2012 IEEE 18th International Conference on Parallel and Distributed Systems

Optimizing Scheduling in Embedded CMP Systems with Phase Change Memory

Jiayin Li[1] Zhiyang Zhang[1] Meikang Qiu[1,*] Ping Zhang[2], Gang Quan[3], Yongxin Zhu[4]
[1] Dept. of Electrical and Computer Engineering, University of Kentucky, Lexington, KY 40506, USA
[2] School of Computer Science and Engineering, South China University of Technology, GZ 510640, China
[3] Dept. of Electrical and Computer Engineering, Florida International University, Miami, FL 33174, USA
[4] School of Microelectronics, Shanghai Jiaotong University, 200240, China

***Abstract*—*Phase Change Memory* (PCM) is emerging as one of the most promising alternative technology to the *Dynamic RAM* (DRAM) when building large-scale main memory systems. Even though the PCM is easy to scale, it encounters serious endurance problems. Writes are the primary wear mechanism in the PCM. The PCM can perform 10^8 to 10^9 times of writes before it cannot be programmed reliably. In addition, the PCM has high write latency. To prolong the lifetime of the PCM as the main memory and enhance the performance, we propose a *Scratch Pad Memory* (SPM) based memory mechanism and an *Integer Linear Programming* (ILP) memory activities scheduling algorithm to reduce the redundant write operations in the PCM. The idea of our approach is to share the data copies among the SPMs, instead of writing back to the PCM main memory each time a modify occurs. Our experimental results show that the ILP scheduling can generate the optimal schedule of memory activities with minimum write operations, reducing the number of write by up to 61%.**

I. Introduction

Dynamic RAM (DRAM) has been the most widely used technology of the main memory for over three decades. However, main memory that consists of entirely DRAM is already reaching the scalability limit [1]. Due to some properties of DRAM, such as destructive reads and low retention time, some specific architecture solutions, such as, write after read operations and the refresh control, are implemented [2]. These extra costs limit the scalability of DRAM. Scaling DRAM beyond 40nm sizes would be questionable in future [3]. *Phase-Change Memory* (PCM) is a potential alterative of the DRAM main memory, due to its many desirable properties [2]. PCM is a non-volatile memory that switches its chalcogenide material between the amorphous and the crystalline states. By setting the resistances of different states, we can store the data in the PCM device. The application of heat that is required by the switch between states can be provided by using electrical pulses. In the PCM write, it relies on analog currents and thermal effects, which means it does not require control over discrete electrons [4]. PCM can provide four times more density than DRAM [5]. A 32-nm device prototype has been demonstrated [6].

Even though PCM is alternative to DRAM as main memory, large efforts are needed to surmount the disadvantage of PCM, compared to DRAM. PCM access latencies, especially in writes, are much slower than those of DRAM. In the read access, PCM is 2x-4x slower than DRAM. Moreover, PCM displays asymmetric timings for reads/writes, which means writes in PCM need 5x-10x more time than reads do. Due to the fact that phase changes in PCM are induced by injecting current into the chalcogenide material and heating it, thermal expansion and contraction in the chalcogenide material make the programming current injection no longer reliable [4]. Writes are the primary wear mechanism in PCM. The number of writes performed before the cell is not able to perform reliably ranges from 10^8 to 10^9. Therefore, writes in PCM limits both the performance and the lifetime of PCM.

In this paper, we propose a PCM main memory optimization mechanism through the utilization of the *Scratch Pad memory* (SPM). Memory activities optimization through the utilization of the SPM is a challenging problem. First of all, to minimize the number of write operations, data need to be shared among SPMs by data migrations. In some cases, multi-hop data migrations, which are necessary for optimal memory activities optimization, cannot be well scheduled by greedy scheduling algorithms. Compared to greedy scheduling algorithms, our ILP method is more promising, because it explores a larger solution space. However, modeling the memory activities scheduling problem through the utilization of the SPM is more sophisticated than the existing ILP-based memory optimization problems [7], [8]. Since the SPM space is limited, the optimization method should decide not only which copies of data should be kept, but also how long the SPM should keep these copies. Moreover, due to data sharing operations among SPMs, there are more kinds of memory activities to schedule than that in the existing ILP memory optimization methods. For example, to have a copy of data in a given SPM, there are three ways: loading the data from the PCM main memory to the SPM; outputting the data from the core to the SPM; and copying the data from a remote SPM via the data migration, which is either for the input requirement of the next task, or just temporary stored for future data migrations. Since copies of data are sharing among SPM via the on-chip network, data migration activities are also subject to the bandwidth of the network. Data dependencies across tasks further complicate the memory activities scheduling. Memory activities should not violate any data dependency. In this paper, we present a comprehensive ILP format that covers different kinds of PCM memory activities when utilizing the SPMs. System and application constraints, such as the size of SPM, the on-chip network bandwidth, and data dependencies, are formulated in our ILP algorithm.

In Section II, we discuss works related to this topic. In Section III, the background knowledge of phase change memory is presented. An illustrating example is given in Section IV. We propose our algorithms in Section V, followed by experimental results in Section VI. Finally, we conclude the paper in Section VII.

II. Related work

The PCM incorporated in the memory hierarchy was studied in [5]. A DRAM based *page cache* was implemented for a large PCM memory. Enhancement approaches, such as read-before-write, row-level rotation and segment swapping, were proposed to improve the lifetime of the PCM [9]. Lee et al. presented a PCM storage device with a bit level read-before-write loop [3]. Ferreira et al. described three lifetime enhancement methods for PCM: N-Chance victim selection replacement policy, bit level writes, and a swap management on page cache writebacks [2]. Although techniques introduced in these papers improve the endurance of the PCM, all of them require significant modifications in the hardware design. In this paper, by utilizing the SPM in CMP, our optimization approach does not require hardware modifications. A preSET method was proposed to enhance the performance of PCM by scheduling the SET operation before the actual write operation [10]. The security issue in PCM has been studied in [11], [12]. The security refresh approach was proposed to protect PCM against malicious wear out [11]. A low overhead method via online attack detection was designed [12]. However, all those approaches bring extra wrties to PCM.

A major trend of techniques of improving the lifetime of non-volatile memories is the application level design. Koc and Kandemir et al. used the recomputation in the SPM to reduce communications among different cores on chip [13], as well as between the cores and off-chip memory [14], which can reduce the number of reads in the main memory. Hu et al. modeled the data migration problem as a shortest path program and decided the best route for a given data to migrate from the source core to the destination core [15]. Nevertheless, the on-chip data traffic was not considered in Hu's approach. Two different optimization approaches for memory activities in CMP were proposed [16], [17]. These two optimization approaches cannot handle the data sharing among SPMs. In our ILP-based optimization approach, we take the capacity constraint in memory, on-chip data bus bandwidth, as well as data dependencies into account. Memory operations such as load, store, and share are well scheduled in the optimal solution generated by our ILP-based optimization.

III. Model and Background

A. Phase-change memory

As one type of non-volatile memory, PCM exploits the unique characteristic of the chalcogenide to store bits. A typical PCM cell consists of a chalcogenide layer and two electrodes on both sides. Two stable states of the chalcogenide, i.e., the crystalline and the amorphous, can be switched between when different amount of heat is applied in the chalcogenide. This procedure is done by injecting current into the PCM cell. When writing the PCM cell, the SET operation heats the chalcogenide layer to temperature between the crystallization temperature (300^{o}C) and the melting temperature (600^{o}C). By this operation, the chalcogenide is in the low-resistance crystalline state, which corresponds to the logic "1". On the other hand, the RESET operation heats the chalcogenide layer above the melting temperature. The corresponding state of the high resistance is amorphous state, i.e., the logic "0". The read operation of the PCM is basically sensing the resistance level of the PCM cell. It is non-destructive and involves much less heat stress, compared to that of the write operation.

Since both the SET and the RESET write operations apply dramatic heat stress into the phase change material, write is the major wear mechanism for the PCM. A PCM cell can perform stably within 10^8 to 10^9 times of writes. Compared to the 10^{15}-time-write endurance of the DRAM, the lifetime of the PCM becomes the major issue in implementing the PCM as the main memory.

The memory controller is one of the crucial parts in the PCM. In the read operation, the memory controller first checks the row buffer. If the target is in the buffer, the memory controller obtains the entry without accessing the memory bank. Otherwise, the memory controller will issue an activate command to move the data to an empty row in the buffer, and a read command to get the data. In the write operation, the memory controller issues the write command and sends the data directly to the memory bank.

B. Scratch pad memory

The SPM is an on-chip memory that can be accessed directly by processors with very low latency. The major difference between the SPM and the cache is that the data storage in the SPM is controlled by the system software, while the cache is automatically controlled by the hardware [17]. Due to the existence of the controllability on data storage in the SPM, we are able to optimize memory activities based on the characteristics of the application running in the system.

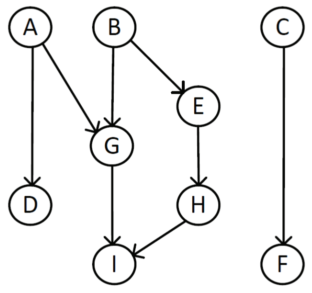
In this paper, we focus on a CMP architecture equipped with the PCM main memory. In this architecture, each core is connected to an SPM array. All SPMs are networked with the memory controller, which is also attached to the PCM main memory. Data are loaded or stored between the SPMs and the PCM main memory, via the memory controller. In addition, copies of data are transferred among the SPMs. When a core is executing a task, it can load data from its own SPM. The resulting data of a task can be written back to the SPM.

C. Application model

We model the application in this paper as a graph $G = \langle T, E, P, R_M, W_M, E_C \rangle$. $T = \langle t_1, t_2, t_3, ..., t_n \rangle$ is the set of n tasks. $E \subseteq T \times T$ is the set of edges where $(u, v) \in E$ means that task u must be scheduled before task v. $P = \langle p_1, p_2, p_3, \ldots, p_m \rangle$ is the set of m pages that are accessed by the tasks. $R_M : T \to P$ is the function where $R_M(t)$ is the set of pages that task t reads from. $W_M : T \to P$ is the function where $W_M(t)$ is the set of pages that task t writes to. $E_C(t)$ represents the execution time of task t while all the required data are in the SPM.

IV. Illustrating Example

A. An example of an application and a system



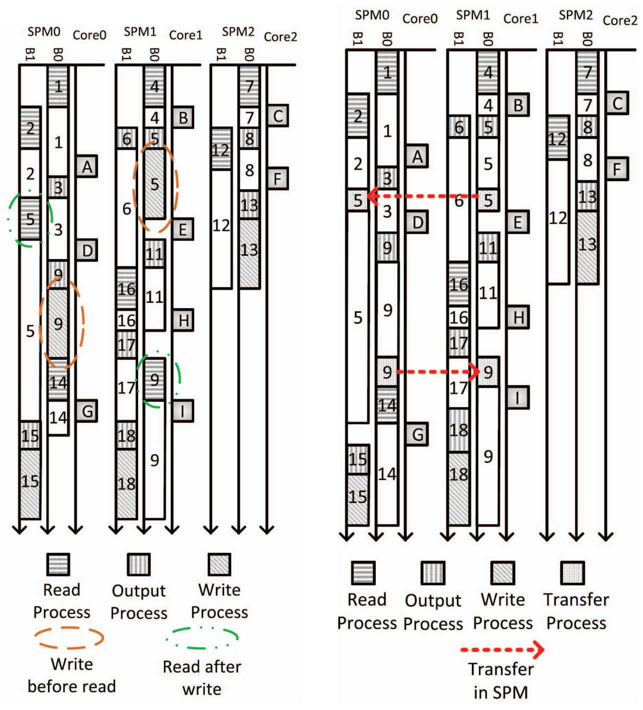
(a)

Task	Read pages	Write pages
A	1,2	3
B	4	5,6
C	7	8
D	5	9
E	6	11
F	8,12	13
G	5,14	15
H	11,16	17
I	9,17	18

(b)

Fig. 1. (a) The DAG of the application in the example, (b) Read pages and write pages of tasks in the application.

First we give an example, in which we reduce the number of writes in the PCM by sharing copies across the SPM. Considering a schedule of an application represented by the DAG in Fig. 1(a) in a three-core system, each task in the application requires up to 2 pages that should be in the SPM before the core executes it. The required pages R_M of each task are shown in the "Read page" column of Fig. 1(b). Moreover, tasks also need to output and modify up to 2 pages, i.e., W_M. The write pages W_M of each task are shown in the "Write page" column. For example, task A requests two pages, <page 1 and page 2>, before its execution, and writes its result in one page, <page 3>.



(a) (b)

Fig. 2. The schedules for the application in Fig. 1 running a three-core CMP system with two SPM blocks per core. The schedule in (a) is without data sharing in SPMs. The schedule in (b) is with data sharing in SPMs. The vertical axis represents the clock cycles. Each core has two SPM blocks, represented as the "B0" and "B1" columns. The blank box with number i in the "Bx (0 or 1)" column indicates that page i resides in SPM block "Bx" at the corresponding cycles. Since the write operation time (800 cycles) is 400 times longer than the core execution time (2 cycles), the scale of these figures does not strictly represent accurate clock cycles, only demonstrating the orders of these schedules.

Using the list scheduling, we have a baseline schedule as follows: task A, D, and G are assigned to core 0; task B, E, H, and I are assigned to core 1; and task C and F are assigned to core 2. A detailed schedule with memory activities is shown in Fig. 2(a). The Y axis represents the clock cycles. We assume the execution time of each task is 8 clock cycles. A core needs 2 cycles to access its own SPM, 5 cycles to a remote SPM. We also assume a read from the PCM main memory takes 80 cycles, while a write takes 800 cycles [15]. The memory activities, i.e., the shaded boxes in Fig. 2(a), are the major time consuming part in this schedule.

We observe that before core 0 reads page 5 in its SPM1, page 5 has been modified by the core 1, which is the output of task B. In this case, transferring pages across the SPMs reduces the write, since it is not necessary to write back page 5 before loading it again in the SPM. In addition, the time of sharing across SPMs should be much shorter than the time of writing and reading in PCM.

We modify the schedule as shown in Fig. 2(b). In this example, instead of writing back page 5 right after the executions of task B, we move the copy of page 5 from the SPM of core1 to the SPM of core0 before the execution of task D, which is represented as a red dotted arrow. The move occurs before the execution of task E on core1, due to the need of space in the SPM of core1 for storing the R_M of task E. After the move, a copy of page 5 is kept in the SPM of core0, until task D is executed by core0. By doing this, an unnecessary write is eliminated. Similarly, we move the copy of page 9 from the SPM of core0 to the SPM of core1 after the execution of task D, which is required by the later executed task I. In the next section, we will discuss our ILP-based optimization algorithms with more details.

V. ILP MEMORY ACTIVITIES OPTIMIZATION ALGORITHM

In this section, we present our ILP memory activities optimization algorithm. There are three major parts in our algorithm: the baseline scheduling, the ILP-based memory activities scheduling, and the post ILP procedure. The baseline scheduling generates a baseline schedule for both the task executions and the SPM assignments. Then, the ILP-based memory activities scheduling will find the optimal memory activities strategy to minimize the memory writes based on the baseline scheduling. Finally, the post ILP procedure will further reduce total execution time by eliminating the idle slots in the schedule.

A. Baseline scheduling

The Min-Min is a popular greedy scheduling algorithm [18], [19]. The Min-Min algorithm generates near-optimal schedules with comparatively low computational complexity [20]. Algorithm 1 shows the procedure of the Min-Min algorithm. Before we schedule a given task executed on a given core, we should schedule the required memory pages allocated in the SPM of the core in advance. We assume that the time of reading a memory page from the SPM is included in the execution time of this given task. We load the modified page in the SPM before it is written back to the PCM main memory. Complicated policies for memory coherence are out of the scope of this paper. We apply some simple policies to keep the memory content among SPMs and the PCM main memory coherent:

- Where a core initiates an SPM modifying process of a given page p, other cores that have a copy of this page in their SPM should initiates an SPM evicting process of this page. By doing this, there is no "dirty" copy of this page exists in the SPMs.

Algorithm 1 Min-Min algorithm

```
Input: A set of $T$ tasks represented by a DAG, $C$ different cores, $E_C$ of tasks
Output: A schedule generated by Min-Min
 1: Form a mappable task set $MT$
 2: while Set $MT$ is not empty do
 3:     for $i$: task $i \in [0, T-1]$ do
 4:         for $j$: core $j \in [0, C-1]$ do
 5:             Calculate $Tpg_{i,j}$ /*The earliest time when all require pages of $i$ are available*/
 6:             $Tfin_{i,j} = Tpg_{i,j} + E_C(i)$
 7:         end for
 8:         Find the core $C_{min}(i)$ that Min$_j(Tfin_{i,j})$
 9:     end for
10:     Find the pair $(k, C_{min}(k))$ with the earliest finish time $Tfin_{i,C_{min}(i)}$ among the task-core pairs generated in the for-loop
11:     Schedule the required pages of task $k$ to the SPM of $C_{min}(k)$ as soon as possible
12:     Assign task $k$ to core $C_{min}(k)$
13:     Schedule the modification of the resulting pages, $W_M(k)$, in the SPM of core $C_{min}(k)$, then the write back process of the resulting pages
14:     Remove $k$ from $MT$, and update the mappable task set $MT$
15: end while
```

- In the baseline scheduling process, we don't consider the data sharing in SPMs. When some tasks require the page that is modified by another task previously, the read process can only be initiated after the modification is finished.
- We implement the *Least Recently Used* (LRU) replacement policy in the SPM management.

B. ILP formatting

1) Resource allocation formatting: To input the baseline schedule to the later memory activities scheduling algorithm, we define several 0-1 matrixes to indicate the task executions and the SPM memory activities. The values in these matrixes are either 0 or 1. For the convenience of the reader, we list the symbols used in the ILP formatting in Table I. We give definitions of four 0-1 matrixes, which are related to the resource allocation, as follows:

(I) *Task assignment matrix ASM.* $ASM_{t,c} = 1$ means that task t is assigned to core c. The matrix ASM has the characteristic as follows:

$$\sum_{c=0}^{C-1} ASM_{t,c} = 1 \qquad \forall \quad t \in [0, T-1] \tag{1}$$

(II) *Task start time matrix St.* When $St_{t,c,s} = 1$, it means that the execution of the task t starts at clock cycle s on core c.

(III) *Core workload matrix WL.* $WL_{t,c,s} = 1$ means that core c is executing task t at clock cycle s. The relationship between St and WL is:

$$WL_{t,c,s} = \sum_{i=s-E_{t,c}-1}^{s} St_{t,c,i} \quad \forall\, t \in [0, T-1],\ c \in [0, C-1] \tag{2}$$

where $E_{t,c}$ is the execution time of task t on core c.

TABLE I
SYMBOLS AND ACRONYMS USED IN THE ILP FORMATTING

Symbol	Description
t	Task t
c	Core c
s	Clock cycle s
p	Memory page p
T	Number of tasks
C	Number of cores
S	Total number of clock cycles
P	Number of pages
$ASM_{t,c}$	Task assignment matrix
$St_{t,c,s}$	Task start time matrix
$WL_{t,c,s}$	Core workload matrix
$Mem_{p,c,s}$	Required memory matrix
$R_M(t)$	A set of page required by task t
$R_{p,c,s}$	Read matrix
$M_{p,c,s}$	Modify matrix
$W_{p,c,s}$	Write matrix
$Ev_{p,c,s}$	Evict matrix
$Si_{p,c,s}$	SPM input matrix
$So_{p,c,s}$	SPM output matrix
$OC_{p,c,s}$	SPM occupation matrix
$PM_{p,c,s}$	SPM page available matrix
$Mo_{p,c,s}$	Move out matrix
$Mi_{p,c,s}$	Move in matrix
$Mih_{p,c,s}$	Move in indicator matrix
$Mr_{p,c,s}$	SPM page modified matrix

(IV) *Required memory matrix* Mem. $Mem_{p,c,s} = 1$ means page p is required by core c at clock cycle s.

$$Mem_{p,c,s} = WL_{t,c,s} \qquad \forall \quad p \in ReqMen(t) \tag{3}$$

where $ReqMen(t)$ is a set of pages required by task t.

2) Basic memory activities formatting: To transform memory activities in the baseline schedule into 0-1 matrixes, we define matrixes for four basic memory activities. Three matrixes are included for modeling a given activity. For example, we define matrix R, $\widehat{R}$, and $\bar{R}$ to model the read from the PCM memory. The element of the matrix with no "hat", i.e. $R_{p,c,s} = 1$, represents that the corresponding activity of page p starts at clock cycle s, initiated by core c. While the one with "$\wedge$", i.e. $\widehat{R}_{p,c,s} = 1$, represents that this corresponding activity of page p is processing at clock cycle s. And the matrix with a bar, i.e. $\bar{R}_{p,c,s} = 1$, indicates this corresponding activity of page p finishes at clock cycle s. The relationships among R, $\widehat{R}$, and $\bar{R}$ are as follows:

$$\widehat{R}_{p,c,s} = \sum_{i=s-len_r+1}^{s} R_{p,c,i} \tag{4}$$

$$\bar{R}_{p,c,s} = R_{p,c,(s-len_r)} \tag{5}$$

where len_r is the length of the corresponding memory activity (the read in this example). Due to the fact that the latter two matrixes are determined by the first one and the length of its corresponding activity, introducing the latter two matrixes does not increase the complexity of solving the ILP problem.

(I) *Read matrixes* R. $R_{p,c,s} = 1$ means page p is read from the PCM memory and loaded into the SPM of core C at clock cycle s.

(II) *Modify matrixes* M. $M_{p,c,s} = 1$ means p is modified by the core C and loaded into the SPM of core C at clock cycle s.

(III) *Write matrixes* W. $W_{p,c,s} = 1$ means P is written back into the PCM memory from core C at clock cycle s.

(IV) *Evict matrixes* Ev. $Ev_{p,c,s} = 1$ means P is evicted from core C at clock cycle s.

3) SPM activities formatting: With the above matrixes, we can transform a baseline schedule as inputs of the ILP problem. In order to formulate the ILP problem, we further introduce four more matrixes, which are related to SPM activities. These four matrixes are determined by the eight matrixes mention above. Thus, they do not increase the complexity of solving the problem.

(I) *SPM input matrixes* Si and $\bar{Si}$. $Si_{p,c,s} = 1$ means page p is loaded into the SPM of core c at clock cycle s. This page can be either read from the PCM memory or store back from the core after it is modified by that core. Thus:

$$Si_{p,c,s} = R_{p,c,s} + M_{p,c,s} \tag{6}$$

$$\bar{Si}_{p,c,s} = \bar{R}_{p,c,s} + \bar{M}_{p,c,s} \tag{7}$$

(II) *SPM output matrixes* So and $\bar{So}$. $So_{p,c,s} = 1$ means page p is evicted from the SPM of core c at clock cycle s. This page could be modified by the core or evicted after read. Thus :

$$So_{p,c,s} = W_{p,c,s} + Ev_{p,c,s} \tag{8}$$

$$\bar{So}_{p,c,s} = \bar{W}_{p,c,s} + \bar{Ev}_{p,c,s} \tag{9}$$

(III) *SPM occupation matrix* OC. $OC_{p,c,s} = 1$ means page p is occupying a part of the SPM of core c at clock cycle s. The SPM occupation matrix OC holds the following equation:

$$OC_{p,c,s} = OC_{p,c,s-1} + Si_{p,c,s} - \bar{So}_{p,c,s} \tag{10}$$

(IV) *SPM page available matrix* PM, $PM_{p,c,s} = 1$ means page p is residing in the SPM of core C at clock cycle s. Note that when $OC_{p,c,s} = 1$, core c may not be able to use the page p at clock cycle s, due to the fact that it may still be in the memory transfer process. And $PM_{p,c,s} = 1$ means that core c can surely use page p at clock cycle s. The SPM page matrix PM holds the following equation:

$$PM_{p,c,s} = PM_{p,c,s-1} + \bar{Si}_{p,c,s} - So_{p,c,s} \tag{11}$$

We will use these 0-1 matrixes represent the baseline schedule in the following ILP-based memory activities scheduling algorithm.

C. ILP-based memory activities scheduling algorithm

With the baseline schedule, we will use our ILP approach to find the optimal memory activities schedule and minimize the number of the PCM activities. In some cases, a page that is needed by a task is residing in the SPM of a remote core. Instead of loading the page from the PCM memory, we can transfer the page from the SPM of the remote memory.

1) Additional ILP formatting for data transferring in SPMs: To represent the sharing activities among the SPMs, we define additional 0-1 matrixes, which also include three matrixes for each activity, similar to matrixes for those four basic memory activities. The definitions are as follows:

(I) *Move out matrix Mo.* $Mo_{p,c,s} = 1$ means page P is moved from the SPM of core C to the SPM of another core at clock cycle s.

(II) *Move in matrix Mi.* $Mi_{p,c,s} = 1$ means P is moved into the SPM of core C from the SPM of another core at clock cycle s.

(III) *Move in indicator matrix Mih.* $Mih_{p,s} = 1$ means P is moved into the SPMs of at least one core at clock cycle s.

$$Mih_{p,s} = \sum_{c=0}^{C-1} Mi_{p,c,s} \quad \forall\, p \in [0, P-1],\ s \in [0, S-1] \tag{12}$$

2) Redefined ILP formatting for data transferring in SPMs: In the previous "ILP formatting" subsection, we define the SPM input/output matrixes $Si_{p,c,s}$, $\bar{S}i_{p,c,s}$, $So_{p,c,s}$, and $\bar{S}o_{p,c,s}$ to determine whether a page is available in the SPM of a give core at clock cycle s. Now, we further modify these definitions by including the consideration of the Mi, Mo, $\bar{M}i$, and $\bar{M}o$, i.e. transferring data among SPMs. The new definition of Si, $\bar{S}i$, So, and $\bar{S}o$ as follows:

$$Si_{p,c,s} = R_{p,c,s} + M_{p,c,s} + Mi_{p,c,s} \tag{13}$$
$$\bar{S}i_{p,c,s} = \bar{R}_{p,c,s} + \bar{M}_{p,c,s} + \bar{M}i_{p,c,s} \tag{14}$$
$$So_{p,c,s} = W_{p,c,s} + Ev_{p,c,s} + Mo_{p,c,s} \tag{15}$$
$$\bar{S}o_{p,c,s} = \bar{W}_{p,c,s} + \bar{E}v_{p,c,s} + \bar{M}o_{p,c,s} \tag{16}$$

We use these new definitions of SPM input/output matrixes to calculate the SPM occupation matrix OC and the SPM page matrix PM in Equation (10) and (11).

3) ILP constraints for memory activities optimization: One of the most critical constraint of the memory activities is that when a task is executed by a given core, all the required memory pages should be placed in the SPM of that core no later than the start time of the execution. This requirement can be expressed as:

$$PM_{p,c,s} \geq Mem_{p,c,s} \quad \forall p \in [0, P-1], c \in [0, C-1] s \in [0, S-1] \tag{17}$$

Another important constraint is that no matter how the pages are transferred, the total amount of pages in the SPM of a core at every clock cycle should not be larger than the capacity of this SPM.

$$\sum_{p=0}^{P-1} OC_{p,c,s} \leq SPM(c) \quad \forall\, s \in [0, S-1],\ c \in [0, C-1] \tag{18}$$

where SPM(c) is the capacity of the core c's SPM.

When data are shared in SPMs, there are several constraints in scheduling sharing activities. For an eligible data sharing in SPMs, the source SPM should have the copy of the target page available when the sharing is initiated.

$$PM_{p,c,s} \geq Mo_{p,c,s} \tag{19}$$

Another constraint we need to set is that only one memory activity can be performed at a clock cycle, due to the arbitration of the data bus across SPMs and the PCM controller. Thus

$$\sum_{p=0}^{P-1} \sum_{c=0}^{C-1} (\widehat{R}_{p,c,s} + \widehat{M}_{p,c,s} + \widehat{Mi}_{p,c,s} + \widehat{W}_{p,c,s} + \widehat{Ev}_{p,c,s} + \widehat{Mo}_{p,c,s}) \leq 1 \quad \forall\, s \in [0, S-1] \tag{20}$$

Remind that we set the rule in our baseline scheduling: when a page is modified by a given core, all the copies in the SPMs of the rest cores should be evicted. There is no conflict data exist in SPMs. To avoid the case that more than one different contents of the same page are copied at the same time, we still need to set a constraint in our ILP model as:

$$\sum_{c=0}^{C-1} Mo_{p,c,s} = 1 \quad \forall\, p \in [0, P-1]],\ s \in [0, S-1] \tag{21}$$

When a page move out process is initiated, there also should be at least one move in process initiated for this page, indicating the source of the sharing. In some cases, maybe multiple cores require this page simultaneously. Then multiple move in processes are initiated. So we can express this constraint as:

$$Mih_{p,s} = \sum_{c=0}^{C-1} Mo_{p,c,s} \quad \forall\, p \in [0, P-1]],\ s \in [0, S-1] \tag{22}$$

To address the memory coherence problems, we set the rule that when a core modifies a given page in its SPM, we will evict all the "dirty" copies of this page in the SPMs of other cores.

$$Ev_{p,c,s} \geq M_{p,c_1,s} \qquad \forall \quad c_1 \neq c \tag{23}$$

The goal of the memory activities optimization is to reduce the number of memory writes. In the baseline scheduling, we do not consider the possible moving of the modified memory. After the page is modified, it will be written back immediately. In this case, we can get the relationship between the SPM modify matrix M and the SPM write matrix W as the following:

$$\sum_{i=0}^{S-1} M_{p,c,i} = \sum_{i=0}^{S-1} W_{p,c,i} \tag{24}$$

The reason why SPM data sharing can reduce the memory writes is that by moving the copy of a given page among SPMs of cores, different tasks can modified this page in serial. And the write back may be initiated after multiple modifications. In this case, Equ. (24) is not necessary. However, even though the number of modifies and the number of writes of a given page may not be equal, at least one write back should be scheduled for a page that had modified previously. Here, we define a 0-1 matrix Mr to indicate whether a page has been modified in the schedule before a give clock cycle. $Mr_{p,s} = 1$ means page p has been modified at least once before the clock cycle s but not written back yet.

$$Mr_{p,s} = Mr_{p,s-1} + \sum_{i=0}^{C-1} (M_{p,i,s} - W_{p,i,s}) \tag{25}$$

In the case that a page has been modified by a given core, but not written back yet, the following tasks that require a copy of this page can only migrate them from the SPM of that core. In other words, the following tasks cannot obtain a copy of this page by reading from the PCM main memory.

$$R_{p,c,s} \leq Mr_{p,s} \qquad \forall \quad c \in [0, C-1] \tag{26}$$

And for every page, it should have a newest copy in the PCM main memory at the end of the schedule. Thus

$$Mh_{p,(S-1)} = 0 \qquad \forall \quad p \in [0, P-1] \tag{27}$$

Finally, our objective of the memory activities scheduling is to minimize the times of write process.

$$\text{Minimize:} \sum_{i=0}^{P-1} \sum_{j=0}^{C-1} \sum_{k=0}^{S-1} W_{i,j,k} \tag{28}$$

D. Post ILP procedure

In our baseline scheduling, we schedule all writes without considering SPM data sharing. Based on this schedule, we optimize the memory activities in our ILP algorithm. Even though the number of writes in the schedule generated by our ILP algorithm is minimized, the start time of each task remains the same as the one in our baseline scheduling. Since the data sharing in SPMs is much less time consuming than the write in the PCM memory, there are a lot of idle slots in which all cores have neither task execution nor memory activities. To improve the system performance, we further eliminate these idle slots in the schedule generated by our ILP algorithm. To remain the data dependencies, we find out these idle slots and push the whole schedule of all cores forward, as long as no data dependency is violated.

VI. Experimental results

A. Experiment setup

TABLE II
THE GROUPING OF BENCHMARKS

Set No.	Benchmarks
Set 1	Convolution, IIR_BIQUAD_N
Set 2	FIR2D, LMS
Set 3	N_REAL_UPDATE, N_COMPLES_UPDATE
Set 4	DOT_PRODUCT, MATRIX_1x3, IIR_BIQUAD_ONE
Set 5	CRC32
Set 6	FFT
Set 7	Blowfish enc
Set 8	Mad
Set 9	PGP sign
Set 10	GSM

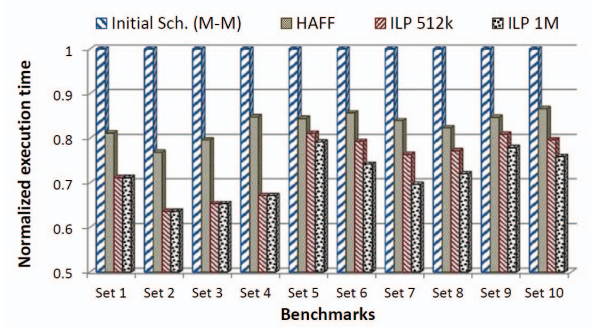
In this section, our proposed ILP algorithm is evaluated by running the DSPstone benchmarks [21] and the MiBench [22]. In our custom simulator, the CMP system has multiple cores, each of which has the similar performance as that of the CoDeL DSP [23]. We compare two different sizes of SPM, which is similar to the SPM setting in [16]. The PCM main memory parameters are set as in [3]. We use the Lingo [24] software to solve the ILP problem.

Since most of the DSPstone benchmarks are embarrassingly parallel, which means there are few data dependencies among tasks, we group multiple DSPstone benchmarks into four benchmark sets. In each set, we create data dependencies by sharing variables among different benchmarks. We also use another six Mibench benchmarks in our experiment, one benchmark per set. The grouping of benchmarks is shown as in Table II.

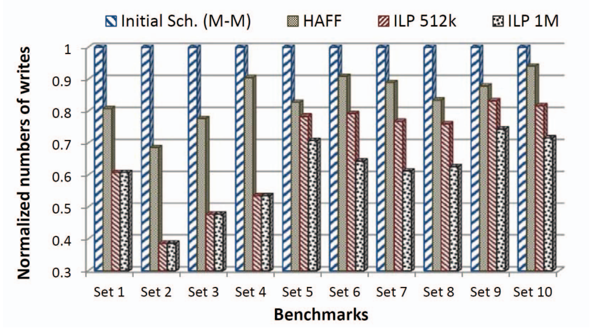
In Fig. 3(a), we compare the performance of our proposed ILP algorithm with that of the HAFF (High Access Frequency First) algorithm [17]. Since its objective is not reducing the numbers of write, we implement a write buffer with 100 entries per core in HAFF to reduce the number of write. In addition, we also compare the numbers of writes in a four-core CMP system in Fig. 3(b). The HAFF has less numbers of writes than that of our baseline scheduling algorithm. Thus, the HAFF outperforms our baseline scheduling algorithm in terms of total execution time. Since our ILP algorithm targets on minimizing the number of writes in the PCM main memory, it outperforms the HAFF algorithm in reducing the numbers of writes by up to 61%. Due to the fact that the write operation in the PCM main memory is the major time consuming operation in the execution of tasks, our algorithm in a four-core CMP system with 512KB SPM reduces the execution time of benchmark set by the percentages from 4.3% to 20.8%, compared to the HAFF algorithm. The performance of CMP with 1MB SPM is slightly better than the one with 512KB SPM, edging by about 5%. Since the DSPstone benchmarks have small size, the size of SPMs makes no difference when running these DSPstone benchmarks.

VII. Conclusions

In this paper, we presented an ILP-based memory activities optimization algorithm for the PCM main memory. In order to increase the lifetime of the PCM memory, we schedule and share the data in SPMs, reducing the redundant writes to the PCM memory in this algorithm. Our experimental results show that our ILP algorithm can significantly reduce the number of write by 61% on average. In addition, the performance of the system is also improved due to less writes that are time-consuming.



(a)



(b)

Fig. 3. (a) The execution time on a four-core CMP system. (b) The numbers of writes on a four-core CMP system. "Initial Sch. (M-M)" is the baseline scheduling with the Min-Min algorithm; "HAFF" is the High Access Frequency First algorithm; "ILP 512K" is our ILP-based algorithm with total 512KB SPMs; and "ILP 1M" is our ILP-based algorithm with total 1MB SPM. All columns are normalized by the corresponding values generated by the baseline scheduling with the Min-Min algorithm.

Acknowledgements

This work was supported in part by the NSF CNS-1249223, NSFC 61071061, the Univ. of Kentucky Start Up Fund; NSF CNS-0969013, CNS-0917021, CNS-1018108; The National High-Technology Research and Development Program of China (863 Program No. 2009AA012201) and the Shanghai International Science and Technology Collaboration Program (09540701900).

References

[1] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. W. Keller, "Energy management for commercial servers," *IEEE Computer*, vol. 36, no. 12, pp. 39–48, 2003.

[2] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mosse, "Increasing PCM main memory lifetime," in *DATE*, 2010, pp. 914–919.

[3] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger., "Architecting phase change memory as a scalable DRAM alternative," in *ISCA*, 2009, pp. 2–13.

[4] B. C. Lee, P. Zhou, J. Yang, Y. Zhang, B. Zhao, E. Ipek, O. Mutlu, and D. Burger, "Phase-change technology and the future of main memory," *IEEE Micro*, vol. 30, no. 1, pp. 131–143, 2010.

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

[6] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y.-C. Chen *et al.*, "Phase-change random access memory: A scalable technology," *Journal of VLSI Signal Processing Systems (JSPS)*, vol. 52, no. 4.5, pp. 465–479, 2008.

[7] M. Qiu, L. Zhang, and E. H.-M. Sha, "ILP optimal scheduling for multimodule memory," in *CODES+ISSS*, 2009, pp. 277–286.

[8] O. Ozturk and M. Kandemir, "Ilp-based energy minimization techniques for banked memories," *ACM Transactions Design Automation of Electronic Systems*, vol. 13, no. 3, pp. 50:1–50:40, 2008.

[9] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in *ISCA*, 2009, pp. 14–23.

[10] M. K. Qureshi, M. M. Franceschini, A. Jagmohan, and L. A. Lastras, "Preset: Improving performance of phase change memories by exploring asysemmetry in write times," in *ISCA*, 2012, pp. 380–391.

[11] N. Seong, D. H. Woo, and H.-H. S. Lee, "Security refresh: protecting phase-change memory against malicious wear out," *IEEE Micro*, vol. 31, no. 1, pp. 119–127, 2011.

[12] M. K. Qureshi, A. Seznec, L. A. Lastras, and M. M. Franceschini, "Practical and secure pcm systems via online attack detection of malicious write streams," in *IEEE International Symposium on High Performance Computer Architecture*, 2011, pp. 478–489.

[13] M. Kandemir, G. Chen, F. Li, and I. Demirkiran, "Using data replication to reduce communication energy on chip multiprocessors," in *ASP-DAC*, 2005, pp. 769–772.

[14] H. Koc, M. Kandemir, E. Ercanli, and O. Ozturk, "Reducing off-chip memory access costs using data recomputation in embedded chip multiprocessors," in *DAC*, 2007, pp. 224–229.

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

[16] V. Suhendra, C. Raghavan, and T. Mitra, "Integrated scratchpad memory optimization and task scheduling for MPSoC architecture," in *CASES*, 2006, pp. 401–410.

[17] L. Zhang, M. Qiu, W.-C. Tseng, and E. H.-M. Sha, "Variable partitioning and scheduling for MPSoC with virtually shared scratch pad memory," *Journal of VLSI Signal Processing Systems (JSPS)*, vol. 58, no. 2, pp. 247–265, 2010.

[18] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, 1977.

[19] J. Li, M. Qiu, J. Niu, and T. Chen, "Battery-aware task scheduling in distributed mobile systems with lifetime constraint," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2011, pp. 743–748.

[20] J. Li, M. Qiu, J. Niu, M. Liu, B. Wang, and J. Hu, "Impacts of inaccurate information on resource allocation for multi-core embedded systems," in *IEEE 10th International Conference on Computer and Information Technology (CIT)*, 2010, pp. 2692 – 2697.

[21] V. Zivojnovic, H. Schraut, M. Willems, and R. Schoenen, "DSPs, GPPs, and multimedia applications - an evaluation using DSPstone," in *The International Conference on Signal Processing Applications and Technology*, 1995, pp. 1–5.

[22] M. R. Guthaus, J. S. Ringenberg, D. Ernst, and T. M. Austin, "Mibench: A free, commercially representative embedded benchmark suite," in *IEEE International Workshop on Workload Characterization*, 2001, pp. 3–14.

[23] N. Agarwal and N. Dimopoulos, "A DSPstone benchmark of CoDeL's automated clock gating platform," in *IEEE Computer Society Annual Symposium on VLSI*, 2007, pp. 508–509.

[24] Lindo Sys. Inc., "LINGO 13.0 - optimization modeling software for linear, nonlinear, and integer programming," http://www.lindo.com, 2011.