Power constraint communication-aware task scheduling in reconfigurable multiprocessors

Xiaoming Chen, Yan Liu*, Renfa Li

College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, Hunan, China

Received 1 June 2014, www.cmnt.lv

Abstract

Heterogeneous multiprocessors with FPGA component have recently received a lot of attention due to its low cost and power consumption. However, most of existing works about task scheduling algorithm focus on minimization of system cost or power consumption. Actually, optimizing multiprocessor performance within a given power budget has recently received a lot of attention. Peak power consumption should be carefully controlled than directly improve computing performance. Furthermore, FPGA component in multiprocessors has essential parallelism ability to execute multiple tasks at same time using dynamic reconfigurable features. In this environment, tasks and communications should be carefully scheduled because their execution orders affect the performance of the whole chip. This paper presents an Integer Linear Programming (ILP) formulation that integrates the resource delay model and FPGA-component with pipelined scheduling and global power control. Moreover, to enhance the computation efficiency, a heuristic algorithm namely PCLS that integrates pipelined scheduling and global power control for heterogeneous multiprocessor architecture is proposed. Experiments show that our ILP method obtains the optimal results when task nodes are less than 35. Proposed PCLS heuristic algorithm achieves on average 10% higher makespan compare with DLS. For heavier synthetic task application, PCLS can provide only about 12% performance degradation under 70% power budgets based on different heterogeneous multiprocessor architectures.

Keywords: multiprocessors, task scheduling, system-on-chip, power control

1 Introduction

Technology scaling continues to support more transistors be integrated into a chip and a typical multiprocessor chip consist of many type of components such as general-purpose CPU, DSP, FPGA and communication bus. An important trend in embedded system is the use of multiprocessor architectures to meet requirements of applications, such as multiprocessor system-on-chip has the potential ability to provide some advantages related to system cost and power. Obtain these improvement depends on designer make applications match with the flexible components and configurability features provided by the multiprocessor platform such as FPGA module. Generally, as the system becomes larger and complicated, the performance of the entire system is affected by the execution order of tasks and communications known as task mapping and scheduling problem.

Heterogeneous multiprocessor chips have potential to obtain better area to performance ratio, high throughput and high speed up. Exploiting inter-core heterogeneity is challenging as it boils down to mapping tasks to most appropriate cores and scheduling well suite task start time. There are some challenges must be faced when solving task scheduling problem on multiprocessor chips. Firstly, only a subset of the total available cores used to execute specific applications and the total number of cores maybe large and heterogeneity including many kinds of computing components and resources. Furthermore, hardware-related component such as FPGA can provide essential ability to execute application parallel. Secondly, designer needs to solve the task scheduling problem by determining the execution order of tasks and communications co-ordinately to run application tasks in a multiprocessor system efficiently. Thirdly, power dissipation has become a first-class constraint in current microprocessor design. Power dissipation increasingly constrains the design and application of multiprocessors. It is important to control the peak power of a multiprocessor chip to allow improved reliability and saved chip cooling and package cost [1]. As the number of cores integrated in multi-core processor chip increased, the power budget of whole chip must be controlled. In other word, compared with the previously studied power minimization problem, now important problem is efficiently control the peak power consumption of a multiprocessor chip to stay below a desired budget at the same time providing ideal performance. This paper addresses the problem of task mapping and scheduling on multiprocessors considers global power control. A linear program based approach considering communication cost is proposed to formulate the pipeline scheduling problem. Further, an efficient heuristic algorithm named PCLS (Power constrain Communication aware List Scheduling) is proposed and validated for task scheduling on power constraint multiprocessor system.

* ***Corresponding author's***-e-mail: liuyan@hnu.edu.cn

1.1 COMMUNICATION-AWARE TASK PIPELINE SCHEDULING

Multiprocessor architectures considered in this paper consist of multiple-ISA processors, reconfigurable processors, memory component, and communication infrastructure. The communication infrastructure can be established by Network-on-Chip or shared bus, etc. The processing elements such as general-purpose CPU, reconfigurable processors, and digital signal processors connected using communication infrastructure. In a typical embedded multiprocessor architecture each processing element has its local memory. Embedded system application domains such as multimedia and network processing demonstrate clear demarcation of producer and consumer tasks with well-defined inter-task communication behaviours [2, 3]. Due to these characteristics and the application specific nature of the design, the designer can statically allocate computing resources for each task and communication task. Communication-aware task scheduling problem involves partitioning and mapping the computation tasks in the application onto the processing elements of the target chip, ordering the execution of the tasks and data transmission between these processors co-ordinately.

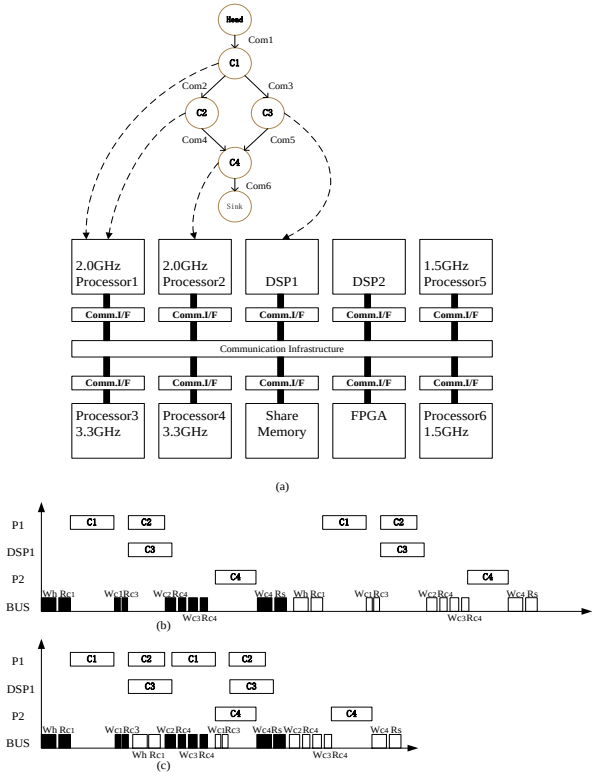


FIGURE 1 A motivational example of pipeline schedule

Figure 1 shows an example of application task with communication detail model scheduling on target multiprocessor chip. Figure 1(a) shows a simple application task allocation. The task graph consists of four computing node, six communication edges and additional empty Head/Sink node to complete the DAG. Assume that there are only Processor1, Processor2, DSP1 in original architecture can execute above example task graph. Then in Figure 1(a), two tasks C1 and C2 are mapped on Processor1, C3 is mapped on DSP1, and C4 is mapped on Processor2. Six communication edges of tasks use shared memory to communicate. Figure 1(b) and Figure 1(c) show time charts for different environment of scheduling. Note that, communication edge represented by $Com_j(Cs,Cd)$ in which Cs and Cd are the source and destination node respectively. More details, we model operation of each communication edge with two additional nodes, one is write data to buffer and the other is read data from buffer. So that communication edge Com_1 in task graph is described by (W_h,R_{c1}) notation in Figure 1. A communication-aware pipeline scheduling of an iterative application task graph be illustrated in Figure 1(b) and Figure 1(c). Assume that two communication channel do not share the physic buffer of shared memory using the same memory interface. In Figure 1(b), task C1 receive data from Head node (empty) and executes computation task. Then task C1 writes data to its output buffer using write operation W_{c1} after running complete. Task C3 can reads from the buffer to obtain data from task C1, then start execute compute procedure. Note that there was no extra communication cost between C1 and C2 because they are mapped on the same Processor1. After task C2 and C3 completes its computation, they write data to its output buffer by write operation W_{c2} and W_{c3} in Figure 1(b). When task C4 completes its previous computation, it read from the two input buffer using read operation labelled R_{c4} and produce final result to sink node. Considering share memory using same physical buffer provide communication, sequence of read/write operation must be obeyed. Using above communication model, more complicate structure can be described like multiple channels or interfaces. Figure 1(c) depicts a possible pipelined schedule compare with Figure 1(b). Processor1 can execute the next loop of task C1 after finishing task C2 and C3 currently. The whole execution time of two loop of application task can be reduced compare with normal status displayed in Figure 1(b).

1.2 CONTRIBUTIONS

An efficient optimizes technique for communication-aware schedule on the multiprocessor architecture considering global power budget control presented in this paper. Contributions are as follows:

1) We present an improved ILP-based formulation that integrates the communication delay model and shared memory model with pipelined scheduling and global power control. Proposed ILP model can describe the multiprocessor architecture more generally as transistor of chips grow in size. Especially, we consider hardware parallel process ability of FPGA component in our model.

2) We present a heuristic algorithm PCLS that integrates pipelined scheduling and global power controlled for designing high efficient computing system implementations. Also the ILP-based method can obtain an optimal result, the heuristic algorithm can handle bigger

problem with result quality relatively close to optimal and requires lower run time cost.

We also present results of extensive experimentation with realistic multimedia applications and synthetic task graphs to evaluate the quality and run-times of our techniques.

The paper is organized as follows: Section II provides the related work, Section III presents the ILP formulations, Section IV presents the heuristic algorithm, Section V discusses the experimental results, and finally Section VI concludes the paper.

2 Related work

In general, application task can be scheduled either at runtime or at design time, named dynamic or static scheduling. In runtime environment, scheduler need determine when processors can execution task and which tasks can be executed frequently. Due to the scheduler overhead and application features, static scheduling considered only in this paper. In design time environment, static scheduler makes decision before system put into market. For a set of application tasks, of which behaviors and details are already known using profile tools.

Integer linear programming (ILP) can be used to find optimal solution for the optimization problem including scheduling problem. In ILP model, a problem is written as a set of linear equations involving integer variables [4]. ILP model can support modelling of different features of heterogeneous multiprocessor architecture or communication infrastructure. But solve ILP formula usually need exponential time cost result in impractical in real system application. The most studied heuristic methods for multiprocessor scheduling problems are list scheduling techniques [5]. In this work, we choose ILP method to find the optimal solution for the communication-aware task scheduling on multiprocessor chip and propose a heuristic algorithm to accelerate scheduling procedure. There are many previously researches focus on design space exploration. They introduce an approach to redesign communication architecture for special application [6]. There are many papers concentrate on performance analysis of communication architecture based on simulation tools [6, 7]. The communication architecture proposed in above works is too complicated, simulation and realization of the whole system is too expensive in time and cost. It is important that more efficient method is needed for fast and accurately communication-aware system design. In [8, 9, 17] they propose ILP-based model for accurate scheduling and timing analysis of task on MPSoC, and they present an efficient scheduler implementation. In this paper, we further consider communication-aware task scheduling problem on multiprocessor chips and use pipelined scheduling improve time efficient, especially considering FPGA component based on previous works [8, 9]. In [2], they propose techniques for system-level power optimization of throughput constrained applications on multiprocessor architectures. We focus on power dissipation control of whole multiprocessor chips. In fact, our goal is control the peak power consumption of a multiprocessor chip to stay blew a desired budget at the same time providing ideal performance.

There are existing global power management methods [10, 11] attempting to maximize performance under power constraints by selecting optimal voltage/frequency level for each core in the context of heterogeneity from process variation. In [11], they model the problem using linear optimization and solve it. In [12], they treated thread scheduling and power management as independent problem same as in [10], and they evaluate the effectiveness and runtime complexity of different power management algorithms. In [13], they using a simple prediction model to support migration of thread because periodically migrate threads to each types of cores is not affordable in terms of performance loss and migration cost. Several works are considering the same problem from other perspectives. The 'uncore' component of system be taken into account for dynamic thread mapping for heterogeneous multicore systems in [14]. In [15], they presented a different approach by dynamically scaling core resources to create adaptive and configurable heterogeneity in hardware. In this paper, we used DAG model instead of thread, and we integrate task mapping, high performance scheduling and low power consumption into a single ILP model. PIE [16] proposed a scheduling framework to predict workload-core mappings. It scales to more than two types of cores. However, it does not provide an efficient method to find optimal mapping for heterogeneous multiprocessor systems.

3 ILP formulation for task scheduling

3.1 SYSTEM SPECIFICATION AND TARGET ARCHITECTURE

An extension direct acyclic graph with communication task node is used in this paper to solver power constrain communication-aware task scheduling on multiprocessor problem [9]. As described in Figure 1(a), we added two empty nodes, head and sink node, which have zero computation and initial communication delays. The source node hasn't predecessor nodes and the sink node hasn't successor nodes. They are used for indicating beginning and end of the task. As displayed in Figure 1(b), communication edge in application DAG transfer to two kinds of nodes and directed edges representing write/read operations and data dependencies of original nodes, such as Com_1 edge label as (W_h,R_{c1}). The task execution times and the communication times are annotated with task nodes and communication nodes, respectively. More details notation about application task will described in next chapter.

Multiprocessor architecture usually consists of hardware/software components and custom hardware. In general, hardware components include general-purpose CPU and DSP, memory components, communication infrastructure and interface. Software components include operating system such as device driver and interrupt service routine, and application software. Additional,

some kinds of custom hardware also added into multiprocessor architecture including FPGA, as described in Figure 1(a). The communication infrastructure can be on-chip buses or Network-on-chip, etc. Communication interfaces are supposed to connect hardware components such as processors, IPs, and memories to the communication networks. Figure 1(a) is an example of target multiprocessor chip consists of nine processors. As simplification example in Figure 1, there are only three components available for running tasks including Processor1, Processor2 and DSP1. Then C1 and C2 running on Processor1, task C4 running on Processor2, the task C3 mapped on DSP1.

3.2 PROBLEM DEFINITION

It is well known that task mapping and scheduling on multiprocessor architecture are highly dependent. The two problems should be solved together to obtain the efficient result. In this paper, we keep concentrate on the communication-aware task scheduling of reconfigurable heterogeneous multiprocessor architecture using pipelined method.

Problem definition: Given an application task graph, a target multiprocessor chip with its parameters, find a mapping and pipelined scheduling of tasks and communications on the target architecture which yields minimum execution time of the task graph while the whole power dissipation below given budget. To solve the problem, we build a flexible model to describe heterogeneous multiprocessor architecture. Then, we use an ILP formulation and a heuristic algorithm to schedule tasks and communications.

3.3 ILP MODEL AND FORMULATION

3.3.1 Variables

Task graph G (V, E) presents the applications and tasks. In G (V, E), contains |V| computation task node and |E| communication edges. Following notations are necessary to describe ILP model used in this paper. Each functionality nodes is labelled as C_i and each communication edge is labelled as $Com_j(C_s,C_d)$. Functionality node can be executed by different type of component such as a processor, DSP, or a custom hardware. Table 1 presents the notations used in our ILP model. And following section states the constraints and objective function.

Based on these notations and decision variables mentioned above Table 1, we can construct improved constraints to model task scheduling problem on multiprocessor chips using pipeline technology based previous research [8].

TABLE 1 Nomenclature of variables

Variables	Description
$Overh(Com_j,n)$	related with communication overhead such as transfer, ISR, context switch
$Type_j$	present a type of component can execute node C_i notational C_i ->*Typej*
PE	the set of all components can be used to running task node
$SType_i$	the set of components can be used to running C_i
$NType_j$	the number of components available for $Type_j$
$Area_{i,j,k}$	the area cost of C_i running on the k^{th} instance of FPGA component *j*
$ExeT_{i,j,k}$	execution time of node C_i running on the k^{th} instance of component type $Type_j$
$ComT_j$	communication time of communication edge $Com_j(Cs,Cd)$
ThrPut	the delay of a pipeline stage in the design
NStage	the number of pipeline stage
$UType_j$	an integer variable which denotes the number of component of $Type_j$ used
MaxP	the maximum number of the final pipelined scheduling
MaxPower	the maximum value of the whole chip power consumption
MaxArea	the maximum size of the FPGA component
BufMax	the maximum size of the shared buffer
Buf(t)	the used buffer size at time *t*
t_i^s and t_i^e	start time and end time of node C_i or communication edge $Com_j(Cs,Cd)$
t_{js}^s and t_{js}^e	start time and end time of write node in communication edge $Com_j(Cs,Cd)$
t_{jd}^s and t_{jd}^e	start time and end time of write node in communication edge $Com_j(Cs,Cd)$
$x_{i,j,k}$	an integer variable associated with node C_i. $x_{i,j,k}$=1 when C_i running on the kth instance of component type $Type_j$; otherwise, zero
$Power_{i,j,k}$	a pre-sampling data, represent power consumption of C_i running on the k^{th} instance of component type $Type_j$
y_i	an integer variable associated with communication edge $Com_j(Cs,Cd)$. y_i=1 when *Cs* and *Cd* nodes running on the different process component; otherwise, zero
$z_{i,NStage}$	set 1 when *Ci* or $Com_j(Cs,Cd)$ is scheduled in pipeline stage *NStage*; otherwise, zero

3.3.2 Constraints

1) General constraints.

Assume that task node is running exactly on one component instance in the target architecture, and the component instance can execute multiple task nodes. After few modifications, following Equation (1) can suit multithread environment. For C_i:

$$\forall C_i, \sum_{Type_j \in SType_j} \sum_{1 \le k \le NType_j} x_{i,j,k} = 1 \tag{1}$$

The decision of used component instance number should smaller or equal to the available component number as Equation (2):

$$UType_j = \sum_{1 \le k \le NType_j} k \times x_{i,j,k} \tag{2}$$

The start and finish time of task node must be satisfied with the dependency constraint of DAG to ensure the precedence relations. In other words, the end time of each incoming communication edge of C_i must be large or equal to the start time, as same as communication edge $Com_j(C_s,C_d)$. The start time and end time of C_i or $Com_j(C_s,C_d)$ can be calculated using following Equation (3) and Equation (4) respectively:

$$t_i^e = t_i^s + \sum_{Type_j \in SType_j} \sum_{1 \le k \le NType_j} x_{i,j,k} \times ExeT_{i,j,k} , \tag{3}$$

$$t_i^e = t_i^s + ComT_i \times y_i . \tag{4}$$

Then, following constraints described as Equation (5) and Equation (6) need be satisfied because of the precedence of write and read operation of nodes and edges.

For task node C_i and its incoming communication edge $Com_j(C_s,C_d)$, equations described as following:

$$t_i^s - t_j^s \ge ComT_j \times y_j , \tag{5}$$

$$t_j^s - t_s^s \ge \sum_{Type_k \in SType_j} \sum_{1 \le k \le NType_j} x_{s,j,k} \times ExeT_{s,j,k} . \tag{6}$$

In above Equation (5) and (6), y_i is associated with communication edge $Com_j(C_s,C_d)$ donate whether the source node and destination node are on the same processor or not.

If per-thread frequency and performance growth were forever stymied, perhaps future scaling could yield more and more cores on a die, to get the throughput performance going for a few generations of a core's lifetime [10]. In fact, power and peak temperature continues to be the key performance limiters compare with other constraints Equation (7).

$$\sum_{Type_j \in SType_j} \sum_{1 \le k \le NType_j} x_{i,j,k} \times Power_{i,j,k} \le MaxPower . \tag{7}$$

2) Pipeline scheduling constraints.

Each task node or communication edge must be scheduled to only one pipeline stage and set to number *NStage*. For each node C_i or edge $Com_j(Cs,Cd)$, the start time and end time must be in the same stage using Equation (8):

$$\begin{cases} t_i^s \ge (\sum_{1 \le NStage \le MaxP} z_{i,NStage} \times NStage - 1) \times ThrPut \\ t_i^s \le (\sum_{1 \le NStage \le MaxP} z_{i,NStage} \times NStage) \times ThrPut \\ \sum_{1 \le NStage \le MaxP} z_{i,NStage} = 1 \end{cases} \tag{8}$$

3) Task execution for shared resources.

To model the behaviour of communication edges using shared memory, $Com_j(Cs,Cd)$ be divided into write and read operators labelled with rectangle *Ws* and *Rd*. For a shared memory, data is pushed to a buffer at the start time of write operation and popped at the finished time of read operation. So, the communication time of edge can calculate with Equation (9).

$$ComT_j = t_{jd}^e - t_{js}^s . \tag{9}$$

Precedence of write/read operation must be satisfied as following Equation (10):

$$\begin{cases} t_{js}^s \ge t_j^s \\ t_{js}^e = t_{js}^s + Overh(Com_j, n) \\ t_{jd}^e \le t_j^e \\ t_{js}^e = t_{js}^s + Overh(Com_j, n) \end{cases} \tag{10}$$

It is difficult deal with the sharing resources contentions. If multiple task nodes are mapped onto the same instance of component, they cannot be executed concurrently. For example, C_{i1} and C_{i2} are mapped onto the same instance k of component type $Type_j$. Therefore, the execution interval of C_i and C_j must not be overlapped with each other. We can use following Equation (11) define overlap condition:

$$(t_{i1}^e - t_{i2}^s) \times m_{i1,i2} + (t_{i2}^e - t_{i1}^s) \times (1 - m_{i1,i2}) \le 0 \tag{11}$$

$m_{i1,i2}$ is an auxiliary binary variable. Processor contention and on-chip communication network contention are similar in that contentions occur when two or more tasks/communication edges are trying to access a single resource at the same time. But in this paper, if a task mapped to FPGA component, above execution overlap constraint can be replaced by Equation (12). Due to the dynamic reconfigurable ability of FPGA, the only limitation is the hardware area while multiple tasks parallel execute on FPGA component. For any task mapped to FPGA, following constraint must be satisfied:

$$\sum_{Type_j \in SType_j} \sum_{1 \le k \le NType_j} x_{i,j,k} \times Area_{i,j,k} \le MaxArea . \tag{12}$$

Considering shared memory capacity constraint, there is different from processors or communication networks because tasks can use the buffer when it is not full. In

another words, the contention of buffer occurs when the sum of the used units by communications is larger than the buffer size. In this paper, we use similar model proposed in [9] to calculate communication edges used buffer size. So, using above method to calculated the used size of a buffer at time t labelled as *Buf*(*t*).

$$Buf(t) \le BufMax . \quad (13)$$

Based above constraints and assumptions, the objective function is minimizing $t^{s}_{\sin k}$.

4 Heuristic algorithm

Considering that ILP belongs to the class of NP-complete problems, we devise a heuristic algorithm based on list scheduling, which is one of the most popular scheduling algorithms [18]. Having defined the problem and model, we now present the heuristic algorithm for the communication-aware task pipelined scheduling problem, an overview of which is presented in Figure 2.

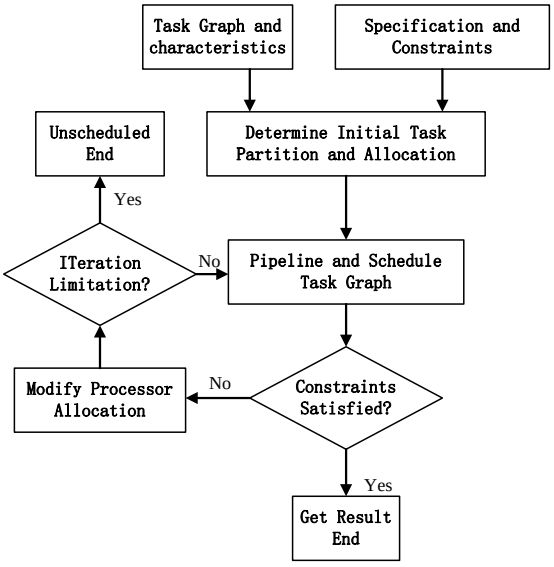


FIGURE 2 Overview of algorithm

Given a task graph specification, power budget and others constraints, the first step consist of determining the application nodes allocation strategy. In this paper, in an attempt to increase total throughput of system, as well as control power dissipation of whole chip, our algorithm try to execute task on fastest component firstly. In this procedure, we determine the number and type of computing components to be used for each of the task nodes. And we then schedule and pipeline the task nodes with a pipe stage and a start time exactly. If it doesn't find a valid schedule and pipeline, it will modify initial allocation strategy and repeat the scheduling and pipeline step. It is repeated until constraints are satisfied or, in the worst case, it will reach the repeat times constraint and quit.

4.1 STEP 1: INITIAL ALLOCATION

Given DAG graph, task execution time and power consumption on different type processor, determining the initial partition is a simple work since our algorithm attempts to execute as many task nodes as possible on powerful processing components. In another words, energy consumption is our first metric to generate initial partition. The estimation of task node power consumption related data can be obtained using profile tools in advance. Our algorithm's primary goal is to perform the communication-aware scheduling and pipelining below the whole chip power consumption budget and provide as possible as high throughput. Data structure *Aavailable_proc_list* record the usage of current processors, and data structure *Allocation_proc_list* save the allocation result. If the final throughput constraint is not satisfied, we choose the less power processor allocation until it is matched in next iteration. The initial processor set consist of the processors on which all the nodes have an execution time that is less than the throughput constraint. Because the number of processing elements of multiprocessors is typically not large, it is not important that we pay much attention to obtain an exactly good initial allocation at first step of proposed algorithm. The initial processor allocation method is very simple, and we could use other more accurate technique in future further research. Step1 is summarized by pseudo code in Figure 3. In pseudo code final step, communication cost of two nodes which mapped on the same processor set zero.

1. Obtaining initial allocation
2. **For** *(each task node in DAG)*
3. maintain Available_proc_list;
4. Allocation_proc_list← max($Power_{i,j,k}$);
5. update Nodes_exe_table;
6. **End For**

FIGURE 3 Obtain initial processor allocation

4.2 STEP 2: SCHEDULING

The goal of algorithm is to pipeline and schedule the DAG graph with the highest throughput while keep the whole chip power consumption constraint. The execution time and power dissipation data of all type processors have known in advance as input of our algorithm. Based on architecture information, execution time, power dissipation data and initial processor allocation, our aim is to determine the schedule and pipeline for the DAG graph that will satisfy the all constraints. In simpler words, our algorithm will assign each task node to a pipe stage and to an exactly starting time within a pipe stage such that predecessor task nodes of finish their execution either in a previous stage or in the same stage before the task node begins its execution.

In our schedule process, the same with ILP model, we must consider shared resource contention problem. For

example, if a task node mapped to a processor, then we need to insure that the selected processor do not execute any other task node during the time interval that it is executing task node.

```
1.  For DAG, compute the longest completion time.
2.  Assign node priority.
3.  Loop
4.     Generate a new ready list.
5.     Loop
6.        Find processor in Allocation_proc_list;
7.        IF (node allocate to FPGA)
8.           Set parallel time slot with pipeline stage;
9.           Update Utili_proc_list;
10.       ELSE
11.          Set time slot with pipeline stage;
12.             IF (no available processor in list)
13.                Stop. No feasible schedule.
14.             Else
15.                Assign processor with node time slot.
16.                Update Utili_proc_list;
17.             End IF
18.       End IF
19.    Mark node as scheduled, remove from ready list.
20.    Until (ready list is empty)
21. Until (nodes in DAG are scheduled)
```

FIGURE 4 Scheduling algorithm

Our algorithm is based the well-known list-scheduling algorithm [18] shown in Figure 4. Algorithm determine the longest completion time from all nodes to all output nodes, assuming that the fastest processor is used to running nodes. This procedure gives the priority for each node. The completion time of node is a direct indication of its criticality. The higher completion time means the higher priority. If task allocated to FPGA component, we assign parallel execution time slot for the tasks, in which mothed the tasks will execute immediately if there are enough resource. In this paper, we use a utilization lists of processors data structure *Utili_proc_list* contain the time tag presents the processor's status. A list of ready nodes that predecessors have already been scheduled is established. As result, we can find the feasible start time of each node in the ready list. A task node in ready list has been mapped to processor and obtained starting time, it is consider scheduled and be removed from ready list. This step is repeated for every node on the ready list, until the ready list is empty and nodes in DAG are scheduled. This procedure is described by pseudo code in Figure 4.

Every node should start executing immediately after all its predecessors have completed. Proposed algorithm starts by finding the longest execution time path from each node until any output task node, assuming execution on the fastest processor. We choose highest priority task node in ready list to find the feasible starting time. If there are multiple choices for candidate, our strategy is selecting the fastest processor that gives us the earliest completion time firstly. In this procedure, task nodes are scheduled with starting time and pipe stage considering processor utilization and pipe stage constraints. For hardware type processing component such as FPGA or ASIC, we calculate the starting time and match area cost constraint due to assume there are some local scheduler for this component exploit task parallel.

4.3 STEP3 MODIFYING PROCESSOR ALLOCATION

After scheduling the DAG graph in the previous step, if there is not a feasible starting time for task node, then modification of processor's allocation used to try again. We use a simple method start with one instance of most powerful consumption processor, and then we can replace it with the slower processor in the available library. For instance, we would favour one faster processor over two slower processors even if the cost of the two were to be less than the cost of the one. This is because with every additional or change processor allocation, the extra communication delays and interface costs could far outweigh the saved dollars in choosing the slower processors. Because the iteration times will set to be a constants number in practical, so that the time complexity of the heuristic is determined by the partitioning algorithm in Step1 and retiming scheduling algorithm in Step2. The main time cost procedure is sorting algorithm and proposed PCLS algorithm has the relevant same time complexity with traditional list scheduling algorithm [18] with additional communication nodes.

5 Results

This section presents the results of experimentation of proposed ILP model and heuristic algorithm. First, we discuss the experimental set-up that includes applications, target architecture and related scheduling techniques. Then, we show comparison of our scheduling method with existing heuristic algorithm to justify that our algorithm is competitive even global power control scenario.

5.1 EXPERIMENTAL SETUP

We evaluated the performance of proposed techniques by using two benchmark sets. The first benchmark consisted of task graph instances derived from practical applications from multimedia applications MJPEG. The task parameters (including execution times, communication delays, and cost) were profiled for a set of 1000 run on a 2.67GHz Intel i5 machine and the Xilinx reconfigurable platform for software processors and FPGA components respectively. The base task dependence graph of MJPEG can be replicated to exploit the pipelined scheduling in this paper. We generated 3 instances for MJPEG application for different replications of the base task graph. The second benchmark was a set of synthetic random task graph instances generated by [19]. These problems were designed to be unbiased towards any particular solver approach and are reportedly harder than other existing benchmarks for scheduling task dependence graphs.

TABLE 2 Experimental parameter descriptions

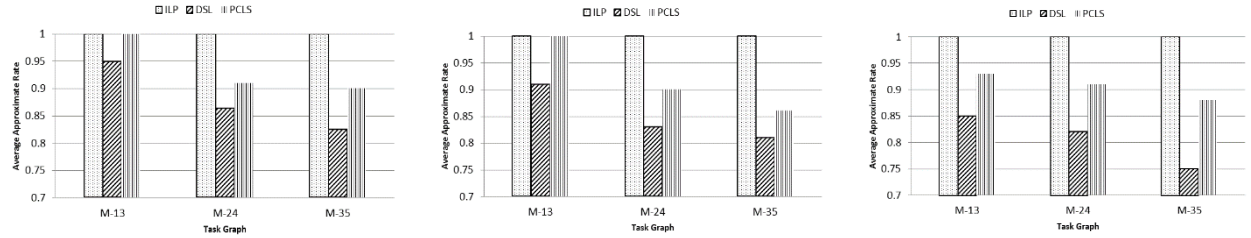
Name	Description
M-13	base task graph derived from MJEPG decoding
M-24	replicated of M-13 which has 24 nodes
M-35	replicated of M-13 which has 35 nodes
S-1	52 nodes and edge density is 10
S-2	52 nodes and edge density is 30
S-3	52 nodes and edge density is 50
S-4	52 nodes and edge density is 70
S-5	102 nodes and edge density is 30
S-6	102 nodes and edge density is 50
S-7	102 nodes and edge density is 70
Arch1	simple heterogeneous 2-core architecture
Arch2	simple heterogeneous 4-core architecture
Arch3	heterogeneous 4-core architecture with FPGA
Arch4	heterogeneous 8-core architecture with FPGA
Arch5	heterogeneous 12-core architecture with FPGA

We established a prototype implementation of the ILP model based on LINGO solver. Proposed heuristic algorithm designed by C++ and whole experiments were conducted on an Intel Dual-Core i5 2.67GHz processor with 4GB RAM running Linux. To evaluate influence of global power control, we consider basic five architectures that perform inter-processor communication using shared memory, three multimedia test benches and seven synthetic task graph test benches. More details description and notation in Table 2. In our simulation framework, different architecture and speed of processors expressed directly with task execution time and cost.

5.2 RESULTS AND DISCUSS

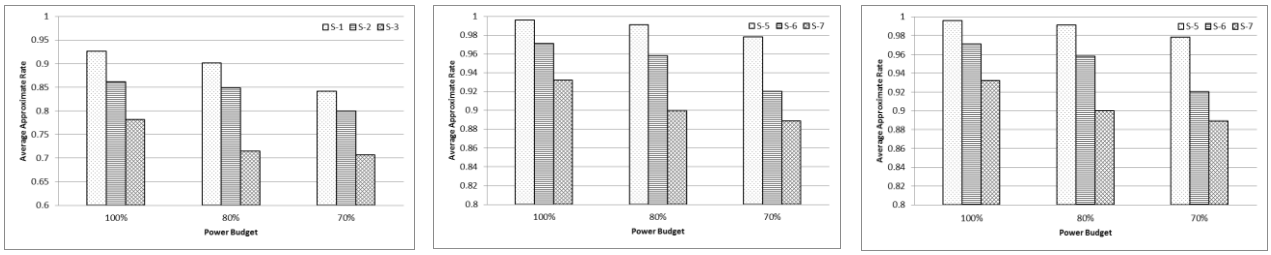
In this experimental, we compare the statistical average makespans obtained by ILP model, DLS [20] and proposed PCLS algorithm which iterative times is set to 50.

Figure 5 shows results of the ILP, DLS and PCLS methods on the first benchmark with realistic MJPEG task graph running on different architectures, Arch1, Arch2 and Arch3. We use approximate rate to report the average percentage approximation of ILP, DLS and PCLS results from the optimal solution by makespan metric.



(a) Arch1 architecture (b) Arch2 architecture (c) Arch3 architecture

FIGURE 5 Average percentage difference of algorithm



(a) Arch3 architecture (b) Arch4 architecture (c) Arch5 architecture

FIGURE 6 Average performance of different power budget

We observe that proposed ILP model can get the optimal result using LINGO solver tools on problem instances with about 35 tasks. In our real experiment, LINGO tools cannot find any feasible solution when problem instance more than 40 in desktop computer. This trend seems to be invariant of the application task graph structure or the number of processors. For realistic application, heuristic algorithm can handle bigger problem. Figure 5(a) reports the results of the three scheduling approaches for multimedia applications on 2 to 4 cores multiprocessor architecture. ILP method can find optimal result. According the increment of task's node, DLS and PCLS can obtain approximate result compare with ILP optimal one. When target architecture is Arch1, there is only 2 cores can execute task. PCLS find out optimal result in M-13 and M-24 bench compare with DLS. When target change to 4 cores architecture Arch2 and Arch3, performance of PLCS can obtain more accurate scheduling result compare with DLS. Especially in Arch3 architecture, PLCS express more attractively performance because our algorithm exploits heterogeneity of Arch3.

As mentioned earlier we did not apply our ILP based technique to the synthetic task graphs due to large designing times. Figure 6 shows results of the proposed PCLS algorithms on the second benchmark with randomly generated task graphs consider global power constraint. The synthetic benchmark is classified by the number of tasks and edge density described in Table 2. The optimal

solutions for these benches were known a priori [19], so we can calculate the average approximate rate to evaluate performance of proposed algorithms. Obviously, the constraint optimization result from ILP model pale in comparison to the PCLS results in many task applications.

In our experiments, the PCLS solution was usually within 85-97% of the optimal on realistic task graphs and problem instances with over 100 tasks. As shown in Figure 6(a), PCLS can obtain about 70% optimal performance based on 70% power budget environment due to the resources limitation of 4-core architecture. For heavier tasks node application, Figure 6 (b) and Figure 6(c) can provide only 12% performance degradation under 70% power budgets based on Arch4 and Arch5.

6 Conclusion

In this paper, we addressed the problem of power constraint communication aware scheduling based on reconfigurable multiprocessor architecture. We presented an ILP formulation that integrated pipelining scheduling and accurate communication model to maximize the whole performance of the application under global power consumption control of chip-level. We presented several formulas that can be used to calculate communication cost and power consumption in our ILP model. Although the ILP method can obtain optimal solutions, its solution time grows exponentially with the number of inputs. Therefore, we also proposed heuristic algorithm based on list scheduling to solve bigger problem in a shorter time.

We performed extensive experimentation with multimedia application MJPEG, as well as large synthetic task graphs. Existing technique such as DLS algorithm was used to compare with proposed PCLS algorithm on several input sets and different architecture, especially heterogeneous architecture with FPGA component. The integration of pipelining and communication aware heuristic algorithm PCLS can obtain relevant better performance under chip wide global power consumption control in comparison to traditional method. For heterogeneous architecture including FPGA component, PCLS algorithm can exploit parallelism of FPGA resources to obtain best trade-off between result quality and solution generation time with given power budget.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant No. 61300037), Hunan Provincial Natural Science Foundation of China (Grant No. 12JJ4057) and the Fundamental Research Funds for the Central Universities.

References

[1] Ma K, Li X, Chen M, Wang X 2011 Scalable power control for many-core architectures running multi-threaded applications *Proc. Int. Symposium on Computer Architecture USA San Jose* 449-60

[2] Srinivasan K, Chatha K S 2007 Integer linear programming and heuristic techniques for system-level low power scheduling on multiprocessor architectures under throughput constraints. *Integration, the VLSI Journal* **40**(3) 326-54

[3] Chatha Karam S, Vemuri R 2002 Hardware-software partitioning and pipelined scheduling of transformative applications *IEEE Trans Very Large Scale Integration Systems* **10**(3)193-208

[4] Chen Y-Y, Hsin-Chu, Hsu Y-C, King C-T 1994 MULTIPAR: behavioral partition for synthesizing multiprocessor architectures *IEEE Trans. Very Large Scale Integration Systems* **2**(1) 21-32

[5] Topcuoglu H, Hariri S, Wu M-y 2002 Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing *IEEE Trans Parallel and Distributed Systems* **13**(3) 260-74

[6] Cescirio W 2002 Component-based design approach for multicore SoCs *Proc. Int'l Design Automation Conference New Orleans USA* 789-94

[7] Ye T-Y, Wolf W 1995 Communication synthesis for distributed embedded systems *Proc. Int'l Conference on Computer-Aided Design* San Jose USA 288-94

[8] Kuang S-R, Chen C-Y, Liao R-Z 2005 Partitioning and Pipelined Scheduling of Embedded System Using Integer Linear Programming *Proc. Int'l Conference on Parallel and Distributed Systems Fukuoka Japan* 37-41

[9] Cho Y 2007 Scheduling with accurate communication delay model and scheduler implementation for multiprocessor system-on-chip *Design Automation for Embedded Systems* **11**(2) 167-91

[10] Isci C, Buyuktosunoglu A, Cher C-Y, Bose P, Martonosi M 2006 An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget *Proc. Int. Symposium on Microarchitecture Orlando USA* 347-58

[11] Teodorescu R, Torrellas J 2008 Variation-Aware Application Scheduling and Power Management for chip multiprocessors *Proc. Int. Symposium on Computer Architecture* Beijing China 363-74

[12] Winter J, Albonesi D, Shoemaker C 2010 Scalable thread scheduling and global power management for heterogeneous many-core architectures *Proc. Int'l Conference on Parallel Architectures and Compilation Techniques Vienna Austria* 29-40

[13] Liu G, Park J, Marculescu D 2013 Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems *Proc. Int. Conference on Computer Design Asheville USA* 54-61

[14] Gupta V, Brett P, Koufaty D, Reddy D 2012 The forgotten'uncore': On the energy-efficiency of heterogeneous cores *Proc. USENIX Annual Technical Conference Boston USA* 1-6

[15] Petrica P, Izraelevitz A, Albonesi D, Shoemaker C 2013 Flicker: A Dynamically Adaptive Architecture for Power Limited Multicore Systems *Proc Int Symposium on Computer Architecture Tel-Aviv Israel* 13-23

[16] Craeynest K V, Jaleel A, Eeckhout L, Narvaez P, Emer J 2012 Scheduling heterogeneous multi-cores through performance impact Estimate on (PIE) *Proc. Int'l Symposium on Computer Architecture Portland USA* 213-24

[17] Lee J, et al 2013 Mapping and Scheduling of Tasks and Communications on Many-Core SoC Under Local Memory Constraint *IEEE Trans Computer-aided Design of Integrated Circuits and Systems* **32**(11) 1748-62

[18] Bakshi S, Gajski D D 1999 Partitioning and Pipelining for Performance-Constrained Hardware/Software Systems *IEEE Trans Very Large Scale Integration Systems* **7**(4) 419-32

[19] Davidovic T, Crainic T G 2006 Benchmark-problem instances for static scheduling of task graphs with communication delays on homogenegous multiprocessor systems *Computers and Operations Research* **33**(8) 2155-77

[20] Sih G C, Lee E A 1993 A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures *IEEE Trans Parallel Distributed System* **4**(2) 175–87

Authors

Xiaoming Chen, 16. 11. 1980, China.

University study: PhD Candidate of Hunan University currently.
Research interests: computer architecture, many-core task scheduling and embedded system.

Yan Liu, 18. 12. 1979, China.

Current position: assistant professor of Hunan University, China.
University study: PhD degree in computer science and technology from Hunan University, China in 2010.
Research interests: computer architecture, embedded system and reconfigurable computing.

Renfa Li, 12. 4. 1956, China.

Current position: professor at Hunan University, China.
University study: PhD degree from Huazhong University of Science and Techonology, China in 2003.
Research interests: computer architecture.