IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 18, 2019, accepted August 7, 2019, date of publication August 9, 2019, date of current version August 26, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2934390

A Memory-Reinforced Tabu Search Algorithm With Critical Path Awareness for HW/SW Partitioning on Reconfigurable MPSoCs

ZHONGFU GUO, XINGMING ZHANG, AND BO ZHAO

China National Digital Switching System Engineering and Technological Reaearch and Development Center, Zhengzhou 450002, China

Corresponding author: Zhongfu Guo (gzf0309@mail.dlut.edu.cn)

This work was supported by the High Security Level Network Infrastructure Key Equipment Core Chip and Software Development under Grant 2017ZX01030301.

ABSTRACT Hardware/software (HW/SW) partitioning and scheduling are the crucial steps in HW/SW co-design. They have a strong effect on performance, area, power and the system itself. In this paper, a memory-reinforced tabu search algorithm with critical path awareness (MTSP) is proposed for solving the HW/SW partitioning problem. First, the critical path (CP) algorithm can locate the critical task queues and output a reduced task graph. Second, the solution to a heuristic algorithm (HA) is used as the initial solution. Third, by introducing hash technology, adding dual memory tables improves the search strength and effectiveness of the tabu search, and the experiment is completed by priority scheduling. MTSP especially has good performance in large task graphs, while it can greatly improve system performance, especially in the case of generating a large communication penalty. The experimental results show that the average improvement over the latest efficient hybrid algorithm is up to 5%. The improvement in algorithm searching time is 66% in comparison to the popular algorithms cited in this paper.

INDEX TERMS Hardware/software partitioning, task graph, heuristic method, tabu search algorithm, MPSoC.

I. INTRODUCTION

As the density of transistors increases, multiple processors system on chip (MPSoC) came into being in order to combat the power wall: with increased processor clock speeds for faster performance came increased power (and heat) output [1]–[3]. System on chip (SoC) platforms composed of microprocessors and FPGAs are called reconfigurable MPSoCs [4], which ensure flexibility and better design parameters. The general-purpose processor that implements software computing and the intellectual property (IP) core that implements hardware computing are collectively referred to as computing resources.

The traditional performance improvement method for MPSoC computing resources is the optimization of task partitioning [5], [6] and scheduling algorithms [7], [8]. However, this method has obvious limitations in two aspects. First, the scheduling algorithm itself has a very limited acceleration for task operation, but its overhead cannot be ignored. Second, the type and quantity of computing resources and the scheduling algorithm affect each other. Combinatorial optimization can obtain the optimal solution. Therefore, the hardware configuration of multicore platforms has caused extensive research.

Efficient techniques for HW/SW co-design [9]–[11] are necessary to realize embedded systems that must meet design constraints while satisfying the shorter time-to-market pressures [12]. HW/SW partitioning [9], [13], [14] is the crucial step during HW/SW co-design, and the HW/SW partitioning algorithm determines which components are implemented in hardware and which components are implemented in software [15]. First, it can guide the design and configuration of computing resources, reducing the overall power to achieve regional optimization; second, the system can be optimized to obtain the maximum acceleration.

HW/SW partitioning is based on Amdahl's law [16]. Twenty percent of the code consumes 80% of the time cost. By performing a small number of tasks in parallel, it combines other overhead costs (communication and memory I/O) analysis to benefit or not. In recent years, much

The associate editor coordinating the review of this article and approving it for publication was Khursheed Aurangzeb.

research has been performed on HW/SW partitioning, which can be divided into structure partitioning and functional partitioning [13], [17], [18]. Structure partitioning has more blocks, usually using functional partitioning.

The HW/SW partitioning algorithm can optimize multiple targets, such as minimizing power consumption [19], [20], hardware area [21], [22], and increasing the acceleration ratio [23]–[25]. Arato *et al.* categorized HW/SW partitioning problems into two types in [26]: a small part can be solved in polynomial time, usually using dynamic programming [27], integer linear programming [28], and accurate algorithms such as branch and bound [29]. Most of the remaining partitioning problems are NP-hard [30]. For NP-hard or larger partitioning problems, heuristic algorithms are the main topic of research.

Traditional heuristic algorithms include genetic algorithm (GA) [31], simulated annealing (SA) [32], and tabu search (TS) [33]. In [34], the comparison of TS, SA and GA algorithms proves the advantages of TS. There are also many hybridizations of heuristic algorithms, such as the greedy simulated annealing (GSA) algorithm combining the greedy and SA algorithms [35], the modified GA algorithm with an efficient crossover operator [36] and the re-excited particle swarm optimization (PSO) algorithm [37], [38] proposed an efficient heuristic algorithm refined by the TS algorithm based on the multiple-choice knapsack problem (MCKP). These hybrid algorithms all have good performance and are mostly used to optimize performance and reserve an extended area to determine the global optimal solution as much as possible.

These methods focus on optimizing the algorithm itself without considering the combination with the specific target platform. Given the differences in the collaborative environment and the lack of a common standard [39], the results obtained cannot be compared with others. This paper focuses on the algorithms applied to reconfigurable MPSoC [40]. Because it is difficult to determine the impact of performance metrics for hardware and software partitioning, this problem requires the introduction of a multiobjective optimization model [41]; Wang *et al.* [14] used an uncertain model and analyzed reconfigurable HW/SW partitioning issues.

This paper presents the MTSP algorithm on reconfigurable MPSoC for the HW/SW partitioning problem. First, the critical path algorithm is proposed. By locating the critical task queues and configuring the crossbar to reduce the communication penalty. Second, because the quality of the solution is significantly reduced when the TS algorithm searches a large task graph, the HA algorithm is introduced to provide an initial solution to improve the search efficiency. Hash technology adds a dual memory table to improve the search intensity, and the solution can match the well architecture of the reconfigurable MPSoC. Finally, the experiments verify the effectiveness of MTSP.

The rest of the paper is organized as follows. In Sect. 2, we introduce some definitions and the architecture utilized in this paper. In Sect. 3, we present the MTSP algorithm for HW/SW partitioning. In Sect. 4, we verify the effectiveness of our algorithm through experiments. Our discussions and performance comparisons are presented. In the final section, we conclude our research and analysis.

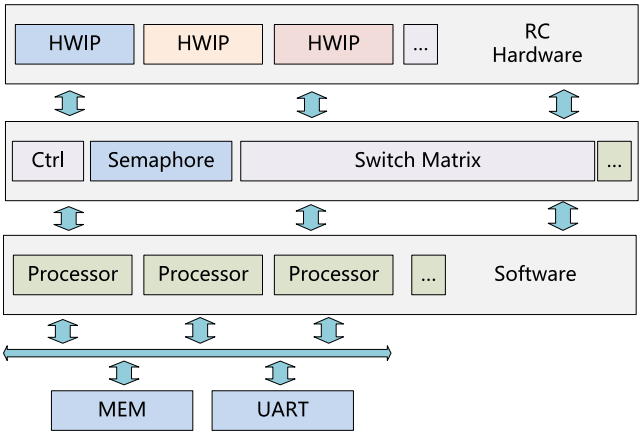


FIGURE 1. Overview of a reconfigurable MPSoC.

II. PRELIMINARIES

A. ARCHITECTURE OF RECONFIGURABLE MPSoCs

The target reconfigurable MPSoC architecture in this paper is illustrated in Figure 1. The algorithm can be compared with existing algorithms in a fair environment [14] without changing the reference architecture. The processing elements (PEs) include the processors and the hardware IP core. The system is built on programmable logic devices such as Xilinx FPGAs, such that the number of PEs for each type is configurable. The processors are isomorphic, and each processor can execute only one type of software task per unit time; each hardware IP core is heterogeneous, and each hardware computing unit is encapsulated into an IP core format and can perform only one hardware task belonging to a specific type of task set each time. In the platform to which the method is applied, in order to enable as many kinds of computing tasks as possible and provide acceleration, there is only one processor along with a plurality of hardware computing units. At the same time, in order to minimize the on-chip area consumed by the hardware platform, there is only one type of hardware IP core of each task type. In an attempt to meet the communication requirements between the processors and the FPGA, the two are connected by a crossbar switch to realize parallel execution of HW/SW tasks. Since sharing the same communication channel is bound to bring time loss, we use latency to quantify this penalty.

B. TASK GRAPH MODEL

Formally, the application to be partitioned is represented as task graph: a directed acyclic graph (DAG) where V is the set of nodes and E is the set of edges. Each node in V represents a task v. The execution time of each task $v \in V$ in software is s_v, displayed in the upper left corner of the corresponding node in G, and the hardware execution time is h_v, displayed

in the lower left corner of the corresponding node in G. Area penalty is displayed below the nodes of the task graph and the value on the edge represents the communication penalty. The dependencies between tasks are represented by edges in the task graph; for example, edge (u, v) indicates that task v depends on the execution of task u; therefore, u is called the predecessor task of v, and v is the successor task of u. The predecessor set of task v is $P(v)$, and the successor set is $S(v)$. The important notation is summarized in Table 1 (*Table* 1).

TABLE 1. Notation and their meanings.

Notation	Meaning
$P(v)$	All predecessor of task v;
$S(v)$	All successors of task v;
s_v	Execution time of task v on software;
h_v	Execution time of task v on hardware;
$c(u,v)$	Communication penalty of task v and u;
$c(P(v),v)$	Total penalty of task v with all its predecessors.
a_v	The area penalty of implementing task v.

For task graphs, if some tasks have no predecessors, these tasks are called start tasks; similarly, if some tasks have no successors, they are called termination tasks. In the DAG, if there are multiple start nodes, V_{start} is added as a predecessor of all the start nodes. Similarly, when there are multiple termination nodes, adding V_{end} after all the termination nodes ensures that the DAG has a unique start node and termination node, V_{start} and V_{end} execution time are 0, and the communication overhead with the task graph is cost free.

Simultaneously, [42] defines the granularity g(G) of the task graph G:

$$g(\mathrm{G}) = \frac{\min\limits_{v=V} s_v}{\max\limits_{u,v\in V} c(u,v)} \tag{1}$$

By definition, when $g(\mathrm{G}) \geq 1$, G is called coarse-grained task graph, all task graphs of this paper are coarse grain, and it is ideal attribute of task graphs.

We add some qualifications to the execution of the task graph on the target platform:

1) Each task in the task graph can be completed only by a specific PE, and the PE is selected according to the HW/SW partitioning algorithm.

2) The software PE executes one software task at a time, and the hardware PE can execute multiple tasks in parallel according to the area cost limit.

3) Once the task starts execution, it cannot be interrupted.

4) The high-speed and low-latency transmission can be realized by the configured crossbar switch, which can be regarded as cost free.

5) Communication time is the total time including read, write and so on.

Given task u, v, if task $u \in P(v)$ and task v is a software task, before task u be executed on software PE, $P(v)$ must have been completed. In order to check the status of the predecessor task, $c(P(v), v)$ is defined as the sum of the $P(v)$ communication penalties in this case. In the other case, if v is implemented on hardware PEs, then $c(P(v), v)$ is calculated by maximizing the communication cost. Formally, $c(P(v), v)$ is defined by:

$$c(P(v), v) = \begin{cases} \sum\limits_{u\in P(v)} c(u,v), & \text{if } u \text{ is an SW task} \\ \max\limits_{u\in P(v)} \{c(u,v)\}, & \text{if } u \text{ is an HW task} \end{cases} \tag{2}$$

We introduce the concept of *Pref* in [43] to evaluate the performance of the system:

$$Pref = \mathrm{Max}\{T_{\mathrm{soft}}, T_{\mathrm{hard}}\} + Pena \tag{3}$$

T_{soft} is the total execution time of the software task, $\mathrm{T_{hard}}$ is the total execution time of the hardware task, and *Pena* is the sum of the communication cost in the execution of all tasks.

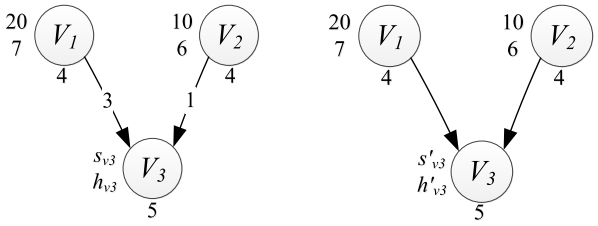


FIGURE 2. An example for task graph G' computing.

As shown in Figure 2, a task graph G with three nodes. After reading the task graph, removing the communication cost by adding it into the execution time of the task, considering tasks execution on software PEs are sequential communication with other tasks, while hardware PEs execute tasks can communicate concurrently. Accordingly, we can simplify the task graph to obtain G′, where s_v' and h'_v replace s_v and h_v in G, respectively:

$$\begin{cases} s'_v := s_v + \sum\limits_{u\in P(v)} c(u,v) \\ h'_v := \max\limits_{u\in P(v)} h_v + c(u,v) \end{cases} \tag{4}$$

Obviously, the new task graph G' retains all the features of the task graph, but the quantity of data is smaller, which is beneficial to the calculation of our partitioning algorithm.

For the execution of the task graph, we make the following assumptions:

1) The condition under which any task v_i can start execution is that $P(v_i)$ is executed, $t_{v_i date\_ready} = \max\limits_{u\in P(v_i)} (t_u + c(u, v_i))$, and the data ready time of the task vi is when the data of the predecessor task has been transmitted to the task node.

2) Tasks assigned to hardware can be executed in parallel.

3) Tasks assigned to the software need to be executed sequentially, depending on the task priority.

4) The priority of the software task: First, check *level*; the task of the previous level in the task graph is executed before the next level of the task can be started. Second, check $num_{\mathrm{successor}}$ for tasks at the same level. The higher the number

of subsequent nodes, the higher the priority. Third, check s_{vi}; tasks with short software execution time are preferred.

$$pri(v_i) = \{(level), (num_{\text{successor}}), (s_{vi})\} \tag{5}$$

C. PROBLEM P AND RELATED METHODS

The resolution of a problem requires the definition of a model representing all the important issues related to the specific problem [39]. In this paper, the input of the HW/SW partitioning algorithm is G′, the task nodes are mapped to the hardware and software PEs, and the algorithm output is represented by x_i, where $x_i \in (0, 1)$, $x_i = 0$ ($x_i = 1$) The computing task is implemented in SW (HW), $W(v_1, v_2, \ldots, v_n)$ represents the execution time of the computing task, and the area cost $A(a_1, a_2, \ldots, a_n)$ of the PEs provided to the corresponding task. The HW/SW partitioning problem discussed in this paper can be formulated as the following nonlinear minimization problem:

$$P := \begin{cases} \text{minimize Pref(G)}, \\ \text{subjuect to } \sum_{i=1}^{n} x_i a_i \leq A, \end{cases} \tag{6}$$

For a given area cost A, the partitioning result with the least task execution time is solved. The area cost of the task execution on software PE is negligible. For the convenience of this discussion, we regard it as cost free.

In the work of predecessors, the TS algorithm has proved that it is the optimal algorithm to solve the problem [34]. Therefore, there are many improved algorithms based on the TS algorithm, such as tabu search simulated annealing (TSSA) [43]. By generating the neighbors through the idea of SA, it can be accepted with a certain probability. The difference solution can effectively avoid the local optimal solution. TSSA has strong mountain climbing ability and combines the advantages of the two algorithms. The genetic algorithm tabu search (GATS) [44] provides the main framework of GATS and uses TS as the mutation operator. TS to search the entire solution space, so it has a strong ability to climb mountains while providing memory capabilities by tabu tables. Both algorithms have a certain degree of improvement over the TS algorithm.

In summary, there was no single approach of absolute advantage in both runtime and solution quality for problem P. Therefore, this manuscript proposes a novel approach to solve problem P.

--A novel critical path algorithm is proposed based on configurable crossbars to ensure that system performance greatly improves, even in the case of generation large communication costs.

--In particular, we combine the advantages of accurate algorithms and heuristic algorithms. A specific solution is obtained by an accurate algorithm, and the solution is used as an initial solution in the input TS algorithm, and the global optimal solution can be accurately searched with fewer iterations.

--Furthermore, our memory-reinforced TS algorithm maintains a dual memory table. In this way, our strategy can enhance the search directivity and eventually improve the solution quality.

Experimental results show that the algorithm is effective.

III. ALGORITHM FOR PARTITIONING

A. OVERVIEW OF PROPOSED MTSP ALGORITHM

In our method, three algorithms are executed sequentially to output the partitioning result. The input of MTSP is task graph G, and the critical path task graph (CG) is output through the critical path algorithm. As a local domain search algorithm, a good initial solution can effectively enhance the quality of the final solution of TS. The solution of the heuristic algorithm as the initial solution of the TS algorithm finally executes the TS algorithm to obtain the partitioning solution. An overview of the proposed MTSP algorithm is shown in Figure 3.

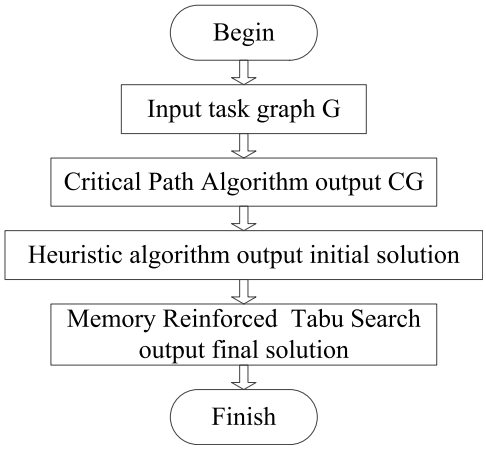


FIGURE 3. Flowchart of the proposed method.

B. CRITICAL PATH (CP) ALGORITHM

In our previous work [45], high-bandwidth, nonblocking, low-latency transmissions were achieved with configurable crossbars.

We selected some of the PEs that performed the same task queue. Such a task queue is called a task chain (TC), and the task chain is satisfied:

$$TC_1(V, E) \cap TC_1(V, E) \cap \ldots \cap TC_j(V, E) = \emptyset \tag{7}$$

Tasks within the TC, providing a communication interface through the Xilinx Fast Simplex Link (FSL) bus in a configurable crossbar that allows Microblaze to directly access the FIFO. We treat the interconnection between tasks in the same TC as a type of local communication with no time cost.

To locate the task chain, we use the critical path algorithm to traverse all the task queues from V_{start} to V_{end} and set them in descending order to the communication cost. Selected TC by the formula below:

$$TC_i = \{V_{start} \rightarrow V_i \rightarrow V_j \rightarrow \ldots \rightarrow V_{end} \,|\, \max c(v)\} \tag{8}$$

Task node N and edge E will be deleted in the task graph when they add to TC. The above steps will be executed

iteratively as soon as there is no path from V_{start} to V_{end}. Furthermore, traversing all the edges with nodes remaining in the task graph as the start and end nodes, the CP algorithm will repeatedly execute formula 7 until there is no edge with the task node as the vertex. All TCs will be output and algorithm terminates.

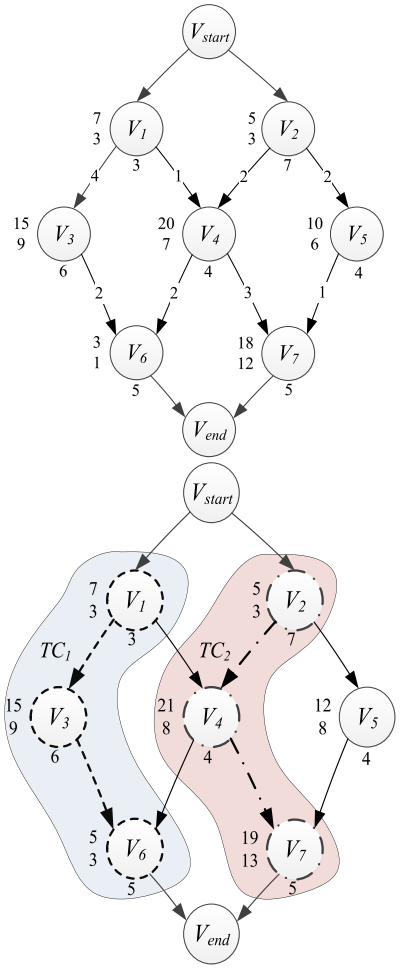


FIGURE 4. An example for DAG simplified as *CG'*.

As shown in Figure 4, it is the task graph simplification of radar signal processing. First, we obtain 6 task queues from V_{start} to V_{end}, calculate the communication cost of each task path, and set them in descending order to the communication cost. Select TC_1 in task queue by formula (7), nodes V_1, V_3, V_6 and edges V_1V_3, V_3V_6 will be deleted at the same time, repeat the above steps twice, only one task node V_5 remains in the task graph, and the algorithm terminates. Output: $TC_1 = \{V_1 \rightarrow V_3 \rightarrow V_6\}$ $TC_2 = \{V_2 \rightarrow V_4 \rightarrow V_7\}$, and finally, we convert CG into CG' according to formula (4). Considering that most of the task graphs have a large size, the whole algorithm maintains an *Open list* and a *Close list*.

The TC can be accurately found by algorithm1. TC will be indicated in CG, interconnection cost will be reduced by

Algorithm 1 CP Algorithm/*Critical Path Algorithm*/

Input: task graph G,
Output: TC, CG'
Begin
1 initialize v_H, *Open list*, *Close list*
/*Fill the task graph G into the open list*/
2 **for each** edge in *Open list* with at least two nodes **do**
  if (*Open list* exist task queue from V_{start} to V_{end})
    Compute the communication cost of task queue
    Choose v_{Hi} in task queue with maximum cost
    Add v_{Hi} and all the edges with v_{Hi} to close list
  else
    Set *Open list* node V_{Hi} as V_{start}
    While (exist edges with v_{Hi})
      Compute the communication cost of e_{Hi}
      Choose the edge e_{Hi} with maximum cost
      Add v_H and e_{Hi} to close list
    end while
  end for
3 Task chain ← E($v_{H1}, v_{H2}, \ldots, v_{Hj}$)
4 CG' ← Reduce CG
5 **Output** TC[1:j], CG'
End

formula (2) and output CG', the time complexity is O[$(m + n) \log n$], m and n represents the number of edges and nodes in the task graph.

C. HEURISTIC ALGOTITHM (HA)

The proposed algorithm utilizes the idea in solving the 0-1 knapsack problem. We initialize the solution of problem P: [0, 0 . . . , 0], all tasks are executed by software PEs, and the tasks are gradually transferred to be executed on hardware PEs through HA. Formally, let b_v denote the benefit of moving the task v to hardware [46], and T^v denote the set of software tasks (including v) lying in the same precedence level of the software task v.

$$b_v = \begin{cases} s'_v, & if\ h'_v \le \sum_{u \in T^v - \{v\}} s'_u, \\ \sum_{u \in T^v - \{v\}} s'_u - h'_v, & otherwise; \end{cases} \quad (9)$$

The area penalty of task executed on hardware PEs is different, and the total area penalty A of reconfigurable MPSoC is limited; efficiency e_v is defined and calculated as follows:

$$e_v = \frac{b_v}{a_v} \quad (10)$$

After observation of the CG', the benefit b_v corresponds to the profit in the knapsack problem. Obviously, the speed of the key nodes has a great impact on the task execution time. Therefore, task v_i with the highest e_{vi} value should be assigned to the hardware PE first.

The time complexity of HA is $O(2n)$.

Algorithm 2 HA/*Heuristic Algorithm*/

Input: CG′, A, $(a_{v1}, a_{v2}, \cdots a_{vn})$ for $(v_1, v_2, \cdots, v_n)$
Output: the heuristic solution [1:n]
Begin
1 $A_{used} = 0$
2 **for** i := 1 to n
  Solution [i]=0
  Calculate b_v, e_{vi}
3 **Repeat**
  Choose the biggest e_{vi}
  $A_{used} = A_{used} + a_i$
  Solution [i]=1
  Update b_v
  /*until no block fits for the residual hardware area, or all the task execute in hardware*/
4 **Output** the solution [1:n]
end

Algorithm 3 MTS /* Memory Reinforced Tabu Search*/

Input: S_{HA}-Initial solution generated by the algorithm HA;
Output: S_{best} -the best-so-far solution found by MTS;
/*q indicates the neighborhood size.*/
Begin
1 S_{local} := S_{HA}, **and** S_{best} := S_{HA};
2 **If** area A is enough for all tasks performed on the hardware PEs
  Then $S_{best} \leftarrow$ sol[1, 1, . . . , 1]
  goto: stop
3 **for** *iter*:=1 to $iter_{MAX}$
  begin
  3.1 Generate q neighbors $N(S)$ of S_{local}:$\{S_{c1}, S_{c2}, \ldots, S_{cq}\}$
  3.2 S_{min_neib}:=the neighbor with the minimal *Pref*($N(S)$)
  3.3 INT_{KEY}=*Hash*(S_{min_neib})
  3.4 **if** *Pref*(S_{min_neib}) < *Pref* (S_{best})
      then S_{local} := S_{min_neib}, and S_{best} := S_{min_neib}
    else begin
      delete $\{S_{currten_iteration}\}\in\{N(s)\cap$hash table$\}$
      if $\{S_{currten_iteration}\}\in\{$tabu list$\}$
        S_{local} := $N(S)$ with the minimal TD
      Else
        S_{local}:= the minimal *Pref*($N(S)$) **and** tabu list free
      end else
      end if
  3.5 **if** *Pref* (S_{local}) < *Pref* (S_{best})
      then S_{best} := S_{local}
    end if
  3.6 Reward the tabu list with *TD*, INT_{KEY}=*Hash*(S_{local});
  3.7 Update tabu list and hash table
  3.8 (Dealing with Conflict) /*if necessary*/
  end for
4 **output** S_{best}
end

D. MEMORY-REINFORCED TABU SEARCH ALGOTITHM (MTS)

TS is a heuristic local search algorithm proposed by Glover and Laguna [47]. The basic idea is to add the history of recent search movements to the tabu list during the search process to prevent the loop of the search process. This paper proposes a memory-reinforced tabu search algorithm applied to the partitioning problem P. A good initial solution can effectively improve the quality of the final solution of the TS algorithm, we use the solution generated in HA as the initial solution of MTS, and MTS maintains a tabu list with a maximum length of l that stores the forbidden movement. Considering that the partition solution in P is a sequence containing only 0 and 1, the movement S_c can be formulated as (11), let S_{local} denote solution for the current iteration, and S'_{local} denote the solution for last iteration.

$$S_c = S'_{local} \oplus S_{local} \tag{11}$$

The tabu list can prohibit l types of movement in the current iteration, and the tabu list is the FIFO queue. Whenever a new value is stored, the earliest stored value will be released. Tabu degree (TD) information will be saved according to the number of times added to the tabu list. The neighborhood search strategy divides the movement $\{S_c\}$ into three categories: not prohibiting $\{U\}$, prohibited but beneficial $\{FB\}$, prohibited but not beneficial $\{FN\}$, when $\{U\}\cup\{FB\} \neq \emptyset$, next iteration of moving $S_c \in \{U\} \cup \{FB\}$.

$S_c \in \{FB\}$ denotes prohibited solution has a good performance, it can 'break the law' to accept this movement, $S_c \in \{FN\}$ denotes all solution after moving within the neighborhood is prohibited and the performance is poor, we select the movement with smallest tabu degree as the S_c of the next iteration.

Considering that the tabu table saves only a short-term memory of the movement, search around the optimal solution neighborhood in the previous iteration. $iter_{MAX}$ denotes the maximum number of TS iterations. During $iter_{MAX}$ iterations, much historical information goes unutilized. Therefore, we added hash technology, which is a widely used search method. The data are converted into the keyword INT_{KEY} by compression and stored in a hash table. The hash table in this paper is equivalent to a long-term memory tabu table and directly determines the partitioning solution that has been searched. Keyword operations are performed by hashing functions. In this manuscript, the solution calculated in each iteration is converted to decimal value, and then the INT_{KEY} is obtained by the middle-square method.

When a solution is searched and the INT_{KEY} is the same as the value stored in the hash table, it is considered a conflict. The conflict resolution problem is a necessary prerequisite for the effective use of hash technology. We evaluate it through problem **P**. The better solution will be determined to be saved as corresponding INT_{KEY}. The movement with no improvement will be added to the tabu table.

In the process of MTS, S_{local} will be updated only when the performance of S_{local} is better than the S_{best} for the given condition. Movement for each time is two bits at random,

i.e., $\sum_{i=1}^{i=n} S_{ci} = 2$, where n is the task number of problem P. When the number of iterations reaches N times and still fails to update S_{best}, the search will be adjusted to $\sum_{i=1}^{i=n} S_{ci} = 1$. This process will be executed only once, the termination criterion is that the number of whole search iterations reaches M times, or when all tasks are transferred to hardware execution, the output S_{best} is the final solution the algorithm.

IV. RESULTS AND DISCUSSION

The proposed algorithms were simulated in C on an Intel(R) Core(TM) CPU@ 2.00 GHz processor with 8G memory. In order to make a fair comparison with TSSA and GATS, considering that these algorithms are heuristics rather than exact algorithms, a general test benchmark is necessary, and our implementations are based on the same type of task graph as used in [21]. All experiment is completed by our previous work priority scheduling [48].

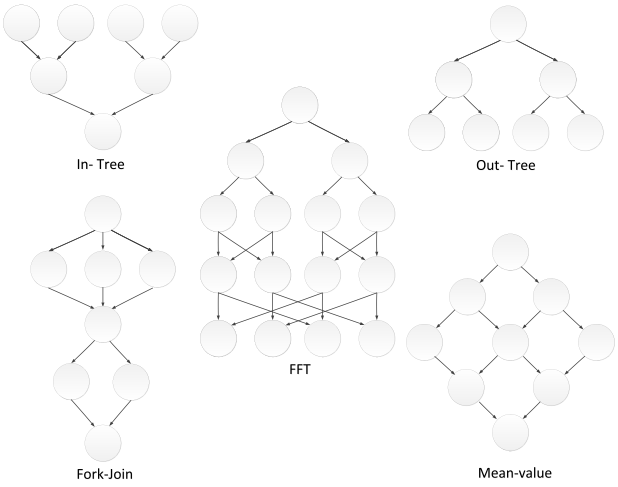


FIGURE 5. Five types of task graphs (DAGs).

The task graph used in this paper is uniform random generation of common structures by task graph for free (TGFF). The number of task nodes is configurable, and task topology includes in-tree, out-tree, fork-joint, mean-value analysis and fast Fourier transform (FFT) task graphs, as shown in Figure 5. The task graph generation parameters are shown in Table 2:

TABLE 2. Parameters defined in DAGs.

Parameters of DAG	Values
Range of software execution time	800 ~ 2000 (ns)
Range of hardware execution time	200 ~ 1200 (ns)
Range of hardware area	100 ~ 400 (unit)
Range of communication time(1)	2 ~ 100 (ns)
Range of communication time(2)	20 ~ 800(ns)

The communication cost can be classified into two cases, each of which follows a uniform random distribution in its interval. This manuscript introduces the concept of the communication-to-computation ratio (CCR). The value range of (1) is the computationally intensive task, corresponding to CCR_L. The value range of (2) is the calculation communication equalization task corresponding to CCR_H. The performance of the algorithm is fully compared for two different CCR test environments.

Assume that the area of hardware PEs required by task i is a_i, let A be the total area for all tasks to be executed by hardware in problem P, $A = \alpha^* \sum a_i$. in consideration of the given PEs are limited. α denotes the hardware coefficient by adjusting the coefficient $\alpha \in [0\%, 100\%]$ to simulate the case of allocating tasks on hardware PEs in an actual application, and the relevant parameters of the MTSP algorithm are shown in Table 3.

TABLE 3. Experiment parameters.

Description	Values
Neighborhood size	20
Maximum iteration number	2000
Non-improvement threshold	200
Length of tabu list	20

To measure the effectiveness of the algorithm, we define two metrics, the acceleration ratio AR and the improvement degree $ImpD$. We assume that the initial search task is executed on the software. We define $sol[0, 0 \ldots, 0]$ as the initial solution, while it has the maximum running time of the task.

$$\mathrm{AR} = \frac{\mathrm{pref(sol\,[0, 0, \ldots, 0])}}{\mathrm{pref(sol[x_1, x_2, \ldots, x_i])}} \tag{12}$$

Here, $ImpD$ is the improvement of algorithm A over algorithm B:

$$ImpD = \left(1 - \frac{(Algorithm\ \mathrm{A})}{(Algorithm\ \mathrm{B})}\right) \times 100\% \tag{13}$$

Let θ_1, θ_2 be the two types of $ImpD$ in our experiments, θ_1 denotes the improvement of the algorithm solution execution time, θ_2 denotes the improvement of the algorithm search time.

First, we investigated the effect of HA on the quality of the MTS solution. HA+MTS denotes MTS starting with the heuristic solution of the HA, RD+MTS denotes the trivial solution chosen as the initial solution of the MTS. The trivial solution is generated by randomly generating a solution for problem P under the limited conditions of area A. HA, RD+MTS and HA+MTS are compared in Figure 6, $\alpha = 50\%$, and the input task graph does not consider communication cost: $c(u, v) = 0, \forall(u, v) \in \mathrm{G}$, as the number of tasks increases, the AR of RD+MTS decreases, from 1.30 to 1.10, HA+MTS always approaches the optimal solution, and the solution of the HA is in the range [1.18, 1.26]. In conclusion, HA+MTS is the best among the three algorithms.

As shown in Figure 7, by adjusting the number of nodes in the task graph, θ_1 of the MTS with CP algorithm (CP+MTS) over MTS with different α, Figure 7a CCR_L is computationally intensive tasks condition, as increases with the number of tasks, θ_1 shows a downward trend when $\alpha = 0.1$ and $\alpha = 0.3$, and the overall decline is up to 66.08%. For $\alpha = 0.5$ and $\alpha = 0.7$, the $ImpD$ is relatively stable. In the case of $\alpha = 0.7$ and $N = 400$, $\theta_1 = 2.23\%$. For CCR_L, when α is small,

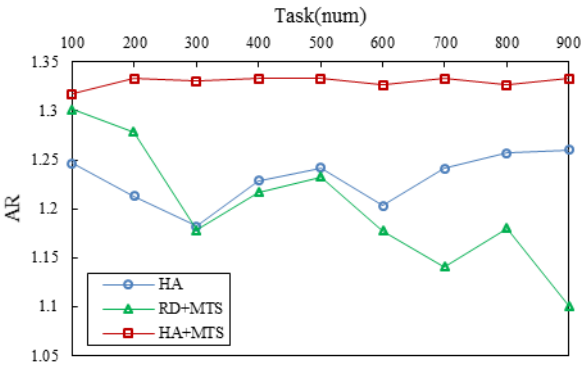
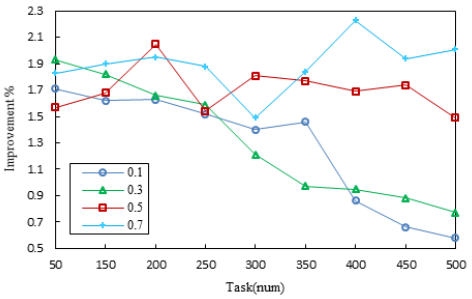
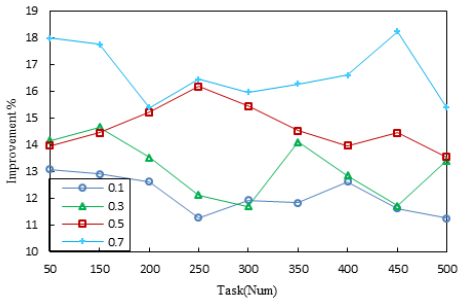


FIGURE 6. Solution comparisons for HA, RD+MTS and HA+MTS.



(a) CCR_L



(b) CCR_H

FIGURE 7. Improvement of CP+MTS over MTS with different values of α.

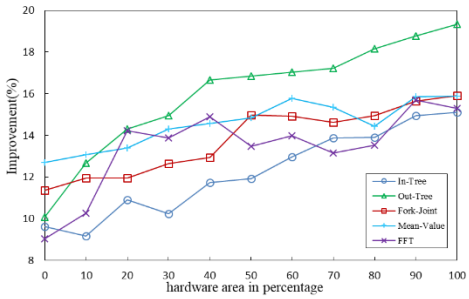
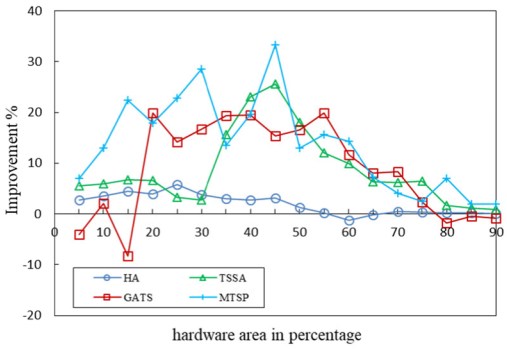
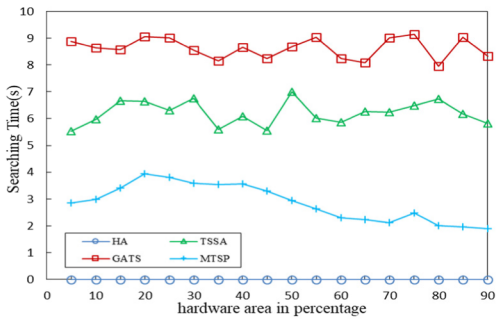


FIGURE 8. Improvement of CP+MTS over MTS on real application graphs.



(a) Solution quality



(b) Searching time

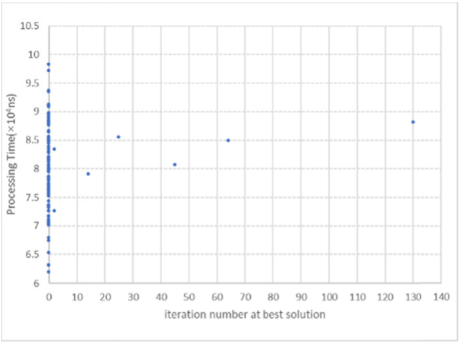
FIGURE 9. Comparisons of solution quality and searching time for the HA, TSSA, GATS and MTSP algorithms over TS, averaged over 10 random instances with 200 nodes.

the influence of communication penalty on the *AR* decreases; however, for larger α, the hardware PEs are sufficient, θ_1 is more stable. As shown in Figure 7b, θ_1 is positively correlated with α, with the increase in α, θ_1 increases significantly on average. In this comparison, the CP+MTS algorithm has obvious advantages and is more stable at CRR-H. In summary, CP+MTS is clearly superior to MTS.

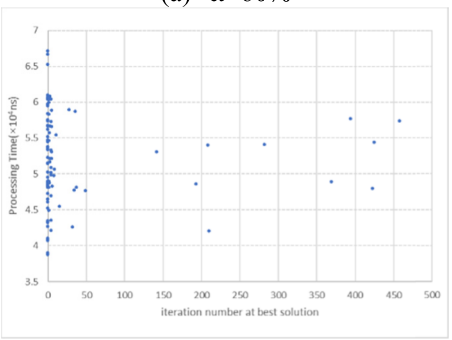
The task graph has different topology, and at the same time, the ratio of nodes to edges is different. It also affects the CP algorithm. The performance of the CP algorithm is analyzed for the task graphs of five different types of topologies. Figure 8 shows the improvements of CP+MTS over MTS for all types of task graphs considered. The communication penalty is set to CCR_H, task number N=100. With increases in α, θ_1 also increases gradually, the performance in various topological is quite different, the average value of θ_1 for out-tree is 16%, and for in-tree is 12.2%. The average number of successor nodes impacts the performance of CP+MTS, which is obviously reflected in the condition when $\alpha \in [0.3, 0.7]$. In other words, CP+MTS performs better when the node-to-edge ratio is larger.

Figure 9 shows the comparisons of solution quality produced by HA, TSSA, GATS and MTSP for the task graph with 200 nodes. When the value of hardware area in percentage α exceeds 80%, considering the solution quality of TS is good enough and it's hard to make great improvement. MTSP and TSSA keep positive returns, MTSP has the largest θ_1 in the range of $\alpha \in [0, 0.3]$, and hybrid algorithms all perform well in the range of $\alpha \in [0.35, 0.55]$. The average θ_1 of MTSP is 13% larger than TSSA (7%) and GATS (7%).

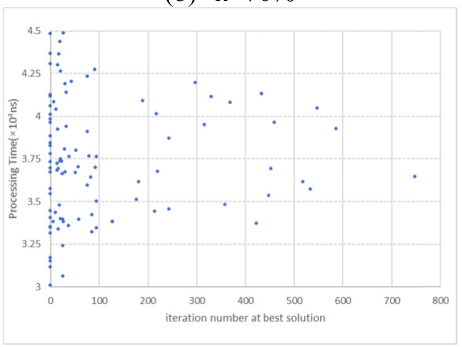
In the comparisons of searching time for algorithms, HA has the fastest searching time, less than 0.02 seconds; however, MTSP, TSSA, especially GATS, suffers from higher search time, and MTSP's search time tends to decrease with increasing α. The average θ_2 of MTSP over TSSA and GATS are 53% and 66%. Because MTSP maintains a dual memory table is more efficient in search, at the same time, when the solution does not improve over a period of time, the search is terminated. Therefore, the optimal solution can be obtained faster; however, GATS is based on GA, and the search speed is slower by maintaining a population to iterate. In summary, MTSP is clearly superior to the other approaches.



(a) α=60%



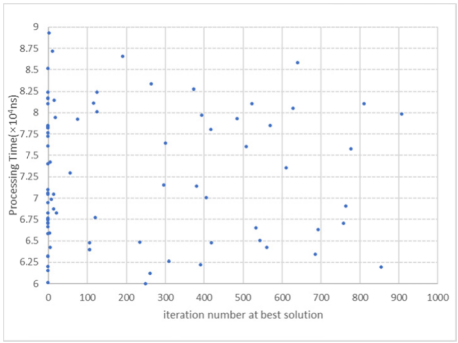
(b) α=70%



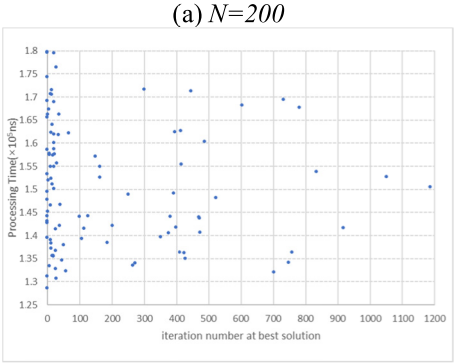
(c) α=80%

FIGURE 10. The distribution of iteration numbers MTSP arrived at the best solution in N=100 tasks, collected from 100 random instances for different values of α.

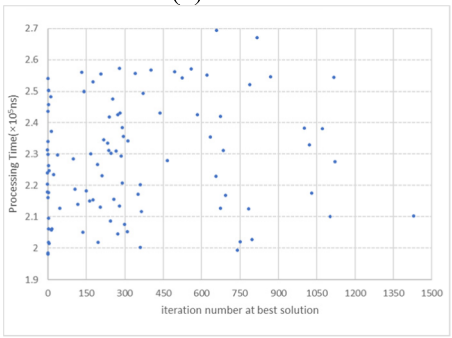
As shown in Figure 10, by changing the hardware area in percentage α and the number of tasks N, we record the iteration number MTSP arriving at the optimal solution for the task graphs with 100 nodes randomly generated. From Figure 10(a), the majority of cases obtain the best solution within 10 iterations; only 5% of the solution is obtained by more than 10 iterations but less than 140 iterations when the hardware area is 60%. According to the nonimprovement threshold, the final number of iterations of MTSP will not exceed 350. Figure 10b and 10c show the case of $\alpha = 70\%$ and $\alpha = 80\%$, respectively. The variance of the iteration number gradually increases, and the maximum value of the iteration number increases from 458 to 748. For Figure 10b, only 10% of the iteration number is obtained in the range of [100, 500]. However, for Figure 10c, the iteration number in the range of [100, 800] is 24%. This is because with the increase in α; the quality of the solution obtained by the HA algorithm decreases gradually; hence, MTSP requires more iterations to obtain the optimal solution.



(a) *N=200*



(b) *N=400*



(c) *N=600*

FIGURE 11. The distribution of iteration numbers MTSP arrived at the best solution with α=80%, collected from 100 random instances for different task number N.

For task graphs with more than 200 nodes, we enlarge the length of the tabu list and neighborhood size to 50. As shown in Figure 11, the available hardware area is set

to $\alpha = 80\%$, and as the number of task nodes increases, the variance of the iteration number value gradually increases. When N=200, 61% of the optimal solution iteration number is less than 100. When N increased to 600, only 40% of the optimal solution iteration number was less than 150. Considering that the quality of the initial solution is degraded with the increase in task node number, thanks to the dual memory table, MTSP can still obtain the optimal solution through a large number of iterations, while MTSP eliminates the search restriction [21] caused by the memory limitation in the large-scale task graph of the TS algorithm. When the task number=600, the maximum number of iterations is 1431; therefore, a maximum of 2000 iterations is quite enough to locate the optimal solution when N$\leq$600. Because of the nonimprovement threshold, the invalid search time is reduced. Figures 10 and 11 show that with increasing α and N, MTSP can locate the optimal solution for large-sized problems.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a memory-reinforced tabu search algorithm with critical path awareness for HW/SW partitioning on reconfigurable MPSoCs. The contributions of this work to field are as follows. First, we simplify the input task graph by reducing the quantity of data and retaining the complete information to improve processing efficiency. Second, through the critical path algorithm, the crossbar is configured according to the output task chain, and the communication penalty of the task graph is reduced. Third, the solution of an HA algorithm is used as the initial solution of the TS algorithm, which caters to the starting-point-sensitive requirements of TS search, significantly improving search efficiency. Fourth, by introducing hash technology, adding dual memory tables improves the search strength and effectiveness of the algorithm. The above technical details and strategies have fully utilized the system characteristics of MPSoCs, greatly reducing the runtime and significantly improving the quality of the solution.

Although numerous experiments confirm the effectiveness of MTSP, there are still many areas worthy of further research, such as making full use of partial reconstruction of FPGA and adopting effective methods to solve the dynamic partition problem; the proposed technique is designed for an MPSoC with a processor that executes only one task at a time, not allowing interruption. This kind of system unsuitable to support not only multithreaded software execution but also execution of the interrupt service routine (ISR). All these requirements in architecture lead us in future work to explore algorithms for HW/SW partitioning on different types of developed systems, combine this with more runtime task problems, etc.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their valuable and constructive comments.

REFERENCES

[1] S. Borkar, "Design challenges of technology scaling," *IEEE Micro*, vol. 19, no. 4, pp. 23–29, Jul./Aug. 1999.

[2] L. S. Umrao, D. P. Mahato, and R. S. Singh, "Recent trends in parallel computing," in *Encyclopedia of Information Science and Technology*, 3rd ed. Hershey, PA, USA: IGI Global, 2015, pp. 3580–3589.

[3] G. Stitt, F. Vahid, and S. Nematbakhsh, "Energy savings and speedups from partitioning critical software loops to hardware in embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 1, pp. 218–232, Feb. 2004.

[4] A. Dutta and M. Bayoumi, "Introducing a novel smart design framework for a reconfigurable multi-processor systems-on-chip (MPSoC) architecture," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, May 2016, pp. 1–3.

[5] H.-R. Li, F.-Z. He, and X.-H. Yan, "IBEA-SVM: An indicator-based evolutionary algorithm based on pre-selection with classification guided by SVM," *Appl. Math. A, J. Chin. Univ.*, vol. 34, no. 1, pp. 1–26, 2019.

[6] J. Zhou, J. Yan, J. Chen, and T. Wei, "Peak temperature minimization via task allocation and splitting for heterogeneous MPSoC real-time systems," *J. Signal Process. Syst.*, vol. 84, no. 1, pp. 111–121, Jul. 2016.

[7] W. Feng, S. Gu, Y. Yang, Q. Zhuge, and E. H.-M. Sha, "Efficient task assignment and scheduling on MPSOC with STT-RAM based hybrid SPMs considering data allocation," in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl. (ISPA/IUCC)*, Dec. 2017, pp. 794–800.

[8] Y. Zhou, F. He, and Y. Qiu, "Dynamic strategy based parallel ant colony optimization on GPUs for TSPs," *Sci. China Inf. Sci.*, vol. 60, no. 6, 2017, Art. no. 068102.

[9] X. Yan, F. He, N. Hou, and H. Ai, "An efficient particle swarm optimization for large-scale hardware/software co-design system," *Int. J. Cooperat. Inf. Syst.*, vol. 27, no. 01, 2018, Art. no. 1741001.

[10] J. L. Risco-Martín, S. Mittal, J. C. Fabero, P. Malagón, and J. L. Ayala, "Real-time hardware/software co-design using devs-based transparent M&S framework," in *Proc. Summer Comput. Simul. Conf.*, Jul. 2016, Art. no. 45.

[11] V. Migliore, M. M. Real, V. Lapotre, A. Tisserand, C. Fontaine, and G. Gogniat, "Hardware/software co-design of an accelerator for FV homomorphic encryption scheme using Karatsuba algorithm," *IEEE Trans. Comput.*, vol. 67, no. 3, pp. 335–347, Mar. 2018.

[12] J. Wu, T. Srikanthan, and G. Chen, "Algorithmic aspects of hardware/software partitioning: 1D search algorithms," *IEEE Trans. Comput.*, vol. 59, no. 4, pp. 532–544, Apr. 2010.

[13] I. Mhadhbi, S. B. Othman, and S. B. Saoud, "A comprehensive survey on hardware/software partitioning process in co-design," *Int. J. Comput. Sci. Inf. Secur.*, vol. 14, no. 3, p. 263, Mar. 2016.

[14] R. Wang, W. N. Hung, G. Yang, and X. Song, "Uncertainty model for configurable hardware/software and resource partitioning," *IEEE Trans. Comput.*, vol. 65, no. 10, pp. 3217–3223, Oct. 2016.

[15] N. Hou, F. He, Y. Zhou, Y. Chen, and X. Yan, "A parallel genetic algorithm with dispersion correction for HW/SW partitioning on multi-core CPU and many-core GPU," *IEEE Access*, vol. 6, pp. 883–898, 2017.

[16] J. Nutaro and B. Zeigler, "How to apply Amdahl's law to multithreaded multicore processors," *J. Parallel Distrib. Comput.*, vol. 107, pp. 1–2, Sep. 2017.

[17] J. I. Hidalgo and J. Lanchares, "Functional partitioning for hardware-software codesign using genetic algorithms," in *Proc. 23rd EUROMICRO Conf., New Frontiers Inf. Technol.*, Sep. 1997, pp. 631–638.

[18] N. Hou, X. Yan, and F. He, "A survey on partitioning models, solution algorithms and algorithm parallelization for hardware/software co-design," *Design Autom. Embedded Syst.*, vol. 23, nos. 1–2, pp. 57–77, 2019.

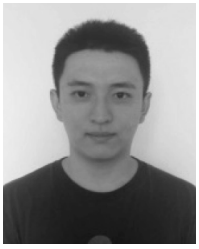
[19] E. Sha, L. Wang, Q. Zhuge, J. Zhang, and J. Liu, "Power efficiency for hardware/software partitioning with time and area constraints on MPSoC," *Int. J. Parallel Program.*, vol. 43, no. 3, pp. 381–402, 2015.

[20] S. B. H. Hassine, M. Jemai, and B. Ouni, "Power and execution time optimization through hardware software partitioning algorithm for core based embedded system," *J. Optim.*, vol. 2017, Feb. 2017, Art. no. 8624021.

[21] W. Jigang, T. Srikanthan, and T. Jiao, "Algorithmic aspects for functional partitioning and scheduling in hardware/software co-design," *Des. Autom. Embedded Syst.*, vol. 12, no. 4, pp. 345–375, 2008. doi: 10.1007/s10617-008-9032-0.

[22] A. Iguider, O. Elissati, A. En-Nouaary, and M. Chami, "HW/SW Partitioning Algorithms for Multi-objective Optimization in Embedded Systems," *Int. J. Inf. Sci. Technol.*, vol. 2, no. 2, pp. 19–28, 2019.

[23] J. Bjärmark and M. Strandberg, "Hardware accelerator for duo-binary CTC decoding: Algorithm selection, HW/SW partitioning and FPGA implementation," M.S. thesis, Dept. Electron. Eng. Linköping Univ., Linköping, Sweden, 2006.
[24] D. D. Gajski, F. Vahid, S. Narayan, and J. Gong, "SpecSyn: An environment supporting the specify-explore-refine paradigm for hardware/software system design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 6, no. 1, pp. 84–100, Mar. 1998.
[25] Anirudh B. K, V. Venkatraman, A. R. Kumar, and S. David S, "Accelerating real-time computer vision applications using HW/SW co-design," in *Proc. Int. Conf. Comput., Commun. Electron.*, Jul. 2017, pp. 458–463.
[26] P. Arató, Z. A. Mann, and A. Orbán, "Algorithmic aspects of hardware/software partitioning," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 10, no. 1, pp. 136–156, Jan. 2005. doi: 10.1145/1044111.1044119.
[27] J. Wu and T. Srikanthan, "Low-complex dynamic programming algorithm for hardware/software partitioning," *Inf. Process. Lett.*, vol. 98, no. 2, pp. 41–46, 2006. doi: 10.1016/j.ipl.2005.12.008.
[28] W. Wolf and J. Staunstrup, *Hardware/Software CO-Design: Principles and Practice*. Norwell, MA, USA: Kluwer, 1997.
[29] W. Jigang, B. Chang, and T. Srikanthan, "A hybrid branch-and-bound strategy for hardware/software partitioning," in *Proc. 8th IEEE/ACIS Int. Conf. Comput. Inf. Sci.*, Jun. 2009, pp. 641–644.
[30] D. Corus, P. S. Oliveto, and D. Yazdani, "Artificial immune systems can find arbitrarily good approximations for the NP-hard partition problem," in *Proc. Int. Conf. Parallel Problem Solving Nature*. Coimbra, Portugal: Springer, Aug. 2018, pp. 16–28.
[31] S. G. Li, F. J. Feng, H. J. Hu, C. Wang, and D. Qi, "Hardware/software partitioning algorithm based on genetic algorithm," *JCP*, vol. 9, no. 6, pp. 1309–1315, 2014.
[32] M. C. Bhuvaneswari and M. Jagadeeswari, "Hardware/software partitioning for embedded systems," in *Application of Evolutionary Algorithms for Multi-objective Optimization in VLSI and Embedded Systems*. New Delhi, India: Springer, 2015, pp. 21–36.
[33] N. Hou, F. He, Y. Zhou, and H. Ai, "A GPU-based tabu search for very large hardware/software partitioning with limited resource usage," *Int. J. Cooperation Inf. Syst.*, vol. 11, no. 5, 2017, Art. no. JAMDSM0060.
[34] T. Wiangtong, P. Y. K. Cheung, and W. Luk, "Comparing three heuristic search methods for functional partitioning in hardware– Software codesign," *Des. Automat. Embedded Syst.*, vol. 6, no. 4, pp. 425–449, 2002.
[35] Y. Jing, J. Kuang, J. Du, and B. Hu, "Application of improved simulated annealing optimization algorithms in hardware/software partitioning of the reconfigurable system-on-chip," in *Proc. Int. Conf. Parallel Comput. Fluid Dyn.* Berlin, Germany: Springer, 2013, pp. 532–540.
[36] R. Faraji and H. R. Naji, "An efficient crossover architecture for hardware parallel implementation of genetic algorithm," *Neurocomputing*, vol. 128, pp. 316–327, Mar. 2014.
[37] M. B. Abdelhalim and S. E.-D. Habib, "An integrated high-level hardware/software partitioning methodology," *Des. Autom. Embedded Syst.*, vol. 15, no. 1, pp. 19–50, 2011. doi: 10.1007/s10617-010-9068-9.
[38] J. Wu, Q. Sun, and T. Srikanthan, "Algorithmic aspects for multiple-choice hardware/software partitioning," *Comput. Oper. Res.*, vol. 39, no. 12, pp. 3281–3292, 2012. doi: 10.1016/j.cor.2012.04.013.
[39] M. López-Vallejo and J. C. López, "On the hardware-software partitioning problem: System modeling and partitioning techniques," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 8, no. 3, pp. 269–297, 2003.
[40] C. Wang, X. Li, Y. Chen, Y. Zhang, O. Diessel, and X. Zhou, "Service-oriented architecture on FPGA-based MPSoC," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2993–3006, Oct. 2017.
[41] M. C. Bhuvaneswari, Ed., and G. Subashini, "Introduction to multi-objective evolutionary algorithms," in *Application of Evolutionary Algorithms for Multi-objective Optimization in VLSI and Embedded Systems*. New Delhi, India: Springer, 2015, pp. 1–20.
[42] O. Sinnen, *Task Scheduling for Parallel Systems*. Hoboken, NJ, USA: Wiley, 2007.
[43] P. Liu, J. Wu, and Y. Wang, "Hybrid algorithms for hardware/software partitioning and scheduling on reconfigurable devices," *Math. Comput. Model.*, vol. 58, nos. 1–2, pp. 409–420, 2013. doi: 10.1016/j.mcm.2012.11.001.
[44] L. Li and M. Shi, "Software-hardware partitioning strategy using hybrid genetic and tabu search," presented at the Int. Conf. Comput. Sci. Softw. Eng., 2008.
[45] Z. Guo, X. Zhang, and G. Yanzhao, "A configurable high speed cross-bar network on FPGA," *J. Phys., Conf. Ser.*, vol. 1237, no. 4, 2019, Art. no. 042010.
[46] C. Li, X. Li, C. Wang, X. Zhou, and F. Zeng, "A dependency aware task partitioning and scheduling algorithm for hardware-software codesign on MPSoCs," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.* Berlin, Germany: Springer, 2012, pp. 332–346.
[47] F. Glover and M. Laguna, "Tabu search," in *Handbook of Combinatorial Optimization*. Boston, MA, USA: Kluwer, 1998, pp. 2093–2229.
[48] Z. Guo, X. Zhang, and B. Zhao, "MPSOC workflow scheduling method and scheduling trigger mechanism based on virtual microscope technology," *Acta Microscopica*, vol. 28, no. 2, pp. 215–225, 2019.



ZHONGFU GUO was born in 1994. He received the B.S. degree from the Dalian University of Technology, China, in 2016, and the M.S. degree from PLA Information Engineering University, Zhengzhou, China, in 2018. He is currently pursuing the Ph.D. degree with the China National Digital Switching System Engineering and Technology Research Center, Zhengzhou. His research interests include multiple processors system-on-chip (MPSoC), hardware/software co-design, and heuristic methods.

XINGMING ZHANG was born in 1963. He received the B.S. and M.S. degrees from PLA Information Engineering University, Zhengzhou, China, in 1985 and 1989, respectively, and the Ph.D. degree from the China National Digital Switching System Engineering and Technology Research Center, Zhengzhou. He is currently a Professor with the Provincial Key Laboratory of System on a Chip, National Digital Switching System Engineering and Technology Research Center. He has authored over 100 journal and conference publications. His research interests include network security, signal processing, hardware/software co-design, broadband networks, and deep learning.

BO ZHAO was born in 1981. He received the B.S. and M.S. degrees from PLA Information Engineering University, Zhengzhou, China, in 2003 and 2007, respectively, and the Ph.D. degree from the China National Digital Switching System Engineering and Technology Research Center, Zhengzhou. He is currently an Associate Professor with the Provincial Key Laboratory of System on a Chip, National Digital Switching System Engineering and Technology Research Center. He has authored over 20 journal and conference publications. His research interests include network security, mimetic defense architecture, and hardware/software co-design.