Energy Optimization for Data Allocation With Hybrid SRAM+NVM SPM

Yan Wang, Kenli Li, *Senior Member, IEEE*, Jun Zhang, and Keqin Li, *Fellow, IEEE*

***Abstract*— The gradually widening disparity in the speed of the CPU and memory has become a bottleneck for the development of chip multiprocessor (CMP) systems. Increasing penalties caused by frequent on-chip memory access have raised critical challenges in delivering high memory access performance with tight energy and latency budgets. To overcome the memory wall and energy wall issues, this paper adopts CMP systems with hybrid scratchpad memories (SPMs), which are configured from SRAM and nonvolatile memory. Based on this architecture, we propose two novel algorithms, i.e., energy-aware data allocation (EADA) and balancing data allocation to energy and write operations (BDAEW), to perform data allocation to different memories and task mapping to different cores, reducing energy consumption and latency. We evaluate the performance of our proposed algorithms by comparison with a parallel solution that is commonly used to solve data allocation and task scheduling problems. Experiments show the merits of the hybrid SPM architecture over the traditional pure memory system and the effectiveness of the proposed algorithms. Compared with the AGADA algorithm, the EADA and BDAEW algorithms can reduce energy consumption by 23.05% and 19.41%, respectively.**

***Index Terms*— Data allocation, energy consumption, nonvolatile memory, write operations.**

Manuscript received December 5, 2016; revised May 4, 2017 and June 4, 2017; accepted June 22, 2017. Date of publication July 14, 2017; date of current version January 5, 2018. This work was supported in part by the Key Program of National Natural Science Foundation of China under Grant 61432005, in part by the National Outstanding Youth Science Program of National Natural Science Foundation of China under Grant 61625202, in part by the International (Regional) Cooperation and Exchange Program of National Natural Science Foundation of China under Grant 61661146006, in part by the National Natural Science Foundation of China under Grant 61370095 and Grant 61472124, in part by the International Science & Technology Cooperation Program of China under Grant 2015DFA11240, in part by the National Key R&D Program of China under Grant 2016YFB0201402 and Grant 2016YFB0201303, and in part by the Innovation Team Project of Guangdong Province University under Grant 2016KCXTD017. This paper was recommended by Associate Editor A. Sangiovanni Vincentelli. *(Corresponding authors: Yan Wang; Kenli Li.)*

Y. Wang is with the College of Information Science and Engineering, Hunan University, Changsha 410082, China, and also with the College of Computer Science and Educational Software, Guangzhou University, Guangzhou 510086, China (e-mail: bessie11@yeah.net).

K. Li is with the College of Information Science and Engineering, Hunan University, Changsha 410082, China, and also with the National Supercomputing Center in Changsha, Changsha 410082, China (e-mail: lkl@hnu.edu.cn).

J. Zhang is with the College of Information Science and Engineering, Hunan University, Changsha 410082, China, and also with the Department of Electrical and Computer Engineering, New York University, New York City, NY 11201 USA (e-mail: jeffjunzhang@gmail.com).

K. Li is with the College of Information Science and Engineering, Hunan University, Changsha 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2017.2720678

I. Introduction

BECAUSE the performance gap between the CPU and memory is expanding, energy consumption has become more important and received extensive attention in chip multiprocessor (CMP) systems. To bridge the performance gap, solutions adopt the SRAM cache as on-chip memory. SRAM caches have facilitated the layered memory hierarchy and improved memory performance. However, SRAM caches account for up to 25%-50% of the overall CMP's energy consumption and do not guarantee predictability of cache misses [2]. Therefore, it is desirable to integrate NVM like flash memory or phase change memory (PCM) for CMP systems because it is nonvolatile and consumes less energy than SRAM. For instance, if a 4GB SRAM on-chip memory is replaced by a 4GB NVM, 65% of energy consumption can be saved in intensive write access on CMP systems [31]. The disadvantages of NVM are explicit. First, the speed and cost of read operations and write operations in NVMs are asymmetric. Second, there is a maximum number of write operations that NVM can perform. Third, memory access in NVM is slower than in SRAM. Considering the properties of DRAM and PRAM, in this paper, we utilize a hybrid on-chip memory composed of a SRAM and a NVM to achieve energy-efficient CMP systems. With hybrid on-chip memory, a substantial performance gain is achieved by the proposed techniques, while consuming less energy and extending the lifetime of NVMs.

To develop alternative energy-efficient techniques, in this paper, the hybrid on-chip memory uses a software controllable hybrid Scratch-Pad memory (SPM). SPM has been widely employed in CMP systems to replace the hardware-controlled cache [10], [20]. This is because SPM has three major advantages compared with the cache. First, SPM is directly addressing and does not need the comparator and tag SRAM. Second, SPM generally guarantees the single-cycle access latency. Third, SPM is purely managed by software, either directly via the application program or through the automated compiler support [19]. To efficiently manage SPMs, in this paper, we use compiler-analyzable data access patterns to strategically allocate data. The proposed technique benefits energy consumption while minimizing performance degradation and endurance caused by the physical limitation of NVM.

When an application with data dependencies is executed on a CMP system with hybrid SPMs, the following problems cannot be overlooked, i.e., reducing energy consumption, improving the endurance of NVM (reducing the number of write operations), and minimizing scheduling time. In this paper, we reconsider the variable partitioning problem of a DSP application on CMP systems with hybrid SPMs. Unfortunately, different objectives may conflict in this hybrid memory architecture. For example, placing data in NVM can reduce energy consumption but may lead to more writes to NVM, which shortens system lifetime and degrades performance. Therefore, techniques are proposed to address the trade-offs between low energy consumption and the performance and endurance degradation caused by write activities on an NVM. These improvements are achieved through the following novel contributions of this paper.

- We first propose a data allocation algorithm for single core systems to reduce energy consumption while controlling the number of write operations on NVM.
- Then, we propose two novel algorithms, i.e., EADA and BDAEW, for CMP systems with hybrid SPMs to solve the energy optimization problems of data allocation and task scheduling. The two algorithms generate a well-planned data allocation and task scheduling scheme so that all requirements can be met and the total energy consumption can be minimized while reducing the number of write operations on NVM.

Experimental results show that our proposed algorithms perform better than parallel solutions [5]. On average, reduction in the total energy consumption of the EADA and BDAEW algorithms is 16.44% and 27.8% compared to parallel solutions. The number of write operations on NVM can be reduced greatly by EADA and BDAEW algorithms. This means the lifetime of NVM can be prolonged. If the original life of NVM is 5 years, our proposed techniques can extend the life of NVM to at least 12 years.

The remainder of this paper is organized as follows. Section II reviews related work. In Section III, we present our CMP with hybrid SPMs architecture and computational model. In Section IV, we use an example for illustration. In Section V, we first propose a data allocation approach for a single core system, and then propose two heuristic algorithms to solve the energy optimization problem of data allocation and task scheduling on CMP systems. In Section VI, we evaluate and analyze our techniques compared with the parallel solution. Section VII concludes this paper and discusses future work.

II. Related Work

Numerous sophisticated SPM data allocation and management techniques have been proposed to reduce energy consumption or improve performance. Wang *et al.* [27] presented algorithms for WCET-aware energy-efficient static data allocation on SPM, i.e., selectively allocating data variables to SPM to minimize program's energy consumption, while respecting a given WCET upper bound. Udayakumaran *et al.* [22], proposed a heuristic algorithm to allocate global and stack data for SPMs to minimize allocation cost. Udayakumaran and Barua [21] proposed a dynamic data allocation method for allocating heap data on SPMs to improve performance. The above techniques target SPMs consisting of pure SRAM. None of the techniques above can apply to the architecture in this paper when integrating the lifetime issues of NVM.

Many data allocation techniques have also been proposed to extend the lifetime of the NVM-based memory subsystem, while reducing energy consumption and improving overall system performance [13], [14], [17], [28]. Monazzah *et al.* [16] presented algorithms for fault-tolerant data allocation on hybrid SPM that consist of NVM and SRAM, protected with error-correcting code (ECC), and parity. Qiu *et al.* [18] proposed a novel genetic algorithm to solve the data allocation problem of heterogeneous SPM with SRAM and NVMs. Dhiman *et al.* [7] proposed an architecture and system policy for managing a hybrid SRAM+PRAM memory. Hu *et al.* [9] considered a hybrid SPM configuration consisting of SRAM and PCM-based NVM, and presented a dynamic data allocation algorithm for reducing write operations on NVM by preferentially allocating read-intensive data variables into NVM, and write-intensive data variables into SRAM. Wang *et al.* [27] considered a multitasking system with hybrid main memory consisting of PCM and DRAM, and addressed the problem of partitioning and allocating data variables to minimize average power consumption while guaranteeing schedulability. In this paper, we address the data allocation problem for CMP systems with hybrid SPM architecture by proposing novel scheduling algorithms. The goal is to reduce the energy consumption and extend the lifetime of the NVMs.

Due to the influence of task scheduling on system performance, data allocation problems have been extensively studied to incorporate task scheduling. Various heuristic algorithms were proposed in [1], [4], [12], [14], [23], [24], [29], and [30]. These works mainly focus on optimizing the performance of a system, where the algorithms provide quality solutions to minimize the application's total execution time. Together with the increasing demand for high-performance CMP systems, the energy consumption problem has also become more important and attracts extensive attention. Banikazemi *et al.* [3] proposed a novel low-overhead, user-level meta-schedule to improve both system performance and energy consumption. Wang *et al.* [26] proposed an optimal ILP based algorithm, and two heuristic algorithms, TAC-DA and TRGS algorithms, to solve heterogeneous data allocation and task scheduling problems; minimizing energy consumption and satisfying the time constraint. Their methods achieve a well-planned data allocation and task scheduling approach. However, the data allocation and task scheduling problem in CMP with hybrid SPMs differs from existing data allocation problems for non-uniform memory access architectures, since the write and read operations to one component of the architectures are asymmetric, and it is desirable to avoid writes to that component. Compared with the above approaches, this paper has several unique aspects. First, we target the CMP embedded systems with hybrid SRAM+NVM SPMs to solve the energy optimization problem of data allocation and task scheduling. Second, we propose several novel algorithms to obtain a well-planned data allocation and task scheduling approach such that the overall performance can be improved while reducing total energy consumption.

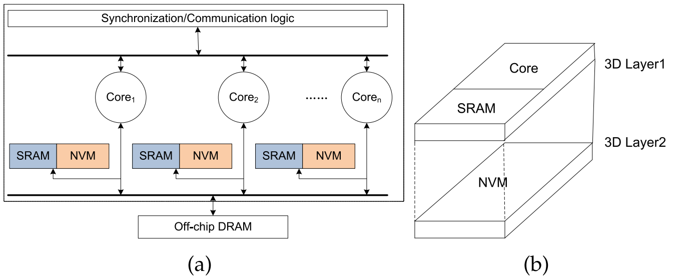


Fig. 1. (a) An architecture model (b) SPM architecture.

TABLE I

TIME AND ENERGY CONSUMPTION FOR ACCESS TO DIFFERENT MEMORIES

Op	LS		RS		LN		RN		DM	
	Ti	En	Ti	En	Ti	En	Ti	En	Ti	En
read	1	5	2	10	2	1	4	2	60	60
write	1	5	2	3	12	6	6	12	13	60	60

III. THE MODEL

A. Architecture Model

In this paper, we target the embedded chip multi-cores with hybrid local memories. As shown in Figure 1, the architecture model consists of a set of connected homogeneous cores denoted by $core = \{core_1, core_2, \ldots, core_n\}$, where n is the number of homogeneous cores. Each core is tightly coupled with an on-chip local memory composed of a SRAM and a NVM. The SRAM and NVM of each core $core_i$ are denoted by M_{2i-1} and M_{2i}. All cores share a DRAM main memory with large capacity. Each core can access its own local memory and other cores' local memories. We call core access from local memory *local access*, while access from an SPM held by another core is referred as *remote access*. Generally, remote access is supported by an on-chip interconnect and all cores access the off-chip DRAM through a shared bus. The IBM CELL processor, which includes a multi-channel ring structure to allow communication between any two cores without intervention from other cores, is an example that adopts this architecture. We can safely assume that the data transfer cost between cores is constant. Local access is faster and consumes less energy than remote access while accessing the off-chip DRAM incurs the longest latency and consumes the most energy. Table I, which is introduced from [18], shows the time and energy consumption for access to different memories. In the table, the columns of "LS", "RS", "LN", "RN", and "DM" indicates the memory access cost to local SRAM, remote SRAM, local NVM, remote NVM, and off-chip DRAM. "Ti" and "En" are time and energy consumption.

It is important to note that the hybrid local memories in the architecture model can not be pure caches because issues such as cache consistency and cache conflict are not considered. In this paper, the architecture employs SPMs as on-chip local memories. To make hybrid SPMs possible, researchers proposed several hybrid hardware/software support SPMs [6], [15]. For example, [18] employs hybrid SPMs composed of a SRAM and two NVM to study cost optimization incurred by data allocation. [8] explores hybrid nonvolatile SPM architectures. In a hybrid SPM, SRAM and NVM share the same address space with the main memory. The CPU can load data from both of them directly.

As shown in Figure 1(b), the hybrid SPM can be fabricated with 3-D chips because 3-D integration is a feasible and promising approach to fabricating the hybrid SPM [8]. In 3-D chips, multiple active device layers are stacked together with short and fast vertical interconnects. For fabrication, SRAM can be fitted into the same layer as the core and NVM can be fitted into a separate layer, so that designers can take full advantage of the attractive benefits that NVM provides.

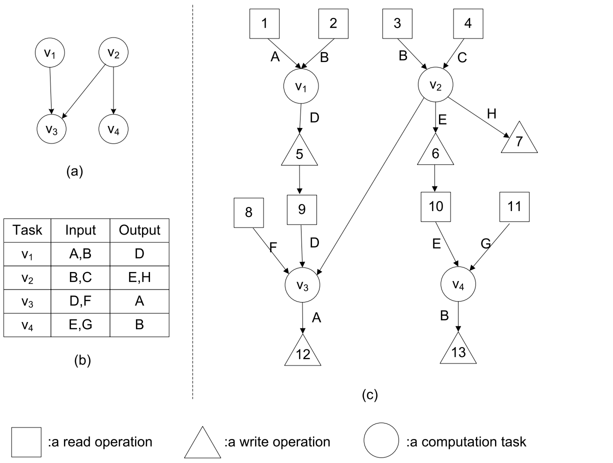


Fig. 2. An input DAG. (a) Precedence constraints among tasks. (b) The input data and output data of each task. (c) An MDFG combined tasks with data.

B. Computational Model

In this subsection, we describe the *memory access data flow graph* (MDFG), which is used to model an application to be executed on the target embedded chip multiprocessors. Before we formally describe the MDFG model, we first introduce a *directed acyclic graph* (DAG) model as shown in Figure 2(a). In this paper, we use a DAG as a description of a given input graph.

Definition 1: A DAG is a node-weighted directed graph represented by $G = (V, E, D, in, out, Nr, Nw)$, where $V = \{v_1, v_2, \ldots, v_N\}$ is a set of task nodes, and $E \subseteq V \times V$ is a set of edges that describe the precedence constraints among nodes in V. D is a set of data. $in(v_i) \subseteq D$ is a set of input data of task v_i, and $out(v_i) \subseteq D$ is a set of output data of task v_i. $Nr(v_i)$ is used to represent the read number of task v_i for different input data, i.e., $Nr(v_i) = (nr_1(i), nr_2(i), \ldots, nr_n(i))$, where $nr_h(i)$ denotes the read time of v_i for input data h. $Nw(v_i)$ is used to represent the write number of task v_i for different output data, i.e., $Nw(v_i) = (nw_1(i), nw_2(i), \ldots, nw_n(i))$, where $nw_h(i)$ denotes the write time of v_i for output data h.

If we treat a memory access operation as a node, we can redefine a DAG to obtain a memory access data flow graph (MDFG) defined as the following.

Definition 2: An MDFG is a node-weighted directed graph by $G' = (V_1, V_2, E, D, var, Nr, Nw, P, M)$, where $V_1 = \{v_1, v_2, \ldots, v_{N_1}\}$ is a set of N_1 task nodes, and $V_2 = \{u_1, u_2, \ldots, u_{N_2}\}$ is a set of N_2 memory access operation nodes. $E \subseteq V \times V$ $(V = V_1 \bigcup V_2)$ is a set of edges. An edge $(\mu, \nu) \in E$ represents the dependency between node μ and node ν, indicating that task or operation μ has to be executed before task or operation ν. D is a set of data. $var : V_1 \times V_2 \times D \rightarrow \{\texttt{true}, \texttt{false}\}$ is a binary function, where $var(v_i, u_l, h)$ denotes whether the memory access operation $u_l \in V_2$ is transmitting datum $h \in D$ for task $v_i \in V_1$. $Nr(v_i, h) = nr_h(i)$ is the read time of task v_i for his input data h. $Nw(v_i, h) = nw_h(i)$ is the write time of task v_i for his output data h. $P = \{core_1, core_2, \ldots, core_n\}$ is a set of cores, and $M = \{M_1, M_2, \ldots, M_{2n}\}$ is a set of on-chip memories. The SRAM and NVM of each core $core_i$ denoted by M_{2i-1} and M_{2i}, respectively.

An example of MDFG from DAG is shown in Figure 2. In the example, Figure 2(a) shows the DAG, there are $N = 4$ tasks, i.e., v_1, v_2, v_3, v_4. Figure 2(a) shows the precedence constraints among the tasks. The data set is $D = \{A, B, C, D, E, F, G, H\}$, and Figure 2(b) shows the input data and output data of each task. For example, task a reads input data A and B before it is started, and writes output data D after it is finished. If we treat a memory access operation as a node, we can obtain an MDFG from the DAG is shown in Figure 2(c), where we have $N_1 = 4$ task nodes and $N_2 = 13$ memory access operation nodes. For example, the node 1 is memory access operation node and represents reading data A.

C. Problem Definition

Assume that we are given a multi-core systems with n cores, where each core is integrated with a SPM which consists of a SRAM and a NVM. The access time and energy consumption of each processor in accessing a unit data from different memories are known in advance. The capacity of each core's SRAM and NVM is also known in advance. The *energy optimization problem of data allocation and task scheduling* can be defined as follows: Given an DAG $G = (V, E, D, in, out, Nr, Nw)$, we treat a memory access operation as a node and reconstruct the DAG to obtain an MDFG $G' = (V_1, V_2, E, D, var, Nr, Nw, P, M)$. The objectives of an *energy optimization problem of data allocation and task scheduling* are to find (1) a data allocation Mem: $D \longrightarrow M$, where $Mem(h) \in M$ is the memory to store $h \in D$; (2) a task assignment A: $V_1 \longrightarrow P$, where $C(v_i)$ is the core to execute task $v_i \in V_1$, such that the total energy consumption can be minimized, the write operations on NVM can be reduced, and the scheduling length can be shortened. In this problem, we assume each core can access SRAM and NVM in its local SPM, every remote SPM, and off-chip DRAM with different time and energy consumption. The time and energy consumption of access to different memories is given in Table I. The *objective function* of the target problem is described as:

Objective 1: Energy consumption is minimized. For each available assignment of data, we obtain the number of local read operations N_{lr}, local write operations N_{lw}, remote read operations N_{rr}, and remote write operations N_{rw}; the corresponding energy consumption is indicated as E_{lr}, E_{lw}, E_{rr}, and E_{rw}; the energy consumption of each data can be formulated as follows.

$$E_h = N_{lr}(h) \times E_{lr} + N_{lw}(h) \times E_{lw} + N_{rr}(h) \times E_{rr} + N_{rw}(h) \times E_{rw} \quad (1)$$

However, the above equation does not consider the case that data h is allocated in main memory. If one data h is allocated in main memory, we would use the following equation to compute the energy consumption:

$$E_h = (total_r + total_w) \times E_{dm} \quad (2)$$

where $total_r$ and $total_w$ are the total number of read operations and write operations of data h, respectively. E_{dm} is the energy consumption when an access operation takes place in main memory.

Given the energy consumption of each task E_{vi}, the total energy consumption of a MDFG can be formulated as:

$$E_{total} = \sum_{vi \in V_1} E_{vi} + \sum_{h \in D} E_h \quad (3)$$

Objective 2: The number of write operations on NVM is minimized. For each NVM, the number of write operations can be formulated as:

$$N_{NVM}(i) = \sum_{M(h)=M_i, i=2k} (N_{lw}(h) + N_{rw}(h)) \quad (4)$$

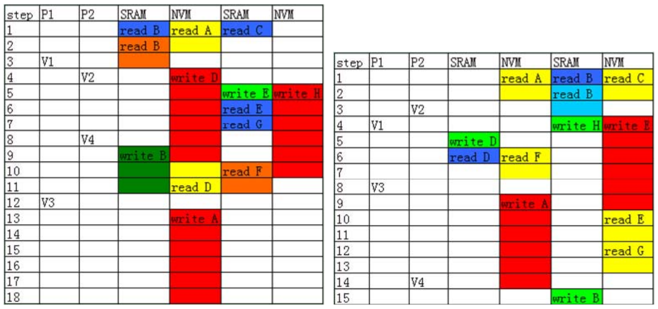
The NVM of each core is denoted by M_{2k}, where k is the id of the corresponding cores. The total number of write operations on NVM is:

$$TN_{NVM} = \sum_{M_i, i=2k} (N_{NVM}(i)). \quad (5)$$

IV. Motivation Example

In this section, we use an example to illustrate the effectiveness of our proposed algorithms. The example of MDFG application in Figure 2 is executed on a two-core system. As shown in Figure 1, each core is equipped with a hybrid local memory, composed of a SRAM and an NVM. The access latency and energy consumption of the target two-core system are shown in Table I. Based on the dependency constraints in the MDFG shown in Figure 2, two solutions of data allocation and tasks scheduling are generated by two compared algorithms shown in Figure 3.

Figure 3(a) shows a schedule generated by parallel solution [5]. Conventionally, the parallel solutions to attack the task scheduling and data allocation problem would be to minimize scheduling time by mapping tasks and allocating data using shortest processing time policy. In this schedule, task v_1 and v_3 are scheduled on $core_1$, task v_2 and v_4 are scheduled on $core_2$, the data A, D, and H are allocated on NVM, and all other data are allocated on SRAM. The completion time of this schedule is 18, the total energy consumption is 77, and the number of write operations on NVM is 3. However, this approach may not produce a good result in terms of energy consumption. Since reducing write operations on NVM is also one of the objectives, we should explore a better trade-off approach to



step	P1	P2	SRAM	NVM	SRAM	NVM
1			read B	read A	read C	
2			read B			
3	V1					
4		V2		write D		
5					write E	write H
6					read E	
7					read G	
8		V4				
9			write B			
10					read F	
11				read D		
12	V3					
13				write A		
14						
15						
16						
17						
18						

(a)

step	P1	P2	SRAM	NVM	SRAM	NVM
1				read A	read B	read C
2					read B	
3		V2				
4	V1				write H	write E
5			write D			
6			read D	read F		
7						
8	V3					
9				write A		
10						read E
11						
12						read G
13						
14		V4				
15					write B	

(b)

Fig. 3. The data allocation and task schedule of the example (a) parallel solutions with energy consumption 77 and the number of writes on NVM 3 (b) the proposed solution with energy consumption 57 and the number writes on NVM 2.

minimize energy consumption and reduce the number of write operations on NVM.

Figure 3(b) shows an improved schedule, which considers energy consumption and the number of write operations on NVM. In this schedule, tasks v_1 and v_3 are scheduled on $core_1$, tasks v_2 and v_4 are scheduled on $core_2$, the data A, C, E, F, and G are allocated in NVM. The other data are allocated in SRAM. The schedule length of the improved schedule is 15, the total energy consumption is 57, and the number of write operations on NVM is 1. Energy consumption is reduced by $(77-57)/57 = 35.08\%$ compared with the parallel solution. From the above example, we can see that energy consumption can be reduced by exploring data allocation and task scheduling on CMP systems with hybrid SPMs.

V. Algorithm

In this section, we first discuss the data allocation mechanism for single core embedded chip system. Then, we propose several methods for CMP to solve the energy optimization problem of data allocation and task scheduling based on hybrid SRAM+NVM local memory.

A. Data Allocation for Single Core Embedded Chip System

A single core embedded chip system is a special embedded chip multiprocessor system. We only consider data allocation for hybrid local memory when an MDFG is run in a single core embedded chip system. This is because all tasks will be assigned to the same core leading to a constant execution time and energy consumption for task nodes. In this section, we propose the data allocation for the single core (DASC) algorithm as shown in Algorithm 1. For the hybrid SPM, there are two disadvantages based on NVM: 1) the limited number of write operations and asymmetric access speed and energy consumption in read and write operations; and 2) errors in storing information when updating operations of an NVM cell is beyond the limited number of write operations. Therefore, we use two thresholds Tr_w and max_w to prevent the NVMs from wearing out. Tr_w restricts NVMs from storing data whose write operations are more than Tr_w. In this paper, Tr_w is equal to the average number of write operations of data in an application. The maximum write operations of a NVM is max_w. If the total number of write operations on a NVM exceeds max_w, data with write operations should not be allocated to the NVM. Data that will be allocated in NVM must satisfy the following properties.

Algorithm 1 Data Allocation for Single Core

```
Input: (1) An application MDFG $G' = (V1, V2, E, D, var)$;
    (2) a hybrid SRAM and NVM local memory; (3) a write
    threshold $Tr_w$ for NVM. (4) setting maximum write oper-
    ations $max_w$ on NVM for an application.
Output: Assign data into SRAM or NVM.
find all data with $cw > Tr_w$, put them in set $L$
while $L$ is not full do
  select a data with maximum writes on $L$
  if SRAM have space to allocate the data then
    allocate it in SRAM
  else
    allocate it in main memory
  end if
  $Nflag(h) = 0$
  remove it from $L$
end while
for each un-allocated data do
  compute the energy consumption $ES(h)$ and $EN(h)$
  if $ES(h) > EN(h)$, NVM have space to allocate the
  data, the total write operations on NVM is less than $max_w$
  then
    allocate the data in NVM, $Nflag(h) = 1$
  else
    if SRAM has space to allocate the data then
      allocate it in SRAM
    else
      allocate it in main memory
    end if
  end if
  $Nflag(h) = 0$
end for
```

Property 3: If data h can be allocated in NVM, then

$$cw(h) \leq Tr_w$$

where $cw(h)$ is the total write operations of data h.

Property 4: Let the binary variable $Nflag(h)$ denotes whether allocated data h in NVM. The total number of write operations on NVM must be less than max_w:

$$\sum_h (Nflag(h) \times cw(h)) \leq max_w$$

where $Nflag(h) = 1$ means data h is allocated in NVM.

In the following, we will discuss the DASC algorithm about how to allocate data in memories to avoid the disadvantages of NVM and reduce total energy consumption for a single core embedded chip system.

In Algorithm 1, data are divided into two categories according to the number of write operations and the threshold Tr_w. If the total number of write operations of one data is more than Tr_w, the data is the first type of categories and is put in set L (Line 1). For the data in set L, if SRAM has enough space to hold the data, the data is allocated in the SRAM; otherwise, the data is allocated in main memory (Lines 2-12). For the data not in set L, we first calculate the energy consumption of each available assignment for data h and use minimum energy consumption $\min(ES(h), EN(h))$ as a measurement for deciding the memory to store the data (Lines 13-14). If data h is allocated in SRAM, energy consumption of data h on SRAM can be formulated as:

$$ES(h) = N_{lw}(h) \times es_{lw} + N_{lr} \times es_{lr} \qquad (6)$$

where es_{lw} and es_{lr} are the energy consumption of each local write operation and each local read operation on SRAM, respectively. And, if the data is allocated in NVM, the energy consumption of data h can be obtained as

$$EN(h) = N_{lw}(h) \times en_{lw} + N_{lr} \times en_{lr} \qquad (7)$$

where en_{lw} and en_{lr} are the energy consumption of each local write operation and each local read operation on NVM, respectively. In computing the energy consumption, $ES(h)$ and $EN(h)$, we only consider local memory access operations. This is because each data allocated in SPM only have local read operations and local write operations since the target system is a single core embedded chip system.

The algorithm also confirms whether the total number of write operations on NVM exceed the max_w. For data to be allocated in the NVM, the following three conditions must be satisfied: 1) the total number of write operations on NVM is less than max_w; 2) NVM is not full; 3) the data with $ES(h) > EN(h)$ (Lines 15-17). If one of the above conditions cannot be met, we will determine the free space of SRAM. If SRAM has sufficient space to hold the data, the data is allocated in SRAM (Lines 18-21). Otherwise, the data is allocated in main memory (Lines 22-27).

The data allocation for a single core algorithm considers two objectives. For the endurance of NVM, it is detrimental to place data with too many writes on NVM; the algorithm controls the maximum write operations on NVM. For energy consumption, it places data into a memory with minimum energy consumption among all available assignments. The complexity of data allocation for a single core algorithm is $O(H)$, where H is the amount of data.

B. Chip Multiprocessors System

CMPs generally consist of multiple cores sharing an off-chip main memory. In this subsection, the target architecture is a CMP shown in Figure 1. For solving the energy optimization problem of data allocation and task scheduling incurred by applications execution on a CMP with N cores (each of these cores is integrated with a hybrid SPM which consists of a SARM and a NVM), we propose two algorithms, i.e., energy-aware data allocation (EDAC) algorithm and balance data allocation with energy and writes (BDAEW) algorithm.

In The EDAC algorithm as shown in Algorithm 2, we first call the parallel algorithm [5] to find an effective mapping for each task. The parallel algorithm in [5] is used to solve the task scheduling problem, where all requirements are met and the scheduling length is minimized. After task mapping, we find an allocation for data using task assignments. In the following, we will discuss in detail how to assign data nodes in different memories. Data allocation consists of two phases. The first phase finds a proper core for the data so that remote memory access operations can be reduced. Since data may be needed by different tasks, more than one memory access operation may be associated with the data. For data h, we first calculate the number of memory access operations on each core $core_i$ as follows:

Algorithm 2 Energy-Aware Data Allocation

```
Input: (1) An application MDFG $G' = (V1, V2, E, D, var)$;
    (2) an embedded chip multiprocessors with hybrid SRAM
    and NVM local memory; (3) a write threshold $Tr_w$ for
    NVM. (4) setting maximum write operations $max_w$ on
    NVM for an application.
Output: A data allocation and task scheduling with minimized energy.
 1: call parallel algorithm to find an effective map for each task nodes
 2: /*data allocation*/
 3: for each data $h$ do
 4:    for each processors $p_i$ do
 5:       compute the number of read operations $Nr(h, p_i)$, and
          the number of write operations $Nw(h, p_i)$
 6:    end for
 7:    choose the processor with maximum $(Nr(h, p_i) + Nw(h, p_i))$
       to assign the data into its corresponding hybrid local memory
 8: end for
 9: for each processor $p_i$ do
10:    call the Algorithm 1
11: end for
```

$$Nr(h, core_i) = \sum_{C(v_j)=core_i} (Nr(j, h)), \quad \forall e(h, v_j) \in G',$$
$$Nw(h, core_i) = \sum_{C(v_j)=core_i} (Nw(j, h)), \quad \forall e(v_j, h) \in G' \qquad (8)$$

where $C(v_j)$ is the core to execute the task v_j. Then, we use maximum memory access operations $\max(Nr(h, core_i) + Nw(h, core_i))$ as a measurement to decide in which core's SPM to place the data (Lines 3-8). In the second phase, we find data allocation according to the first phase. For each core, we call the Algorithm 1 to decide which memory is allocated data (Lines 9-11).

In the EADA algorithm, it takes $O(|VE|)$ time to find a better mapping for each task, where V represents the number of tasks and E represents the number of edges between tasks. To find a better data allocation, it takes $O(|VHP|)$ determine which processor to allocate data and takes $O(H)$ to allocate data to a determinate memory, where H is the number of data and P is the number of cores. Therefore, if P is treated as a constant, the time complex of the EADA algorithm is $O(|VE| + |VH| + |H|)$

Algorithm 2 has two objectives, minimizing energy consumption and reducing write operations on NVM. However, the two objectives may conflict: assigning data in NVM can save energy consumption but may cause many write operations on NVM. Therefore, we propose BDAEW algorithm as shown in Algorithm 3 to balance the conflict of minimizing energy consumption and reducing write operations on NVM. Before the details of the algorithm are presented, several theorems on our algorithms are built as follows.

Theorem 5: For all $h \in in(v_i)$, if and only if the data h and task v_i are allocated the same core, the binary variable $Rflag(v_i, h) = 1$. The total local read number for data h can be formulated as:

$$N_{lr}(h) = \sum_{v_i}(Rflag(v_i, h) \times Nr(v_i, h) \times in(v_i, h))$$

and the total remote read number for data h can be formulated as:

$$N_{rr}(h) = \sum_{v_i}((1 - Rflag(v_i, h)) \times Nr(v_i, h) \times in(v_i, h))$$

where $Nr(v_i, h)$ is the read number of data h for task v_i and binary variable $in(v_i, h) = 1$ denotes h is a input of task v_i.

Proof: For each task and data pair (v_i, h), if task v_i and data h are allocated the same core, the read operations for pair (v_i, h) are local read operations. Otherwise, the read operations for pair (v_i, h) are remote read operations. Thus, for pair (v_i, h), the local reads number is $Rflag(v_i, h) \times Nr(v_i, h) \times in(v_i, h)$ and the remote reads number is $(1 - Rflag(v_i, h)) \times Nr(v_i, h) \times in(v_i, h)$. Furthermore, for each data h, we can obtain the total number of local read operations and remote read operations as Theorem 5.

Theorem 6: For all $h \in out(v_i)$, if and only if the data h and task v_i are allocated the same core, the binary variable $Wflag(v_i, h) = 1$. The total local write number for data h is:

$$N_{lw}(h) = \sum_{v_i}(Wflag(v_i, h) \times Nw(v_i, h) \times out(v_i, h))$$

and the total remote write number for data h is:

$$N_{rw}(h) = \sum_{v_i}((1 - Wflag(v_i, h)) \times Nw(v_i, h) \times out(v_i, h))$$

where $Nw(v_i, h)$ is the write number of data h for task v_i and binary variable $out(v_i, h) = 1$ denotes h is a output of task v_i.

Proof: The proof is similar to the proof of Theorem 5.

Summing up all of these memory access operations for each data $h \in D$, we can obtain the total number of each type of memory access operations as follows:

- The total local read number $N_{lr} = \sum_h N_{lr}(h)$
- The total remote read number $N_{rr} = \sum_h N_{rr}(h)$
- The total local write number $N_{lw} = \sum_h N_{lw}(h)$
- The total remote write number $N_{rw} = \sum_h N_{rw}(h)$

Since data may be needed by different tasks, we should calculate the energy consumption of data for each available allocation. For each available allocation $Mem(h) = M_i$, given the energy consumption of each local read operation $E_{lr}(M_i)$, each remote read operation $E_{rr}(M_i)$, each local write operation $E_{lw}(M_i)$, and each remote write operation $E_{rw}(M_i)$, the energy consumption can be formulated as:

$$\begin{aligned} En(h, M_i) = {} & N_{lr}(h) \times E_{lr}(M_i) + N_{rr}(h) \times E_{rr}(M_i) \\ & + N_{lw}(h) \times E_{lw}(M_i) + N_{rw}(h) \times E_{rw}(M_i) \end{aligned} \tag{9}$$

Additionally, the really energy consumption of each data can be formulated as follows:

$$E_h = \sum_{M_i}(En(h, M_i) \times flag(h, M_i)) \tag{10}$$

where $flag(h, M_i)$ is a binary variable, denoting whether allocated data h is in M_i. If $flag(h, M_i) = 1$ it means data h is allocated in M_i.

In algorithm BDAEW as shown in Algorithm 3, we first use the parallel algorithm to find a better mapping for each task. Then, we find better allocation for data to meet all requirements and to minimize total energy consumption while reducing the number of write operations on NVMs. Data allocation consists of two phases. The first phase finds a minimum energy consumption assignment for the data, and the second phase allocates write operations on NVMs in such a way as to balance write operations on NVMs and total energy consumption.

In the first phase, we first calculate the energy consumption of each available assignment for each data h. Then, we use $\min\{En(h, M_i)\}$ as a measurement to decide which memory is assigned data h. In other words, for each data h, we choose a memory M_i with minimum energy consumption $En(h, M_i)$ among all available assignment of data h to hold the data (Lines 3-8). In the second phase, for each processor, we first determine if all data allocated in NVM meet the write constraints. If there is data with $cw(h) > Tr_w$ on NVM, we reassign the data to SRAM (SRAM has enough space to hold the data) or main memory (SRAM is full), where $cw(h)$ is the total write operations of data h, and is equal to $N_{lw}(h) + N_{rw}(h)$ (Lines 9-19). Then, we determine if the total number of write operations on NVM meets the constraint $T_{cw} < max_w$. If the total number of write operations on NVM $T_{cw} < max_w$, we obtain a solution; otherwise, we reallocate some data; In reallocating data to satisfy the constraint $T_{cw} < max_w$, we use *read-to-write* ratio = $\frac{cw(h)}{cw(h)+cr(h)}$ as a measurement to select a data in NVM with the maximum *read-to-write* ratio to be moved into SRAM or main memory, where $cr(h)$ is the total number of read operations of data h (Lines 20-28). After adjustment of data allocation, the algorithm finds a new data allocation and reduces write operations on NVM until the constraint $T_{cw} < max_w$ is satisfied.

In the BDAEW algorithm, it takes $O(|VE|)$ time to find a better mapping for each tasks and takes $O(|VMH|)$ to find a original data mapping , where V represents the number of tasks and E represents the number of edges between tasks, H is the number of data, and M is the number of memories. To reallocate data, it takes at most $O(|\log_2(HM)|)$ to obtain a better allocation where the maximum number of write operations on NVM is controlled. Thus, the time complexity of BDAEW algorithm is $O(|VE| + |VH| + |\log_2 H|)$.

Algorithm 3 Balance Data Allocation With Energy and Write Operations

```
Input: (1) An application MDFG $G' = (V1, V2, E, D, var)$; (2) an embedded chip multiprocessors with hybrid SRAM and NVM local memory; (3) a write threshold $Tr_w$ for NVM. (4) setting maximum write operations $max_w$ on NVM for an application.
Output: A data allocation and task scheduling with minimized energy.
call parallel algorithm to find an effective map for each task nodes
/*data allocation*/
for each data $h$ do
  for each memory $M_i$ do
    compute the energy consumption if the data is assigned in the memory $En(h, M_i)$
  end for
  choose the memory with minimum $En(h, M_i)$ to allocate the data, and marked $flag(h, M_i) = 1$
end for
for each processor $p_i$ do
  while NVM $M_{2i}$ exist data with $cw > Tr_w$ do
    select a data $h$ with maximum writes $cw(h)$ on $M_{2i}$
    let $flag(h, M_{2i}) = 0$
    if SRAM $M_{2i-1}$ has enough space to hold the data then
      reallocate the data h on $M_{2i-1}$, $flag(h, M_{2i-1}) = 1$,
    else
      reallocate the data h on main memory
    end if
  end while
  compute the total number of write operations $T_{cw}$ on its NVM
  while $T_{cw} > max_w$ do
    find a data in NVM with maximum $ratio = \frac{cw}{cr+cw}$, where $cr$ is the number of read operations on NVM for this data
    let $flag(h, M_{2i}) = 0$
    if SRAM $M_{2i-1}$ is not full then
      reallocate the data in $M_{2i-1}$,$flag(h, M_{2i-1}) = 1$
    else
      reallocate the data in main memory
    end if
  end while
end for
```

VI. Experimental Results

A. Experiment Setup

In this section, we present experimental results to illustrate the effectiveness of the proposed algorithms. We use the following benchmarks from DSPstone [32], i.e., IIR, Allope, Floyd, Elliptic, Volterra, and 8-lattic. These benchmarks are frequently used in multicore systems research. We compile each benchmark using GCC and obtain the task graphs accompanied by the read/write data sets. There are three notes. First, the source codes must be compiled with profiling option on (`-fprofile-generate`). Then, the compiled binary must be executed by feeding a data set corresponding to the use case. Finally, the source code must be compiled again with both profile-guided optimization and ABSINTH enabled (`-fprofile-use-fabsinth`). The `pass_absinth_bbs` traverses all RTL expressions within each basic block. For each expression, `pass_absinth_bbs` analyzes whether it is an instruction or not, and generates one execute primitive per each instruction [11]. Then, the task graphs and access sets are fed into our simulator. Our simulator requires data to be processed by the extracted graphs. To make the experiment more rigorous, we reuse the same task graph but feed various data volume. The amount of data needed in the graph is modeled as $N_d = \alpha \times \sqrt{V} \times \sqrt{E}$, where V is the amount of tasks in the graph and E is the number of edges in the MDFG. The α is a tuning parameter which is randomly selected from the Poisson distribution where λ is picked from a uniform distribution in the range [0,10]. As α grows, the number of data increases and the dependency between tasks associated with the data is stronger. For each task node, the number of read/write access of data is set randomly from a uniform distribution in the range [0,20]. To thoroughly evaluate the proposed algorithms, we conducted a rigorous simulation with different α settings.

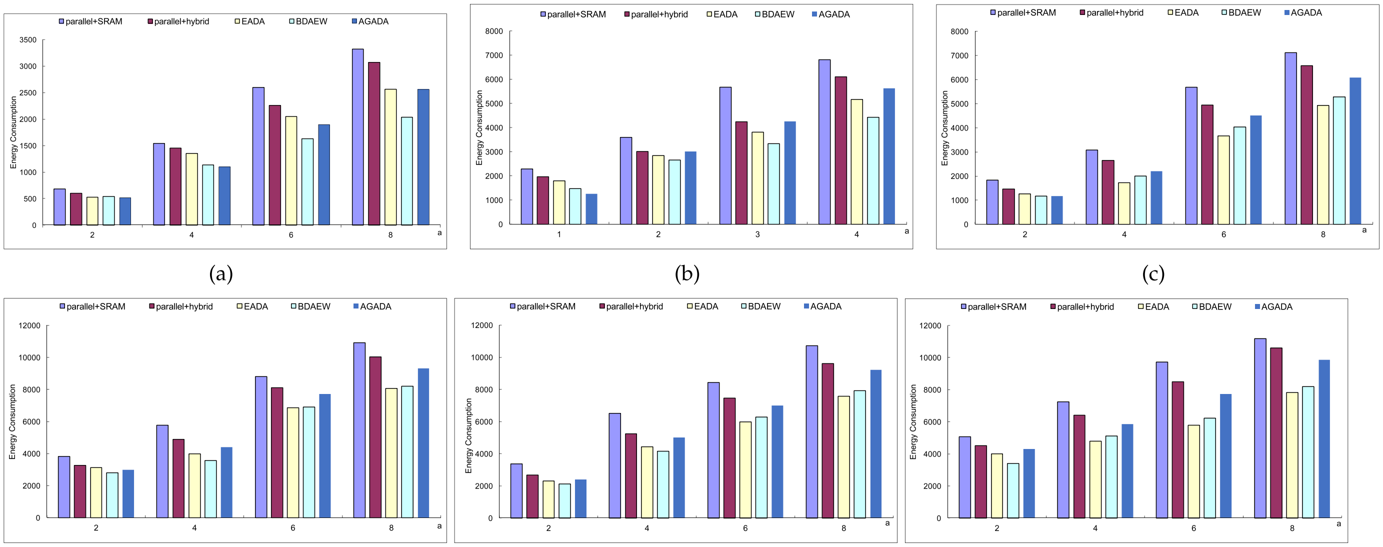
TABLE II

Performance Parameters for the Target Memory Modules

Device	Parameter
CPU	Number of cores: 3, frequency:2.9GHz
SRAM	size: 1M, local read energy:0.226nJ local write energy: 0.226nJ, leakage power: 1.004nW local read latency:0.565ns, local write latency:0.565ns remote read energy: 0.441nJ, remote write energy: 0.475nJ remote read latency: 1.13ns, remote write latency: 1.275ns
PRAM	size: 526KB, read energy:0.319nJ write energy: 1.725nJ, leakage power: 0.125nW read latency:0.694ns, write latency:4.290ns remote read energy: 0.785nJ, remote write energy: 2.689nJ remote read latency: 1.695ns, remote write latency: 8.386ns
main memory	size:512MB, access energy:20.083nJ, access latency:21.04ns, leakage power:102.56W

The experiments for benchmarks are conducted on an architecture model which is defined in Section III. The target architecture consists of three cores. Each core is equipped with hybrid local memory units composed of a SRAM and a PRAM. The configurations of the target architecture systems are shown in Table II. We integrated all these parameters into our in-house simulator to verify the effectiveness of our proposed algorithms. All the simulations run on an Intel® Core^{TM}2 Duo Processor E7500 2.93GHz with a 2GB main memory operated by Red Hat Linux 7.3.

We compared the performance of our proposed algorithms to that of the parallel solution [5] and AGADA algorithm [18]. AGADA algorithm is a recently published algorithm to minimize the total cost of data allocation on hybrid memories with NVM. The parallel solution is a classical algorithm to solve the task scheduling and data allocation problem. Therefore, the AGADA algorithm and parallel solution are the most related works and two excellent candidates for benchmarking. In this paper, the AGADA algorithm has been evolved so that it is



(a) (b) (c)

(d) (e) (f)

Fig. 4. The energy consumption of benchmarks under different approaches when change α. (a)iir, (b) allope, (c) floyd, (d) elliptic, (e) voltera, (f) 8_lattice.

TABLE III

THE NUMBER OF WRITES ON PRAM

Bench	node	edge	α	para+hybrid	AGADA	EADA	$\frac{E_p-E}{E_p}$%	$\frac{E_A-E}{E_A}$%	BDAEW	$\frac{E_p-E}{E_p}$%	$\frac{E_A-E}{E_A}$%
iir	8	7	2	194	159	149	23.71%	6.20%	152	21.65%	4.40%
			4	350	286	231	34.0%	19.23%	251	28.28%	12.24%
			6	472	366	295	37.5%	19.39%	313	34.11%	14.48%
			8	661	509	389	41.14%	23.57%	417	36.92%	18.07%
allope	15	17	2	403	365	326	19.30%	8.40%	308	23.76%	15.62%
			4	757	664	580	23.38%	12.65%	545	28.01%	17.92%
			6	1203	1009	875	27.26%	13.28%	825	31.42%	18.59%
			8	2021	1553	1207	40.27%	22.27%	1105	45.32%	28.84%
floyd	16	20	2	564	464	346	38.65%	25.43%	334	40.78%	28.01%
			4	965	771	563	41.65%	26.97%	541	43.93%	29.83%
			6	1351	1063	743	45.01%	30.13%	708	47.59%	33.40%
			8	1824	1360	950	47.89%	30.15%	915	49.80%	32.72%
elliptic	34	47	2	1005	793	494	50.84%	37.95%	491	51.14%	38.08%
			4	1748	1449	882	49.54%	39.13%	824	52.86%	43.13%
			6	2622	2356	1289	50.87%	45.28%	1178	55.07%	45.36%
			8	3539	3099	1764	50.95%	43.07%	1580	49.70%	49.15%
voltera	27	34	2	682	579	497	27.12%	13.73%	498	26.97%	13.98%
			4	1220	1007	859	29.59%	14.70%	861	29.42%	14.49%
			6	1927	1632	1305	32.27%	20.03%	1219	36.74%	25.30%
			8	2794	2321	1827	34.72%	21.28%	1694	37.97%	27.01%
lattice	42	59	2	1173	806	499	57.45%	38.08%	496	57.71%	38.46%
			4	1970	1412	871	55.78%	38.32%	803	59.23%	43.13%
			6	3073	2205	1364	55.61%	38.14%	1175	62.57%	46.71%
			8	4425	3529	2125	51.66%	39.78%	1786	59.57%	46.35%
average	-	-	-	1543	1243	863	42.06%	29.57%	798	48.28%	35.80%

comparable to our model to consider the energy consumption problem of task scheduling and data allocation. The parallel solution is originally used in the system with (a) a pure SRAM local memory, and (b) a hybrid local memory composed of a SRAM and a PRAM. To make fair comparisons, we implemented all four algorithms, i.e., parallel solution, AGADA, EADA, and BDAEW, in the same scheduling framework. By doing so, we ensured that the performance loss of the parallel solution and AGADA algorithm is not due to different settings of the implementations. The results for energy consumption are shown in Figure 4. The results for the number of writes on PRAM are shown in Table III. Last, the results for execution time are shown in Figure 5.

B. Results and Analysis

This section presents the experimental results to illustrate the effectiveness of our proposed algorithms. The results of total energy consumption are represented by the statistical comparison of different approaches when changing α. As we can observe, with the increase of the data parameter α, the energy consumption of all five approaches increase and

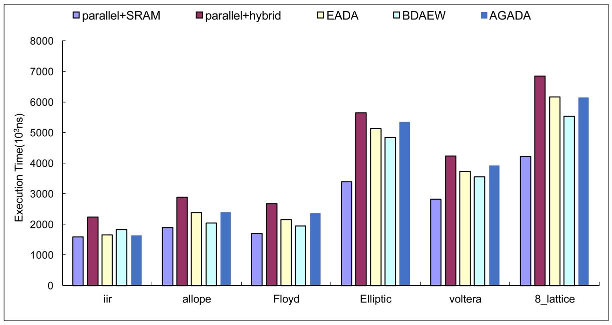


Fig. 5. The comparison of execution time under different approaches. $\alpha = 2$, $max_w = 500$.

the gap in energy consumption among the five algorithms become larger. For all benchmarks, the energy consumption of EADA and BDAEW algorithms is less than that of the parallel solution using a hybrid PRAM+SRAM local memory. Additionally, the energy consumption of the parallel solution using a pure SRAM is the maximum of the five different approaches. Compared with parallel+SRAM, the EADA and BDAEW can reduce energy consumption by 47.52% and 36.65%, respectively on average. The EADA and BDAEW algorithms also can reduce energy consumption by 29.08% and 25.47% on average compared with the parallel solution using a hybrid local memory, respectively. Therefore, algorithms EADA and BDAEW save energy better than the parallel solution. From Figure 4, we also observe that the energy consumption of EADA and BDAEW algorithms is less than that of AGADA algorithm in most cases. On average, the EADA and BDAEW algorithms can reduce energy consumption by 23.05% and 19.41%, respectively. Although the energy consumption of AGADA algorithm is less than that of EADA and BDAEW algorithms in several cases, the AGADA algorithm does not consider data-dependency, which will result in overhead write operations.

The number of write operations on PRAM has a large effect on the PRAM's lifetime. In this paper, we use the following formulation to compute the number of write operations on PRAM:

$$N_{PRAM} = \sum_{h} (N\_lw(h) + N\_rw(h)) \times Nflag(h) \qquad (11)$$

Although the parallel solution is used as a baseline technique to evaluate the PRAM's write operations on NVM of our proposed algorithms, our proposed algorithm is not comparable with the parallel solution using a pure SRAM. This is because there are no write operations on PRAM in parallel+SRAM. The results for write operations on PRAM are shown in Table III, which is the statistical comparison of all four algorithms for all benchmarks based on the target architectural model. In Table III, the eighth and ninth columns show the ratio of the reduction of write operations on PRAM by EADA compared with the parallel+hybrid and AGADA algorithm. The eleventh and twelfth columns show the reduction ratio of write operations on PRAM by BDAEW compared with the parallel+hybrid and AGADA algorithm. From the table, we can observe that our algorithm EADA and BDAEW can achieve better write operation reduction than the parallel solution and AGADA algorithm. Compared with parallel+hybrid, EADA and BDAEW can reduce the number of write operations on PRAM by 42.06% and 48.28%, respectively on average. Compared with AGADA, EADA and BDAEW can reduce the number of write operations on PRAM by 29.57% and 35.80%, respectively on average. The lifetime improvement ratio of PRAM can be estimated by $(\frac{M/W'-M/W}{M/W})$ [25], where M stands for the maximum write operations of PRAM, W is the number of write operations on PRAM when using parallel solutions, and W' stands for the number of write operations on PRAM when using our proposed technique. Approximately, 28.82% and 29.93% reduction on the number of write operations is equivalent to a 144.03% and 155.76% increase on the lifetime on PRAM. It means that our proposed techniques can prolong the lifetime of PRAM to 12 years if the PRAM's original lifetime is 5 years.

However, NVM introduces longer latency. For example, when $\alpha = 2$, the statistical comparisons of execution time under different approaches are shown in Figure 5. From the figure, we can see that the scheduling length of EADA and BDAEW algorithms are longer than the parallel solution using pure SRAMs, but shorter than AGADA and parallel solution using hybrid SPMs. However, as we can see from the results, the negative impact on applications' execution time is not significant. This is because we can use PRAM with write buffers and write operations are relatively insensitive to memory in hierarchies that are far from the CPU [31]. As shown in Figure 5, EADA and BDAEW algorithms can reduce the execution time of benchmarks by 15.54% and 21.49% compared with parallel solutions using hybrid SPMs, respectively on average. Compared with AGADA algorithm, EADA and BDAEW algorithms can reduce the execution time of benchmarks by 5.83% and 12.44%, respectively on average.

In order to further illustrate the effectiveness of the proposed algorithm, we compared the scheduling length and overhead energy consumption of the five approaches using different benchmarks. The overhead energy consumption is a result of the scheduling cost, the cost of computing all data- dependencies, and other logistic costs. The results of scheduling length and overhead energy consumption are shown in Figures 6 and 7, respectively. From the two figures, we can observe that the scheduling time and overhead energy consumption of the parallel solution are less than that of the other four algorithms, and that of the EADA and BDAEW algorithms are less than AGADA algorithm in most cases. However, as the data-dependency grows, the gap between the AGADA algorithm and the proposed algorithm decreases. When the data-dependency application is represented by a large MDFG, the scheduling time and overhead energy consumption of AGADA algorithm is less than the proposed algorithms. This is because the time complexity of AGADA is $O(G * P * H)$, where G and P represent the maximum number of iterations and the population size of the genetic algorithm, respectively. In more detail, the scheduling time and overhead energy consumption of the proposed algorithms

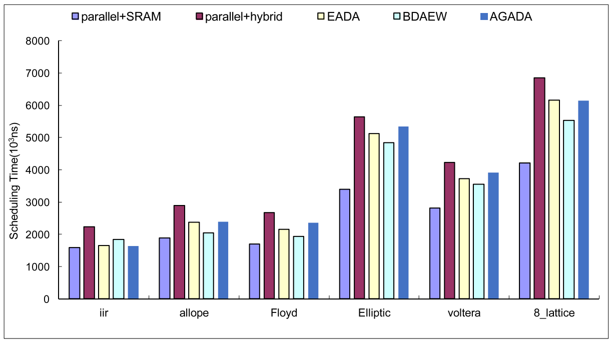


Fig. 6. The comparison of scheduling time under different approaches. $\alpha = 2$, $max_w = 500$.

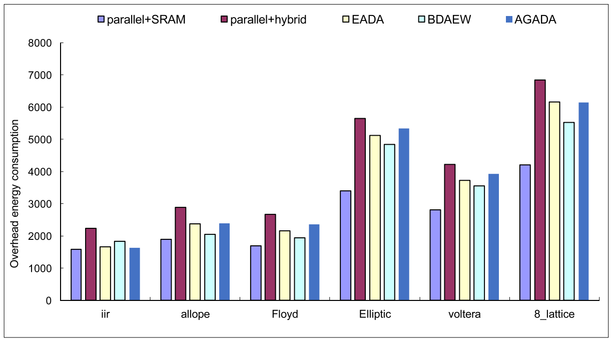


Fig. 7. The comparison of overhead energy consumption under different approaches. $\alpha = 2$, $max_w = 500$.

are not simply dependent on the number of data but depends on the product of data-dependency, the amount of data, and the amount of memories, while that of AGADA increases linearly with the growth of data. When the data-dependency and size of applications grow to a certain size that is greater than $G * P$, the scheduling time and overhead energy consumption of AGADA are less than that of the proposed algorithm. For example, if the population size is set as 100 and maximum generation is set as 1000, the scheduling time and overhead energy consumption of AGADA will be less than that of the proposed algorithm, when the number of tasks and data increase to 50 and 350, respectively. However, even in this case, the net execution time and the net energy consumption of the proposed algorithm are still less than the AGADA. Hence, the benefits we gain by the proposed technique outweigh the extra overheads.

In summary, for CMP with hybrid SPMs composed of a SRAM and NVM, the EADA and BDAEW algorithms can obtain a well-planned assignment such that the total energy consumption is minimized with little degradation in performance and endurance of PRAM. The EADA and BDAEW algorithms are also evaluated with experimental results showing that the EADA and BDAEW algorithms can obtain a better solution in energy consumption and the number of write operations on PRAM than parallel solutions and the AGADA algorithm.

VII. Conclusion and Future Work

Hybrid local memory is an effective approach to reduce energy consumption and memory access latency for multi-core systems. In this paper, we propose two novel heuristic algorithms, EADA and BDAEW. Based on the hybrid SRAM+NVM SPM architecture, data are allocated efficiently and tasks are scheduled reasonably, such that the total energy consumption is minimized with little degradation in performance and endurance caused by NVM. In experimental studies, we employed hybrid SRAM+PRAM SPM for multi-core systems to execute various applications. The results show that both the EADA and BDAEW algorithms achieve noticeable average reduction rates of total energy consumption compared with parallel solutions and AGADA algorithm. Both the EADA and BDAEW algorithms can reduce the number of write operations on NVM. This means that the lifetime of NVM can be extended when EADA and BDAEW are used in the hybrid SPM architecture.

The proposed algorithms can be extensible to heterogeneous cores. To achieve this, the cost of memory operations in each memory must be redefined and the method of computing energy consumption changed. A modern high-performance computing system normally consists of heterogeneous computing and communication resources, i.e., heterogeneous processors, heterogeneous memories, and heterogeneous communication interconnections. In heterogeneous processors, the same type of operations can be processed by different processors with various execution times and energy consumption. This makes the task scheduling and data allocation problem more complicated. Although the proposed algorithm can be extensible for heterogeneous cores, the performance and effectiveness need more precise investigations. Therefore, we will study the task and data allocation problem for heterogeneous processors with hybrid on-chip memory in the future research work.

References

[1] K. Bai and A. Shrivastava, "Heap data management for limited local memory (LLM) multi-core processors," in *Proc. IEEE/ACM/IFIP Int. Conf. Hardw./Softw. Codesign Syst. Synthesis (CODES+ISSS)*, Oct. 2010, pp. 317–325.

[2] 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. 10th Int. Symp. Hardw./Softw. Codesign*, 2002, pp. 73–78.

[3] M. Banikazemi, D. Poff, and B. Abali, "PAM: A novel performance/power aware meta-scheduler for multi-core systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2008, pp. 1–12.

[4] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generat. Comput. Syst.*, vol. 28, no. 5, pp. 755–768, 2012.

[5] C. Boyd, "Data-parallel computing," *Queue*, vol. 6, no. 2, pp. 30–39, 2008.

[6] Y.-T. Chen *et al.*, "Dynamically reconfigurable hybrid cache: An energy-efficient last-level cache design," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2012, pp. 45–50.

[7] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid pram and dram main memory system," in *Proc. 46th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2009, pp. 664–669.

[8] J. Hu, C. J. Xue, Q. Zhuge, W.-C. Tseng, and E. H. Sha, "Data allocation optimization for hybrid scratch pad memory with SRAM and nonvolatile memory," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 6, pp. 1094–1102, Jun. 2012.

[9] J. Hu, Q. Zhuge, C. J. Xue, W.-C. Tseng, and E. H.-M. Sha, "Management and optimization for nonvolatile memory-based hybrid scratchpad memory on multicore embedded processors," *ACM Trans. Embedded Comput. Syst.*, vol. 13, no. 4, p. 79, 2014.

[10] M. Kandemir and A. Choudhary, "Compiler-directed scratch pad memory hierarchy design and management," in *Proc. 39th Annu. Design Autom. Conf.*, 2002, pp. 628–633.

[11] J. Kreku and K. Tiensyrjä, and G. Vanmeerbeeck, "Automatic workload generation for system-level exploration based on modified GCC compiler," in *Proc. Conf. Design, Autom. Test Eur.*, 2010, pp. 369–374.

[12] K. Li, X. Tang, B. Veeravalli, and K. Li, "Scheduling precedence constrained stochastic tasks on heterogeneous cluster systems," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 191–204, Jan. 2015.

[13] S. Matsuno, M. Tawada, M. Yanagisawa, S. Kimura, N. Togawa, and T. Sugibayashi, "Energy evaluation for two-level on-chip cache with non-volatile memory on mobile processors," in *Proc. IEEE 10th Int. Conf. ASIC (ASICON)*, 2013, pp. 1–4.

[14] S. Mittal and J. S. Vetter, "AYUSH: A technique for extending lifetime of SRAM-NVM hybrid caches," *IEEE Comput. Archit. Lett.*, vol. 14, no. 2, pp. 115–118, Dec. 2014.

[15] J. C. Mogul, E. Argollo, M. A. Shah, and P. Faraboschi, "Operating system support for NVM+ dram hybrid main memory," in *Proc. HotOS*, 2009, p. 14.

[16] A. M. H. Monazzah, H. Farbeh, S. G. Miremadi, M. Fazeli, and H. Asadi, "FTSPM: A fault-tolerant scratchpad memory," in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2013, pp. 1–10.

[17] C. D. Nicholson, W. J. Westerinen, C. Ergan, M. R. Fortin, and M. Iyigun, "Reliability of diskless network-bootable computers using non-volatile memory cache," U.S. Patent 7 036 040, Apr. 25, 2006.

[18] M. Qiu *et al.*, "Data allocation for hybrid memory with genetic algorithm," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 4, pp. 544–555, Dec. 2015.

[19] V. Suhendra, C. Raghavan, and T. Mitra, "Integrated scratchpad memory optimization and task scheduling for MPSOC architectures," in *Proc. Int. Conf. Compil., Archit. Synthesis Embedded Syst.*, 2006, pp. 401–410.

[20] H. Takase, H. Tomiyama, and H. Takada, "Partitioning and allocation of scratch-pad memory for priority-based preemptive multi-task systems," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2010, pp. 1124–1129.

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

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

[23] A. N. Udipi, N. Muralimanohar, N. Chatterjee, R. Balasubramonian, A. Davis, and N. P. Jouppi, "Rethinking dram design and organization for energy-constrained multi-cores," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, pp. 175–186, 2010.

[24] W. Wang, P. Mishra, and S. Ranka, "Dynamic cache reconfiguration and partitioning for energy optimization in real-time multi-core systems," in *Proc. 48th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2011, pp. 948–953.

[25] 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. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2012, pp. 1553–1556.

[26] Y. Wang, K. Li, H. Chen, L. He, and K. Li, "Energy-aware data allocation and task scheduling on heterogeneous multiprocessor systems with time constraints," *IEEE Trans. Emerg. Topics Comput.*, vol. 2, no. 2, pp. 134–148, Feb. 2014.

[27] Z. Wang, Z. Gu, and Z. Shao, "WCET-aware energy-efficient data allocation on scratchpad memory for real-time embedded systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 11, pp. 2700–2704, Nov. 2015.

[28] J. Xing, A. Serb, A. Khiat, R. Berdan, H. Xu, and T. Prodromakis, "An FPGA-based instrument for en-masse RRAM characterization with ns pulsing resolution," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 6, pp. 818–826, Jun. 2016.

[29] Y. Yi, W. Han, X. Zhao, A. T. Erdogan, and T. Arslan, "An ILP formulation for task mapping and scheduling on multi-core architectures," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, 2009, pp. 33–38.

[30] L. Zhang, K. Li, Y. Xu, J. Mei, F. Zhang, and K. Li, "Maximizing reliability with energy conservation for parallel task scheduling in a heterogeneous cluster," *Inf. Sci. Int. J.*, vol. 319, pp. 113–131, Oct. 2015.

[31] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *ACM SIGARCH Comput. Archit. News*, vol. 37, pp. 14–23, Jun. 2009.

[32] V. Zivojnovic, J. M. Velarde, C. Schlager, and H. Meyr, "DSPSTONE: A DSP-oriented benchmarking methodology," in *Proc. Int. Conf. Signal Process. Appl. Technol.*, 1994, pp. 715–720.

Yan Wang received the B.S. degree in information management and information technology from Shenyang Aerospace University in 2010 and the Ph.D. degree from the College of Information Science and Engineering, Hunan University, Changsha, China, in 2016. Her research interests include modeling and scheduling in parallel and distributed computing systems, and high performance computing.

Kenli Li received the Ph.D. degree in computer science from Huazhong University of Science and Technology,China, in 2003. He was a Visiting Scholar with the University of Illinois at Urbana–Champaign from 2004 to 2005. He is currently the Dean of the School of Information Science and Technology, Hunan University, and also the Deputy Director of the National Supercomputing Center, Changsha. He has authored over 160 papers in international conferences and journals, such as the IEEE TC, the IEEE-TPDS, JPDC, ICPP, and ICDCS. His major research includes parallel computing, grid and cloud computing, and big data. He is an associate editor of IEEE-TC and an outstanding member of CCF.

Jun Zhang received the bachelor's and master's degrees in computer science from Hunan University, Changsha, China. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, New York University. His research interests include computer architecture, FPGA, real time embedded systems, and machine learning.

Keqin Li (F'14) is currently a SUNY Distinguished Professor of Computer Science. He has authored over 480 journal articles, book chapters, and refereed conference papers. His current research interests include parallel computing and high-performance computing, distributed computing, energy-efficient computing and communication, heterogeneous computing systems, cloud computing, big data computing, CPU–GPU hybrid and cooperative computing, multicore computing, storage and file systems, wireless communication networks, sensor networks, peer-to-peer file sharing systems, mobile computing, service computing, Internet of Things, and cyber-physical systems. He has received several best paper awards. He is currently or has served on the editorial boards of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, the IEEE TRANSACTIONS ON SERVICES COMPUTING, and the IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING.