Data Placement and Duplication for Embedded Multicore Systems With Scratch Pad Memory

Yibo Guo, Qingfeng Zhuge, Jingtong Hu, *Student Member, IEEE,* Juan Yi, Meikang Qiu, *Senior Member, IEEE,* and Edwin H.-M. Sha, *Senior Member, IEEE*

***Abstract*—Scratch pad memories (SPM) are attractive alternatives for caches on multicore systems since caches are relatively expensive in terms of area and energy consumption. The key to effectively utilizing SPMs on multicore systems is the data placement algorithm. In this paper, two polynomial time algorithms, regional data placement for multicore (RDPM) and regional data placement for multicore with duplication (RDPM-DUP), have been proposed to generate near-optimal data placement with minimum total cost. There is only one copy for each data in RDPM, while RDPM-DUP allows data duplication. Experimental results show that the proposed RDPM algorithm alone can reduce the time cost of memory accesses by 32.68% on average compared with existing algorithms. With data duplication, the RDPM-DUP algorithm further reduces the time cost by 40.87%. In terms of energy consumption, the proposed RDPM algorithm with exclusive copy can reduce the total cost by 33.47% on average. When RDPM-DUP is applied, the improvement increases up to 38.15% on average.**

***Index Terms*—Data duplication, data placement, embedded systems, multicore, scratch pad memory.**

I. Introduction

As DEMANDS for higher performance keep growing, multicore systems have become one of the most promising designs in modern embedded systems. A multicore embedded system has potential to provide near-linear performance improvement. For instance, two smaller processor cores that occupy the same space and use the same energy as the large core can potentially provide 70%–80% higher performance, as compared to only 40% from a large monolithic core [1].

Manuscript received October 26, 2011; revised November 4, 2012; accepted December 22, 2012. Date of current version May 15, 2013. This work was supported in part by NSF CNS-1015802, NSF CNS-1249223, Texas NHARP 009741-0020-2009, NSFC 61173014, National 863 Program 2013AA013202, and the Natural Science Foundation of Chongqing through CSTC2012ggC40005. The work of Meikang Qiu was supported in part by NSF CNS-1249223. This paper was recommended by Associate Editor Y. Xie.

Y. Guo and J. Hu are with the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080 USA (e-mail: yxg091020@utdallas.edu; jthu@utdallas.edu).

M. Qiu is with the Department of Electrical and Computer Engineering, University of Kentucky, Lexington, KY 40506 USA (e-mail: mqiu@engr.uky.edu).

J. Yi and Q. Zhuge are with the College of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: jenni@cqu.edu.cn; qfzhuge@cqu.edu.cn).

E. Sha is with the College of Computer Science, Chongqing University, Chongqing 400044, China, and the Department of Computer Science, University of Texas at Dallas, Richardson, TX 75080 USA (e-mail: edwinsha@cqu.edu.cn).

Digital Object Identifier 10.1109/TCAD.2013.2238990

However, with the number of cores increasing, the cost of adopting hardware-controlled caches in embedded systems also becomes extremely high. There are two reasons for the cost increase. First, the power and performance overheads of automatic memory management in hardware is growing prohibitively [2], [3], [38]. Caches consume about half of the processor's energy for a single-core processor [4]. Second, cache coherency protocols do not scale well with the number of cores. Therefore, scratch pad memories (SPMs), also known as software-controlled on-chip memories, have been widely adopted in many embedded systems as a substitute for caches [5]–[10].

There are several reasons for this fact. One is that SPMs have a 34% smaller area and 40% lower power consumption than caches of the same capacity, as shown by Banakar *et al.* [7], [8]. They also revealed that the runtime measured in cycles was 18% better with an SPM using a simple knapsack-based allocation algorithm [11]. Besides the hardware advantages of SPMs, application-specific embedded system applications normally have compiler analyzable data access patterns, and therefore an optimizing compiler would be in a better position than hardware to manage data transfers across memory hierarchies. Given the power, cost, and performance advantages of SPMs, many existing multicore embedded processors, including TI's TNETV3010 CMP [12] and IBM's Cell Processor [13], are employing SPMs as their on-chip memories.

In order to utilize on-chip SPMs effectively, compilers usually have to carefully determine the data placement for programs so that the cost of memory accesses can be minimized [8]. In most cases, a profiling process will first be carried out to obtain the memory access information [14]. Since embedded systems normally have limited or fixed input sets [3], [15], it is feasible to obtain the memory access information by profiling. Based on profiled information, the compiler determines the best data placement with the minimal memory access cost during compile time.

Data placement methods proposed in existing work can be categorized into two types, global-fixed or regional, depending on whether data placement changes [16]. The global-fixed data placement method generates a single data placement for the whole program, and the data placement remains the same during the execution of the whole program [8], [17]. In regional data placements, programs are divided into program regions, each of which has its own data placement determined

by compilers [18], [19]. During compile time, data movement instructions are inserted in between regions. Each program region will have its own data placement as the program executes. It has been experimentally proven that regional data placement methods achieve better performance and lower energy costs than global-fixed data placement methods [6], [20], since they can take advantage of the data locality of different program regions. Therefore, the regional data placement approach is preferred. It is also the method adopted in this paper.

This paper targets the multicore embedded system. Both single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) are possible. Data sharing activities in SIMD applications are scarce, but prevalent in MIMD applications [21]. In traditional data placements for SPMs, each data only has one copy in one SPM or main memory. Therefore, the core that does not have the data in the local SPM has to access the data from a remote core's SPM or main memory. In this case, a remote access occurs, which is more expensive than accessing its own SPM. To reduce remote memory accesses, in this paper, we introduce a data duplication algorithm along with a data placement algorithm. In data duplications, certain data are purposely chosen to be duplicated in multiple cores' SPMs in order to reduce the total cost of memory accesses.

In this paper, two polynomial-time algorithms are proposed to determine the near-optimal data placement for programs running on SPM-equipped multicore embedded systems. In these two algorithms, programs are divided into parallel regions. Each parallel region is a block of code that can be executed in parallel on multiple cores. The first algorithm, regional data placement for multicore (RDPM), minimizes the total memory access cost for each parallel region when there is only one exclusive copy of data in one of the SPMs or main memory. The second algorithm, regional data placement for multicore with duplication (RDPM-DUP), minimizes the total memory access cost for each parallel region when multiple copies of data are allowed.

According to the experimental results, the proposed RDPM algorithm reduces the time cost of memory access for an eight-core system by 32.68% on average, compared with a greedy algorithm derived from Udayakumaran's algorithm [18]. With data placement and data duplications, the RDPM-DUP algorithm reduces memory access cost by 40.87% on average. For energy consumption, the proposed data placement method alone reduces the energy cost by 33.47%, and together with data duplications, by 38.15%.

The major contributions of this paper include the following.

1) We propose a polynomial-time data placement algorithm for multicore systems equipped with SPMs to minimize the total cost of memory accesses.
2) We present a data duplication technique and integrate it into the data placement algorithm. This data duplication technique can further reduce the memory access cost when multiple data copies are allowed.

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

II. Background and Related Works

SPMs are widely adopted in single-core embedded processors [8], [27], [28]. There are many data placement approaches proposed for SPM in single-core processors [8], [11], [18], [29]. Existing SPM data placement algorithms can be categorized into two types: global-fixed data placement and regional data placement. In global-fixed approaches, the compiler profiles the data access information and determines data placement for the whole program. The data placement remains the same during the execution of the whole program. In [17] and [27], Panda *et al.* proposed techniques to partition the scalar variables and arrays to the SPMs and main memory in order to minimize the execution cost. Avissar *et al.* [8] proposed an integer linear programming (ILP) formulation based on the cost and number of memory accesses for the data placement problem of SPMs. Sjödin *et al.* [30] also proposed a global-fixed data placement method. Global-fixed data placements cannot handle the regional behavior of the program and achieve satisfactory performance.

In order to achieve better performance, regional data placement techniques are proposed. In these methods, programs are divided into small regions. The compiler generates different data placements for different regions. At the beginning of each program region, data movement instructions are inserted to construct a data placement for the current region so that the memory access cost of the current region is reduced. Thus, the regional data placement is able to take advantage of the data locality of the program. Udayakumaran *et al.* [18] proposed a greedy data placement algorithm with the regional data placement approach on a single-core system. Their algorithm divided a program into several regions by program points. Each region has its own data placement according to the memory access information. This technique works properly for single-core systems since the communication cost can be ignored. However, for the multicore systems, the greedy algorithm cannot achieve optimal results, as shown in Section IV.

There are also several works on the data placement algorithms for multicore systems equipped with SPMs. Che *et al.* [31] proposed an ILP formulation and heuristic approach in order to maximize the throughput of stream programs for multicore processors. Two drawbacks exist in this approach. First, it is a global-fixed placement approach. Therefore, it could not take advantage of data locality. Second, ILP takes exponential time. Zhang *et al.* [24] introduced a data partitioning and scheduling method for SPMs on multicore system. In their work, the data placement scheme used could not generate optimal data placement for one region. Kandemir *et al.* [32] proposed a data tiling technique that focused on loop-intensive applications. Their methods only worked for arrays and greatly relied on the characteristics of loops. Suhendra *et al.* [33] proposed an ILP formulation for task scheduling and data allocation on MPSoC architectures. Their

ILP formulation considered task mapping, scheduling, SPM partitioning, and data allocation. Since ILP is known for exponential completion time, the completion time of their technique is too long for large applications. Moreover, their data placement is global fixed. The techniques proposed in this paper employ the regional data allocation method, which can better adapt to the regional behavior of programs and take advantage of the program data locality.

Data duplication is often used for data assurance. Li *et al.* [34] proposed a method to duplicate blocks in order to protect SPMs from soft errors. Issenin *et al.* [35] proposed a data reuse technique for multicore systems with SPMs. However, if the number of memory accesses to the shared data is enormous, the shared buffer will be the bottleneck and many remote accesses will occur. Therefore, it is beneficial to duplicate data if the data are accessed many times from different cores. Even though a limited number of initial off-chip memory accesses occur when duplicating, the savings gained from the later local SPM accesses can offset the initial cost. If the gains cannot offset the initial cost, data will not be duplicated in the proposed algorithm. In our research, with appropriate data duplication, the time or energy cost for a program region on multicore systems can be further minimized.

III. Hardware and Software Model

A. Hardware Architecture

This paper targets embedded multicore systems that employ virtually shared SPMs (VS-SPM) [25]. Fig. 1 shows the VS-SPM architecture. Every core has its own private on-chip SPM, while all cores share the main memory. Each core can access its own private SPM. It can also access data items on other cores' SPMs by the interconnecting bus. Each core can access all other SPMs. There is no limit for the number of remote SPMs that a core can access in the architecture. The Cell processor [13] is one of the examples that adopted this architecture. Any two cores can communicate with each other without contention.

In VS-SPM architecture, the private SPM of a core is called the local SPM, while the SPMs on the other cores are called remote SPMs. The cost of accessing remote SPMs is higher than accessing the local SPM since a communication cost exists when there are remote accesses. In our architecture, each core can access all remote SPMs. Let d be the distance between two cores. The cost of remote SPMs accessing is a nondecreasing function f of d. The function is different for different bus architectures.

B. Execution Model

The execution model is shown in Fig. 2. Basically, the program is divided into many small tasks. In this paper, we treat each basic block as an independent task. Tasks of the same depth in the task graph form a parallel region. Between each parallel region, the compiler adds data placement instructions, which assign data to each core's SPM for the incoming parallel region.

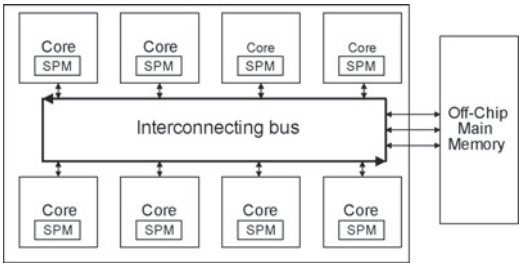


Fig. 1. VS-SPM architecture.

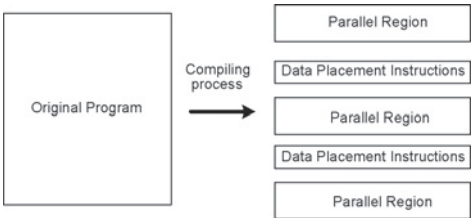


Fig. 2. Demonstration for parallel regions.

After the regions are determined, the number of accesses on each variable for the region is obtained through profiling. Based on the profiled information, the proposed algorithms determine the best data placement for each region.

Please note that if the profiling information is based on a single input, the proposed algorithms will find the optimal data placement for this particular input. If the profiling information is based on a set of inputs, then the data placement generated is near optimal for a particular input. For some embedded systems that repeatedly carry out the same tasks, the proposed algorithms can achieve optimal results. However, if the input patterns are dynamic and unexpected, the proposed algorithms can only obtain near-optimal results.

IV. Motivational Example

In the motivational example, the goal of optimization is to minimize the total memory access cost of a parallel region. The memory access cost can either be memory accessing time or energy consumption. If the optimization goal is memory accessing time, the energy cost is not necessarily the optimal, and vice versa. However, if we choose either one of the two costs to be the optimization objective, the most accessed data will be allocated to the memory area with fewer costs. Since, for memory accesses, the time cost is directly proportional to the energy cost, reducing one of the two costs will reduce the other as well.

For the motivational example, we are assuming that there are two cores in the system. Each core is equipped with an on-chip SPM. For the purpose of simplicity and illustration, we are assuming that each SPM can only hold two data items in this motivational example. All cores can access a shared main memory, which is large enough to hold all data for the program.

Before presenting the example, a list of notations that will be used for the whole paper is defined in Table I. The column Notation contains the names of variables and the column Definition contains the definition of each variable.

TABLE I
NOTATIONS IN THIS PAPER

Notation	Definition
$Size_{S_i}$	The capacity of the SPM on core i
CR_{S_i}	The cost of reading a data item from its own SPM for core i
CW_{S_i}	The cost of writing a data item to its own SPM for core i
CR_{M_i}	The cost of reading a data item from main memory for core i
CW_{M_i}	The cost of writing a data item to main memory for core i
$CR_{i\to S_j}$	The cost of reading a data item from core j's SPM for core i
$CW_{i\to S_j}$	The cost of writing a data item to core j's SPM for core i
$CM_{S_i\to S_j}$	The cost of moving a data item from SPM of core i to SPM of core j
$CM_{S_i\to M}$	The cost of moving a data item from SPM of core i to main memory
$CM_{M\to S_i}$	The cost of moving a data item from main memory to SPM of core i

TABLE II
NUMBER OF MEMORY ACCESSES FOR EACH CORE

	A	B	C	D	E
Core 1	50	25	20	5	0
Core 2	49	10	19	0	5

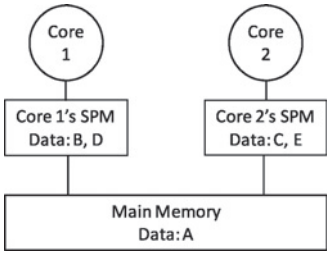


Fig. 3. Initial state of SPM and main memory.

In the motivational example, we assume that these two cores access five data items: A, B, C, D, and E. The number of memory accesses of each data is shown in Table II. The rows Core 1 and Core 2 show the number of memory accesses of each data item for Core 1 and Core 2, respectively. Assume that initially, data A is in the shared main memory; data B and data D are in the SPM of Core 1 (SPM_1); data C and data E are in the SPM of Core 2 (SPM_2) (Fig. 3). Also, the cost of a single memory access for the two cores and the shared main memory are defined in Table III. For illustration purposes, it is assumed that the cost of reading and writing data from a private SPM is 1, and the cost of reading and writing data from the main memory is 50. The nondecreasing remote SPM accessing cost function is a linear function $f(d) = 10 \times d$ for this example. Since we only have two cores in the example, the distance d equals 1. Therefore, the remote SPM accessing costs $CR_{S_1\to S_2}$, $CR_{S_2\to S_1}$, $CW_{S_1\to S_2}$, and $CW_{S_2\to S_1}$ are 10.

Udayakumaran's algorithm proposed in [18] is a greedy algorithm, in which programs are divided into regions, and the most accessed data items in each region are allocated into the SPM. However, their approach only targets single-core processors. We adopted their algorithm and applied it to multicore systems for comparison purposes. The derived Udayakumaran algorithm works as follow: first, data items that need to be accessed are sorted according to the total number of accesses. The total number of accesses is the sum of the number of accesses from all cores to this data item. Then, the data item with the most total number of accesses is chosen by the compiler. There might be many cores that access this data. The compiler allocates the data into the available SPM of the core that accesses the data most times. When all SPMs of the cores are full, the data are allocated into the main memory. The data placement generated by the derived Udayakumaran algorithm is not optimal. For the motivational example, the derived greedy Udayakumaran algorithm will allocate data A and C in Core 1's SPM; data B and D in Core 2's SPM; and data E in the main memory. The total memory accessing cost is 1445.

TABLE III
ACCESS COST TABLE FOR MOTIVATIONAL EXAMPLE

Data	Cost
CR_{S_1}, CR_{S_2}	1
CW_{S_1}, CW_{S_2}	1
CR_{M_1}, CR_{M_2}	50
CW_{M_1}, CW_{M_2}	50
$CR_{1\to S_2}$, $CR_{2\to S_1}$	10
$CW_{1\to S_2}$,$CW_{2\to S_1}$	10
$CM_{S_1\to S_2}$, $CM_{S_2\to S_1}$	11
$CM_{M\to S_1}$, $CM_{M\to S_2}$	51
$CM_{S_1\to M}$, $CM_{S_2\to M}$	51

Rather than the greedy data placement, if data B and D are placed in SPM_1, data A and C are in SPM_2, and data E are in the main memory, the total memory access cost is 1250. The cost of memory access is reduced by 13.49% compared with the greedy data placement. Actually, this is the data placement with the minimum memory access cost for the example. Section V-A will show how to achieve this optimal solution with a dynamic programming approach.

Data duplication is a technique that trades space for time efficiency. In a multicore environment, data may be accessed by different cores, and sometimes it is beneficial to duplicate data and place multiple copies of the same data in different locations. In the motivational example, it is assumed that all data are only read and not modified during the current parallel region. Thus, they may all have duplications. With data duplication, there is another possible data placement: Data A, B in SPM_1, data A, C in SPM_2, and data D, E in the main memory. The memory access cost can be further reduced to 1107 using this data placement. Compared with the derived Udayakumaran algorithm, the cost of memory access is reduced by 23.39%. In Section V-B, the RDPM-DUP algorithm, which determines the optimal data placement and duplication for each program region, will be presented.

V. DATA PLACEMENT AND DUPLICATION

In this section, two novel data placement algorithms are presented. In Section V-A, the SPM data placement with

$$CostMin[j, i_1, i_2, ..., i_C] = \begin{cases} \sum_{j=1}^{N_d} Cost_M(D_j), & \text{if } \forall k = 1, 2, ..., n,\ i_k = Size_{S_k}, \\ \sum_{j=2}^{N_d} Cost_M(D_j) + Cost_{S_k}(D_1) & \text{if } j = 1,\ \exists i_k = Size_{S_k} - 1, \\ & \forall k' \in \{1, 2, ..., C\} - \{k\}, i_{k'} = Size_{S_k} \\ \min(CostMin[j-1, i_1, i_2, ..., i_C], & \\ CostMin[j-1, i_1+1, i_2, ..., i_C] - Cost_M(D_j) + Cost_{S_1}(D_j), & \\ CostMin[j-1, i_1, i_2+1, ..., i_C] - Cost_M(D_j) + Cost_{S_2}(D_j), & \\ \dots, & \\ CostMin[j-1, i_1, i_2, ..., i_C+1] - Cost_M(D_j) + Cost_{S_n}(D_j) & \text{if } \sum_{k=1}^{C} i_k \geq \sum_{k=1}^{C} Size_{S_k} - j, \\ \infty & \text{if } \sum_{k=1}^{C} i_k < \sum_{k=1}^{C} Size_{S_k} - j \text{ or} \\ & \exists k \in \{1, 2, 3, \dots, C\}\ i_k > Size_{S_k}. \end{cases} \tag{2}$$

minimum cost problem on multicore is formally defined. After that, the RDPM algorithm is proposed. Section V-B will present how to determine the optimal data duplication and propose the RDPM-DUP.

A. Regional Data Placement for Multicore

Before introducing the proposed RDPM algorithm, the formal definition of the SPM data placement with minimal cost problem on multicore is presented.

1) *Problem Definition:* The inputs are: A collection of data $D = (d_1, d_2, \dots, d_{N_d})$; the initial data placement for each core's on-chip SPM; capacity of each core's SPM S_i for core i; number of data N_d; number of cores C; read and write cost to the local on-chip SPM R_{S_i}, W_{S_i} for core i; the read and write cost from core i to core j's SPM $R_{C_i \to S_j}$, $W_{C_i \to S_j}$; the read and write cost to the main memory R_{M_i}, W_{M_i} for core i.

Definition 1: SPM data placement with minimal cost problem on multicore systems: Given the inputs, what is a data placement for all cores' SPMs and the shared main memory that the total time/energy cost of memory accesses is minimized?

The output is: A data placement for all cores' SPMs and main memory, under which the total cost of memory access is minimized.

2) *RDPM Algorithm:*

a) *Compute cost of accessing remote SPM:* Let C be the number of cores in the system. Let d be the distance between two cores. Let the nondecreasing remote SPM accessing cost function be f. In this step, we will compute $CR_{i \to S_j}$ and $CW_{i \to S_j}$ for every pair of (i, j) using $f(d)$.

b) *Build cost table:* After we have computed all $CR_{i \to S_j}$ and $CW_{i \to S_j}$, the cost table T can be built. Let there be $C + 1$ columns in the table T indicating $C + 1$ different placements for a data item. The costs of each data in different locations are computed as shown in (1), which is the sum of executing cost and placement cost. To compute the execution cost of each data, we first count the number of memory accesses to the data and multiply it by the cost of each access for all the cores. Then, all cores' costs are summed up for the data to obtain the execution cost. The placement cost is the cost of moving the data from its initial placement to the new placement, which includes a read operation from the original memory and a write operation to the target memory. If the data is not moved, the placement cost is 0.

$$\sum_{1}^{C} \text{executing\_cost} + \text{placement\_cost} \tag{1}$$

c) *Dynamic programming scheme:* After the cost table is built, in the second step, a dynamic program algorithm as shown in (2) is proposed to find the data placement.

First, a C + 1-dimensional dynamic programming table *CostMin* is constructed as follows: the first dimension of the table represents data items; each of the other dimensions represent the available space on a certain core's SPM, assuming the shared main memory is large enough to hold all the data items of the program.

Let $CostMin[j, i_1, i_2, ..., i_C]$ be the minimal cost of memory accesses when the placement of data j ($j \leq N_d$, N_d is the number of data for the current region) has been optimally determined while the rest of the data k ($j < k \leq N_d$) are in the main memory, and there are i_1 empty memory units on SPM of core 1, i_2 empty memory units on SPM of core 2, ..., and i_C empty memory units on SPM of core C.

The complexity of the RDPM algorithm is in polynomial time. The total number of iterations is $N_d \cdot Size_{S_1} \cdot Size_{S_2} \cdot \dots \cdot Size_{S_C}$. Since the architecture is determined, C is a constant. Inside the most inner loop, there are $C + 1$ if/else branches to decide the value of the current cell. Thus, the complexity of RDPM is $O(n^{C+1})$.

B. Data Duplication on Multicore Systems

In multicore systems, traditionally, a data item only has one copy in either one of the SPMs or in the main memory. It is common that multiple cores may access the same data in one parallel region. In this section, a data duplication strategy for read-only data on SPMs of multiple cores is proposed. First, the duplication mechanism in this paper is introduced. Then,

$$CMD[j, i_1, i_2, ..., i_C] = \begin{cases} \sum_{j=1}^{N_d} Cost_M(D_j), & \text{if } \forall k = 1, 2, ..., n, \\ & i_k = Size_{S_k}, \\ \sum_{j=2}^{N_d} Cost_M(D_j) + Cost_{S_k}(D_1) & \text{if } j = 1, \ \exists i_p = Size_{S_p} - 1, \\ & \forall k \in \{1, 2, ..., C\} - \{p\}, i_k = Size_{S_k} \\ \min(CMD[j-1, i_1, i_2, ..., i_C], & \\ CMD[j-1, i_1+1, i_2, ..., i_C] - Cost_M(D_j) + Cost_{S_1}(D_j), & \\ CMD[j-1, i_1, i_2+1, ..., i_C] - Cost_M(D_j) + Cost_{S_2}(D_j), & \\ \ldots, & \\ CMD[j-1, i_1, i_2, ..., i_C+1] - Cost_M(D_j) + Cost_{S_C}(D_j)), & \\ CMD[j-1, i_1+1, i_2+1, ..., i_C] - Cost_M(D_j) + Cost_{S_{1+2}}(D_j)), & \\ CMD[j-1, i_1, i_2+1, i_3+1, ..., i_C] - Cost_M(D_j) + Cost_{S_{2+3}}(D_j)), & \\ CMD[j-1, i_1, i_2, i_3, ..., i_p+1, ..., i_q+1, ..., i_C] & \\ -Cost_M(D_j) + Cost_{S_{p+q}}(D_j)), & \\ \text{(Calculate all two combinations' cost, } p, \ q = 1, 2, ..., \ n) & \\ \ldots & \\ CMD[j-1, i_1, i_2, ..., i_p+1, ..., i_q+1, ..., i_r+1, ..., i_C] & \\ -Cost_C(D_j) + Cost_{S_{p+q+r}}(D_j)), & \\ \text{(Calculate all three combinations' cost,} & \\ p, \ q, \ r = 1, 2, ..., \ n) & \\ \ldots & \\ \text{(Calculate the cost for all combinations of } t \text{ copies, } t \text{ is} & \\ \text{the maximum number of cores allowed to share SPMs.)} & \\ & \text{if } Cost_{S_t \text{ combinations}}(D_j) \neq \infty \text{ and} \\ & \sum_{k=1}^{C} i_k + t \geq \sum_{k=1}^{C} Size_{S_k} - j, \\ \infty & \sum_{k=1}^{C} i_k + t < \sum_{k=1}^{C} Size_{S_k} - j \text{ and} \\ & \text{if } Cost_{S_t \text{ combinations}}(D_j) = \infty \text{ or} \\ & \exists \ k \in \{1, 2, 3, \ldots, C\}, \ i_k > Size_{S_k}. \end{cases} \tag{3}$$

TABLE IV
SYSTEM SPECIFICATION FOR THE EIGHT-CORE ARCHITECTURE

Component	Description
CPU Core	Number of cores: 8, frequency: 1.0 GHz
SRAM SPM	Size: 8 kB, access latency: 0.305 ns, access energy: 0.014 nJ
Main memory	DDR SDRAM, Size: 512 MB, Access latency: 19.51 ns access energy: 0.607 nJ

how to integrate data duplication into the data placement algorithm presented in the previous section in order to determine the best data placement and duplication is shown.

When there is no data duplication mechanism, if a data item is intensively accessed by multiple cores, it will incur a lot of remote accesses. Wherever the data is placed, there is only one core being benefited from the local SPM. Data duplication method will solve this problem by allocating a copy of the data item to each SPM that may be benefited. As a result, the time and energy cost incurred by remote data accesses is reduced.

For exclusive copy mode, there is no need to worry about the data consistency. However, for data duplication mechanisms, the data consistency problem becomes a key issue. In such cases, inconsistency will occur when there are multiple cores that want to write to the same data. It is true that in writing heavy applications, duplicating to-be-written data may be beneficial with a well-designed data consistency protocol. However, the overhead caused by maintaining data consistency may offset the benefits of duplicating written data. Therefore, in this paper, only read data is allowed to be duplicated.

The first step in integrating data duplication into RDPM is modifying the cost tables. In Section V-A, data has the cost for each memory placement in the cost table. In data duplication, each data can be duplicated in multiple SPMs. Therefore, there are many different possible ways of duplication for the same data. All the possible data duplication configurations need to be considered. In the cost table, a new column is added for each possible way of data duplication for duplicated data. For instance, in the motivational example, there is a possibility

TABLE V
COMPARISON OF TIME COST AMONG VARIOUS ALGORITHMS ON THE EIGHT-CORE SYSTEM

Benchmarks	Che (μJ)	Uday (μJ)	RDPM (μJ)	Imprv-Che (%)	Imprv-Uday (%)	RDPM-DUP (μJ)	Imprv-Che (%)	Imprv-Uday (%)
basicmath	12696.38	10434.65	6260.70	50.69	40.01	5858.08	53.86	43.85
btcount	751.55	672.14	388.08	48.36	42.26	283.28	62.31	57.85
qsort	19005.03	12889.47	9540.48	49.80	25.98	8558.41	54.97	33.60
susan	3246.32	1806.13	1331.83	58.97	26.26	1131.25	65.15	37.36
dijkstra	914.73	686.12	416.91	54.42	39.24	351.56	61.57	48.76
patricia	10853.42	6618.85	5059.12	53.39	23.57	4408.67	59.38	33.39
stringsearch	1712.88	1098.17	827.56	51.69	24.64	675.50	60.56	38.49
rijndael	13513.36	8354.76	5895.92	56.37	29.43	5534.15	59.05	33.76
SHA	7362.61	4864.20	3347.06	54.54	31.19	3066.68	58.34	36.95
CRC32	6279.95	4172.32	2629.81	58.12	36.97	2475.66	60.58	40.66
FFT	5363.44	4126.83	2481.25	53.73	39.89	2274.12	57.60	44.91
Average	–	–	–	**53.64**	**32.68**	–	**59.40**	**40.87**

TABLE VI
COMPARISON OF ENERGY COST AMONG VARIOUS ALGORITHMS ON THE EIGHT-CORE SYSTEM

Benchmarks	Che (μJ)	Uday (μJ)	RDPM (μJ)	Imprv-Che (%)	Imprv-Uday (%)	RDPM-DUP (μJ)	Imprv-Che (%)	Imprv-Uday (%)
basicmath	295.87	235.48	158.78	46.33	32.57	149.27	49.55	36.61
btcount	43.31	40.07	16.13	62.75	59.74	13.57	68.67	66.13
qsort	515.03	291.58	212.35	58.77	27.30	210.84	59.06	27.70
susan	114.23	62.81	45.65	60.03	27.31	43.62	61.82	30.55
dijkstra	44.88	28.97	20.65	54.01	28.71	18.26	59.33	36.98
patricia	430.52	307.66	205.06	52.37	33.35	186.85	56.59	39.26
stringsearch	88.46	60.57	45.44	48.63	24.99	43.08	51.30	28.87
rijndael	310.65	205.62	134.54	56.69	34.57	125.83	59.50	38.81
SHA	219.45	176.57	119.23	45.67	32.48	110.55	49.62	37.39
CRC32	190.27	142.70	97.55	48.73	31.64	91.26	52.03	36.05
FFT	206.50	165.42	106.49	48.43	35.62	97.03	53.01	41.34
Average	–	–	–	**52.95**	**33.47**	-	**56.41**	**38.15**

that each data has two copies, and the copies are in SPM_1 and SPM_2. If any data item cannot have multiple copies according to the restriction, their costs in the new column are set to be infinity.

Second, we define a new $C + 1$-dimensional dynamic programming table CMD for duplication method. Let $CMD[j, i_1, i_2, ..., i_C]$ be the minimal memory accessing cost when the placement and duplication of the jth data ($j \leq N_d$) are optimally determined, while the rest of data is in the shared main memory, and there are i_k empty memory units on SPM of core k ($k \leq C$). Here, C is the number of cores.

Third, the recursive function of RDPM should be modified for the data duplication method. The new recursive function is shown in (3). The new equation needs to consider the cost of all possible number of copies, and all possible places that hold the copies.

VI. EXPERIMENTS

Experiments are performed on a selected set of benchmarks from Mibench [36] to compare both time and energy costs of memory accesses for four data allocation techniques: the Che's algorithm [31], the derived Udayakumaran algorithm for multicore, the RDPM algorithm, and the RDPM-DUP algorithm. The experimental results show promising improvement for the algorithms proposed in this paper compared with the existing greedy algorithm.

A. Experimental Setup

All experiments are conducted on a custom simulator. The simulator is flexible for different hardware configurations. In this paper, we conduct the experiments on an eight-core system. The hardware configuration of the system is shown in Table IV. The costs of memory accesses are obtained from HP CACTI 5.3 [37]. All cores share an off-chip DRAM main memory, and any pair of cores can access each other's local SPM.

The nondecreasing cost function f of accessing remote SPMs that we used in the experiments is a linear function $f = d \times \beta$, where d is the distance between the two cores and β is a constant cost. For the eight-core system, β equals 0.305 ns when we compute remote time cost and β equals 0.014 nJ for the remote energy cost.

The benchmarks used in the experiments are from Mibench [36]. Eleven applications are selected from the Mibench benchmark suite: qsort, susan, basicmath, bitcount, dijkstra, patricia, stringsearch, rijndael, sha, CRC32, and FFT. The memory traces of these benchmarks are the input for the simulator.

B. Experimental Results

In this subsection, the comparisons of time and energy cost for the eight-core system are shown in Tables V and VI.

Tables V and VI reflect the experimental results on an eight-core environment. The average time cost reductions

are 53.64% for Che's algorithm and 32.68% for the derived Udayakumaran algorithm. The average energy cost reductions are 56.41% and 38.15%, respectively.

The reason that RDPM and RDPM-DUP are significantly better than Che's algorithm is that the goal of Che's algorithm is to achieve the maximum throughput. It does not include sufficient techniques that reduce the memory access cost. Also, in Che's algorithm, all data have to be moved into the SPM before being accessed. This leads to a large number of unnecessary data movements, which significantly increases the total cost.

From the experimental results, it is easy to see that the RDPM and RDPM-DUP algorithms have a better performance for time latency, as well as energy consumption, compared with both of two baseline algorithms. Furthermore, the RDPM-DUP algorithm determines the optimal number of copies for a heavily accessed data and places the data copies into appropriate SPMs. It will always generate a data placement at least as good as that of the RDPM algorithm. When there is no suitable duplication for any data, the RDPM-DUP algorithm will have the same optimal solution as the RDPM algorithm.

VII. Conclusion

In this paper, two polynomial time regional data placement algorithms were proposed to minimize the cost of memory accesses for multicore systems. The RDPM algorithm can achieve near-optimal data placement for each region with exclusive copy, while the RDPM-DUP algorithm is able to generate near-optimal data placement and duplication when the multiple copies for a single data item are allowed. Experimental results show that the proposed RDPM algorithm alone can reduce the time cost of memory accesses by 32.68% on average compared with existing algorithms. With data duplication, the RDPM-DUP algorithm further reduces the time cost by 40.87%. For energy consumption, the proposed RDPM algorithm with exclusive copy can reduce the total cost by 33.47% on average. The improvement increases up to 38.15% on average when RDPM-DUP is applied.

References

[1] S. Borkar, "Thousand core chips: A technology perspective," in *Proc. DAC*, 2007, pp. 746–749.

[2] M. Qiu, Z. Shao, Q. Zhuge, C. Xue, M. Liu, and E. H.-M. Sha, "Efficient assignment with guaranteed probability for heterogeneous parallel DSP," in *Proc. ICPADS*, 2006, pp. 623–630.

[3] J. Xue, T. Liu, Z. Shao, J. Hu, Z. Jia, and E. H.-M. Sha, "Address assignment sensitive variable partitioning and scheduling for DSPS with multiple memory banks," in *Proc. ICASSP*, 2008, pp. 1453–1456.

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

[5] S. Gilani, N. S. Kim, and M. Schulte, "Scratchpad memory optimizations for digital signal processing applications," in *Proc. DATE*, 2011, pp. 1–6.

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

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

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

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

[10] Y. He, C. Xue, C. Xu, and E. H.-M. Sha, "Co-optimization of memory access and task scheduling on MPSoC architectures with multilevel memory," in *Proc. ASP-DAC*, 2010, pp. 95–100.

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

[12] S. Kaneko, H. Kondo, N. Masui, K. Ishimi, T. Itou, M. Satou, N. Okumura, Y. Takata, H. Takata, M. Sakugawa, T. Higuchi, S. Ohtani, K. Sakamoto, N. Ishikawa, M. Nakajima, S. Iwata, K. Hayase, S. Nakano, S. Nakazawa, K. Yamada, and T. Shimizu, "A 600-MHz single-chip multiprocessor with 4.8-Gb/s internal shared pipelined bus and 512-kb internal memory," *IEEE J. Solid-State Circuits*, vol. 39, no. 1, pp. 184–193, Jan. 2004.

[13] H. P. Hofstee, "Power efficient processor architecture and the Cell processor," in *Proc. HPCA*, 2005, pp. 258–262.

[14] S. Udayakumaran and R. Barua, "An integrated scratch-pad allocator for affine and non-affine code," in *Proc. DATE*, 2006, pp. 925–930.

[15] Y. Guo, Q. Zhuge, J. Hu, and E.-M. Sha, "Optimal data placement for memory architectures with scratch-pad memories," in *Proc. ICESS*, 2011, pp. 1045–1050.

[16] Q. Zhuge, Y. Guo, J. Hu, W.-C. Tseng, S. J. Xue, and E.-M. Sha, "Minimizing access cost for multiple types of memory units in embedded systems through data allocation and scheduling," *IEEE Trans. Signal Process.*, vol. 60, no. 6, pp. 3253–3263, Jun. 2012.

[17] P. R. Panda, N. D. Dutt, and A. Nicolau, "On-chip vs. off-chip memory: The data partitioning problem in embedded processor-based systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 5, pp. 682–704, Jul. 2000.

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

[19] M. Kandemir, M. J. Irwin, G. Chen, and I. Kolcu, "Compiler-guided leakage optimization for banked scratch-pad memories," *IEEE Trans. Very Large Scale (VLSI) Syst.*, vol. 13, no. 10, pp. 1136–1146, Oct. 2005.

[20] Y. Guo, Q. Zhuge, J. Hu, M. Qiu, and E.-M. Sha, "Optimal data allocation for scratch-pad memory on embedded multi-core systems," in *Proc. ICPP*, 2011, pp. 464–471.

[21] R. Buchty, V. Heuveline, W. Karl, and J.-P. Weiss, "A survey on hardware-aware and heterogeneous computing on multicore processors and accelerators," *Concurrency Comput.: Practice Experience*, vol. 24, no. 17, pp. 663–675, 2012.

[22] J. Chang and G. S. Sohi, "Cooperative cache partitioning for chip multiprocessors," in *Proc. ICS*, 2007, pp. 242–252.

[23] G. E. Suh, L. Rudolph, and S. Devadas, "Dynamic cache partitioning for simultaneous multithreading systems," in *Proc. IASTED PDCS*, 2001, pp. 116–127.

[24] L. Zhang, M. Qiu, and W.-C. Tseng, "Variable partitioning and scheduling for MPSoC with virtually shared scratch pad memory," *J. Signal Process. Syst.*, vol. 50, no. 2, pp. 247–265, 2010.

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

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

[27] P. R. Panda, N. D. Dutt, and A. Nicolau, "Efficient utilization of scratch-pad memory in embedded processor applications," in *Proc. ED&TC*, 1997, p. 7.

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

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

[30] J. Sjödin and C. von Platen, "Storage allocation for embedded processors," in *Proc. CASES*, 2001, pp. 15–23.

[31] W. Che, A. Panda, and K. S. Chatha, "Compilation of stream programs for multicore processors that incorporate scratchpad memories," in *Proc. DATE*, 2010, pp. 1118–1123.

[32] M. Kandemir, J. Ramanujam, and A. Choudhary, "Exploiting shared scratch pad memory space in embedded multiprocessor systems," in *Proc. DAC*, 2002, pp. 219–224.

[33] V. Suhendra, C. Raghavan, and T. Mitra, "Integrated scratchpad memory optimization and task scheduling for MPSoC architectures," in *Proc. CASES*, 2006, pp. 401–410.
[34] M. K. F. Li and G. Chen, "Improving scratch-pad memory reliability through compiler-guided data block duplication," in *Proc. ICCAD* , 2005, pp. 1002–1005.
[35] I. Issenin, E. Brockmeyer, B. Durinck, and N. Dutt, "Multiprocessor system-on-chip data reuse analysis for exploring customized memory hierarchies," in *Proc. DAC*, 2006, pp. 49–52.
[36] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. WWC*, 2001, pp. 3–14.
[37] S. J. E. Wilton and N. P. Jouppi, "CACTI: An enhanced cache access and cycle time model," *IEEE J. Solid-State Circuits*, vol. 31, no. 5, pp. 677–688, May 1996.
[38] M. Qiu and E. H.-M. Sha, "Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 2, pp. 1–30, Apr. 2009.

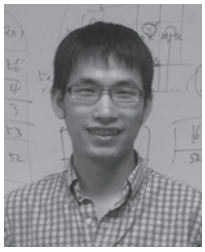
Yibo Guo received the B.S. degree in information security from Hunan University, Hunan, China, in 2009, and the M.S. degree in computer science from the University of Texas at Dallas, Richardson, TX, in 2011, where he is currently pursuing the Ph.D. degree in computer science.

His current research interests include memory scheduling and data allocation on MPSoc.

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

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

Dr. Zhuge received the Best Ph.D. Dissertation Award in 2003.



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

His current research interests include low power and high-performance embedded systems, wireless sensor networks, memory optimization, nonvolatile memory, and compiler optimization.

Juan Yi received the B.E. degree from the School of Software Engineering at Chongqing University, Chongqing, China, in 2006 and is currently pursuing the Ph.D degree from the Department of Computer Science at the same university.

Her current research interests include multicore architecture optimization and high-performance parallel computing.

Meikang Qiu (SM'07) received the B.E. and M.E. degrees from Shanghai Jiao Tong University, Shanghai, China, and the M.S. and Ph.D. degrees in computer science from the University of Texas at Dallas, Richardson, TX, in 2003 and 2007, respectively.

He was with the Chinese Helicopter Research and Development Institute and was also with IBM. He is currently an Assistant Professor of ECE at the University of Kentucky, Lexington. He also holds three patents and has published three books. His current research interests include embedded systems, computer security, and wireless sensor networks.

Dr. Qiu is an ACM Senior member. He has published 160 peer-reviewed papers, including 16 IEEE/*ACM Transactions on Networking* papers and more than 60 journal papers. He is the recipient of the *ACM Transactions on Design Automation of Electronic Systems* 2011 Best Paper Award. He also received four other Best Paper Awards (IEEE EUC'09, IEEE/ACM GreenCom'10, IEEE CSE'10, and IEEE ICESS) and one best paper nomination in the last four years. He was named to the Navy Summer Faculty in 2012 and SFFP Air Force Summer Faculty in 2009. His research is supported by the National Science Foundation, Navy, and Air Force. He has held various chair positions and served as a TPC member for many international conferences. He served as the Program Chair of IEEE EmbeddCom'09 and EM-Com'09.

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

From August 1992 to August 2000, he was with the Department of Computer Science and Engineering at the University of Notre Dame, Notre Dame, IN. Since 2000, he has been a Tenured Full Professor at the Department of Computer Science at the University of Texas at Dallas, Richardson, TX. Since 2012, he has been serving as the Dean of the College of Computer Science at Chongqing University, Chongqing, China. He has published more than 280 research papers in refereed conferences and journals. He has served as an editor for many journals, on program committees, and as a Chair for numerous international conferences.

Dr. Sha received the Teaching Award, Microsoft Trustworthy Computing Curriculum Award, NSF CAREER Award, NSFC Overseas Distinguished Young Scholar Award, and Chang Jiang Honorary Chair Professorship. He is a member of the China Thousand-Talent Program.