage optimization for banked SPM is proposed. Instead of targeting general applications, they only focus on array-intensive regular loop nests. Even though they can reduce the leakage power to some extent, the leakage power consumption is still large due to the ever-growing number of CMOS transistors in pure static random-access memory (SRAM) SPMs.

The recent emergence of various nonvolatile memories (NVMs), such as magnetic RAM (MRAM) [3], phase change memory (PCM) [4]–[6], and embedded dynamic RAM (eDRAM) [7], gives us a new way of addressing the memory power consumption problem. NVMs have many attractive characteristics such as low leakage power, high density, nonvolatility, and high immunity to soft errors. Several previous works [5], [6], [8]–[13] confirm that NVM main memory can achieve significant energy savings with comparable performance to that of a DRAM main memory. Besides using NVMs as main memory, research using NVMs to build cache hierarchies [14]–[18] also show that NVMs have advantages over SRAM when configured and used properly. However, to the best of our knowledge, there is no existing research about SPMs using NVMs. In this paper, we explore a novel hybrid SPM architecture consisting of SRAM and NVM. With this hybrid SPM architecture, we can achieve many benefits, such as high density, nonvolatility, and ultralow leakage power, promised by NVMs.

Even though NVMs have many advantages as described above, there are two challenges that need to be addressed before we can practically adopt NVMs as on-chip memory.

Manuscript received October 25, 2011; revised April 19, 2012; accepted May 18, 2012. Date of publication July 24, 2012; date of current version May 20, 2013. This work was supported in part by the National Science Foundation (NSF) under Grant CNS-1015802, the Texas NHARP under Grant 009741-0020-2009, the NSFC under Grant 61173014 and Grant 61133005, China's Thousand-Talent Program, and grants from the Research Grants Council of the Hong Kong Special Administrative Region, China, under Project 123811, Project 123210, and Project 123609.

J. Hu and W.-C. Tseng are with the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080 USA (e-mail: jthu@utdallas.edu; wxt043000@utdallas.edu).

C. J. Xue is with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong (e-mail: jasonxue@cityu.edu.hk).

Q. Zhuge is with the College of Computer Science and Engineering, Chongqing University, Chongqing 400044, China (e-mail: qfzhuge@gmail.com).

E. H.-M. Sha is with the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080 USA, and also with the College of Computer Science and Engineering, Chongqing University, Chongqing 400044, China (e-mail: edsha@utdallas.edu).

Digital Object Identifier 10.1109/TVLSI.2012.2202700

First, a write to NVM incurs more energy and latency compared with a read. Second, the endurance of NVM is limited compared to that of SRAM. Both challenges can be addressed by reducing the write activities on the NVMs. In order to overcome such disadvantages of NVMs, a smart management policy that aims to reduce the write activities on NVMs is essential. The data allocation problem in hybrid SPM differs from the existing data allocation problem for nonuniform memory access architectures in that reads and writes to one of the memory components are asymmetric, and it is desirable to avoid writes to that component.

There are many static and dynamic data management algorithms for SPM [19]–[21]. However, none of the previous data management algorithms differentiates between reads and writes, nor considers reducing the write activities on NVMs because they are targeting SPMs consisting of pure SRAM.

In this paper, the target platform is application-specific single-core embedded systems, on which programs can be comprehensively analyzed. We first propose a novel hybrid SPM which consists of SRAM and NVM to take advantage of the ultralow leakage power and high density of NVMs. Then, we propose the novel hybrid SPM data management algorithms, which will move the most written data into SRAM and the most read data into NVM, to make use of the full potential of the NVM. By taking advantage of the benefits of both NVMs and SRAM, this paper aims to provide a low-power, yet long-lasting, on-chip memory solution. We evaluate the proposed architecture and data allocation techniques. According to the experimental results, with the help of proposed algorithm the novel hybrid SPM architecture can reduce the memory access time by 18.17%, the dynamic energy by 24.29%, and the leakage power by 37.34% compared with a baseline pure SRAM SPM with the same size area.

The major contributions of this paper include the following.

1) To our knowledge, this is the first paper to explore hybrid nonvolatile SPM architectures.
2) We propose a novel dynamic data management algorithm for the novel SPM architecture which can reduce the memory access time and dynamic access energy while extending the NVM's lifetime.
3) We propose an algorithm to select the best optimal data allocation from a global point of view.
4) We develop a hybrid SPM simulator to evaluate our architecture and 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 Sections V and VI. Experiments are presented in Section VII. Finally, Section VIII concludes this paper.

II. Background and Related Works

In this section, we introduce the background of data management for SPMs and related works. Unlike cache, in which the data replacement is purely managed by hardware, data replacement is managed by software in SPM architectures. Either compilers or programmers need to decide how to manage the data. Many researchers have proposed SPM data allocation and management techniques to minimize the application execution time.

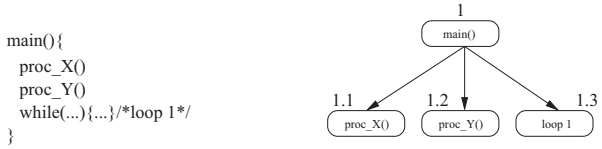


Fig. 1. Example of a program outline and its regions.

In order to achieve efficiency and take advantage of temporal locality, dynamic data allocation algorithms have been proposed. In these approaches, basically, the program is divided into regions that are delineated by *program points*: 1) the start of each procedure and 2) the start of every loop. Before starting the execution of each region, data management code is executed to generate a data allocation that 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. Udayakumaran *et al*. [19], [22] have proposed methods for allocating global and stack data. Dominguez *et al*. [23] have proposed methods for allocating heap data. Fig. 1 shows an example of such methods. The left part shows a program outline, and the right part shows the corresponding regions. In this example, the program consists of three regions, including two procedures, namely proc_X(), proc_Y(), and one while loop. The regions are marked with timestamps as described in their works. The regions will execute in an ascending order of timestamps. Since it is possible that multiple paths lead to the same program point, a program point can have multiple timestamps. In such cases, each timestamp will have a corresponding data allocation. This approach does not have as high an overhead as the first type of methods and also can achieve better locality than the second type of methods. Therefore, this is the approach that we also take in this paper for the novel hybrid SPM architectures. All existing regional data allocation schemes are for pure SRAM SPM and are not suitable for hybrid SPM architectures. In this paper, we propose an optimal data allocation algorithm for each predetermined region for hybrid SPM.

The recent emergence of various NVMs, such as MRAM [3], [24], [25], and PCM [4], [26] has attracted a lot of researchers' interest thanks to their appealing characteristics, such as their low cost, resistance to shock, nonvolatility, high density, and power economy (almost no leakage power). Several previous works [5], [6], [12], [13], [27]–[34] have confirmed that an 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 have also been proposed to be used as caches. Mangalagiri *et al*. [15] proposed a low-power PCM-based hybrid cache architecture. Dong *et al*. [16] conducted circuit and microarchitecture evaluation of 3-D stacking MRAM as a

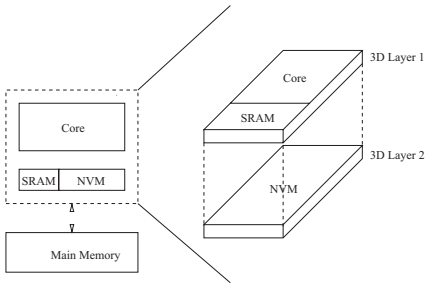


Fig. 2. SPM architecture.

universal memory replacement, in which they use MRAM as on-chip cache. Wu *et al*. proposed hybrid cache architectures with NVMs in [17]. Joo *et al*. [18] proposed energy- and endurance-aware design of PCM cache. All these works integrate NVMs into cache hierarchies. The above works naturally imply that it is technically feasible to integrate NVMs with SRAM into on-chip memory. They also propose cache replacement algorithms to reduce the write activities on NVMs to reduce the energy consumption and prolong NVMs' endurance. However, those algorithms are either unsuitable for SPM management or cannot achieve the best results for SPMs.

III. Hybrid SPM With SRAM and NVM

In this section, the hardware model used in this paper is described. In our architecture, we are use a hybrid of SRAM and NVM as SPM. SRAM and NVM 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.

Hybrid SPM can be fabricated with either 2-D or 3-D chips. However, 3-D integration is a feasible and promising approach to fabricate the hybrid SPM. In 3-D chips, multiple active device layers are stacked together with short and fast vertical interconnects. Among the several benefits offered by 3-D integrations, the mixed-technology stacking is especially attractive for stacking NVM on top of CMOS logics, providing a means to mount NVM layers on top of logic layers, so that designers can take full advantage of the attractive benefits that NVM provides.

The proposed architecture is shown in Fig. 2. As shown, the on-chip memory consists of SRAM and NVM. For fabrication, SRAM can be fitted into to the same layer as the core and NVM can be fitted into a separate layer.

IV. Motivational Example

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

There are two steps in the proposed techniques. In the first step, the minimal memory access cost in a particular region and the corresponding optimal data allocations are determined.

TABLE I
Notations Used in This Paper

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

There might be several data allocations that can achieve the minimal memory access cost. Since the data allocation determined for the current region is the initial data allocation for the next region, it affects the memory access cost of the next region. In the second step, one of the several optimal data allocations is chosen so that the next region can achieve the best result.

In this paper, the region partitioning method in [19] is used to determine the regions. Basically, the program is divided into regions that are delineated by *program points*: 1) the start of each procedure and 2) 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. The proposed algorithm in this paper can generate optimal data allocations for each region determined by the partitioning method.

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

Let us use the example described in Fig. 1 as the motivational example. Here we have three regions: proc_X(), proc_Y(), and a while loop. We will only focus on the regions proc_X() and proc_Y() in this example. proc_X() executes prior to proc_Y(). Assume that in region proc_X() six data are accessed: A, B, C, D, E, and F. For simplicity, all data have the same size in this example. The number of reads and writes is shown in Table II. Also we assume that in region proc_Y(), data A, B, and C are accessed 10 times, as shown in the Table II. Initially, we assume that 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 costs. The costs used in this example are shown in Table III.

We define three costs for each data. The first cost S(D_i) is the total cost for this data if D_i is in SRAM in a particular region. 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. Let Reads(D_i) be the number of reads for data D_i in a region. Let Writes (D_i) be the number of writes for data D_i in a region. These three

TABLE II
NUMBER OF ACCESSES

Data	Reads	Writes
proc_X()		
A	1	6
B	2	5
C	3	4
D	4	3
E	5	2
F	6	1
proc_Y()		
A	1	9
B	1	9
C	1	9

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

costs can be computed with the following equations.

$$S(D_i) = \text{Reads}(D_i) \times RS + \text{Writes}(D_i) \times WS + (\text{moving\_cost}) \quad (1)$$

$$N(D_i) = \text{Reads}(D_i) \times RN + \text{Writes}(D_i) \times WN + (\text{moving\_cost}) \quad (2)$$

$$M(D_i) = \text{Reads}(D_i) \times RM + \text{Writes}(D_i) \times WM + (\text{moving\_cost}). \quad (3)$$

Three different costs of each data for proc_X() is shown in Table IV. The cell corresponding to A/SRAM shows the cost for A if A is in SRAM during the execution of this region. We compute the value for this cell as $1 \times \text{RS} + 6 \times \text{WS} + M \rightarrow S = 58$. Notice that F/SRAM is 7 because F is already in SRAM; it does not have the cost of moving data from main memory to SRAM ($M \rightarrow S$). The cost table will be used in Section V to generate the optimal data allocation for proc_X().

Since there is no algorithm designed for a hybrid SPM architecture, we use the algorithm designed for pure SRAM SPM architecture in [22] for comparison purposes. It is a greedy algorithm. For each region, the most accessed data is moved into the SPM. In our example, all six data have the same number of accesses in this region which is 7. So Udayakumaran's algorithm will choose any five of them to put into the NVM and SRAM. One possible solution is:

TABLE IV
THREE DIFFERENT COSTS FOR EACH DATA OF proc_X()

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

TABLE V
DATA ALLOCATION COMPARISON

Algorithm	Data Allocation	No. of writes on NVM	Cost
Udayakumaran's algorithm	A, B in NVM, C, D, E in SRAM, and F in main memory	11	780
DAHS	A, B, C in SRAM, E, F in NVM, D in main memory	3	639

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. It is computed as $\text{N}(A) + \text{N}(B) + \text{S}(C) + \text{S}(D) + \text{S}(E) + \text{M}(F) = 105 + 100 + 58 + 58 + 58 + 401 = 780$. Here, the cost could be access time or dynamic energy consumption. With this data allocation, there are 11 writes on the NVM.

However, with another data allocation, not only the cost 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; and D in main memory, is used, the cost of this program region is reduced to 639. The cost can be computed as $\text{N}(E) + \text{N}(F) + \text{S}(A) + \text{S}(B) + \text{S}(C) + \text{M}(D) = 85 + 30 + 58 + 58 + 58 + 350 = 639$. There are only three 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 region proc_X(). The data allocation for hybrid SPM (DAHS) algorithm presented in the next section can generate the optimal data allocation. Table V shows the comparison of the data allocations of these two difference algorithms. The main reason why DAHS 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 DAHS will be presented in Section V.

DAHS can find the optimal data allocations for each predetermined program region. However, there might be several different data allocations that can achieve the minimal cost. For this example, among all the data allocations for region proc_X(), the data allocation described above is not

TABLE VI
OPTIMAL DATA ALLOCATIONS

Optimal data allocations	SRAM	NVM	Main memory	Cost
I	A, B, C	E, F	D	639
II	B, C, D	E, F	A	639
III	A, C, D	E, F	B	639
IV	A, B, D	E, F	C	639

the only one that can achieve the minimal memory access cost. Including the data allocation described above, four different data allocations can achieve the same results. They are shown in Table VI.

These four data allocations can achieve the same results for region proc_X(). However, as the initial data allocation for proc_Y(), they will lead to different memory access cost for proc_Y(). If we use allocation 1 as the initial data allocation for proc_Y(), the minimal cost for proc_Y() will be 30. If we use any of the other three data allocation, the minimal cost for proc_Y() will be 132. Therefore, the best data allocation for proc_Y() will be the first data allocation. The multiple region data allocation (MURDA) algorithm described in Section VI will show how to determine the best data allocation for the next region among all the optimal data allocations for current region.

V. REGIONAL DYNAMIC DATA MANAGEMENT FOR HYBRID SPM

In this section, we present the details of the proposed algorithms that generate optimal data allocations for each predetermined program region. First, we formally define our problem. Then we will present the algorithms.

A. Problem Definition

In this subsection, we formally define the problem.

Definition 5.1 Regional Optimal Data Allocation Problem: Given the initial data allocation in the on-chip SPM, size of SRAM, size of NVM, number of accesses to each data in this region (can be obtained using profiling), RS, RN, RM, WS, WN, WM, S→N, N→S, S→M, M→S, N→M, and M→N, what is the optimal data allocation under which the total cost of the execution of this region is minimized?

The inputs to our algorithm are the initial data allocation in the on-chip SPM, size of SRAM Size_s, size of NVM Size_n, access counts in this region, RS, RN, RM, WS, WN, WM, S→N, N→S, S→M, M→S, N→M, and M→N.

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

B. Optimal DAHS

In this section, we introduce the process of finding the optimal data allocation for each predetermined region. The process consists of three parts. In the first step, we will compute costs for each data. Then we will determine the optimal data allocation for the region with the DAHS so that the cost for the execution of this region is minimized. In the third step, based on the tables generated by DAHS, we construct the optimal solutions. We will use an example alongside the presentation to illustrate the ideas of our algorithm.

For illustration purposes, we use proc_X() in the example presented in Section IV to demonstrate the process of the DAHS algorithm.

The step of computing costs for each data was already shown in Section IV. After we have the costs for each data, as shown in Table IV, a dynamic programming algorithm is used to compute the best data allocation for the region. In this subsection, we first define a cost array C. Then we will present the optimal substructure. Lastly, according to the optimal substructure, we will present the DAHS algorithm.

Let $\text{Size}(D_i)$ be the size of data i. Let $C[i, j, k]$ be the cost of the whole program region where there are j bytes empty in the SRAM and k bytes empty in the NVM. For $x \leq i$, data D_x has been optimally determined. For $y > i$, data D_y is in the main memory.

The recursive function is shown in (4), shown at the bottom of the page. This equation has three parts. In the first part, if $i = 0, j = \text{Size}_s, k = \text{Size}_n$, which means that all the data is in the main memory and all on-chip memories are empty. In this case, $C[i, j, k]$ should be the sum of all $M(D_i)$ since all the data is still in the main memory. In the second part, for any $j > \text{Size}_s$ or $k > \text{Size}_n$, $C[i, j, k]$ will be infinity since j cannot be larger than Size_s and k cannot be larger than Size_n. Also, if $\sum_{x=1}^{i} \text{Size}(D_x) + j + k < \text{Size}_s + \text{Size}_n$, $C[i, j, k]$ is set to be infinity since it is impossible. In the third part, if $\sum_{x=1}^{i} \text{Size}(D_x) + j + k \geq \text{Size}_s + \text{Size}_n$, we set $C[i, j, k]$ to be the minimum of the following three values: 1) $C[i-1, j, k]$, which means that Data_i is still in the main memory; 2) $C[i-1, j+\text{Size}(D_i), k] - (M(D_i) - S(D_i))$, which means that Data_i is moved into the SRAM; and 3) $C[i-1, j, k+\text{Size}(D_i)] - (M(D_i) - N(D_i))$, which means that Data_i is moved into the NVM.

According to the recursive function, the DAHS algorithm is presented in Algorithm 1. In line 1, $C[i, j, k]$ is initialized

$$C[i, j, k] = \begin{cases} \sum_i M(D_i), & \text{if } i = 0, j = \text{Size}_s, k = \text{Size}_n \\ \infty, & \text{if } j + k < \text{Size}_s + \text{Size}_n - \sum_{x=1}^{i} \text{Size}(D_x) \\ & \text{or } j > \text{Size}_s \text{ or } k > \text{Size}_n \\ \min(C[i-1, j, k], & \\ C[i-1, j+\text{Size}(D_i), k] - (M(D_i) - S(D_i)), & \\ C[i-1, j, k+\text{Size}(D_i)] - (M(D_i) - N(D_i))), & \text{if } j + k \geq \text{Size}_s + \text{Size}_n - \sum_{x=1}^{i} \text{Size}(D_x) \end{cases} \quad (4)$$

Algorithm 1 DAHS Algorithm

Require: Number of data N_d, Size_s, Size_n, $\text{S}(D_i)$, $\text{N}(D_i)$, $\text{M}(D_i)$ for each data D_i.

Ensure: a data allocation under which the total cost of the execution of this region is minimized.

1: C[0, Size_s, Size_n] $\leftarrow \sum_i M(D_i)$;
2: **for** i $\leftarrow$ 1 to N_d **do**
3: **for** j $\leftarrow$ Size_s to 0 **do**
4: **for** k $\leftarrow$ Size_n to 0 **do**
5: **if** $\sum_{x=1}^{i} \text{Size}(D_x) + j + k < \text{Size}_s + \text{Size}_n$ **then**
6: $C[i, j, k] \leftarrow \infty$;
7: **end if**
8: **end for**
9: **end for**
10: **end for**
11: **for** $i \leftarrow 1$ to N_d **do**
12: **for** $j \leftarrow \text{Size}_s$ to 0 **do**
13: **for** $k \leftarrow \text{Size}_n$ to 0 **do**
14: **if** $i + j + k \geq \text{Size}_s + \text{Size}_n$ **then**
15: $C[i, j, k] \leftarrow \min(C[i-1, j, k], C[i-1, j+\text{Size}(D_i), k] - (M(D_i) - S(D_i))), C[i-1, j, k+\text{Size}(D_i)] - (M(D_i) - N(D_i))$;
16: **if** $C[i, j, k] = \text{C}[i-1, j, k]$ **then**
17: Record$[i, j, k] \Leftarrow$ ("data D_i in main memory," $(i-1, j, k)$);
18: **end if**
19: **if** $C[i, j, k] = C[i-1, j+\text{Size}(D_i), k] - (M(D_i) - S(D_i))$ **then**
20: Record$[i, j, k] \Leftarrow$ ("data D_i into SRAM,"$(i-1, j+\text{Size}(D_i), k)$);
21: **end if**
22: **if** $C[i, j, k] = C[i-1, j, k+\text{Size}(D_i)] - (M(D_i) - N(D_i))$ **then**
23: Record$[i, j, k] \Leftarrow$ ("data D_i into NVM,"$[i-1, j, k+\text{Size}(D_i)]$);
24: **end if**
25: **end if**
26: **end for**
27: **end for**
28: **end for**

according to the first part of 4. From line 1 to line 1, $C[i, j, k]$ is initialized to infinity according to the second part of 4. From line 1 to line 1, we compute $C[i, j, k]$ according to part 3 of 4. In lines 1, 1, and 1, we use a 3-D array Record$[i, j, k]$ to keep track of the actions and previous locations so that we can construct the solutions. The content of each unit of Record$[i, j, k]$ is a set of 2-tuples. The first element in each tuple is the action for Data_i, i.e., where to place Data_i; and the second element is the previous $C[i, j, k]$ from which we can construct an optimal solution. Note that we use "$\Leftarrow$" here, which means adding a new tuple to Record$[i, j, k]$. Each unit in Record$[i, j, k]$ can hold multiple tuples. Using Record$[i, j, k]$, optimal solutions can be easily constructed. The time complexity of Algorithm 1 is $\text{O}(N_d \times \text{Size}_s \times \text{Size}_n)$.

Algorithm 2 MURDA

Require: Optimal data allocations from previous region DL, data access information for the next region.

Ensure: The best optimal data allocation in DL.

1: best_layout$\leftarrow$NULL;
2: optimal_cost$\leftarrow\infty$;
3: **for** d in DL **do**
4: use d to compute cost tables;
5: call Algorithm 1 (DAHS);
6: get the optimal cost CNR for the next region;
7: **if** CNR$\leq$optimal_cost **then**
8: best_layout$\leftarrow$d;
9: optimal_cost$\leftarrow$CNR;
10: **end if**
11: **end for**

VI. Global Data Allocation

From the motivational example, we know that there may be several possible optimal solutions for each predetermined region. For example, there are four different optimal data allocations for region proc_X. All the data allocations have the same access cost for region proc_$X()$. However, since the data allocation for current region will be the initial data allocation for the next region proc_$Y()$, each optimal data allocation will lead to a different memory access cost for the region proc_$Y()$. Therefore, we need a method to choose the best data allocation for the next region among all the optimal data allocations for region proc_$X()$. Algorithm 2 is proposed to achieve this goal.

In Algorithm 2, we enumerate all the possible optimal data allocations and use them as the initial data allocation to compute the optimal data allocation for the next region. Then we choose the data allocation that will generate the least cost-optimal data allocation for the next region. Since the number of regions and the number of optimal data allocations in each region are usually not large, this enumeration is reasonable.

VII. Experiments

In the experiments, we first evaluate the proposed algorithms presented in this paper. Then we evaluate our novel architecture. Experimental setup is presented in Section VII-A. The experimental results of the proposed algorithms are presented in Section VII-B1 and the experimental results for the proposed hybrid SPM are presented in Section VII-B2.

A. Experimental Setup

In our experiments, we evaluate a hybrid SPM which consists of SRAM and PCM. The reason why we chose PCM among all the NVMs is that it is currently the most promising NVM technology in terms of lifetime and access latency.

We used the PCM memory simulator NVsim [35], which is a PCM-supporting variant of the CACTI tool, to estimate the read/write latencies and their energy consumption for a given size of PCM memory. We used 45-nm technology with the tool. We also used NVsim to obtain the access latencies and energy consumption for a given size of SRAM memory. Area was used as the optimization goal in NVsim.

TABLE VII
TARGET SYSTEM SPECIFICATION

Component	Description
CPU	Frequency: 1.0 GHz
On-chip SPM SRAM part	Size: 16 kB, access latency: 3.95 ns, access energy: 0.034 nJ, leakage power: 7.99 mW
On-chip SPM NVM part	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

TABLE VIII
RUNNING TIME OVERHEAD

Benchmarks	Udayakumaran's algorithm	DAHS + MURDA
	Time (s)	Time (s)
qsort	47.14	359.96
susan	34.07	150.10
basicmath	41.6	521.87
bitcount	1.31	25.48
dijkstra	37.2	329.04
patricia	47.95	494.72
stringsearch	5.16	172.64
rijndael	7.91	142.16
sha	10.2	56.22
CRC32	5.7	44.51
FFT	4.07	58.32

We developed a trace-driven hybrid SPM simulator and integrated the 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 specification used for our experiments is shown in Table VII.

We ran benchmarks from Mibench [36] in Simplescalar and collected the memory trace for each region. We then fed the memory trace for each region into our simulator. We used the criteria described in [22] to determine regions. There are two kinds of program points: 1) the start of each procedure and 2) just before the start of every loop. Codes between two program points are defined as one region. We selected 11 applications from the Mibench benchmark site: qsort, susan, basicmath, bitcount, dijkstra, patricia, stringsearch, rijndael, sha, CRC32, and fast Fourier transform (FFT).

We implemented our algorithm as a standalone program which takes the memory trace as input and profiles through the memory trace. 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.

B. Experimental Results

1) Evaluation of the Proposed Algorithms: In this section, we compare the proposed algorithms with the greedy algorithm presented in [19], which is designed for pure SRAM SPM. Table VIII shows the running time (in seconds) of Udayakumaran's algorithm and the proposed algorithms. From the table, we can see that Udayakumaran's algorithm can finish in less than 1 min for all the benchmarks. The proposed DAHS and MURDA algorithms together can finish in less than 10 min for all the benchmarks.

TABLE IX
ACCESS TIME COMPARISON

Benchmarks	Udayakumaran's algorithm	DAHS + MURDA	
	Time (μs)	Time (μs)	Imprv.
qsort	21 404.09	18 086.06	15.50%
susan	6458.90	5312.89	17.74%
basicmath	23 698.34	23 275.89	1.78%
bitcount	2112.62	657.93	77.38%
dijkstra	17 837.25	16 081.10	9.85%
patricia	20 533.64	20 410.26	0.60%
stringsearch	3978.83	414.52	14.18%
rijndael	25 721.65	24 960.97	2.96%
sha	15 933.03	13 974.33	12.29%
CRC32	12 582.21	9411.98	25.20%
FFT	12 911.70	11 407.70	11.65%
Average			17.19%

TABLE X
DYNAMIC ENERGY CONSUMPTION COMPARISON

Benchmarks	Udayakumaran's algorithm	DAHS + MURDA	
	Energy (μJ)	Energy (μJ)	Imprv.
qsort	613.61	539.00	12.16%
susan	172.00	139.22	19.06%
basicmath	726.12	690.79	4.87%
bitcount	65.79	12.15	81.53%
dijkstra	526.48	450.45	14.44%
patricia	592.32	572.70	3.31%
stringsearch	115.87	100.08	13.63%
rijndael	783.40	733.21	6.41%
sha	423.11	356.54	15.73%
CRC32	342.04	190.15	44.41%
FFT	388.72	335.34	13.73%
Average			20.84%

Table IX shows the experimental results when we use time as the optimization objective. In this set of experiments, both algorithms are used to determine the hyrbid SPM data allocation for each program region. For each algorithm, we collected the on-chip memory accesses time. The first column of Table IX shows the benchmarks' names. The second column shows the memory access time when we use Udayakumaran's algorithm to determine the data allocation for each region. The third column shows the memory access time when we use our algorithms to determine the data allocation for each region. The fourth column shows the percentage that the proposed DAHS and MURDA algorithms can reduce compared with Udayakumaran's algorithm. From the table, we can see that the proposed algorithms can reduce the memory access time by 17.19% on average.

TABLE XI
NUMBER OF WRITES COMPARISON

Benchmarks	Udayakumaran's algorithm	DAHS + MURDA	
	Writes	Writes	Imprv
qsort	71 148	15 226	78.60%
susan	4488	133	97.04%
basicmath	46 374	13 833	70.17%
bitcount	135	13	90.37%
dijkstra	66 684	21 297	68.06%
patricia	60 405	14 651	75.75%
stringsearch	7154	90	98.74%
rijndael	15 359	12 302	19.90%
sha	100 322	37 577	62.54%
CRC32	53 083	41	99.92%
FFT	92 388	16 467	82.18%
Average			76.66%

TABLE XII
BASELINE SYSTEM SPECIFICATION

Component	Description
CPU	Frequency: 1.0 GHz
SRAM SPM	Size: 32 kB, access latency: 5.72 ns, access energy: 0.061 nJ, Leakage power: 15.96 mW
Main memory	DDR SDRAM, Size: 512 MB, Access latency: 104.4 ns Access energy: 3.26 nJ, Leakage power: 200.685 mW

Table X shows the experimental results when we use dynamic energy as the optimization objective. From the table, we can see that the proposed algorithms can reduce the dynamic energy by 20.84% on average.

Table XI shows the experimental results when we use the number of writes on NVM as the objective. From the table, we can see that the proposed algorithms can reduce the number of writes on NVM by 76.66% on average. Since the NVM normally has a limited number of writes, by reducing the number of writes on the NVM, we can extend the NVM's lifetime. By reducing 76.66% of the number of writes on NVM, our method can extend the lifetime of NVM by over four times.

2) Evaluation of Hybrid SPM: In this section, we compare the hybrid SPM architecture with a traditional purely SRAM SPM in terms of memory access time and dynamic energy consumption. The hybrid architecture is shown in Table VII, while the baseline architecture is shown in Table XII.

The hybrid SPM has 16 kB SRAM and 64 kB PCM, while the baseline SPM has 32 kB SRAM. Since the PCM can be made four times denser than SRAM, these two SPMs occupy similar areas. We use Udayakumaran's algorithm to manage the data allocation for pure SRAM SPM and use the DAHS algorithm to manage the data allocation for hybrid architecture.

On average, the hybrid architecture can reduce the memory access time by 18.17% and the dynamic energy by 24.29% in our experiments. The static leakage power consumption for the hybrid architecture is 10 mW, as shown in Table VII, and the leakage power for the SRAM architecture is 15.96 mW, as shown in Table XII. For the same running time, the new architecture can reduce the leakage power by 37.34%.

VIII. CONCLUSION

In this paper, we proposed a novel hybrid SPM that consisted of SRAM and NVM to take advantage of the ultralow leakage power and higher density of NVM. Also, we proposed a novel dynamic data management algorithm to make use of the full potential of NVM. According to the experimental results, with the help of the proposed algorithms, the novel hybrid SPM architecture could reduce the memory access time by 18.17%, the dynamic energy by 24.29%, and the leakage power by 37.34% compared with a baseline pure SRAM SPM with same area size.

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 *Proc. CODES*, 2002, pp. 73–78.

[2] M. Kandemir, M. J. Irwin, G. Chen, and I. Kolcu, "Banked scratch-pad memory management for reducing leakage energy consumption," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, Nov. 2004, pp. 120–124.

[3] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM," in *Proc. ISLPED*, 2005, pp. 459–462.

[4] F. Yeung, S.-J. Ahn, Y.-N. Hwang, C.-W. Jeong, Y.-J. Song, S.-Y. Lee, S.-H. Lee, K.-C. Ryoo, J.-H. Park, J.-M. Shin, W.-C. Jeong, Y.-T. Kim, G.-H. Koh, G.-T. Jeong, H.-S. Jeong, and K. Kim, "$Ge_2Sb_2Te_5$ confined structures and integration of 64 Mb phase-change random access memory," *Jpn. J. Appl. Phys.*, vol. 44, no. 4B, pp. 2691–2695, 2005.

[5] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *Proc. 36th Annu. Int. Symp. Comput. Arch.*, 2009, pp. 14–23.

[6] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in *Proc. 36th Annu. Int. Symp. Comput. Arch.*, 2009, pp. 2–13.

[7] K. C. Chun, P. Jain, and C. H. Kim, "A 0.9 V, 65 nm logic-compatible embedded DRAM with >1 ms data retention time and 53% less static power than a power-gated SRAM," in *Proc. 14th ACM/IEEE Int. Symp. Low Power Electron. Design*, Aug. 2009, pp. 119–120.

[8] Y. Wang, J. Du, J. Hu, Q. Zhuge, and E. H.-M. Sha, "Loop scheduling optimization for chip-multiprocessors with non-volatile main memory," in *Proc. ICASSP*, 2012, pp. 1553–1556.

[9] Q. Li, Y. Zhao, J. Hu, C. J. Xue, E. H.-M. Sha, and Y. He, "MGC: Multiple graph-coloring for non-volatile memory based hybrid scratch-pad memory," in *Proc. INTERACT-16*, 2012, pp. 1–6.

[10] J. Hu, Q. Zhuge, C. J. Xue, W.-C. Tseng, and E. H.-M. Sha, "Optimizing data allocation and memory configuration for non-volatile memory-based hybrid SPM on embedded CMPs," in *Proc. HPPAC*, 2012, pp. 976–983.

[11] J. Hu, C. J. Xue, Q. Zhuge, W.-C. Tseng, and E. H.-M. Sha, "Write activity reduction on non-volatile main memories for embedded chip multi-processors," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 3, pp. 9-1–9-25, Apr. 2012.

[12] 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 *Proc. 20th Symp. Great Lakes Symp. VLSI*, 2010, pp. 91–94.

[13] 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 *Proc. 18th IEEE/IFIP VLSI Syst. Chip Conf.*, Sep. 2010, pp. 131–136.

[14] 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 *Proc. Design Autom. Test Eur. Conf.*, Apr. 2009, pp. 737–742.

[15] 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 *Proc. 20th Symp. Great Lakes Symp. VLSI*, 2008, pp. 395–398.

[16] 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 *Proc. 45th Annu. Design Autom. Conf.*, 2008, pp. 554–559.

[17] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in *Proc. 36th Annu. Int. Symp. Comput. Arch.*, 2009, pp. 34–45.

[18] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energy- and endurance-aware design of phase change memory caches," in *Proc. Design Autom. Test Eur. Conf.*, 2010, pp. 136–141.

[19] 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.

[20] M. Kandemir, J. Ramanujam, J. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh, "Dynamic management of scratch-pad memory space," in *Proc. Design Autom. Conf.*, 2001, pp. 690–695.

[21] M. T. Kandemir, J. Ramanujam, M. J. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh, "A compiler-based approach for dynamically managing scratch-pad memories in embedded systems," *IEEE Trans. Comput.-Aided Integr. Circuits Syst.*, vol. 23, no. 2, pp. 243–260, Feb. 2004.

[22] S. Udayakumaran and R. Barua, "Compiler-decided dynamic memory allocation for scratch-pad based embedded systems," in *Proc. Int. Conf. Compil., Arch. Synthesis Embedded Syst.*, 2003, pp. 276–286.

[23] 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.

[24] 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 *Proc. Design Autom. Test Eur. Conf.*, 2010, pp. 148–153.

[25] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "Energy reduction for STT-RAM using early write termination," in *Proc. Int. Conf. Comput.-Aided Design*, 2009, pp. 264–268.

[26] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé, "Increasing PCM main memory lifetime," in *Proc. Design Autom. Test Eur. Conf.*, 2010, pp. 914–919.

[27] 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 *Proc. 8th IEEE Symp. Appl. Specific Process.*, Jun. 2010, pp. 7–12.

[28] 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 *Proc. Design Autom. Conf.*, 2010, pp. 350–355.

[29] J. Hu, C. J. Xue, Q. Zhuge, W.-C. Tseng, and E. H.-M. Sha, "Toward energy efficient hybrid on-chip scratch pad memory with non-volatile memory," in *Proc. Design Autom. Test Eur. Conf.*, 2011, pp. 136–141.

[30] J. Hu, W.-C. Tseng, C. Xue, Q. Zhuge, Y. Zhao, and E.-M. Sha, "Write activity minimization for nonvolatile main memory via scheduling and recomputation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 584–592, Apr. 2011.

[31] C. J. Xue, Y. Zhang, Y. Chen, G. Sun, J. J. Yang, and H. Li, "Emerging non-volatile memories: Opportunities and challenges," in *Proc. 9th Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Oct. 2011, pp. 325–334.

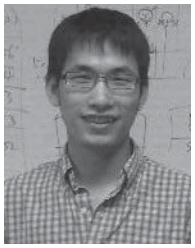
[32] T. Liu, Y. Zhao, C. J. Xue, and M. Li, "Power-aware variable partitioning for DSPs with hybrid PRAM and DRAM main memory," in *Proc. Design Autom. Conf.*, 2011, pp. 405–410.

[33] J. Li, L. Shi, C. J. Xue, C. Yang, and Y. Xu, "Exploiting set-level write non-uniformity for energy-efficient NVM-based hybrid cache," in *Proc. ESTImedia*, 2011, pp. 19–28.

[34] Y. Huang, T. Liu, and C. J. Xue, "Register allocation for write activity minimization on non-volatile main memory," in *Proc. 16th Asia South Pacific Design Autom. Conf.*, 2011, pp. 129–134.

[35] X. Dong, N. P. Jouppi, and Y. Xie, "PCRAMsim: System-level performance, energy, and area modeling for phase-change RAM," in *Proc. Int. Conf. Comput.-Aided Design*, 2009, pp. 269–275.

[36] 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 *Proc. Int. Workshop Workload Character.*, Dec. 2001, pp. 3–14.



Jingtong Hu (S'09) received the B.E. degree from the School of Computer Science and Technology, Shandong University, Shandong, China, in 2007, and the M.S. degree from the Department of Computer Science, University of Texas at Dallas, Richardson, in 2010, where he is currently pursuing the Ph.D. degree.

His current research interests include wireless sensor networks, memory and compiler optimization for embedded systems, and nonvolatile memory.

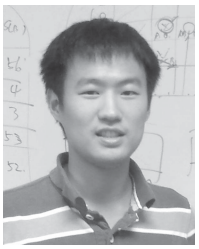
Chun Jason Xue (M'04) received the B.S. degree in computer science and engineering from the University of Texas, Arlington, in 1997, and the M.S. and Ph.D. degrees in computer science from the University of Texas at Dallas, Richardson, in 2002 and 2007, respectively.

He is currently an Assistant Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong. His current research interests include memory and parallelism optimization for embedded systems, software/hardware codesign, and real-time systems.

Qingfeng Zhuge received the B.S. and M.S. degrees in electronics engineering from Fudan University, Shanghai, China, in 1992 and 1996, respectively, and the Ph.D. degree from the Department of Computer Science, University of Texas at Dallas, Richardson, in 2003.

She is currently a Full Professor with Chongqing University, Chongqing, China. She has authored more than 60 research articles published in premier journals and conference proceedings. Her current research interests include parallel architectures, embedded systems, supply-chain management, real-time systems, optimization algorithms, compilers, and scheduling.

Dr. Zhuge was a recipient of the Best Ph.D. Dissertation Award in 2003.



Wei-Che Tseng received the B.S. degree in electrical engineering and the M.S. degree in computer science from the University of Texas at Dallas, Richardson, in 2007 and 2010, respectively, where he is currently pursuing the Ph.D. degree from the Department of Computer Science.

His current research interests include embedded systems, compiler optimization, virtual machines, and computer architecture.

Edwin H.-M. Sha (S'88–M'92–SM'04) received the Ph.D. degree from the Department of Computer Science, Princeton University, Princeton, NJ, in 1992.

He was with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, from August 1992 to August 2000. Since 2000, he has been a Tenured Full Professor with the Department of Computer Science, University of Texas at Dallas, Richardson. Since 2012, he has been serving as the Dean of the College of Computer Science and Engineering, Chongqing University, Chongqing, China. He has published more than 280 research papers in refereed conference proceedings and journals.

Prof. Sha was a recipient of the Teaching Award, the Microsoft Trustworthy Computing Curriculum Award, the National Science Foundation CAREER Award, the National Science Foundation Council Overseas Distinguished Young Scholar Award, the Chiang Jiang Honorary Chair Professorship, and the China Thousand Talents Program Award.