Journal of Systems Architecture 59 (2013) 1424–1432

Contents lists available at SciVerse ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

Clustering scheduling for hardware tasks in reconfigurable computing systems

Zhi Chen [a,b], Meikang Qiu [a], Zhong Ming [b,*], Laurence T. Yang [c], Yongxin Zhu [d]

[a] Department of Electrical and Computer Engineering, University of Kentucky, Lexington, KY 40506, USA
[b] College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, GD 518060, China
[c] Department of Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada
[d] School of Microelectronics, Shanghai Jiaotong University, Shanghai 200240, China

ARTICLE INFO

Article history:
Available online 31 May 2013

Keywords:
Reconfigurable systems
Scheduling
Clustering scheduling strategy (CSS)

ABSTRACT

Reconfigurable computing systems have been used widely in various areas due to their attractive features in low-power and high-precision. However, how to increase utilization and throughput while reducing configuration and execution time overheads on large-scale data has become a great challenge for reconfigurable computing systems. In this paper, we employ a *directed acyclic graph* (DAG) to represent the tasks in an application. With considerations of task dependencies and resource constraints that are not sufficiently studied in literature, we propose two clustering scheduling strategies to reduce the number of configurations and the execution time of applications, while enhancing the utilization of *field programmable gate array* (FPGA) devices: One is a heuristic scheduling strategy and the other is a dynamic programming scheduling strategy. Experimental results indicate that our dynamic programming scheduling strategy can significantly reduce the number of configurations and improve the FPGA utilization, compared to the heuristic scheduling strategy.

1. Introduction

Conventional embedded systems mainly include *application-specified-integrated-circuit* (ASIC) [1,2] and *general-purpose-processor* (GPP) [3–5]. ASICs are usually superior in computing speed and precision. However, low flexibility of them is a major throttle for their further development and extensive use. In addition, the high integration of the current circuit easily leads to a number of potential errors in circuit design, which dramatically aggravate the design cost and impact on time-to-market. GPP systems, although, are competitive in flexibility, development period, and reprogrammable ability, they are not sufficient to satisfy the growing demands for computing-intensive applications due to their low computing speed and precision [6,7].

Reconfigurable computing devices, integrated advantages of both ASIC and GPP systems, are widely employed to accelerate various applications with high precision, fast speed, and low power consumption requirements. These applications range from data encryption [8], target recognition [9], Golomb Ruler Derivation [10], to transitive closure of dynamic graphs [11], etc. However, the further development of reconfigurable devices is still overwhelmed by threefold major challenges. First, the big data they process is several orders of magnitude larger than the processing amounts of the traditional computing systems. And they need to guarantee the high precision and resource utilization at the same time. Second, the configuration time of existing (*field programmable gate array*) FPGA devices accounts for a significant portion of time overhead for applications, which is a limiting factor to the performance of reconfigurable systems. Third, to satisfy the real-time requirements of applications, it requires efficient algorithms to shorten the execution time of overall tasks. Hence, it becomes more and more important to design reasonable and highly efficient algorithms to reduce configuration and execution time overheads while increasing utilization and throughput of reconfigurable chips.

This paper mainly focuses on hardware tasks scheduling, which is a process after the completion of hardware/software partitioning. Hardware task scheduling bears many similarities to general software task scheduling which involves static scheduling and dynamic scheduling, depending on when the task to reconfigurable device mapping is performed. Static scheduling is carried out before execution of tasks, and the mapping remains fixed during application execution. This kind of scheduling is easy to implement, but it cannot capture the dynamic behavior of applications. In contrast, dynamic scheduling performs the tasks to reconfigurable device mapping during the execution of applications, but it

* Corresponding author.
E-mail addresses: zch229@uky.edu (Z. Chen), mqiu@engr.uky.edu (M. Qiu), mingz@szu.edu.cn (Z. Ming), ltyang@gmail.com (L.T. Yang), zhuyongxin@ic.sjtu.edu.cn (Y. Zhu).

1383-7621/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.sysarc.2013.05.015

usually incurs high overhead. A variety of static scheduling strategies, including modified list scheduling [12,13], are used in literature. In order to capture the real-time behavior of application, a number of dynamic scheduling approaches are emerged recently, such as priority-based scheduling [14], horizon and stuffing scheduling [15–17].

However, prior works, such as [15,17–22], mainly focus on the utilization of chips, the rate of acceptance for hardware tasks, and the overhead of scheduling during hardware tasks scheduling. Some vital factors in real reconfigurable computing environments, such as the precedence relations and communication overhead between tasks, and reconfiguration cost of a FPGA, are not well considered. Since tasks are seldom executed independently, we believe that their precedence constraints should be scrutinized during scheduling.

Generally, the tasks in an application, such as image processing, are closely dependent. We use a *directed acyclic graph* (DAG) [23] to represent the dependencies between tasks of an application in a reconfigurable system. To cut the overall execution time of the tasks in a DAG and the reconfiguration cost of a FPGA, while reasonably improving the utilization of reconfigurable resources, we propose two *clustering scheduling strategies* (CSS): a heuristic method and a dynamic programming method. During the scheduling, we consider the resource utilization constraints of reconfigurable platforms and precedence relationships between tasks. The key idea of these strategies are to fully exploit the parallelism of reconfigurable devices, without negatively impacting the schedulability of tasks.

The scheduling bears some similarities to knapsack problem [24,25] that aims to maximize the value of overall items in a package, under the limitation of the package's capacity. The method employed to solve knapsack problem will be also adaptable to our hardware task scheduling. To demonstrate the effectiveness of the presented algorithms, we perform extensive simulations on random data. Experimental results show that our algorithms are efficient in dwindling the number of configurations, reducing the overall execution time, and improving the resource utilization. In addition, the dynamic programming algorithm outperforms the heuristic strategy, since it achieves an average reduction up to 13.87% and 11.40% in the number of configurations and the FPGA utilization.

The main contributions of our work compared to previous research are:

- We design a new heuristic scheduling algorithm to schedule tasks with dependencies. Given a DAG, our heuristic scheduling strategy can efficiently reduce the number of configurations and execution of the application.
- We propose a dynamic programming scheduling algorithm to optimally solve the hardware tasks scheduling problem.

The remainder of this paper is organized as follows: the model adopted in this paper is given in the next section. An example to illustrate our ideas is presented in Section 4. The details of our algorithms are introduced in Section 5. Experimental results and conclusions are provided in Sections 6 and 7.

2. Related work

Many researchers have investigated the task scheduling problem in reconfigurable embedded systems. Previously, most works focus on reconfigurable operating systems [26,17,18], hardware/software partitioning [19,20], and energy control [27,28]. However, to the best of our knowledge, the clustering scheduling strategies with consideration of precedence dependencies between tasks, are not sufficiently studied.

In [29], the authors first introduced best-fit, first-fit and bottom left bin-packing approaches for both on-line and off-line hardware scheduling. The time complexity of their algorithms to search an empty rectangle is $O(nlogn + K)$, where K is the number of tasks. Targeted improving the efficiency for placement, relocation and defragmentation of tasks, an algorithm for on-line tasks to find an empty area on reconfigurable devices was proposed in [30]. In their work, a linked list is utilized to store and manage maximum empty areas. With the consideration of I/O communications, the first-fit and best-fit placement strategies are improved in [31]. However, the tasks in their work are independent and can be scheduled without considering the relationships to other tasks.

A method to manage and maintain occupied space and free space on FPGA is proposed in [21]. The maintenance overhead is $O(nlogn)$. For high communication costs of multitasking reconfigurable system, two clustering approaches are proposed in [32]. One approach clusters tasks with close run-time, and the other one focuses on reducing the ratio of inter-task communication to resource utilization. They employ first-fit strategy to search a suitable areas for tasks. In order to maximize the performance of reconfigurable systems, a methodology for designing these systems was proposed by Merino et al. [33]. They took the area of FPGA as a constraint, and proposed a methodology for coarse-grained partitioning. While these work can improve resource utilization of FPGA platforms, they did not consider the reconfiguration overhead.

Much research has been done in HW/SW co-design [34,35,23]. By construction of a stochastic model, authors in [23] proposed two task scheduling algorithms for embedded systems with heterogeneous functional units. In [36], a RDMS algorithm to reduce the movements for data blocks is proposed, with consideration of the task dependencies and inter-task data communication. However, to the best of our knowledge, most of the previous literature view the tasks to be independent and targeted the improvement of chip utilization. The dependencies between tasks and the configurations of reconfigurable platforms have not been sufficiently studied.

Considering the utilization constraint and tasks dependencies, our work focuses on the reduction of the execution time and configuration time as well as the increase of resource utilization of a FPGA. The main advantages of our clustering scheduling technologies are: (1) By executing tasks in a cluster, the total number of reconfigurations will be decreased, which will eventually lead to the improvement of the performance of reconfigurable systems. (2) By improving the resource utilization, we can ameliorate the overall resource utilization efficiency for FPGAs.

3. System model

3.1. Task model

We model a task in a reconfigurable computing system as a 2-tuple T. For any task $T_i = \langle u_i, e_i \rangle$, u_i is the resource utilization for the execution of task T_i, e_i is the execution time of task T_i. Hence, all tasks in the system can be specified as a set $\Gamma = \langle T_1, T_2, \ldots, T_N \rangle$, where N is the number of tasks.

3.2. Scheduling model

We use a *directed acyclic graph* (DAG) to represent the relationships among tasks in an application. A DAG $G = \langle \Gamma, E \rangle$, where Γ represents a set of task nodes. $E \subseteq \Gamma \times \Gamma$ is a set of edges that represents the precedence dependencies between tasks in the set Γ. For any edge $e(u \rightarrow v)$, it indicates that task v cannot be executed until the completion of task u. We define an notation $\prec$ to

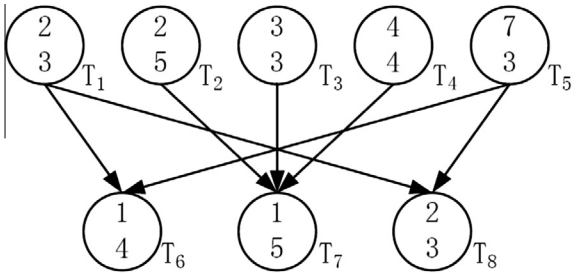


Fig. 1. An instance for DAG.

represent the precedence constraints of these two nodes as $u \prec v$. Fig. 1 illustrates an instance of a DAG. In a node, the values at the above and below of it represent the resource utilization and the execution time of a task.

Considering the characteristics of the reconfigurable platform, such as the heterogeneity and high parallelism, without loss of generality, we make the following assumptions: (1) Compared to the overhead of configuration time, parallel execution delay can be overlooked. (2) The execution of hardware task is non-preemptive.

We aim to configure all tasks in a DAG and schedule them on FPGA. The minimal cost for the execution of a DAG is defined as *MinC(G)*. Due to the high parallelism of FPGA, executing as many tasks as possible concurrently on it under the area constraint will be beneficial to the reduction of overall task execution and system reconfiguration time. The task scheduling is to divide the DAG into M clusters $G = \langle G_1, G_2, \ldots, G_M \rangle$, where $G_i = \langle T_1, T_2, \ldots, T_K \rangle$ is a configuration includes K tasks.

The problem is similar to knapsack problem, which aims to put a set of items into a given package to maximize the total value of these items, while satisfying the limited package capacity. Many algorithms are proposed to solve this problem, such as heuristic algorithm, dynamic programming algorithm and integer linear programming, etc. Correspondingly, the task scheduling problem can be viewed as a knapsack problem, the resource utilization and execution time of a task can be regarded as the weight and value of an item, respectively. The reason to maximize the execution time of a group of tasks is because all the tasks in the same cluster can be executed simultaneously. Therefore, if we execute as many as possible tasks on FPGA, it will efficiently reduce the overall execution time of an application. Our attention is to repeatedly select a group of tasks whose in-degrees are 0 from a DAG, until all the tasks are scheduled. The problem can be formulated as Eq. (1).

$$\begin{cases} \textit{maximize}: & E = \sum_{i=1}^{N} x_i e_i. \\ \textit{satisfy}: & U = \sum_{i=1}^{N} x_i u_i \leqslant A, \\ & \forall t_1 \prec t_2, t_1 \text{ is executed before } t_2. \end{cases} \tag{1}$$

where e_i and u_i are the execution time and resource utilization of task T_i; E is the total execution time of a set of tasks, x_i indicates if a task is selected. If so, $x_i = 1$, otherwise 0; A is the maximum utilization area of a FPGA device. Therefore, the minimum execution cost of a DAG *MinC(G)* can be calculated as $\langle G_1, G_2, \ldots, G_M \rangle = \sum_{i=1}^{M} E_i$, where E_i is the maximum execution time for ith cluster, and K indicates the number of cluster that can be divided from the DAG.

However, the computation time of a certain cluster is also sensitive to some other factors, such as the memory access time and configuration of FPGA, etc. Therefore, the total execution time of a certain cluster is computed as Eq. (2).

$$\begin{cases} E_{G_i} = T_{conf} + T_{lowest} + T_{mem} \\ T_{conf} = \rho \times U_{G_i}, \ \rho \in (0,1) \\ T_{lowest} = Max_{j \in G_i}(e_j) \end{cases} \tag{2}$$

where T_{conf} is the configuration time of a FPGA, T_{lowest} is the task with the highest execution time in the cluster G_i, and T_{mem} is the memory access time. U_{G_i} is the total utilization of the cluster G_i. Compared to the configuration overhead of FPGA, the memory access overhead is insignificant, we assume this port of time is invariant. Therefore, the overall execution time of the DAG can be further expressed as $MinC(G) = \sum_{i=1}^{M} E_{G_i}$.

4. Motivational example

In this section, we will give an example to illustrate the hardware task scheduling in a DAG through classifying them into different clusters. We assume a set of tasks in an application is represented by the DAG shown in Fig. 1.

In order to schedule a group of tasks into a FPGA, we first check the dependencies between tasks. The most intuitive way is to select those tasks whose in-degrees are 0, because the precedence constraints can be satisfied by this way. In a cluster, these tasks thus can be executed in parallel. To simplify the example, the following assumptions are made: (1) The maximum chip area of FPGA is 10. (2) The resource utilization and execution time of a task are less than 10. (3) In the same cluster, the communication cost and pipeline delay between tasks are negligible. (4) The configuration overhead and memory access cost are constant. We assume they are 10 and 1, respectively.

We first apply the heuristic algorithm, which is similar to the one used in [37], to divide the DAG into groups. The only difference is that we employed a selection function to break the tier when two tasks are qualified to be scheduled. We define the selection function $S(i) = (e_i + c_i)/u_i$, which is used to heuristically make optimal choice at each step by the heuristic strategy, where e_i, c_i, and u_i represent the execution time, number of children, and resource utilization of task T_i. In each stage, we select the nodes from those which do not have parent. Repeat the procedure, until the number of tasks in DAG is 0, then we get all the clusters.

In this example, $\{T_1, T_2, T_3, T_4, T_5\}$ are candidates for the first selection. The values of the selection function for each task are $\{S(1), S(2), S(3), S(4), S(5)\} = \{2.5, 3.0, 1.33, 1.25, 0.71\}$. Then, $\{T_1, T_2, T_3\}$ are elected as the first cluster. The total resource requirement of it is $u_1 + u_2 + u_3 = 7$. If we continue putting T_4 in the cluster, the resource utilization will be 11, which exceeds the limited capacity of reconfigurable device, so $\{T_1, T_2, T_3\}$ is the optimal combination for the heuristic selection at the first iteration. Then, we remove them from the DAG, and resume the selecting operations. In the next iteration, $\{T_4, T_5\}$ will be served as the candidates. Table 1 shows the concrete steps.

From above results obtained by applying the heuristic algorithm on the DAG, we can calculate the overall execution time: $(5 + 10 + 1) + (4 + 10 + 1) + (5 + 10 + 1) + (4 + 10 + 1) = 62$, and the average resource utilization: $R_U = (7 + 4 + 8 + 3)/(4 \times 10) = 55.0\%$.

Table 1
The procedures of applying the heuristic algorithm on the DAG in Fig. 1 to partition it into clusters.

Step	*CS*	$S(i)$	Selected tasks
1	$\{T_1, T_2, T_3, T_4, T_5\}$	$\{2.5, 3.0, 1.33, 1.25, 0.71\}$	$\{T_1, T_2, T_3\}$
2	$\{T_4, T_5\}$	$\{1.25, 0.71\}$	$\{T_4\}$
3	$\{T_5, T_7\}$	$\{0.71, 5.0\}$	$\{T_5, T_7\}$
4	$\{T_6, T_8\}$	$\{4.0, 1.5\}$	$\{T_6, T_8\}$

Table 2
The first iteration of applying the dynamic programming scheduling strategy on the DAG in Fig. 1.

u_i	e_i	j										
		0	1	2	3	4	5	6	7	8	9	10
2	3	0	0	3	3	3	3	3	3	3	3	3
2	5	0	0	5	5	8	8	8	8	8	8	8
3	3	0	0	5	5	8	8	8	11	11	11	11
4	4	0	0	5	5	8	8	9	11	12	12	12
7	3	0	0	5	5	8	8	9	11	12	12	12

Table 3
The second iteration of applying dynamic programming scheduling strategy on the DAG in Fig. 1.

u_i	e_i	j										
		0	1	2	3	4	5	6	7	8	9	10
3	3	0	0	3	3	3	3	3	3	3	3	3
7	3	0	0	3	3	3	3	3	3	3	3	6

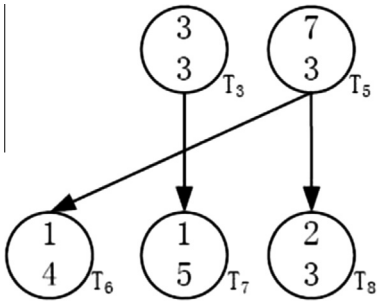


Fig. 2. After the first iteration.

We apply the dynamic programming scheduling algorithm on the DAG. The concrete procedures to select the clusters of nodes are illustrated in Tables 2 and 3. We tend to select the node with highest value of selection function defined above when we schedule the nodes with the same execution time.

During the first scheduling, task $\langle T_1, T_2, T_4\rangle$ are selected from the nodes that have no parents $\langle T_1, T_2, T_3, T_4, T_5\rangle$. The DAG is changed into Fig. 2 after this scheduling. Then, repeat the process, and choose nodes $\langle T_3, T_5\rangle$ from the updated DAG, as shown in Table 3. After these two steps, all precedence dependencies in the DAG are removed, and the remaining tasks are schedulable in the same cluster.

Therefore, the DAG is divided into three clusters through the whole scheduling, which are $\{T_1, T_2, T_4\}$, $\{T_3, T_5\}$, and $\{T_6, T_7, T_8\}$. The overall execution time of the DAG can be computed as: $E_{DAG} = (5 + 10 + 1) + (3 + 10 + 1) + (5 + 10 + 1) = 46$, and the overall ratio of resource utilization is $R_U = (U_{cluster1} + U_{cluster2} + U_{cluster3})/(3 \times 10) = 70.3\%$.

It can be analyzed from these two strategies that the dynamic programming method is more efficient in the reduction of execution time and the improvement of utilization of chip areas. More precisely, compared to the heuristic algorithm, the execution time of the dynamic programming strategy is decreased by 26%, while the resource utilization is increased by 15.3%. In addition, the dynamic programming strategy is more advantageous in the configuration time of a FPGA, since it only needs three configurations while the heuristic approach needs four configurations.

5. Algorithm

In this section, we describe the details of our heuristic algorithm and dynamic programming algorithm.

5.1. Heuristic scheduling algorithm

Heuristic algorithm is a method to heuristically make a locally optimal choice at all stages, with the hope that this choice will eventually lead to the global optimum. We apply the heuristic algorithm on a DAG repeatedly to get a locally optimal execution time for each cluster. We define the selection function as Eq. (3).

$$S(i) = (e_i + \beta c_i)/u_i, \tag{3}$$

where e_i, c_i, and u_i correspond to the execution time, number of children, and resource utilization of task i. β is a coefficient to normalize the number of child to the utilization of task. Let $\beta = A/10$, where A is the maximal available chip utilization.

The basic idea of the strategy is as follows. First, we calculate $S(i)$ for tasks without parent (lines 3–4). The reason for choosing these tasks is to make sure the precedence constraints are met. Second, we sort them in descending order of $S(i)$ (line 5), and schedule them sequentially. Finally, delete these tasks from the current DAG (line 13). The pseudo-code is described as Algorithm 3.

5.2. Dynamic programming algorithm

Dynamic programming is an approach for addressing some complicate problems via breaking them down into simpler subproblems [38]. Our objective is to maximize total execution time of a cluster, while subject to the constraint that overall utilization of the cluster $\sum_{i=1}^{M} U_{G_i} \leqslant A$. The mathematical form for the recursion procedure is shown in Eq. (4).

$$opt(i,j) = \begin{cases} opt(i-1,j) & \text{if } u_i > j \\ max(opt(i-1,j), opt') & \text{otherwise} \end{cases} \tag{4}$$

where $opt(i,j)$ represents an optimal combination when task i is selected, and the current maximum value is j, opt' is defined as Eq. (5). We assume the resource utilization of all tasks are integers, and for $\forall i \in \{1, 2 \ldots, N\}$, $u_i < 100$.

$$opt' = \begin{cases} opt(i-1, j-u_i) + e_i & \text{if task } i \text{ can be scheduled} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Require: A DAG$\langle \Gamma, E\rangle$ with n nodes, which are stored in an adjacency matrix $G[0...n, 0...n]$. A list $task[0...n]$ is used for storage of tasks
Ensure: The number of clusters K and the total execution time $MinE$
1: Let N and N_{subset}be the remaining number of tasks in task list and the number of tasks in the current $subset$. Let $subset \leftarrow \varnothing$ be a set of tasks has been chosen. Define $C_U = 0$ as the current resource utilization.
2: **while** $N > 0$ **do**
3: $tmp\_set \leftarrow$ Get the nodes whose in-degrees are 0
4: $S(i) \leftarrow$ Calculate the value selection function for each task in tmp_set
5: Sort the tasks in tmp_set in descending order of $S(i)$
6: **while** $C_U \leq A$ && $i \leq$ the length of tmp_set **do**
7: insert $tmp\_set(i) \rightarrow subset$
8: $C_U \leftarrow C_U + u_{tmp\_set(i)}$
9: $i \leftarrow i + 1$
10: **end while**
11: $N \leftarrow N - N_{subset}$
12: $K \leftarrow K + 1$
13: $MinE \leftarrow MinE + \sum_{i=1}^{N_{subset}} e_i$
14: Remove the tasks in $subset$ from the DAG
15: **end while**
16: **return** K

Fig. 3. Heuristic scheduling.

The necessary and sufficient condition for a task can be scheduled is that it can pass the admission test given in Algorithm 6. The algorithm mainly performs two examinations for the input task. The first one is to check out the resource utilization constraints, the other one is to inspect the precedence dependencies for the given tasks.

From the algorithms, it can be analyzed that the procedure for clustering scheduling adheres to the following execution sequences: activating tasks in Algorithm 4, searching all the tasks without parent nodes, performing admission test for a given task in Algorithm 6, and putting the one which is approved by admission test into a certain list *sublist* by executing Algorithm 5. Finally, we calculate the execution time of this cluster in Algorithm 4. We repeat this procedure until the entire tasks in DAG are scheduled.

5.3. Complexity analysis

We analyze the time complexity of the proposed algorithms as follows:

(a) *The heuristic algorithm:* The heuristic approach includes following processes: (1) Finding the nodes without parent node. (2) Sorting these nodes. (3) Greedily selecting a group of tasks from these n nodes. The time complexity for these three parts are $O(n)$, $O(nlogn)$, and $O(nlogn)$, respectively. Hence, the time complexity of using the heuristic strategy to finding a cluster of tasks is $O(nlogn)$. The overall time complexity of the heuristic scheduling, in the worst case, can be calculated as: $nlogn + (n-1)log(n-1) + \cdots + 2log2 = \int_1^n xlog(x)dx = \frac{1}{2}[x^2logx]_1^n - \frac{1}{2}\int_1^n x^2 d logx = \frac{1}{2}n^2(log(n) - \frac{1}{2ln2})$. Therefore, the worst case time complexity of the heuristic clustering scheduling algorithm is $O(n^2logn)$.

(b) *The dynamic programming algorithm:* Applying a dynamic programming algorithm on the knapsack problem to get an optimal solution requires the time complexity of $O(nA)$, where A is the maximum area of reconfigurable device. Therefore, in worst case that we can just get a single task at each iteration, the time complexity is $n \times A + (n-1) \times A + \ldots + 2 \times A + A = \frac{n \times (n+1)}{2} A$. Hence, the overall time complexity is $O(n^2A)$.

6. Experiments

We have done extensive experiments to verify the effectiveness of the proposed algorithms which are implemented in C++. The simulations are performed on a Dell INSPIRON 4010 laptop with 2 AMD Athlon P360 (2.3 GHz) CPU and 2 GB memory. We implemented the algorithms as standalone modules of our customized simulator on Ubuntu 11.04. All the DAGs and input values are randomly generated and the run time of each simulation is collected by the Linux time function. We performed a host of experiments with 500 sets of randomly generated hardware tasks. The following assumptions are made for the input data. (1) The maximum resource utilization of reconfigurable device is 100 ($A \leqslant 100$). (2) The execution time of each task is no more than 100 ($e_i \leqslant 100$). (3) The configuration time of reconfigurable devices are constant.

Require: A DAG$\langle \Gamma, E \rangle$ with n nodes, which are stored in an adjacency matrix $G[0...n, 0...n]$. A list $task[0...n]$ for storage of tasks
Ensure: The number of clusters M and the total execution time $minE$
1: Let N be the remaining number of tasks in task list
2: **while** $N > 0$ **do**
3: $cluster_i \leftarrow$ Get a cluster of tasks by applying Algorithm 5 on task list $task$
4: Remove the precedence edges which are connected by the tasks in the cluster from the DAG and delete these tasks from task
5: $T_{C_i} \leftarrow$ Apply Equation (2) to calculate the execution time of $cluster_i$
6: Increase the execution time for the scheduled tasks by $cluster_i$, $minE \leftarrow minE + T_{C_i}$
7: $M \leftarrow M + 1$
8: Count the number of tasks in $Cluster_i$, let it be N_{C_i}
9: $N \leftarrow N - N_{C_i}$
10: **end while**
11: **return** M and $minE$

Fig. 4. Dynamic programming scheduling.

Require: 1. A DAG$\langle \Gamma, E \rangle$ with n nodes, which are stored in an adjacency matrix $G[0...n, 0...n]$, with average execution time e_i and resource utilization u_i of every task in the DAG. 2. An adjacency matrix $path[0...n, 0...n]$, which records the selection paths of tasks. 3. An optimal execution time matrix $opt[0...A, 0...A]$, where A is the maximum area of FPGA. 4. A task list $sublist$, which is used to store the optimal combination of tasks
Ensure: A group of tasks that is selected
1: Let $maxE = 0$ be the current maximum execution time
2: **for** $i = 1 \rightarrow n$ **do**
3: **for** $j = 1 \rightarrow A$ **do**
4: **if** $j \geq u_i$ **then**
5: **if** The admission test returns $feasible$ by Algorithm 6 **then**
6: $optimal \leftarrow opt(i-1, j-u_i) + e_i$
7: $path(i, j) \leftarrow 1$
8: **if** $optimal > maxE$ **then**
9: $maxE \leftarrow optimal$
10: update $sublist$
11: **end if**
12: **else**
13: $optoptimal \leftarrow 0$
14: **end if**
15: Calculate $opt(i, j)$ by Equation (4).
16: **end if**
17: **end for**
18: **end for**
19: **return** $sublist$

Fig. 5. Get a cluster.

Require: A DAG $\langle \Gamma, E \rangle$, with n nodes, which are stored in an adjacency matrix $G[0...n, 0...n]$. A specific task i. An adjacency matrix $path[0...n, 0...n]$, which records the selection path of tasks. The current optimal resource utilization cur_opt, the current task ID tid, and the current optimal combination $sublist$
Ensure: The feasibility of scheduling task i
1: Define a temporary list $tmp\_set \leftarrow \varnothing$
2: Get the optimal resource utilization tmp_opt before the addition of task i, $tmp\_opt = cur\_opt - u_i$
3: Recursively search the tasks that can form currently optimal combination.
4: **for** $i = (tid - 1) \rightarrow 0$ **do**
5: **if** $path(i, tmp\_opt) = 1$ **then**
6: insert the task $T_i \rightarrow tmp\_set$
7: **end if**
8: Check the precedence dependencies of task tid.
9: **for** $j = 0 \rightarrow n$ **do**
10: **if** task j is the parent node of task tid **then**
11: **if** task j is not in the tmp_set **then**
12: **return** $infeasible$
13: **end if**
14: **end if**
15: **end for**
16: **end for**
17: insert task $tid \rightarrow sublist$
18: **return** $feasible$

Fig. 6. Admission test for a given task.

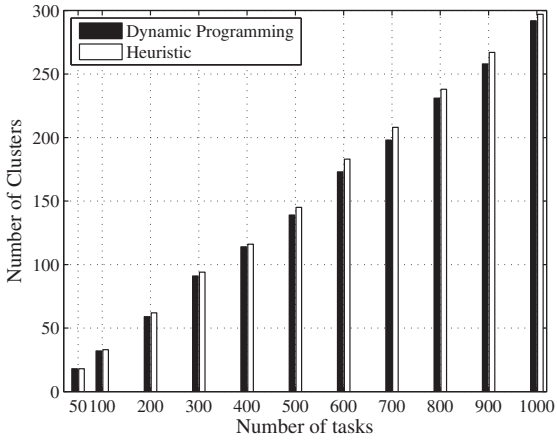


Fig. 7. The relation between the number of clusters and the number of tasks, when the maximum resource utilization $u_i \leqslant 40$.

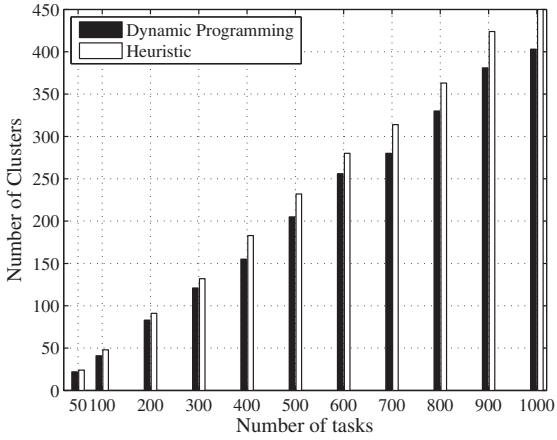


Fig. 8. The relation between the number of clusters and the number of tasks, when the maximum resource utilization $u_i \leqslant 60$.

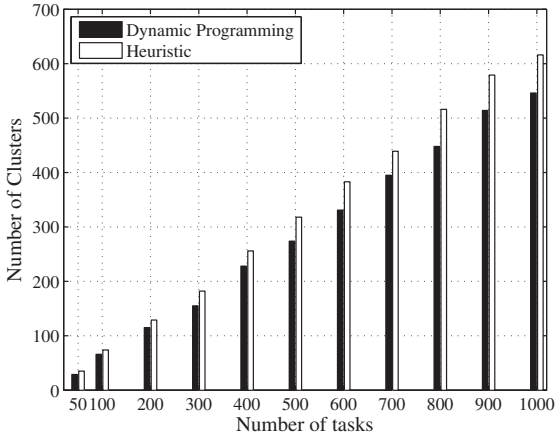


Fig. 9. The relation between the number of clusters and the number of tasks, when the maximum resource utilization $u_i \leqslant 80$.

(4) All tasks can be executed simultaneously in the same configuration. (4) Each node has no more than five children. (5) We assume the size of the simulation chip is 80×120, which is the same as the XCV2000E devices.

As shown in Figs. 7–9, we compare the number of the clusters obtained by dynamic programming algorithm and that of by

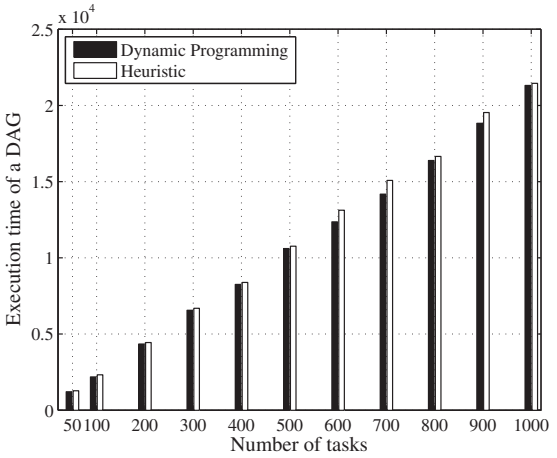


Fig. 10. The relation between the execution time of a DAG and the number of tasks, when the maximum resource utilization $u_i \leqslant 40$.

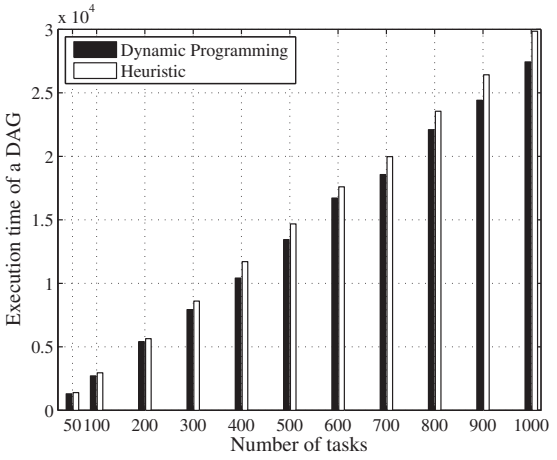


Fig. 11. The relation between the execution time of a DAG and the number of tasks, when the maximum resource utilization $u_i \leqslant 60$.

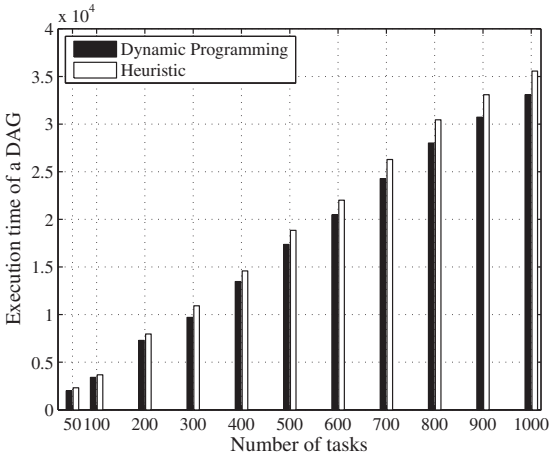


Fig. 12. The relation between the execution time of a DAG and the number of tasks, when the maximum resource utilization $u_i \leqslant 80$.

heuristic algorithm, with the increase of tasks. The experiment includes three groups of comparisons, with the maximal resource utilization of a task changing from 40, 60, to 80. The group of figures illustrate that the dynamic programming scheduling strategy

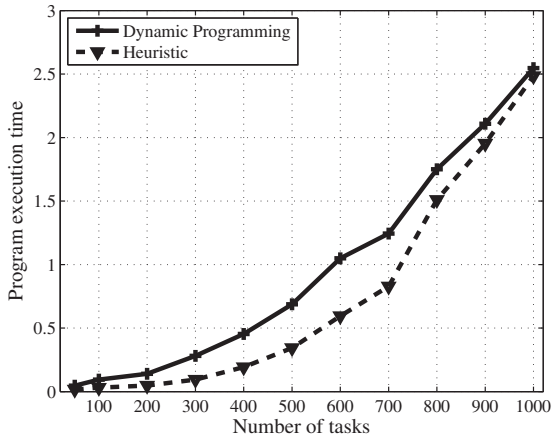


Fig. 13. The comparison of the computation time of the algorithms with the increase of tasks, when the maximum resource utilization $u_i \leqslant 40$.

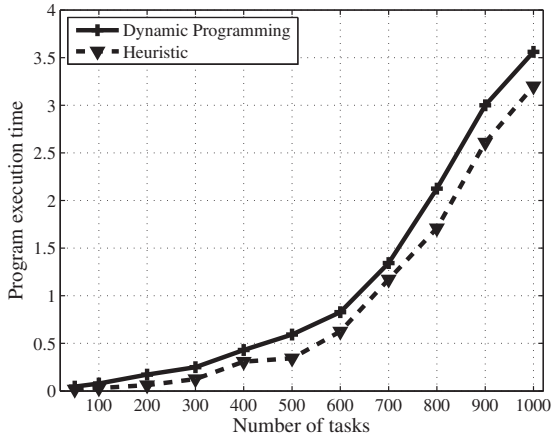


Fig. 14. The comparison of the computation time of the algorithms with the increase of tasks, when the maximum resource utilization $u_i \leqslant 60$.

leads to fewer clusters for an application. This result is mainly attributed to the accurate search of dynamic programming algorithm, which guarantees an optimal combination of tasks at each step.

With the increase of the maximum resource utilization of tasks, dynamic programming clustering scheduling approach is even more superior than the heuristic algorithm. This is because when the resource utilization of all tasks are relatively low, as shown in Fig. 7, the fragments incurred by scheduling is not very obvious. Therefore, the reduction in the number of clusters is insignificant. However, with the increase of resource utilizations of tasks, the improvement will be substantial. As shown in Fig. 9, compared to the heuristic scheduling approach, when the maximum resource utilization of a task is 80, the number of clusters generated by dynamic programming scheduling algorithm is reduced by 13.87% on average. The shrinking number of clusters will efficiently decrease the reconfiguration overhead of FPGA.

Figs. 10–12 show the overall execution time of a DAG by using the dynamic programming approach and the heuristic scheduling

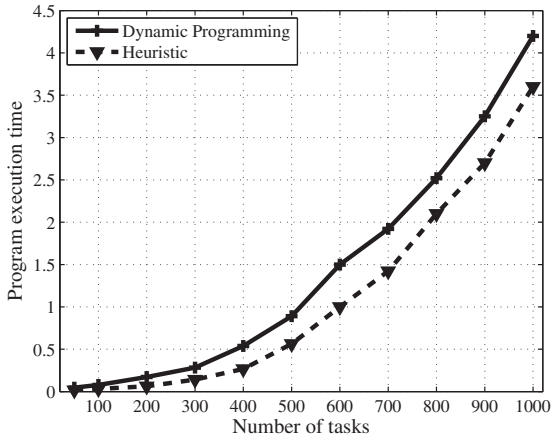


Fig. 15. The comparison of the computation time of the algorithms with the increase of tasks, when the maximum resource utilization $u_i \leqslant 80$.

strategy. As illustrated in these three figures, applying dynamic programming to partition a DAG can save more execution time. This is primarily because the dynamic programming scheduling strategy optimally selects task nodes from a DAG, while the heuristic scheduling approach mainly performs according to the selection function at each single step.

Figs. 13–15 indicate the whole computation time of a DAG for heuristic algorithm and dynamic programming scheduling algorithm. It can be seen that the heuristic algorithm has trivial advantage with respect to the algorithm computation time. However, compared to the configuration time and tasks execution time, the computation time only accounts for a very small portion. For example, normally, it just needs around 4 s to run our algorithms, even when the number of tasks reaches 1000 and the maximum resource utilization is 80. This is the worst case in our experiments. The computation time will be much shorter with the decreasing number or the maximal resource utilization of tasks. Therefore, although the dynamic programming scheduling algorithm consumes a little more running time than that of the heuristic scheduling algorithm, it is much more competitive in the reduction of configurations and the improvement of resource utilization.

To verify the chip utilization of these two algorithms, we define the ratio of average resource utilization as $R_u = \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N_i} u_i/A$, where M and N_i represent the total number of clusters and the number of tasks in the cluster i. Table 4 shows the average resource utilizations with the growing number of tasks, when the resource utilization for each task i satisfies $u_i \leqslant 60$. As shown in Table 4, the utilization of the dynamic programming scheduling strategy is increased by 11.4% on average, compared to that of the heuristic method.

7. Conclusion

In this paper, we proposed a heuristic scheduling strategy and a dynamic programming algorithm to divide a DAG into several clusters on reconfigurable computing systems. During the clustering scheduling, we considered resource utilization and precedence dependencies between tasks. The dynamic programming is able

Table 4
The resource utilization of the dynamic programming algorithm and the heuristic algorithm for clustering scheduling, when $u_i \leqslant 60$.

Algorithm	The number of tasks										
	50	100	200	300	400	500	600	700	800	900	1000
Dynamic programming	0.7677	0.8429	0.8342	0.8471	0.8779	0.8457	0.8066	0.8732	0.8315	0.8446	0.8546
Heuristic	0.7375	0.72	0.7609	0.7765	0.7436	0.7473	0.7374	0.7787	0.7559	0.759	0.7654

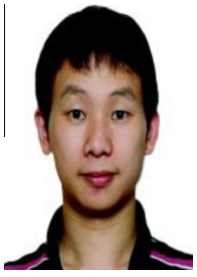
to provide more accurate and efficient solution to schedule the tasks in a DAG, by reducing the number of configurations and improving the resource utilization. Experimental results verified the effectiveness of our algorithm. We will further apply our approach to improve the performance while considering the thermal and power-performance issues of reconfigurable computing devices.

Acknowledgements

This work was supported in part by the NSF CNS-1249223, NSFC 61071061 of Meikang Qiu; NSFC61170077, NSFGD:2012B091 100198, 10351806001000000, Shen zhen S&T proj. of 2013 of Zhong Ming.

References

[1] B. Dave, G. Lakshminarayana, N.K. Jha, Power management in high-level synthesis, IEEE Trans. Very Large Scale Integr. VLSI Syst. 7 (1) (1999) 92–104.
[2] R. Dick, N.K. Jha, MOGAC: a multiobjective genetic algorithm for hardware–software cosynthesis of distributed embedded systems, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 17 (10) (1998) 920–935.
[3] U. Kapasi, W.J. Dally, S. Rixner, J.D. Owens, B. Khailany, The Imagine stream processor, in: Proceedings of the 2002 IEEE International Conference on Computer Design, 2002, pp. 282–288.
[4] M.I. Gordon, W. Thies, M. Karczmarek, J. Lin, A. Meli, A.A. Lamb, A stream compiler for communication-exposed architectures, SIGOPS Oper. Syst. Rev. 36 (5) (2002) 291–303.
[5] J.D. Owens, U.J. Kapasi, P. Mattson, B. Towles, B. Serebrin, S. Rixner, W.J. Dally, Media processing applications on the Imagine stream processor, in: Proceedings of the IEEE International Conference on Computer Design, 2002, pp. 295–302.
[6] I. Kuon, J. Rose, Measuring the gap between FPGAs and ASICs, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 26 (2) (2007) 203–215.
[7] K. Compton, S. Hauck, Reconfigurable computing: a survey of systems and software, ACM Comput. Surv. 34 (2) (2002) 171–210.
[8] A.J. Elbirt, C. Paar, An FPGA implementation and performance evaluation of the serpent block cipher, in: Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays (FPGA), 2000, pp. 33–40.
[9] M. Rencher, B.L. Hutchings, Automated target recognition on SPLASH 2, in: Proceedings of the Fifth IEEE Symposium on FPGA-Based Custom Computing Machines (FCCM), 1997, pp. 192–200.
[10] E. Sotiriades, A. Dollas, P. Athanas, Hardware–software codesign and parallel implementation of a Golomb ruler derivation engine, in: IEEE Symposium on Field-Programmable Custom Computing Machines, 2000, pp. 227–235.
[11] L. Huelsbergen, A representation for dynamic graphs in reconfigurable hardware and its application to fundamental graph algorithms, in: Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays (FPGA), 2000, pp. 105–115.
[12] S.M. Loo, B.E. Wells, Task scheduling in a finite-resource, reconfigurable hardware/software codesign environment, INFORMS J. Comput. 18 (2) (2006) 151–172.
[13] B. Mei, P. Schaumont, S. Vernalde, A hardware–software partitioning and scheduling algorithm for dynamically reconfigurable embedded systems, in: Proceedings of the 11th ProRISC Workshop on Circuits, Systems, and Signal Processing, 2000, pp. 405–411.
[14] J. Noguera, R. Badia, Dynamic run-time hardware–software scheduling techniques for reconfigurable architectures, in: Proceedings of the 17th International Conference on Hardware Software Codesign (CODES), 2002, pp. 205–210.
[15] C. Steiger, H. Walder, M. Platzner, L. Thiele, Online scheduling and placement of real-time tasks to partially reconfigurable devices, in: Proceedings of the 24th IEEE International Real-Time Systems, Symposium (RTSS'03), 2003, pp. 224–235.
[16] H. Walder, M. Platzner, Non-preemptive multitasking on FPGAs: task placement and footprint transform, in: Proceedings of the Second International Conference on Engineering of Reconfigurable Systems and Architectures (ERSA), 2002, pp. 24–30.
[17] C. Steiger, H. Walder, M. Platzner, Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks, IEEE Trans. Comput. 53 (11) (2004) 1393–1407.
[18] V. Nollet, P. Coene, D. Verkest, S. Vernalde, R. Lauwereins, Designing an operating system for a heterogeneous reconfigurable SoC, in: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, 2003.
[19] J. Wu, T. Srikanthan, T. Lei, Efficient heuristic algorithms for path-based hardware/software partitioning, Math. Comput. Modell. 51 (7–8) (2010) 974–984.
[20] J. Madsen, J. Grode, P.V. Knudsen, M.E. Peterson, A. Haxthausen, LYCOS: the Lyngby co-synthesis system, Math. Comput. Modell. 2 (2) (1997) 195–235.
[21] A. Ahmadinia, C. Bobda, S. Fekete, J. Teich, J.C. Veen, Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices, IEEE Trans. Comput. 56 (5) (2007) 673–680.
[22] T. Marconi, Y. Lu, K. Bertels, G. Gaydadjiev, Online hardware task scheduling and placement algorithm on partially reconfigurable devices, in: Proceedings of the International Workshop on Applied Reconfigurable Computing (ARC), 2008.
[23] M. Qiu, E.H.-M. Sha, Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems, ACM Trans. Des. Automat. Electron. Syst. 14 (2) (2009) 1–30 (Best Paper Award).
[24] H. Kellerer, U. Pferschy, D. Pisinger, Knapscak Problems, Springer, 2004.
[25] M. Silvano, T. Paolo, Knapsack Problems: Algorithms and Computer Interpretations, Wiley-Interscience, 1990.
[26] P. Hsiung, C. Huang, Y. Chen, Hardware task scheduling and placement in operating systems for dynamically reconfigurable SoC, J. Embedded Comput. 3 (1) (2009) 53–62.
[27] M. Qiu, L.T. Yang, Z. Shao, E.H.-M. Sha, Dynamic and leakage energy minimization with soft real-time loop scheduling and voltage assignment, IEEE Trans. Very Large Scale Integr. VLSI Syst. 18 (3) (2010) 501–504.
[28] M. Qiu, C. Xue, Z. Shao, E.H.-M. Sha, Energy minimization with soft real-time and DVS for uniprocessor and multiprocessor embedded systems, in: IEEE/ACM Design, Automation and Test in Europe (DATE), 2007, pp. 16–20.
[29] K. Bazargan, R. Kastner, M. Sarrafzadeh, Fast template placement for reconfigurable computing systems, IEEE Des. Test 17 (1) (2000) 68–83.
[30] M. Handa, R. Vemuri, An efficient algorithm for finding empty space for online FPGA placement, in: Proceedings of the 41st Annual Design Automation Conference (DAC'04), 2004, pp. 960–965.
[31] M. Tomono, M. Nakanishi, S. Yamashita, K. Nakajima, K. Watanabe, A new approach to online FPGA placement, in: Proceedings of 40th Information Science and Systems (CISS), 2006, pp. 145–150.
[32] A. Ahmadinia, C. Bobda, J. Teich, Temporal task clustering for online placement on reconfigurable hardware, in: Proceedings of the IEEE International Conference on Field-Programmable Technology, 2003, pp. 359–362.
[33] P. Merino, M. Jacome, J.C. Lpez, A methodology for task based partitioning and scheduling of dynamically reconfigurable systems, in: Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM'98), 1998, pp. 965–970.
[34] S. Shukla, J. Talpin, Guest editorial: special issue on models and methodologies for co-design of embedded systems, J. ACM Trans. Embedded Comput. Syst. 4 (2) (2005) 225–227.
[35] I. Robertson, J. Irvine, A design flow for partially reconfigurable hardware, J. ACM Trans. Embedded Comput. Syst. 3 (2) (2004) 257–283.
[36] M. Huang, V. Narayana, H. Simmler, O. Serres, T. EL-Ghazawi, Reconfiguration and communication-aware task scheduling for high-performance reconfigurable computing, J. ACM Trans. Reconfigurable Technol. Syst. 3 (4) (2010) 1–24.
[37] P. Saha, Automatic software–hardware co-design for reconfigurable computing systems, in: Proceedings of International Conference on Field Programmable Logic and Applications, 2007, pp. 507–508.
[38] T.H. Corment, C.E.M. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press & McGraw-Hill, 2001.



Zhi Chen received the BS and MS degree from Huaihua University and Hunan University, China, in 2008 and 2011, respectively. And now he is pursuing his Ph. D. degree in the Department of Electrical and Computer Engineering (ECE), University of Kentucky. His research interests include heterogeneous embedded systems and high performance computing.

Meikang Qiu (SM'07) received the B.E. and M.E. degrees from Shanghai Jiao Tong University, China. He received the M.S. and Ph.D. degrees of Computer Science from University of Texas at Dallas in 2003 and 2007, respectively. He had worked at Chinese Helicopter R&D Institute and IBM. Currently, he is an assistant professor of ECE at University of Kentucky. He is an IEEE senior member and ACM Senior member. He has published more than 170 papers, including 15 IEEE/ACM Transactions papers. He is the recipient of the ACM Transactions on Design Automation of Electronic Systems (TODAES) 2011 Best Paper Award. He also received four other best paper awards (IEEE EUC'09, IEEE/ACM GreenCom'10, IEEE CSE'10, and IEEE ICESS'12) and one best paper nomination. He also holds 2 patents and has published 3 books. He has also been awarded Navy summer faculty award in 2012

and SFFP Air Force summer faculty in 2009. He has been on various chairs and TPC members for many international conferences. He served as the Program Chair of IEEE EmbeddCom'09 and EM-Com'09. His research interests include embedded systems, computer security, and wireless sensor networks.

Zhong Ming is a professor at College of Computer and Software Engineering of Shenzhen University. He is a member of a council and senior member of China Computer Federation. His major research interests are software engineering and embedded systems. He led two projects of National Natural Science Foundation, and two projects of Natural Science Foundation of Guangdong province, China.

Laurence T. Yang received B.E in Computer Science from Tsinghua University, China and Ph.D in Computer Science from University of Victoria, Canada. He is a professor in computer science and the director of the Parallel and Distributed Computing Lab, and Embedded and Ubiquitous Computing Lab at St. Francis Xavier University, Canada. His research interests include parallel and distributed computing, embedded and ubiquitous/pervasive computing. His research has been supported by NSERC and CFI of Canada.

Yongxin Zhu (Winson) is an Associate Professor with the School of Microelectronics, Shanghai Jiao Tong University, China. He is a senior member of China Computer Federation and a member of ACM/IEEE. He received his B.Eng. in EE from Hefei University of Technology, and M.Eng. in CS from Shanghai Jiao Tong University in 1991 and 1994 respectively. He received his Ph.D. in CS from National University of Singapore in 2001. His research interest is in computer architectures, embedded systems, medical electronics and multimedia.