Optimizing Task and Data Assignment on Multi-Core Systems with Multi-Port SPMs

Shouzhen Gu, Qingfeng Zhuge, Juan Yi, Jingtong Hu, *Member, IEEE*, and Edwin Hsing-Mean Sha, *Senior Member, IEEE*

Abstract—Multi-core processors have been adopted in modern embedded systems to meet the ever increasing performance requirements. Scratchpad memory (SPM), a software-controlled on-chip memory, has been used in embedded systems as an alternative to hardware-controlled cache due to its advantage in die area, power consumption, and timing predictability. SPMs in multi-core systems can be accessed by both local core and remote cores. In order to alleviate data contention on a SPM unit, multi-port SPMs are employed in multi-core systems. In such systems, proper task scheduling and data assignment can significantly improve the overall performance by exploring the parallelism of computation tasks and concurrent data accesses on SPMs. Since scheduling for multi-core systems is NP-Complete in general. In this paper, we propose an ILP formulation to optimally determine the task scheduling and data assignment on multi-core systems with multi-port SPMs. Since ILP takes exponential time to finish, we also propose a heuristic method, including the *task assignment with remote access reduced* (TARAR) algorithm and the *minimum memory access cost* (MMAC) algorithm, to obtain near optimal solutions within polynomial time. According to the experimental results, the ILP formulation can improve the system performance by 23.02 percent over the HAFF algorithm on average, while the heuristic algorithm can improve the system performance by 16.48 percent over HAFF on average.

Index Terms—Task scheduling, data assignment, scheduling, multi-core systems, multi-port SPMs

1 Introduction

To satisfy the ever-growing performance demands of high-performance applications such as digital signal processing (DSP), multimedia, and telecommunications, etc., more and more cores are integrated on a single chip [1], [2], [3]. Scratchpad memory (SPM), a software-controlled on-chip memory, has been used in embedded systems as an alternative to hardware-controlled cache. Compared to hardware-controlled cache, software-controlled SPMs present multiple advantages such as small die area, low energy consumption, and better timing predictability. Systems employing SPMs include Texas Instruments TMS370Cx [4], Motorola 68HC12 [5], and IBM's CELL processor. Different from cache, the data transfer between SPMs and off-chip memory is explicitly managed by software [6]. SPMs in multi-core systems can be accessed by both local core and remote cores. In order to alleviate data contention on an SPM unit, multi-port SPMs are employed in multi-core systems. With proper task and data assignment, compact schedules can be generated for multi-core systems with multi-port SPMs. In this paper, we study the problem of task and data assignment along with scheduling for multi-core systems with multi-port SPMs.

There have been research efforts on task assignment for multi-core systems [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17]. However, these studies target on different architectures from the one considered in this paper. Therefore, their techniques cannot produce optimal results for this paper's architecture.

Scheduling for multi-core systems is NP-Complete in general. In order to achieve optimal results for this architecture, in this paper, we propose an *integer linear programming (ILP) formulation* that determines task and data assignment and scheduling simultaneously to get an optimal schedule with minimum length. ILP formulation can guarantee optimal results. However, ILP takes exponential time to finish, which is not acceptable for large inputs. In order to solve large inputs efficiently, we propose a *polynomial-time near-optimal heuristic algorithm*, which consists of three phases. First, we determine task assignment by exploring opportunity of parallelism of data accesses and tasks. The first phase will generate a core-to-data matrix which describes the access frequency of each core to each data. In the second phase, according to the core-to-data matrix obtained from task assignment, we allocate each data to the core which minimizes the total memory access cost. In the third phase, we determine the start time of tasks and data accesses operations. To the best of our knowledge, the proposed techniques are the first methods that solve task and data assignment for multi-core systems with multi-port SPMs. The main contributions of this paper include:

- An ILP formulation is developed to generate optimal solution for the problem of task and data assignment and scheduling on multi-core systems with multi-port SPMs to achieve minimum schedule length.

- S. Gu and J. Yi are with the College of Computer Science, Chongqing University, Chongqing, China. E-mail: shannon1229.gu@gmail.com, 20101402118@cqu.edu.cn.
- Q. Zhuge and E.H.-M. Sha are with the College of Computer Science, Chongqing University and the Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing 400044, China. E-mail: {qfzhuge, edwinsha}@gmail.com.
- J. Hu is with the School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74078. E-mail: jthu@okstate.edu.

Manuscript received 13 Feb. 2014; revised 8 July 2014; accepted 18 Aug. 2014. Date of publication 7 Sept. 2014; date of current version 7 Aug. 2015.
Recommended for acceptance by S.-Q. Zheng.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2014.2356194

- We design a polynomial-time algorithm, the *task assignment with remote access reduced (TARAR) algorithm*, to determine task assignment by exploring parallelism of tasks and data accesses.
- We propose a dynamic programming algorithm, the *minimum memory access cost (MMAC) algorithm*, to generate a data assignment which minimizes the total memory access cost.

We implement the proposed techniques and evaluate them with benchmarks from Mediabench benchmarks. The effectiveness of the proposed techniques is evaluated by comparing schedule length generated by three different techniques: the high access frequency first (HAFF) algorithm proposed in [12], the ILP method, and the proposed heuristic algorithm (HA). The experimental results show that compared with HAFF algorithm the heuristic algorithm can achieve an improvement of 16.48 percent on average. The ILP can achieve 23.02 percent improvement on average.

The rest of the paper is organized as follows: related works are discussed in Section 2. In Section 3, hardware and software models are introduced. A motivational example is presented in Section 4 to illustrate the basic ideas. The ILP formulation is presented in Section 5. The heuristic algorithm is presented in Section 6. The experimental results are presented in Section 7. Section 8 concludes the whole paper.

2 Related Works

Multi-core systems have been in the marketplace for the past several years and are expected to be available in even greater variety in the future.

Huang and Xu [8] proposed an energy-efficient task allocation and scheduling for multi-mode multi-processor system-on-chip under lifetime reliability constraint. Huang et al. [9] proposed a novel customer-aware task allocation and scheduling technique for multi-mode multi-processor system-on-chip for lifetime reliability improvement and energy reduction. Again in [10], Huang and Xu proposed performance yield-driven task allocation and scheduling for multi-processor system-on-chip under process variation. Wang et al. [18] proposed a variation-aware task allocation and scheduling approach for multi-processor system-on-chip to mitigate the impact of parameter variations. All these techniques are used during the design stage. Huang et al. [19] presented a case study of designing multi-processor system-on-chip for Motion-JPEG and H.264 based on their Simulink tools. Demontes et al. [20] showed their study of moving an MPEG4 decoder from generic RISC platforms to CELL processor manually. Their work showed that an automatic task assignment and scheduling algorithm after the design stage is also of vital importance for users.

There have been research efforts on task and data assignment for multi-core system to improve performance [16], [13], [15], [21], [22], [23], [24], [25], [26], [27]. Xue et al. [16] introduced a data prefetching technique to hide the memory latency for off-chip memory accessing. Zhang et al. [12] proposed two variable partitioning heuristics based on an initial schedule and presented a loop pipeline scheduling algorithm to improve overall throughput. In [12], the virtually shared scratch-pad memory (VS-SPM) architecture has a single port SPM on each core. Hence, concurrent data accesses are not allowed on one SPM. Suhendra et al. [17] proposed an integer linear programming based technique which integrated task mapping, scheduling, SPM partitioning, and data allocation. Chang et al. [28], [29] explored how to assign a set of independent implicit-deadline sporadic tasks onto an island-based multi-core platform to meet the timing constraints with the considerations of different memory access latencies. Chen et al. [30] proposed a workload-aware task scheduling (WATS) scheme to improve the performance of parallel applications in asymmetric multi-core architectures. Saifullah et al. [31] addressed the hard real-time scheduling of a set of generalized DAGs sharing a multi-core machine. However, none of them applies the architecture of multi-core systems with multi-port SPM. Therefore, all the techniques above cannot apply on the architecture of this paper.

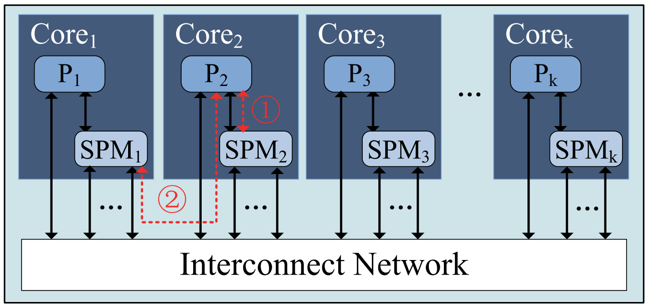


Fig. 1. Multi-core system with multi-port SPMs.

There are also research works on data assignment for single-core processor with SPM. Zhuge et al. [32] proposed an optimal data allocation for scalar variables on a single CPU with multiple memory types. Zhang et al. [33] proposed a dynamic programming approach to allocate the scalar and array variables on a single processor with multiple types of memory units. Neither of them targeted multiprocessor architecture.

Multi-bank and Multi-port memory provides high memory bandwidth which is required for data intensive applications [34], [35], [36]. Some previous works focused on multi-bank main memory. However, none of them focus on multi-port memories. Zhuge et al. [14] explored variable partitioning and scheduling on multiple-memory-module architectures and proposed a variable independence graph (VIG) model for approaching the variable partitioning problem. Qiu et al. [37] proposed an integer linear programming model to optimize the performance and energy consumption of multi-module memories by solving variable assignment, instruction scheduling and operating mode setting problems simultaneously. Their techniques cannot solve the problems of this paper on multi-core systems with multi-port SPM.

3 Basic Models and Problem Statement

In this section, we first introduce the target hardware architecture. Then, the *directed acyclic graph (DAG)* model which we use to model the input application is presented. At last, the formal definition of the problem is presented.

3.1 Target Hardware Architecture

The target hardware architecture is shown in Fig. 1. There are multiple cores on a single chip. Each core is equipped

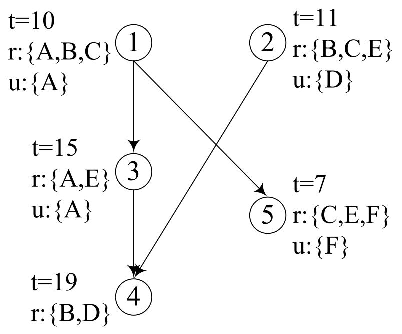


Fig. 2. The DAG model of motivational example.

with an SPM. The SPM serves as a local on-chip memory. Each SPM has multiple ports and can be concurrently accessed by multiple cores through interconnect network. Each core can access its local SPM and all the other SPMs. If a core accesses its own SPM, we call it a *Local Access*. The access latency on a local SPM is small. If a core accesses an SPM in other cores, we call it a *Remote Access*. It incurs a larger latency than local access. We assume that local access can overlap with remote access. In this work, we focus on latency of remote accesses during task assignment and scheduling. Thus, the data accesses refer to remote accesses in this paper. Please note that the SPMs used in the target hardware architectural are multi-port SPMs. Hence, multiple data accesses can be performed in parallel on each SPM. However, the number of ports of each SPM is limited by a constant.

Formally, the target hardware architecture T is a set of K cores $\{C_1, C_2, \dots, C_k\}$. Each core is equipped with an SPM. The number of ports of SPMs is denoted by MA. The size of SPM is denoted by $Msize$.

3.2 DAG Model

We use directed acyclic graph to model the tasks. The DAG is defined as follows:

Definition 3.1. *A DAG $G = <V, E, t, r, u>$ is a node-weighted directed graph, where V represents a set of tasks, $E \subseteq V \times V$ is a set of edges. Each edge $e(v_1, v_2) \in E$ represents precedence relation between nodes $v_1, v_2 \in V$. $t(v)$ represents the computation time of task $v \in V$. $r(v)$ is a set of data that task v needs request. $u(v)$ is a set of data that task v needs update during execution.*

In the algorithm, we need a table to reflect access frequency of each data by each task. We define the task-to-data access table (TDAT) here. The notation N_d denotes the set of data accessed by the application.

Definition 3.2 (Task-to-data access table). *A TDAT A is a table for which $a_{i,j}$ is a pair of non-negative integers $(r_{i,j}, u_{i,j})$, where $i \in [1, |V|]$ and $j \in [1, |N_d|]$. $r_{i,j}$ represents the frequency of the jth data requested by the ith task. $u_{i,j}$ represents the frequency of the jth data updated by the ith task.*

3.3 Problem Statement

The problem we are solving in this paper is defined as: given a DAG G, a TDAT A and target hardware architecture T, the goal of this paper is to find appropriate assignment for both tasks and data on a set of cores and SPMs such that the schedule length can be minimized. To achieve this, our proposed methods need to solve the following problems:

TABLE 1
The TDAT of Motivational Example

	A	B	C	D	E	F
	1	2	3	4	5	6
1	(1,1)	(1,0)	(3,0)	(0,0)	(0,0)	(0,0)
2	(0,0)	(3,0)	(2,0)	(0,2)	(3,0)	(0,0)
3	(1,1)	(0,0)	(0,0)	(0,0)	(2,0)	(0,0)
4	(0,0)	(2,0)	(0,0)	(1,0)	(0,0)	(0,0)
5	(0,0)	(0,0)	(1,0)	(0,0)	(1,0)	(2,2)

- *Task assignment*. A legal assignment for tasks should obey both precedence relations and resource constraints.
- *Data assignment*. Data is assigned to SPMs so that memory access cost can be minimized. We only keep one copy of data in the system.
- *Scheduling*. A schedule determines the start time of tasks and memory access operations.

4 Motivational Example

After presenting the basic model and problem, in this section, we illustrate the main techniques proposed in this paper with a motivational example.

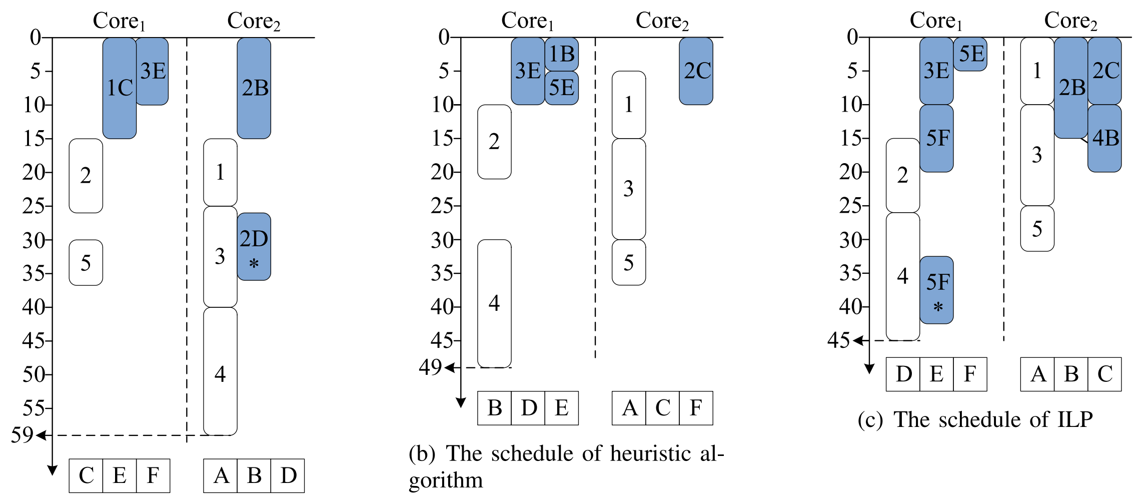
In this example, we assume the latency of a remote access L_r is five clock cycles. The target architecture for this example has two cores. Each core has one on-chip SPM of size 3 data units. The number of concurrent accesses allowed on each SPM is 2.

Fig. 2 shows the input DAG for the motivational example. Each node represents a task. "t" is the computation time of each task. For example, task 1 needs 10 clock cycles to complete. "r" stands for the set of data that a task needs to request and "u" stands for the set of data that a task needs to update. For example, task 1 request data A, B and C before computation, and update data A after computation.

We construct the task-to-data access table A for the motivational example. As shown in Table 1, the frequency of data "B" requested by task 2 is 3 and updated is 0. From the TDAT, we can get a global view of relations between tasks and data.

Fig. 3 shows three different schedules. The horizontal axis in Fig. 3 shows two different cores which are separated by dotted line. Each core has a 2-port SPM which can be accessed by at most two memory access operations concurrently. The vertical axis shows the schedule length in clock cycles. In the schedules, the white rectangles represent tasks and the blue rectangles represent request or update operations. We use notation "$*$" to distinguish request and update operations. For example, in Fig. 3a, the white rectangle "2" represents task 2, the blue rectangle "1C" represents a request operation of data "C" by task 1, and the blue rectangle "2D$*$" represents a update operation of data "D" by task 2. The data assignment of each schedule is shown below the schedule.

All the existing task and data assignment and scheduling techniques are designed for SPM that can only be exclusively accessed. If we use the *high access frequency first* technique proposed in [12] to schedule this example, we can get a schedule as shown in Fig. 3a. In Fig. 3a, tasks 2 and 5 and data C, E, and F are assigned to $Core_1$, the other tasks and data are assigned to $Core_2$. In this schedule, "1C", "2B",



(a) The schedule of HAFF

(b) The schedule of heuristic algorithm

(c) The schedule of ILP

Fig. 3. Schedules of different approaches.

"2D∗", and "3E" are remote accesses. According to the task-to-data access table A, the latency of remote accesses are 15, 15, 10, and 10 clock cycles respectively. The length of schedule equals the latency of "1C" plus the completion time of tasks 1, 3, and 4. The total length is 59 clock cycles.

The above schedule can be improved. In Fig. 3b, by assigning tasks 2 and 4 and data "D" to $Core_1$, the remote access "2D∗" can be eliminated. In this schedule, data B and E are also assigned to $Core_1$, which allows task 2 to start at the tenth clock cycle. Furthermore, by assigning data A and C to the $Core_2$ together with task 1, task 1 can start at the fifth clock cycle. The schedule length can be reduced from 59 to 49 clock cycles by above task and data assignment.

However, the second schedule is still not optimal. By switching the assignment of data B and F and proper scheduling of data accesses, we can further reduce the total scheduling length. Fig. 3c is an optimal schedule with 45 clock cycles.

From this example, we can see that with proper task and data assignment/scheduling the schedule length can be reduced from 59 clock cycles to 49 clock cycles and to minimum 45 clock cycles. The performance is improved by 23.7 percent. The optimal schedule shown in Fig. 3c can be obtained from the ILP proposed in Section 5. The near optimal schedule shown in Fig. 3b can be obtained by the heuristics proposed in Section 6 in polynomial time.

5 The ILP Model

For multi-core systems, scheduling is NP-Complete in general. In this section, we propose an ILP formulation to achieve optimal task and data assignment and scheduling for multi-core systems that allow concurrent data accesses on each SPM. First, the notations and theorems used to construct the ILP formulation is presented in Section 5.1. Then, the details of the ILP formulation is presented in Section 5.2.

5.1 Notations and Theorems

Before presenting ILP formulation, notations and theorems that the ILP bases on will be presented. The notations that will be used in the ILP formulation and their definitions are shown in Table 2.

In the ILP formulation, we will use the following two theorems to model two *if and only if* statements with linear constraints.

Theorem 5.1. *Let x be an integer variable whose value is between $-L$ and L, and y is a binary variable. The statement:"$x \le 0$ if and only if $y = 0$"can be modelled as the following linear form:*

$$-(L+1) + y \times (L+1) < x \le L \times y. \tag{1}$$

Proof. If $y = 0$, according to (1), x belongs to interval $(-L-1, 0]$, that is, x can be any integer from $-L$ to 0. Since x is an integer between $-L$ and L, $x \le 0$ is true.

TABLE 2
Notations Used in the ILP Formulations

Notation	Definition
$x_{u,s,k}$	$= 1$ if and only if a computation task or memory operation u starts to execute on core k at step s.
$y_{u,s,k}$	$= 1$ if and only if there is a computation task u is being executed on processor k at step s.
$z_{u,s,k}$	$= 1$ if and only if there is a memory access operation u is being executed on SPM k at step s.
$m_{d,k}$	$= 1$ if and only if data variable d is allocated to SPM k.
$\|V_1\|$	Number of computation tasks
$\|V_2\|$	Number of memory access operations
S	Upper bound of steps that can be evaluated from some basic algorithms.
K	Number of processors or SPMs
N_d	Number of data in G
$MSize_k$	Size of SPM k
MA	Number of ports of SPMs.
$Size_d$	Size of data d
L_r	Access latency of remote access
$t(u)$	The execution time of a node u
$var(u)$	The data that memory access operation u accesses
$con(u)$	The set of memory access operations which is request or update by task u.
$P(u)$	The which executes task u
$M(u)$	The SPM which is accessed by memory access operation u
$R(u)$	$= 1$ if and only if memory access operation u is a remote access.
$fr(u)$	The frequency of memory access operation u.

On the other hand, if $x \leq 0$, that is, x can be any integer from $-L$ to 0, then the left-hand side of Equation (1) must be less than $-L$ and the right-hand side must be greater than or equal to 0. We use contradiction to prove y must be 0.

If $y = 1$, then the left side of Equation (1) is 0. This contradicts that it must be less than $-L$. Thus, y must be 0. □

Theorem 5.2. *Let x be an integer variable whose value is between $-L$ and L, and y is a binary variable. The statement:"$x \geq 0$ if and only if $y = 0$"can be modeled as the following linear form:*

$$-L \times y \leq x < L + 1 - y \times (L + 1). \tag{2}$$

Proof. Variable x in Theorem 5.2 is just the opposite value in Theorem 5.1. Thus, we can apply Theorem 5.1 to prove Theorem 5.2 is true. □

Theorem 5.3. *Let w be an integer variable whose value is between $-L$ and L, and x, y and z are binary variables. Let $x = 0$ if and only if $w = 0$, $y = 0$ if and only if $w \leq 0$, and $z = 0$ if and only if $w \geq 0$. The statement:"$x = 0$ if and only if $y = z = 0$"can be modelled as the following linear form:*

$$x = y + z. \tag{3}$$

Proof. According to the definition of y and z, if $y = z = 0$, then w must be 0. Since $w = 0$ can infer $x = 0$, $x = 0$ is true.

According to the definition of x, if $x = 0$, then w must be 0. Since $w = 0$ can infer $y = 0$ and $z = 0$, $y = z = 0$ is true. □

Now that we have presented the notations and theorems which will be used in the ILP formulations, we are ready to construct the ILP formulations.

5.2 The ILP Formulations

For our problem, we wish to assign tasks and data and schedule computation tasks and memory access operations such that the schedule length is minimum. We will present all the constraints first. After all the constraints are presented, the objective function follows.

Task mapping constraint. Each computation task and memory access operation is executed exactly once at any time.

$$\sum_{s=1}^{S}\sum_{k=1}^{K} x_{u,s,k} = 1, \forall u \in V_1 \cup V_2. \tag{4}$$

Data allocation constraints. During data allocation, we need to consider three constraints. First, each data can be allocated to one SPM.

$$\sum_{k=1}^{K} m_{d,k} = 1, \forall d \in [1, N_d]. \tag{5}$$

Second, the total size of data allocated to an SPM should not exceed the size of SPM. It is formulated as follows:

$$\sum_{d=1}^{N_d} Size_d \times m_{d,k} \leq MSize_k, \forall k \in [1, K]. \tag{6}$$

Third, data should be allocated to the same core for corresponding memory access operations. Then we have

$$\sum_{k=1}^{K} k \times m_{var(u),k} = \sum_{s=1}^{S}\sum_{k=1}^{K} k \times x_{u,s,k}, \forall u \in V_2. \tag{7}$$

Memory access time. A memory access can be a local or a remote memory access depending on whether the task and accessed data are assigned to the same core. Since the costs for local access and remote access are different, we need to first find out the type of a memory access according to the task and data assignment. Then we can compute the memory access time for each memory access.

The core $P(u_1)$ which executes task u_1 can be $1, \ldots, K$.

$$P(u_1) = \sum_{s=1}^{S}\sum_{k=1}^{K} k \times x_{u_1,s,k}, \forall u_1 \in V_1. \tag{8}$$

The SPM $M(u_2)$ which is accessed by memory access operation u_2 can be SPM $1, \ldots, K$.

$$M(u_2) = \sum_{s=1}^{S}\sum_{k=1}^{K} k \times x_{u_2,s,k}, \forall u_2 \in V_2. \tag{9}$$

Then, $R(u)$ can be formulated as

$$R(u_2) = \begin{cases} 0, & \text{if } P(u_1) = M(u_2), \\ 1, & \text{others}, \end{cases} \quad \forall u_2 \in con(u_1).$$

Unfortunately, this equation is a non-linear condition. We need to transform this into a linear constraint. In order to modelled this type of non-linear condition with linear constraint, two auxiliary binary variables, $q(u)$ and $r(u)$, are used. Let $q(u_2) = 0$ if and only if $P(u_1) - M(u_2) \leq 0$. Let $r(u_2) = 0$ if and only if $P(u_1) - M(u_2) \geq 0$.

According to Theorem 5.1, the relation of $P(u_1) - M(u_2)$ and $q(u_2)$ can be represented by following linear constraint.

$$q(u_2) \times K - K < P(u_1) - M(u_2) \leq (K - 1) \times q(u_2). \tag{10}$$

According to Theorem 5.2, the relation of $P(u_1) - M(u_2)$ and $r(u_2)$ can be represented by following linear constraint.

$$(1 - K) \times r(u_2) \leq P(u_1) - M(u_2) < K - r(u_2) \times K. \tag{11}$$

From the definition of $q(u)$ and $r(u)$, we can infer that "$P(u_1) = M(u_2)$ if and only if $q(u_2) = r(u_2) = 0$". Then, $R(u_2)$ can be formulated as follows:

$$R(u_2) = q(u_2) + r(u_2). \tag{12}$$

Then, the execution time of memory access operation u_2 can be formulated as

$$t(u_2) = L_r \times fr(u_2) \times R(u_2), \forall u_2 \in con(u_1). \tag{13}$$

Dependency constraint. A dependency edge $e(u, v) \in E$ indicates that node v cannot be executed until node u is finished, where $u, v \in V_1 \cup V_2$.

$$\sum_{s=1}^{S}\sum_{k=1}^{K} s \times x_{u,s,k} + t(u) \leq \sum_{s=1}^{S}\sum_{k=1}^{K} s \times x_{v,s,k}, \quad \forall e(u, v) \in E, \forall u, v \in V_1 \cup V_2. \tag{14}$$

Resource constraints. There is only one computation task can be executed on core k at any time, as shown in Equation (15):

$$\sum_{u=1}^{|V_1|} y_{u,s,k} \le 1, \forall s \in [1,S], \forall k \in [1,K]. \tag{15}$$

There are up to MA concurrent remote accesses on an SPM at any time.

$$\sum_{u=1}^{|V_2|} z_{u,s,k} \le MA, \forall s \in [1,S], \forall k \in [1,K]. \tag{16}$$

Execution constraints. If a computation task u starts to execute on core k at step s, it will stay on core k at following $t(u)-1$ steps. The relationship between start time $x_{u,s,k}$ and execution time $y_{u,s,k}$ can be formulated as follows:

1) If $x_{u,s,k}=1$, then $\{y_{u,s,k}, y_{u,s+1,k}, \ldots, y_{u,s+t(u)-1,k}\}$ all equal to 1.

$$t(u) \times x_{u,s,k} \le \sum_{j=s}^{s+t(u)-1} y_{u,j,k}, \quad \forall u \in V_1, \forall s \in [1,S], \forall k \in [1,K]. \tag{17}$$

2) If $y_{u,s,k}=1$, then one of $\{x_{u,s-t(u)+1,k}, x_{u,s-t(u)+2,k}, \ldots, x_{u,s,k}\}$ must be 1.

$$y_{u,s,k} \le \sum_{j=s-t(u)+1}^{s} x_{u,j,k}, \quad \forall u \in V_1, \forall s \in [1,S], \forall k \in [1,K]. \tag{18}$$

Similar to computation task, if memory access operation starts to execute on SPM k at step s, then it will stay on SPM k at following $t(u)-1$ steps.

1) The execution time of memory access operation is $t(u)$. We can formulate that as

$$\sum_{k=1}^{K}\sum_{s=1}^{S} z_{u,s,k} = t(u), \forall u \in V_2. \tag{19}$$

2) If memory access operation u is remote access and $x_{u,s,k}=1$, then $\{z_{u,s,k}=1, z_{u,s+1,k}=1, \ldots, z_{u,s+L_r*fr(u)-1,k}\}$ all equal to 1.

$$L_r \times fr(u) \times (x_{u,s,k} + R(u)) - L_r \times fr(u) \le \sum_{j=s}^{s+L_r*fr(u)-1} z_{u,j,k}, \quad \forall u \in V_2, \forall s \in [1,S], \forall k \in [1,K]. \tag{20}$$

3) If $z_{u,s,k}=1$, then one of $\{x_{u,s-L_r*fr(u)+1,k}, x_{u,s-L_r*fr(u)r+2,k,\ldots,} x_{u,s,k}\}$ must be 1.

$$z_{u,s,k} \le \sum_{j=s-L_r*fr(u)+1}^{s} x_{u,j,k}, \quad \forall u \in V_2, \forall s \in [1,S], \forall k \in [1,K]. \tag{21}$$

Objective function. The goal of ILP model is finding the schedule whose length is minimum. The objective function can be formulated as follows:

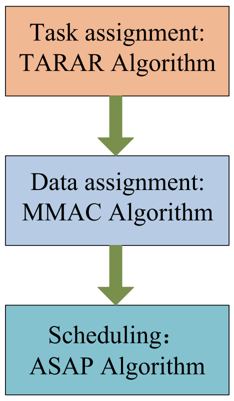


Fig. 4. Three steps in the proposed method.

$$\min \sum_{s=1}^{S}\sum_{k=1}^{K} s \times x_{u_0,s,k}. \tag{22}$$

where u_0 is a dummy node added to DAG to compute the finish time of the whole graph. We also added one outgoing edge from each node with no child to u_0.

6 Polynomial-Time Heuristic Method

ILP can always generate optimal solution. However, it takes exponential time to finish. Therefore, we need a polynomial-time solution for large inputs. In this section, we propose a heuristic method which consists of three phases to achieve a near-optimal solution in polynomial-time. First, the *task assignment with remote access reduced* algorithm generates task assignment. Then, a dynamic programming algorithm, the *minimum memory access cost* algorithm, generates data assignment with minimum memory access cost. Finally, according to the task and data assignment, the as soon as possible (ASAP) scheduling algorithm generates a near-optimal schedule. The three steps are shown in Fig. 4.

6.1 Task Assignment with Remote Access Reduced Algorithm

The goal of the TARAR algorithm is that a task is assigned to the core on which it can be finished earliest and the remote access is avoided as much as possible. The task assignment with remote access reduced algorithm is shown in Algorithm 6.1.

In Algorithm 6.1, a matrix $Freq[k][d]$ is used to keep track of the access information of each core k to each variable d. This matrix is initialized to 0 at the beginning. A priority queue will also be built for all tasks. First, we will get a task m from priority queue to assign to an available core n. When this task is assigned, the value of $TDAT[m][d]$ is added to $Freq[n][d]$ for every variable d that task m accesses. Therefore, matrix $Freq[k][d]$ will always keep track of the data access information for each core to each variable under current task assignment.

In the next step, we get an another task m' from priority queue to assign to an available core n'. In this step, we will compare the task data access information and core data access information in order to find a task to core assignment such that they share the most common accessed data. This is because that if they have the most common accessed data and we assign task m' to core n', it is more likely there will be less remote accesses and task m' can finish early. After we assign task m' to n', matrix $Freq[k][d]$ is updated to reflect the latest data access information for each core. After that, we continue to assign the next task.

Algorithm 6.1. Task Assignment with Remote Access Reduced Algorithm

```
Input: (1) A graph model DAG $G = <V, E, t, d, u>$; (2) A table
    TDAT $A$; (3) Architectural model $T$.
Output: A task assignment and the data access frequency
    matrix $Freq$.
1: Build a priority queue $Q$ of tasks.
2: Initialize $Available(k) \leftarrow 0$.
3: Initialize a $K \times N_d$ matrix $Freq[k][d] \leftarrow 0$.
4: while ($Q$) do
5:   $q \leftarrow$ Dequeue($Q$);
6:   $Est \leftarrow$ The earliest executable time of $q$;
7:   for core $k$ do
8:     $Min \leftarrow \infty$;
9:     $Num(k) \leftarrow \sum_{d \in D_q} Freq[k][d]$,
       where $D_q$ is the subset of data variables that task $q$
       needs access.
10:    $R_p(k) \leftarrow \sum_{d' \in D'_q} TDAT[q][d']$,
       where $D'_q$ is the subset of data variables that task $q$
       needs access and $Freq[k][d'] = 0, \forall d' \in D'_q$.
11:    $St(k) \leftarrow$ The earliest start time of $q$ on core $k$ according
       to $Available(k)$ and $Est$;
12:    if (($Min > St(k) + L_r \times R_p(k)/MA$) or ($Min = St(k) +
       L_r \times R_p(k)/MA$ and $Num(k) > Num(Core\_id)$)) then
13:      $Min \leftarrow St(k) + L_r \times R_p(k)/MA$;
14:      $Core\_id \leftarrow k$;
15:    end if
16:  end for
17:  $ass(q) \leftarrow Core\_id$;
18:  Update $Available(Core\_id)$ by removing the execution
     time slot of task $q$.
19:  Update $Freq$ by adding $TDAT[q][d]$ to $Freq[Core\_id][d]$ for
     every variable $d$ that task $q$ accesses.
20: end while
21: return a task assignment $ass$ and the data access frequency
    matrix $Freq$.
```

Initially, all the variables are located in main memory and there is no task assigned to any core. The algorithm builds a priority queue of tasks for a given DAG. It assigns priority to tasks based on the number of children nodes. If two tasks have same number of children nodes, the task who is in critical path has higher priority. For example, in the motivational example, tasks 2 and 3 have same number of children nodes. However, task 3 has a higher priority because it is in critical path. An array $Available(k)$ is used to record the available time slots on core k for task assignment.

In each iteration from line 4 to line 20, the algorithm tries to assign a task q to a core k. Variable Est records the earliest executable time of task q. It is the earliest finish time of ancestors of task q. From lines 7 to 16, the algorithm checks all the cores and finds the core on which task q can be finished earliest and the remote access is avoided as much as possible. Variable Min records the earliest finish time of task q. In line 9, variable $Num(k)$ records the total number of core k's accesses on the data that are commonly accessed by core k and task q. In line 10, variable $R_p(k)$ records the estimated number of remote accesses if task q is assigned to core k. For each variable d that task q accesses, if $Freq[k][d] = 0$, it means that core k does not access that data. We treat this access on variable d as remote access. For example, when the algorithm tries to assign task 3 to core 2, $Freq[2] = \{2, 1, 3, 0, 0, 0\}$. Since $TDAT[3] = \{2, 0, 0, 0, 2, 0\}$, core 2 and task 3 only commonly access the first data. Then $Num(2) = 2$ and $R_p(2) = 2$. $St(k)$ is the earliest executable time slots of $Available(k)$ which is greater than or equal to Est. The algorithm regard $St(k)$ as the start time of task q on core k.

From line 12 to line 15, we compare the finish time of task q on each core and update the value of Min with the earliest finish time. The finish time of task q on core k depends on the start time $St(k)$, estimated number of remote accesses $R_p(k)$, and the number of accesses on common accessed data $Num(k)$. We estimate the finish time with $St(k) + L_r \times R_p(k)/MA$, where $L_r \times R_p(k)/MA$ represents the latency of remote accesses. If more than one core have same finish time, we choose the core whose $Num(k)$ is maximum as the assignment of task q.

In line 17, the algorithm records the assignment of task q on array $ass(q)$. At following steps, we update $Available(Core\_id)$ by removing the execution time slot of task q and $Freq$ by adding $TDAT[q][d]$ to $Freq[Core\_id][d]$ for every variable d that task q accesses.

The complexity of the TARAR algorithm is $O(|V||K||N_d|)$ where $|V|$ is the number of tasks, $|K|$ is the number of cores and $|N_d|$ is the number of data.

$$A[j, i_1, i_2, \ldots, i_K] = \begin{cases} \sum_j Cost(m_0, d_j), & \text{if } i_k = Cap_k, \forall k \in \{1, 2, \ldots, K\}; \\ \infty, & \text{if } \sum_k i_k < \sum_k Cap_k - j \text{ or} \\ & \text{if } \exists k \in \{1, 2, \ldots, K\} i_k > Cap_k; \\ \min(A[j-1, i_1, i_2, \ldots, i_K], & \\ A[j-1, i_1+1, i_2, \ldots, i_K] & \\ \quad -(Cost(m_0, d_j) - Cost(m_1, d_j)), & \\ A[j-1, i_1, i_2+1, \ldots, i_K] & \\ \quad -(Cost(m_0, d_j) - Cost(m_2, d_j)), & \\ \quad \vdots & \\ A[j-1, i_1, i_2, \ldots, i_K+1] & \\ \quad -(Cost(m_0, d_j) - Cost(m_K, d_j)) & \text{if } \sum_k i_k \geq \sum_k Cap_k - j. \end{cases} \tag{23}$$

6.2 Minimum Memory Access Cost Algorithm

After tasks are assigned, we will assign data such that the memory access cost can be minimized. In this section, we first present some definitions. Some notations used in MMAC are shown in Table 3. Then, we formally define the problem of minimum memory access cost on multi-core systems with multi-port SPMs. After that, we propose the *minimum memory access cost* algorithm using dynamic programming approach.

Definition 6.1 (Data assignment function). *The data assignment of a data is defined as a function DA: $D \rightarrow M$. A function $DA(d_j) = m_i$ represents data $d_j \in D$ is assigned to memory $m_i \in M$.*

TABLE 3
Notations Used in the MMAC Algorithm

Notation	Definition
D	A set of data $\{d_1, d_2, \ldots, d_N\}$.
M	A set of memories $\{m_0, m_1, \ldots, m_K\}$, where m_0 is the main memory.
Cap_i	The capacity of a memory m_i
$C_{x,i}$	The cost of memory accesses that the xth core accesses the ith memory m_i, where $x \in \{1, 2, \ldots, K\}$.

Definition 6.2 (Cost of memory accesses for a data). *Let $cost(DA(d_j), d_j)$ be the cost of memory accesses for data d_j under data assignment $DA(d_j)$. It is defined as follows:*

$$cost(DA(d_j), d_j) = \sum_{x=1}^{K} Freq[x][j] \cdot C_{x,DA(d_j)}.$$

Now let us formally define the problem of minimum memory access cost on multi-core systems with multi-port SPMs.

Definition 6.3 (Minimum memory access cost problem). *Given a set of data D, a set of memories M, the capacity of memories Cap and data access frequency matrix $Freq$. The minimum memory access cost problem is to find a data allocation $DA(d_j)$ for each data d_j, such that $\sum cost(DA(d_j), d_j)$ is minimum.*

After all definitions are presented, we are ready to present the *minimum memory access cost* algorithm. It consists of two major steps. In the first step, the costs of memory accesses for each data are computed. In the second step, the optimal data assignment is generated by dynamic programming such that memory access cost is minimum.

6.2.1 Computing Cost of Memory Accesses

Given a set of data $D = \{d_1, d_2, \ldots, d_N\}$, a set of memories $M = \{m_0, m_1, \ldots, m_K\}$ and the data access frequency matrix $Freq$, the cost of memory accesses $cost(DA(d_j), d_j)$ is computed for each data d_j on all memories based on Definition 6.2.

6.2.2 Determining Optimal Data Assignment Using Dynamic Programming

After the cost of memory accesses are computed, we are ready to present the dynamic programming part.

Let $A[j, i_1, i_2, \ldots, i_K]$ be the minimum memory access cost when data d_j is allocated on a certain memory and there are i_1 unoccupied slots on m_1, i_2 unoccupied slots on m_2, etc. The first dimension is represented by a data d_j. For $x \le j$, the assignment of data d_x has been optimally determined. For $x > j$, the data d_x is in the main memory. Each one of other dimensions is represented by unoccupied space on a certain memory $m_i \in M$ except for m_0. We assume that memory m_0 is the main memory which is large enough to hold all data. Initially, the unoccupied space of memory m_i is equal to its capacity Cap_i.

The recursive function is shown in Equation (23). This equation has three parts. In the first part, if $i_k = Cap_k$, which means that all the data is in the main memory and all on-chip memories are empty. In this case, $A[j, i_1, i_2, \ldots, i_K]$ should be the summation of all $cost(m_0, d_j)$ since all the data is still in the main memory. In the second part, for any $i_k > Cap_k$, $A[j, i_1, i_2, \ldots, i_K]$ will be infinity since i_k cannot be larger than Cap_k. Also, if $\sum_k i_k < \sum_k Cap_k - j$, $A[j, i_1, i_2, \ldots, i_K]$ is set to be infinity since it is impossible. In the third part, if $\sum_k i_k \ge \sum_k Cap_k - j$, we set $A[j, i_1, i_2, \ldots, i_K]$ to be the minimum of following K values:

- $A[j-1, i_1, i_2, \ldots, i_K]$, which means data d_j is still in the main memory;
- $A[j-1, i_1+1, i_2, \ldots, i_K] - (Cost(m_0, d_j) - Cost(m_1, d_j))$, which means data d_j is moved into memory m_1;

  $\vdots$
- $A[j-1, i_1, i_2, \ldots, i_K+1] - (Cost(m_0, d_j) - Cost(m_K, d_j))$, which means data d_j is moved into memory m_K.

Algorithm 6.2. Minimum Memory Access Cost Algorithm

```
Input:(1) A set of data $D = \{d_1, d_2, \ldots, d_N\}$; (2) A set of memories $M = \{m_0, m_1, \ldots, m_K\}$ and the array of capacity $Cap$; (3) a $K \times N$ matrix $Freq[k][d]$ records data access frequency of each core to each data.
Output: Data assignment results with minimum memory access cost.
1:  Compute the cost function $cost(DA(d_j), d_j)$ for each data $d_j$ on all memories.
2:  Initial array $A[N, Cap_1, Cap_2, \ldots, Cap_K] \leftarrow \infty$.
3:  for $j \leftarrow 1$ to $N$ do
4:    $A[j, Cap_1, Cap_2, \ldots, Cap_K] \leftarrow \sum_{d_j \in D} cost(DA(d_j), d_j)$.
5:  end for
6:  for $j \leftarrow 1$ to $N$ do
7:    for $i_1 \leftarrow Cap_1$ to 0 do
8:      for $i_2 \leftarrow Cap_2$ to 0 do
9:        $\vdots$
10:         for $i_K \leftarrow Cap_K$ to 0 do
11:           Update $A[j, i_1, i_2, \ldots, i_K]$ following Equation (23).
12:           $R \leftarrow Record[j, i_1, i_2, \ldots, i_K]$.
13:           if $A[j-1, i_1, i_2, \ldots, i_K]$ is selected then
14:             $R \leftarrow m_0$.
15:           else if $A[j-1, i_1+1, i_2, \ldots, i_K]$ is selected then
16:             $R \leftarrow m_1$.
17:           else if $A[j-1, i_1, i_2+1, \ldots, i_K]$ is selected then
18:             $R \leftarrow m_2$.
19:               $\vdots$
20:           else if $A[j-1, i_1, i_2, \ldots, i_K+1]$ is selected then
21:             $R \leftarrow m_K$.
22:           end if
23:           $Record[j, i_1, i_2, \ldots, i_K] \leftarrow R$.
24:         end for
25:           $\vdots$
26:       end for
27:     end for
28: end for
29: Find the minimal value in array $A$.
30:   return a map of data assignment by tracing back array $Record$.
```

According to the recursive function, the MMAC algorithm is presented in Algorithm 6.2. In line 1, the cost of memory accesses $cost(DA(d_j), d_j)$ is computed for each data d_j on all memories based on Definition 6.2. In line 2, $A[N, Cap_1, Cap_2, \ldots, Cap_K]$ is initialized to infinity according to the second part of Equation (23). From line 3 to line 5, $A[j, Cap_1, Cap_2, \ldots, Cap_K]$ is set according to the first part of Equation (23). From line 6 to line 28, we compute $A[j, i_1, i_2, \ldots, i_K]$ according to the third part of Equation (23). In line 23, we use an array $Record[j, i_1, i_2, \ldots, i_K]$ record the assignment of data d_j. Using $Record[j, i_1, i_2, \ldots, i_K]$, optimal solutions can be easily constructed. The time complexity of the MMAC algorithm is $O(N \times S^K)$, where N is the number of data, S is the upper bound of memory capacity and K is the number of SPMs.

6.3 As Soon As Possible Algorithm

Now, we have obtained the task and data assignment with TARAR algorithm and MMAC algorithm. In order to get a near-optimal schedule, we employ ASAP scheduling algorithm to decide the start time of tasks and memory access operations. The as soon as possible Algorithm is shown in Algorithm 6.3.

Algorithm 6.3. As Soon As Possible Algorithm

```
Input: (1) A task assignment $ass$; (2) A data assignment $DA$; (3)
    Architectural model $T$: the number of cores $K$, the number
    of concurrent accesses $MA$; (4) A graph model DAG
    $G = <V, E, t, d, u>$ and a TDAT $A$.
Output: A near-optimal schedule $Sch$.
 1: Generate a DAG $G'$ which represents the relationship of
    operations including tasks and remote accesses;
 2: Built a priority queue $Q$ of tasks and data accesses according to $G'$;
 3: Initialize $idle_p(k)$ and $idle_s(k, i)$ to record the idle slots of
    processors and SPMs, where $k \in [1, K]$ and $i \in [1, MA]$;
 4: while Q do
 5: $q \leftarrow$ Dequeue($Q$);
 6: Find the executable resources $e(q)$ of $q$ according to $ass$ or
    $DA$;
 7: $Sch(q) \leftarrow$ Find the earliest start time according to $idle_p(e(q))$
    or $idle_s(e(q), i)$;
 8: Update $idle_p(k)$ or $idle_s(k, i)$ according to $Sch(q)$ and $t(q)$;
 9: end while
10: return a near-optimal schedule $Sch$.
```

In order to deciding the start time of tasks and memory access operations, the relationship of tasks and memory access operations is necessary. In line 1, we generate a new DAG $G' = <V_1, V_2, E, t, d>$ according to graph model G, TDAT A, task assignment ass, and data assignment DA. Graph G' represents the relationship of tasks and memory access operations.

Definition 6.4. *A DAG $G' = <V_1, V_2, E, t, d>$ is a node-weighted directed graph, where V_1 represents a set of tasks, V_2 represents a set of remote accesses. $E \subseteq V \times V$ is a set of edges where $V = V_1 \cup V_2$. Each edge $e(v, u) \in E$ represents precedence relation between nodes $v, u \in V$. $t(v)$ represents the execution time of node $v \in V$. $d(v')$ is the data that remote access $v' \in V_2$ needs request or update.*

For motivational example, task assignment is $ass = \{2, 1, 2, 1, 2\}$ and data assignment is $DA = \{2, 1, 2, 1, 1, 2\}$. According to graph model G, task assignment ass, and data assignment DA, there are 4 remote accesses: requesting data B for task 1, requesting data C for task 2, requesting data E for task 3 and task 5.

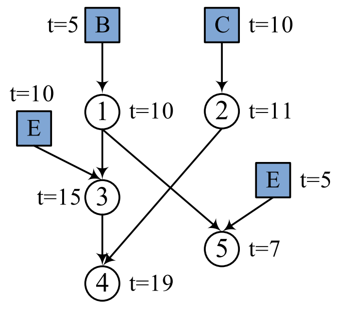


Fig. 5. The DAG model of motivational example.

The new graph G' of motivational example is shown in Fig. 5. The white nodes represent tasks and the blue nodes represent remote accesses. For a task, the execution time t can be obtained from Fig. 2. For a remote access, the execution time t is the product of access frequency fr and access latency L_r. For example, as shown in Table 1, $fr = 2 + 0 = 2$ according to $A(2, C) = (2, 0)$. The product of fr and L_r is 10, such that the execution time of requesting C for task 2 is 10 clock cycles.

In line 2, we build a priority queue Q of nodes in G'. It assigns priority to nodes based on the number of children nodes. If two nodes have same number of children nodes, the node that is on the critical path has higher priority.

In line 3, we initialize an array $idle_p(k)$ and a matrix $idle_s(k, i)$ according to architecture model T. Array $idle_p(k)$ is used to record idle slots of processors k. And matrix $idle_s(k, i)$ is used to record idle slots of i th port of SPM k.

In each iteration from line 4 to line 9, the algorithm tries to decide the start time of node q. For a task, we first find the executable processor $e(q)$ according to task assignment ass; Then, we find the earliest start time $Sch(q)$ of task q according to $idle_p(e(q))$; Finally, we update array $idle_p(k)$ according start time $Sch(q)$ and execution time $t(q)$. For a remote access, we first find the executable processor $e(q)$ according to task assignment ass and data assignment DA; Then, we find the earliest start time $Sch(q)$ of remote access q according to $idle_s(e(q), i)$; Finally, we update array $idle_s(k, i)$ according start time $Sch(q)$ and execution time $t(q)$. In line 10, we use Sch to record the start time of nodes in G'.

The time complexity of ASAP algorithm is $O(N \times |V_1| + |V|)$, where N is the number of data, $|V_1|$ is the number of tasks and $|V|$ is the number of nodes in G'.

7 Experiments

In this section, we present the experimental results of ILP and the proposed heuristic algorithm, composed of the TARAR algorithm, MMAC algorithm and ASAP algorithm. The experimental set-up is first presented in Section 7.1. Then the experimental results is presented in Section 7.2. The experimental results show that the proposed techniques can significantly improve the system performance compared with existing technique.

7.1 Experimental Setup

We developed a custom simulator to conduct the experiments. In the experiments, two set of experiments are done.

TABLE 4
Characteristics of the Benchmarks

Benchmark	#tasks	#var	Benchmark	#tasks	#var
adpcm	6	6	**pegunit**	34	25
ghostscript	14	9	**rasta**	47	25
epic	20	15	**pgp**	48	25
gsm	23	10	**mpeg2**	64	23
jpeg	33	22	**mesa**	68	20

In the first set, the targeting architecture is a eight-core processor. Each core is equipped with a four-port SPM. The other set experiments are done in a four-core processor and each core is equipped with a two-port SPM.

The benchmarks are chosen from the Mediabench benchmarks including Tree and DFG. Characteristics of the benchmarks are shown in Table 4. The #**tasks** column represents the number of tasks in benchmarks. The #**var** column represents the number of variables in benchmarks.

For the ILP experiments, Lingo is used as solver tool. Not all benchmarks can be solved optimally. We terminate the ILP solver after 12 hours and used the best results that solver had. For the heuristic algorithm experiments, all benchmarks can finish in several seconds.

7.2 Experimental Results and Analysis

In this paper, we measure the schedule length by clock cycle. The three different approaches are: (1) HAFF; (2) ILP; (3) Heuristic algorithm consists of *task assignment with remote access reduced* algorithm, *minimum memory access cost* algorithm and the *ASAP* algorithm. Among three approaches, HAFF, proposed in [12], is adapted to solve the multi-core with multi-port SPMs problem in the experiments.

In Table 5, the experimental results on 8-core system with 4-port SPMs are shown. The "SL" column represents the schedule length of different approaches. The "Imprv" column represents the schedule length improvement of ILP and HA over HAFF. ILP is able to produce the optimal schedule length for small benchmarks. However, for most of the benchmarks, ILP cannot find optimal solutions in several hours. Results, marked with "$*$", are best solutions we can find in several hours. Whereas, heuristic algorithm can solve all of them in several seconds.

TABLE 5
Schedule Lengths on a Eight-Core System with Four-Port SPMs

	HAFF	ILP		HA	
Benchmark	SL	SL	Imprv	SL	Imprv
adpcm	64	51	20.31%	51	20.31%
ghostscript	72	55*	23.61%	61	15.28%
epic	77	58*	26.68%	63	18.18%
gsm	71	63*	11.27%	66	7.04%
jpeg	104	73*	29.81%	86	17.31%
pegunit	125	99*	20.80%	113	9.60%
rasta	141	118*	16.31%	129	8.51%
pgp	185	127*	31.35%	130	29.73%
mpeg2	199	145*	27.13%	154	22.61%
mesa	179	138*	22.91%	150	16.20%
Average			23.02 %		16.48%

TABLE 6
Comparison of Number of Remote Accesses on a Eight-Core System with Four-Port SPMs

	HAFF	ILP		HA	
Benchmark	#R	#R	Imprv	#R	Imprv
adpcm	17	11	35.29%	9	47.06%
ghostscript	55	68	−23.64%	41	25.45%
epic	37	29	21.62%	32	13.51%
gsm	69	88	−27.54%	47	31.88%
jpeg	105	73	30.48%	81	22.86%
pegunit	128	103	19.53%	98	23.44%
rasta	168	127	24.40%	117	30.36%
pgp	163	202	−23.93%	130	20.25%
mpeg2	203	156	23.15%	145	28.57%
mesa	238	284	−19.33%	182	23.53%
Average			6.01%		26.69%

On this architecture, our proposed approaches are able to utilize four parallel data accesses as possible and generate compact schedules. The average improvement of ILP and HA over HAFF is 23.02 and 16.48 percent. In the best case, HA reduces the schedule length by 29.73 percent for the "pgp". By applying our proposed techniques, the performance can be improved significantly.

Table 6 shows the comparison of number of remote accesses on a eight-core system with four-port SPMs in each loop. The "#**R**" column represents the number of remote accesses. We can see that the heuristic algorithm can reduce the number of remote accesses by 26.69 percent on average. For the "adpcm", ILP and HA can generate optimal schedules whose length are both 51, shown in Table 5. However, ILP generates 11 remote accesses and HA generates nine remote accesses. This because HA employs the *minimum memory access cost* algorithm, so that it can obtain the schedule with minimum remote accesses. For the "ghostscript", ILP generates a shorter schedule whose length is 55, shown in Table 5. However, it has 68 remote accesses which is more than the number of remote accesses of HAFF and HA. Since task and data assignment are interrelated to each other and significantly affect remote accesses, ILP assigns tasks and data simultaneously to obtain an optimal schedule, irrespective of remote accesses. HAFF algorithm assigns data after task assignment. It employs a greedy strategy in which high access frequency is assigned first to generate a schedule with fewer remote accesses. Therefore, the number of remote accesses generated by ILP may be more than HAFF algorithm.

Table 7 shows the experimental results of scheduling length on 4-core system with 2-port SPMs. On this architecture, our proposed approaches are able to utilize two parallel data accesses as possible and generate compact schedules. The average improvement of ILP and HA over HAFF is 22.75 and 14.06 percent. In the best case, HA reduces the schedule length by 22.46 percent for the "pgp".

Table 8 shows the comparison of number of remote accesses on a 4-core system with 2-port SPMs in each loop. From the table, we can see that the heuristic algorithm can reduce the number of remote accesses by 23.63 percent on average.

TABLE 7
Schedule Lengths on a Four-Core System with Two-Port SPMs

	HAFF	ILP		HA	
Benchmark	**SL**	**SL**	**Imprv**	**SL**	**Imprv**
adpcm	68	51	25.00%	56	17.65%
ghostscript	94	69*	26.59%	82	12.77%
epic	115	89*	22.61%	102	11.30%
gsm	128	92*	28.13%	108	15.63%
jpeg	153	133*	13.07%	140	8.50%
pegunit	211	161*	23.70%	183	13.27%
rasta	256	214*	16.41%	231	9.77%
pgp	276	197*	28.62%	214	22.46%
mpeg2	315	246*	21.90%	269	14.60%
mesa	321	252*	21.50%	274	14.64%
Average			22.75%		14.06%

TABLE 8
Comparison of Number of Remote Accesses on a Four-Core System with Two-Port SPMs

	HAFF	ILP		HA	
Benchmark	**#R**	**#R**	**Imprv**	**#R**	**Imprv**
adpcm	13	13	0%	9	30.08%
ghostscript	39	50	−28.21%	30	23.08%
epic	42	31	26.19%	36	14.29%
gsm	49	61	−24.49%	33	32.65%
jpeg	68	53	22.06%	48	29.41%
pegunit	97	73	24.74%	69	28.87%
rasta	128	107	16.41%	93	27.34%
pgp	131	158	−20.61%	114	12.98%
mpeg2	166	133	19.88%	128	22.89%
mesa	197	159	19.29%	168	14.72%
Average			5.53%		23.63%

From Tables 6 and 8, we can see that the number of remote accesses generated by HA is minimum among three methods. On multi-core systems, memory access contributes major energy consumption. In target architectures, remote access is more time-consuming and energy-consuming than local access. Therefore, a schedule with less remote accesses is efficient in energy consumption. HA adopts the *Minimum memory access cost* algorithm to generate a data assignment with minimum remote accesses. Compared with HAFF algorithm, HA can reduce the number of remote accesses by 26.69 percent on average on an eight-core system with four-port SPMs in each loop. On a four-core system with two-port SPMs, HA can reduce the number of remote accesses by 23.63 percent on average in each loop. Therefore, HA is more efficient in energy consumption.

Compared with the HAFF algorithm, the heuristic algorithm is more effective on task and data assignment and scheduling for multi-core with multi-port SPMs. The reason is that the TARAR algorithm considers effects of both task assignment and data assignment at the same time. It tries to maximize the parallelism of tasks and data accesses by considering the cost of data accesses on remote SPM. The MMAC algorithm can achieve the minimum memory access cost, which makes the schedule more compact. The experimental results also show that the heuristic algorithm is more time-efficient than the ILP method.

8 Conclusion

In this paper, we study the problem of task and data assignment and scheduling on multi-core systems with multi-port SPMs. We build ILP formulation to obtain the optimal schedule length. We also propose efficient heuristic algorithm composed of *task assignment with remote access reduced* algorithm and *minimum memory access cost* algorithm to generate near-optimal results in polynomial time. The experimental results show that the heuristic algorithm are more effective than the HAFF algorithm in exploring parallelism in simulated multi-core system architectures and generating compact schedules.

Acknowledgments

This work was partially supported by the National 863 Program 2013AA013202, Chongqing High-Tech Research Program csct2012ggC40005, NSFC 61173014, and NSFC 61472052. Edwin Hsing-Mean Sha is the corresponding author.

References

[1] W. Wolf, A. A. Jerraya, and G. Martin, “Multiprocessor system-on-chip (MPSoC) technology,” *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 27, no. 10, pp. 1701–1713, Oct. 2008.
[2] Texas Instruments. (2013). *66AK2Hx keystone multicore DSP+ARM* [Online]. Available: http://www.ti.com/lit/ds/sprs866a/sprs866a.pdf
[3] Texas Instruments. (2012). *TMS320C6678 multicore fixed and floating-point system-on-chip* [Online]. Available: http://www.ti.com/lit/ds/symlink/tms320c6678.pdf
[4] Texas Instruments. (1997). *Tms370cx7x 8-bit microcontroller* [Online]. Available: http://www-s.ti.com/sc/psheets/spns034c/spns034c.pdf
[5] Microcontrollers. (2000). *Cpu12 reference manual* [Online]. Available: http://e-www.motorola.com/brdata/PDFDB/MICROCONTROLLERS/16BIT/68HC12FAMILY/REFMAT/CPU12RM.pdf
[6] P. R. Pandaand, 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, 2000.
[7] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang, “Mapping of applications to MPSoCs,” in *Proc. 7th IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2011, pp. 109–118.
[8] L. Huang and Q. Xu, “Energy-efficient task allocation and scheduling for multi-mode MPSoCs under lifetime reliability constraint,” in *Proc. Des., Autom. Test Eur. Conf. Exhib.*, Mar. 2010, pp. 1584–1589.
[9] L. Huang, R. Ye, and Q. Xu, “Customer-aware task allocation and scheduling for multi-mode MPSoCs,” in *Proc. 48th Des. Autom. Conf.*, 2011, pp. 387–392.
[10] L. Huang and Q. Xu, “Performance yield-driven task allocation and scheduling for MPSoCs under process variation,” in *Proc. 47th ACM/IEEE Des. Autom. Conf.*, Jun. 2010, pp. 326–331.
[11] C. Baloukas, L. Papadopoulos, D. Soudris, S. Stuijk, O. Jovanovic, F. Schmoll, D. Cordes, R. Pyka, A. Mallik, S. Mamagkakis, F. Capman, S. Collet, N. Mitas, and D. Kritharidis, “Mapping embedded applications on MPSoCs: The MNEMEE approach,” in *Proc. IEEE Comput. Soc. Annu. Symp. Very Large Scale Integr.*, Jul. 2010, pp. 512–517.
[12] L. Zhang, M. Qiu, W.-C. Tseng, and E. H.-M. Sha, “Variable partitioning and scheduling for MPSoC with virtually shared scratch pad memory,” *J. Signal Process. Syst.*, vol. 58, pp. 247–265, 2010.
[13] O. Ozturk, G. Chen, M. Kandemir, and M. Karakoy, “An integer linear programming based approach to simultaneous memory space partitioning and data allocation for chip multiprocessors,” in *Proc. IEEE Comput. Soc. Annu. Symp. Emerging Very Large Scale Integr. Technol. Archit.*, 2006, p. 50.
[14] Q. Zhuge, E. H.-M. Sha, B. Xiao, and C. Chantrapornchai, “Efficient variable partitioning and scheduling for DSP processors with multiple memory modules,” *IEEE Trans. Signal Process.*, vol. 52, no. 4, pp. 1090–1099, Apr. 2004.

[15] Y. Guo, Q. Zhuge, J. Hu, M. Qiu, and E. H.-M. Sha, "Optimal data allocation for scratch-pad memory on embedded multi-core systems," in *Proc. Int. Conf. Parallel Process.*, 2011, pp. 401–410.

[16] C. Xue, Z. Shao, M. Liu, M. Qiu, and E.-M. Sha, "Loop scheduling with complete memory latency hiding on multi-core architecture," in *Proc. Int. Conf. Parallel Distrib. Syst.*, 2006, pp. 277–286.

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

[18] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan, "Variation-aware task allocation and scheduling for MPSoC," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2007, pp. 598–603.

[19] K. Huang, S.-il Han, K. Popovici, L. Brisolara, X. Guerin, L. Li, X. Yan, S.-I. Chae, L. Carro, and A. Jerraya, "Simulink-based MPSoC design flow: Case study of motion-JPEG and H.264," in *Proc. 44th ACM/IEEE Des. Autom. Conf.*, 2007, pp. 39–42.

[20] L. Demontes, M. Bonaciu, and P. Amblard, "Software for multi processor system on chip: Moving an MPEG4 decoder from generic RISC platforms to CELL," in *Proc. 19th IEEE/IFIP Int. Symp. Rapid Syst. Prototyping*, 2008, pp. 34–40.

[21] P. Agrawal, K. Sugand, M. Palkovic, P. Raghavan, L. V. der Perre, and F. Catthoor, "Partitioning and assignment exploration for multiple modes of IEEE 802.11n modem on heterogeneous MPSoC platforms," in *Proc. 15th Euromicro Conf. Digital Syst. Des.*, 2012, pp. 608–615.

[22] Z. Shao, Q. Zhuge, C. Xue, and E. H.-M. Sha, "Efficient assignment and scheduling for heterogeneous DSP systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 6, pp. 516–525, Jun. 2005.

[23] O. Avissar, R. Barua, and D. Stewart, "An optimal memory allocation scheme for scratch-pad-based embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 1, pp. 6–26, 2002.

[24] J. Hu, C. J. Xue, W.-C. Tseng, M. Qiu, Y. Zhao, and E. H.-M. Sha, "Minimizing memory access schedule for memories," in *Proc. 15th Int. Conf. Parallel Distrib. Syst.*, 2009, pp. 104–111.

[25] S. Gilani, N. Kim, and M. Schulte, "Scratchpad memory optimizations for digital signal processing applications," in *Proc. Des., Autom. Test Eur. Conf. Exhib.*, 2011, pp. 1–6.

[26] S. Gu, Q. Zhuge, J. Hu, J. Yi, and E. H.-M. Sha, "Efficient task assignment and scheduling for MPSoC DSPS with VS-SPM considering concurrent accesses through data allocation," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2013, pp. 2615–2619.

[27] Y. Guo, Q. Zhuge, J. Hu, J. Yi, M. Qiu, and E. H.-M. Sha, "Data placement and duplication for embedded multicore systems with scratch pad memory," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, 2013, vol. 32, pp. 809–817.

[28] C.-W. Chang, J.-J. Chen, T.-W. Guo, and H. Falk, "Real-time partitioned scheduling on multi-core systems with local and global memories," in *Proc. ACM/IEEE Asia South Pac. Des. Autom. Conf.*, 2013, pp. 467–472.

[29] C.-W. Chang, J.-J. Chen, T.-W. Guo, and H. Half, "Real-time task scheduling on island-based multi-core platforms," *IEEE Trans. Parallel Distrib. Syst.*, 2013.

[30] Q. Chen, Y. Chen, Z. Huang, and M. Guo, "WATS: Workload-aware task scheduling in asymmetric multi-core architectures," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp.*, 2012, pp. 249–260.

[31] J. Li, D. Ferry, C. Gill, C. Lu, and K. Agrawal, "Parallel real-time scheduling of DAGs," *IEEE Trans. Parallel Distrib. Syst.*, 2013.

[32] Q. Zhuge, Y. Guo, J. Hu, W.-C. Tseng, C. J. Xue, and E. H.-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.

[33] J. Zhang, T. Deng, Q. Gao, Q. Zhuge, and E. H.-M. Sha, "Optimizing data allocation for loops on embedded systems with scratch-pad memory," in *Proc. IEEE 18th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2012, pp. 184–191.

[34] R. Leupers and D. Kotte, "Variable partitioning for dual memory bank DSPs," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 2001, vol. 2, pp. 1121–1124.

[35] C. J. Xue, T. Liu, Z. Shao, J. Hu, Z. Jia, W. Jia, and E. H.-M. Sha, "Address assignment sensitive variable partitioning and scheduling for DSPs with multiple memory banks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2008, pp. 1453–1456.

[36] M. Qiu, M. Guo, M. Liu, C. J. Xue, L. T. Yang, and E. H.-M. Sha, "Loop scheduling and bank type assignment for heterogeneous multi-bank memory," *J. Parallel Distrib. Comput.*, vol. 69, pp. 546–558, 2009.

[37] M. Qiu, L. Zhang, and E. H.-M. Sha, "ILP optimal scheduling for multi-module memory," in *Proc. 7th IEEE/ACM Int. Conf. Hardware/Softw. Codesign Syst. Synthesis*, 2009, pp. 277–286.

Shouzhen Gu received the BS degree in computer science from Chongqing University in 2010. She is currently working toward the PhD degree at the College of Computer Science, Chongqing University, Chongqing, China. Her research interests include scheduling and data allocation on MPSoC, high performance and low power embedded systems, and non-volatile memory.

Qingfeng Zhuge received the BS and MS degrees in electronics engineering from Fudan University, Shanghai, China and the PhD degree from the Department of Computer Science at the University of Texas at Dallas, in 2003. She is currently a professor at Chongqing University, China. She received the Best PhD Dissertation Award in 2003. She has published more than 90 research articles in premier journals and conferences. Her research interests include parallel architectures, embedded systems, supply-chain management, real-time systems, optimization algorithms, compilers, and scheduling.

Juan Yi received the BE degree from the School of Software Engineering at Chongqing University, Chongqing, China, in 2010 and is currently working toward the PhD degree from the Department of Computer Science at the same university. Her current research interests include temperature modeling and optimization, and reliability analysis of multicore systems.

Jingtong Hu (S'09–M'13) received the BE degree from the School of Computer Science and Technology, Shandong University, China, in 2007 and the MS and PhD degree in computer science from the University of Texas at Dallas, in June 2010 and August 2013, respectively. He is now an assistant professor in the School of Electrical and Computer Engineering at Oklahoma State University. His research interests include high performance and low power embedded systems, wireless sensor network, and non-volatile memory. He is a member of the IEEE.

Edwin Hsing-Mean Sha (S'88–M'92–SM'04) received the PhD degree from the Department of Computer Science, Princeton University, USA, in 1992. From August 1992 to August 2000, he was with the Department of Computer Science and Engineering at University of Notre Dame, USA. Since 2000, he has been a tenured full professor at the University of Texas at Dallas. Since 2012, he served as the dean of College of Computer Science at Chongqing University, China. He has published more than 300 research papers in refereed conferences and journals. His work has been cited over 2,200 times. He received the Teaching Award, Microsoft Trustworthy Computing Curriculum Award, NSF CAREER Award, and NSFC Overseas Distinguished Young Scholar Award, Chang-Jiang Honorary Chair Professorship and China Thousand-Talent Program. He is a senior member of the IEEE.