The Journal of Supercomputing (2023) 79:7244–7271
https://doi.org/10.1007/s11227-022-04955-w

Efficient tasks scheduling in multicore systems integrated with hardware accelerators

Jinyi Xu[1,2,3] · **Hao Shi**[1,3] · **Yixiang Chen**[1,3]

Accepted: 12 November 2022 / Published online: 27 November 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract

Multicore systems integrated with hardware accelerators provide better performance for executing real-time applications in time-critical fields, such as robots, avionics, and aerospace. The integration of hardware accelerators brings new challenges for system scheduling; the software scheduling problem is extended to a hardware–software co-scheduling problem. Efficient co-scheduling strategy maximizes the benefits of hardware acceleration, which is important for time-critical systems. To solve this problem, we propose a co-scheduling strategy to minimize the system execution time. It combines hardware–software resource allocation and a real-time schedule method. Our scheduling can fit the different parallel in software and hardware (e.g., CPUs and FPGAs). The key component of our strategy is its novel hardware–software resource allocation and a high-performance heuristic scheduling algorithm. In the experiments, we evaluate our approach using both simulated and real parallel applications. The results illustrate that our algorithm obtains efficient solutions within reasonable runtimes compared to the state of the art.

Keywords Multicore systems · Task scheduling · Hardware accelerators · Real-time applications

✉ Yixiang Chen
yxchen@sei.ecnu.edu.cn

Jinyi Xu
52184501010@stu.ecnu.edu.cn

Hao Shi
51194501091@stu.ecnu.edu.cn

[1] Software Engineering Institute, East China Normal University, No. 3663, North Zhongshan Road, Shanghai 20062, China

[2] Taran, IRISA INRIA, 35000 Rennes, France

[3] MOE Engineering Research Center for Software/Hardware Co-Design Technology and Application, East China Normal University, No. 3663, North Zhongshan Road, Shanghai 20062, China

1 Introduction

Multicore systems integrated with hardware accelerators, such as field-programmable gate arrays (FPGAs), provide faster data processing and lower power consumption [1–3]. Efficient schedules that take into account both hardware and software components are critical to the performance of such hardware-accelerated multicore systems. Typical scheduling approaches consider optimizations within the software stack such as EDF (earliest deadline first), RMS (rate monotonic), and LLF (least laxity first) [4–6]. However, many of these are not suitable or applicable to the hardware components as well. In this paper, we propose a new hardware–software scheduling algorithm for scheduling applications in the hardware-accelerated multicore systems. Throughout this paper, when we use the term *software*, we denote to *the execution units in traditional systems* without any hardware accelerator (e.g., CPUs on multi-core systems). To this same end, when we use the term *hardware*, we refer to *the execution units in the hardware accelerator* itself (e.g., FPGAs).

The hardware–software co-scheduling problem is challenging because the *hardware* and *software* components have different computational logic when executing tasks. Consider the following example to illustrate the difference between software execution and hardware execution. As shown in Fig. 1a, after the tasks *T*1 and *T*2 have completed, the three tasks *T*3, *T*4 and *T*5 are ready for execution. In a multicore system as Fig. 1b, *T*5 can only be scheduled *after T*3 or *T*4 completes, because each software core can only execute one task at a time. Contrast this with the way that the same tasks can be executed on the hardware accelerator, as shown in Fig. 1c. Here, there are different resources (i.e., execution units) for each task shown in boxes. Consequently, task *T*5 can be scheduled *at the same time* as tasks *T*3 and *T*4 because they do not use the same resources. From this,

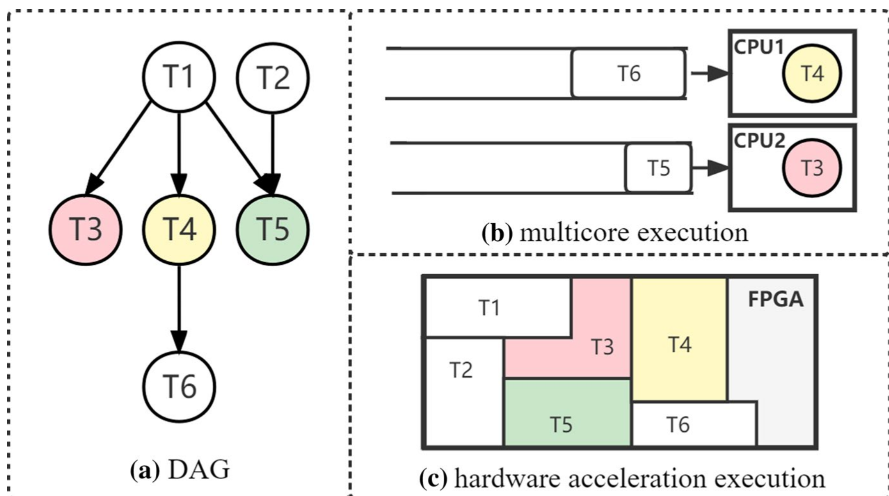


Fig. 1 When task *T*3, *T*4, and *T*5 are released: **b** in two-core system, *T*3 and *T*4 can execute at the same time and *T*5 waits in the queue. **c** Using hardware acceleration, all the tasks *T*3, *T*4, and *T*5 can execute at the same time, but they consume non-reusable FPGA resource

we can see that the number of software cores is a limiting factor on the number of tasks that can be executed in parallel on the software. In hardware acceleration, there is no upper limit to the number of parallel tasks, but the number of tasks is bounded by the size of hardware resources. Unlike typical scheduling problems that only consider the scheduling of tasks on software cores, in hardware-accelerated multicore (i.e., *both* hardware and software) systems, we need to incorporate hardware execution into the scheduling problem.

In the general multicore scheduling problem, real-time applications are always represented by a directed acyclic graph (DAG) consisting of a series of tasks, where nodes represent tasks and edges represent data dependencies. And each real-time task has its own time requirements, such as execution time, release time, and deadline. The multicore platform consists of multiple software cores. The scheduling problem is to find the order and assignment of task on the multicore platform that will achieve the goal. The goals of scheduling problem include, for example, time optimization [7], energy usage reduction [8–13], minimizing the number of computing units [14, 15], and so on. Various strategies have been exploited to minimize the overall completion time (i.e., the makespan). Most renowned strategies can be divided into three categories: list scheduling, task duplication, and other heuristics strategies.

List scheduling is that it maps tasks on a set of pre-defined processors by calculating a priority list of tasks [16–19]. Task duplication method is another popular scheduling strategy that focuses on large-scale multicore systems. It reduces the makespan by executing tasks copies on different processors to reduce the inter-processor communication cost. These algorithms show better performance in scenarios with higher communication costs (i.e., higher computation-to-communication ratio) [20–25]. There are also some other heuristics strategies, for example, weight-based mechanism [26] and genetic algorithms [27, 28].

In this paper, we focus on finding the task scheduling solution in a hardware-accelerated multicore system. Our goal is to obtain the shortest makespan under the hardware resource constraints. We propose an efficient software–hardware co-scheduling strategy (ESHCS).

Our strategy consists of two phases: allocation and scheduling. In the allocation phase, our goal is to efficiently maximize the advantages of hardware parallelism by improving the utilization of limited hardware resources. We cluster tasks into different groups and use linear programming to obtain allocation results under hardware resource constraints. In the scheduling phase, we propose a list-based heuristic strategy to assign pre-allocated tasks to the suitable software cores or hardware acceleration by considering the computation capabilities of cores as well as the real-time requirements of the tasks.

Finally, we evaluate our algorithm by experiments. Our results confirm that ESHCS can obtain the efficient solutions within a reasonable runtime. Compared to prior work [29], ESHCS is more efficient. It greatly improves the running time at the expense of very little speedup performance.

In summary, the contributions of this work are as follows:

- We construct a model for multicore platforms integrated with hardware accelerators based on hardware and software features and give a description of the task features required for hardware–software co-scheduling.
- We propose an effective allocation strategy derived from the idea of linear programming. Compared to linear programming, our method achieves a good allocation results in less time.
- We propose an allocation-based scheduling algorithm. Compared to list scheduling and genetic scheduling, our method obtains a reasonable makespan in the shortest time.
- We present the results of our experiments using both simulation applications and real-world applications. Our algorithm performs the best in both applications.

The rest of this paper is organized as follows. In Sect. 3, we define the hardware–software co-scheduling problem and the related terminology. In Sect. 2, we discuss the related work. And in Sect. 4, the proposed scheduling approach is presented. Section 5 shows the experiments and analysis of our algorithm. Finally, Sect. 6 discusses the conclusions of this research and our future work.

2 Related work

Although multicore scheduling algorithms cannot be directly applied to hardware-accelerated multicore architectures, we can incorporate aspects of these approaches. For instance, heterogeneous earliest finish time (HEFT) is a well-known list scheduling, which calculates the upward rank of the task to set the priorities of the tasks [16]. Critical-path-on-a-processor (CPOP) uses the sum of upward and download rank values for prioritizing. The objective of these two algorithms is to minimize the makespan at lower cost [16]. Heterogeneous edge and task scheduling (HETS) minimize the communication overhead edges to obtain reduced schedule length [18]. Massinissa et al. [28] proposed a genetic approach to scheduling the applications under energy constraints. Du et al. [19] designed a feature-aware list scheduling to assign the tasks to the appropriate processors. The feature they considered includes frequency, data size, and resources utilization.

In addition to the above works, allocation and scheduling for FPGA-based systems could help inform our work. Several approaches have been presented in previous works including optimizing frameworks [30, 31], heuristic algorithms [32], priority-driven algorithms [19], genetic algorithms [33–35], feature extraction [36], etc. Lam et al. proposed two hardware/software partitioning algorithms: a customized Tabu search algorithm and a dynamic programming algorithm. The first algorithm is able to produce good approximate solutions quickly, while the second one yields more accurate solutions in smaller-scale instances [37]. These two algorithms are designed for tree-like task graphs. Chen et al. [38] proposed a multi-model multi-task learning approach for heterogeneous hardware–software co-design. It reduces overall power consumption and critical path latency. Reza presented a

heuristic-based dynamic scheduling technique to schedule task graphs on multi-FPGA systems [39]. Matteo focuses on the efficient scheduling onto cloud FPGAs to minimize the makespan [40]. These two works focus only on scheduling in multiple hardware, without considering the collaboration of hardware and software. Furthermore, Rodriguez et al. [31] designed a HAP scheduler to minimize the energy consumption in the CPU + FPGA systems. Du et al. [19] proposed a speedup estimation model based on the speedup of tasks on FPGA. They used this speedup estimation as well as the runtime resource load balancing strategy to assign the tasks to the appropriate computing cores. Dai et al. [36] proposed a benefit-based scheduling metric to evaluate the task assignment. Based on the metric, they accelerate task execution. This work targets on independent tasks without communication. For the applications with high communication load, Hao proposed an efficient scheduling algorithm using task duplication [41]. However, all these works are designed and proven to get great performance in nonreal-time applications and assume no limitation in hardware resources. For real-time applications, Zhu et al. [30] focus on improving the scheduling efficiency of CPU + FPGA architecture and proposed a scheduling framework for independent tasks. In our previous work [29], we proposed a scheduling approach, called ReTPA, for FPGA-based multicore systems. This work was our first exploration of the real-time hardware–software co-scheduling problem. We further propose this efficient hardware–software co-scheduling algorithm in this paper, which better balances performance and time overhead.

We specifically introduce three representative works: a classic priority-driven approaches HEFT [16], two advanced genetic approaches, GAA and MGAA [33] and our algorithm, ReTPA, in 2022 [29]. The comparison of our algorithms and these algorithms is presented in Sect. 5.

2.1 Heterogeneous earliest-finish-time (HEFT)

The HEFT algorithm proposed by Topcuoglu et al. [16] is a classic multicore scheduling algorithm. Their objective is to simultaneously minimize the makespan and meet low scheduling costs. The multicore structure used in this paper consists of different software cores and involve communication cost. HEFT assigns the task with the highest upward rank value according to an insertion-based policy. The experiments demonstrate the HEFT algorithm shows an impressive performance in terms of both quality and cost of schedule. To compare with our algorithms, we firstly use linear programming or greedy to allocate tasks into software and hardware and then use HEFT for task scheduling. We use LHEFT or GHEFT to denote the HEFT with linear programming or greedy strategy, respectively.

2.2 Genetic algorithm approach (GAA) and modified genetic algorithm approach MGAA

The GAA and MGAA algorithm are two CPU + FPGA scheduling algorithms, proposed by Abdallah et al. [33]. They are two novel GA-based approaches considering the communication cost. GAA assigns the execution sequence of tasks by proposed

strategy A or B and then uses the genetic algorithm to find the best cores for each task. MGAA exploits further the search space to find better solutions by creating different successful sequences in a pre-treatment. They are proven to be capable of finding good solutions while improving the running time compared to other existing works.

2.3 Real-time priority-driven algorithm (RePTA)

The ReTPA algorithm for real-time applications in FPGA-based multicore systems is proposed in [29]. This work proposed a two-step scheduling strategy: allocation and scheduling. In that algorithm, first we use a linear programming approach to obtain the hardware–software allocation results. And then, we proposed a priority-driven scheduling to assign the task based on the allocation results. The performance comparisons illustrate that ReTPA shows the good performance for computation-intensive applications. But there are some shortcomings, for example, linear programming can lead to the optimal solution in NP-hard problem but at a significant time cost. In the scheduling phase, priority calculation can be upgraded.

3 A novel model

The notation $\alpha \mid \beta \mid \gamma$ is used to describe scheduling problems, where α denotes the environment, β denotes properties of the tasks, and γ denotes the goal. Our hardware–software co-scheduling problem is described as $P \mid prec, c_{\mathrm{ij}}, time_{\mathrm{real}} \mid C_{\max}, RSC_{\mathrm{con}}$. P is the set of computing resources (i.e., CPU, FPGA). $prec, c_{\mathrm{ij}}, time_{\mathrm{real}}$ means the tasks are non-independent and real time. The communication costs between tasks are considered. $C_{\max}, RSC_{\mathrm{con}}$ means the goal is to minimize the makespan under the resource constraints.

In this work, the application is aperiodic and is represented by a directed acyclic graph, $G = (V, E)$, where V is the set of v tasks and E is the set of e edges. Each edge $e(i, j)$ represents the precedence constraint such that task τ_i should complete its execution before task τ_j starts. A $v \times v$ matrix $DATA$ is used to present communication data, where $data_{i,j}$ is the amount of data that needs to be transferred from task τ_i to task τ_j. Task execution of a given application is assumed to be non-preemptive and unsuspendable.

The target computing environment consists of a series of software cores and hardware resources, where software cores are represented by $p_1, p_2, \dots, p_q$ and hardware accelerator FPGA is considered as a special computing core represented by p_0. Computing cores are connected in a fully connected topology. In our model, all inter-core communications are assumed to perform without contention. Communication only occurs at the end of tasks.

In addition, there are three very important features that we should characterize in detail:

- Time features of tasks

  The real-time application consists of a series of tasks with their time features: release time, computation cost, and deadline. W is a $v \times 2$ computation cost matrix in which each $w_{i,1}$ gives the estimated execution time to complete task τ_i on software and $w_{i,2}$ gives the estimated execution time to complete task τ_i on hardware. Two v-dimensional vectors RLS and DDL represent release time and deadline, respectively. The release time of a task is the instant of time when the task becomes available for execution. The task can be scheduled and executed at any time at or after its release time, whenever its data dependency conditions are met. If task τ_i can be executed when the system begins execution, it has no release time, hence $RLS_i = 0$. The deadline of a task is the instant of time when its execution is required to be completed. If task τ_i has no deadline, it can be completed at any time during the system execution.
- Computing resources

  The hardware-accelerated multicore platform includes software and hardware resources. The software resource is the number of software cores, represented by q. The hardware resources are FPGA, which composes of look-up table (LUT), registers and flip-flops (FFs). A LUT cannot be deployed to two tasks. Therefore, the number of LUTs limits the total number of tasks in hardware. We consider it as a resource constraint when scheduling the hardware-accelerated multicore system, represented by LUT_{con}. We use a v-dimensional vector LUT to present the number of LUTs for tasks, where LUT_i is the number of hardware resources required to execute task τ_i in hardware.
- Communication cost

  The data transfer rates depends on the architecture of the communication links, such as shared bus, hierarchical bus, and network on chip. The data transfer rates between cores are stored in matrix $BAND$ of size $(q+1) \times (q+1)$, where $BAND_{k,l} = BAND_{l,k}$ for computing cores p_k and p_l. We define the communication cost $c_{i,j}$ when task τ_i executed on computing unit p_k and task τ_j executed on computing unit p_l as

$$c_{i,j} = \frac{data_{i,j}}{BAND_{k,l}} \tag{1}$$

  We assume that the cost of intra-core communication is negligible compared to the cost of inter-core communication. When $k = l$, $c_{i,j}$ is zero.

Our goal is to find a solution for the real-time application to (I) minimize the makespan $C_{\max}$; (II) ensure each task τ_i meets its release time RLS_i and deadline

DDL_i; (III) ensure the actual hardware resources consumption LUT_{sum} is less than or equal to the hardware resource constraints LUT_{con}.

Makespan $C_{\max}$ is defined as

$$L = \max\{AFT_i\} \tag{2}$$

where AFT_i is the actual finish time of task τ_i. The makespan will be the last actual finish time of an exit task.

The overall hardware resources consumption LUT_{sum} is defined as

$$LUT_{\text{sum}} = \sum_{\tau_i \in \mathcal{H}}^{n} LUT_i \tag{3}$$

where $\mathcal{S}$ is the sets of tasks allocated into software cores and $\mathcal{H}$ as the sets of tasks allocated into hardware.

4 Co-scheduling strategy

Before introducing the detail of our co-scheduling strategy, we first introduce the linear programming method in [29] and some notions used in our strategy.

4.1 Related knowledge

Linear Programming The goal of task allocation is to minimize makespan within the constraints of hardware resources. It can easily be described as a mathematical model for linear programming.

We define a v-dimensional vector X as the allocation solution, where $X_i = 0$ indicates that task τ_i is allocated to the software cores and $X_i = 1$ means task τ_i is allocated to FPGA. The target function is then defined as:

$$\begin{cases} \min \quad TimeCost_{\text{sum}} = \sum_{i=1}^{v} (1 - X_i) \times w_{i,1} + X_i \times w_{i,2} \\ \text{s.t.} \quad \sum_{i=1}^{n} X_i \times LUT_i \leq LUT_{\text{con}} \end{cases} \tag{4}$$

As a solution, we obtain a hardware/software allocation scheme X.

Notions We introduce some notions derived from [29].

Definition 1 Given a real-time task τ_i, the *Successor Parallel Rank* of task τ_i, denoted by $SPR(\tau_i)$, is defined as

$$SPR(\tau_i) = \frac{|Suc(\tau_i)|}{\max_{\tau_j \in V} |Suc(\tau_j)|} \tag{5}$$

where $Suc(\tau_i)$ is the set of the direct successor of task τ_i, $|Suc(\tau_i)|$ represents the number of the elements in the set $Suc(\tau_i)$.

Definition 2 Given a real-time task τ_i, the *Execution Time* of task τ_i, denoted by τ_i, is defined as

$$ET(\tau_i) = \begin{cases} w_{i,1}, & \tau_i \in \mathcal{S}, \\ w_{i,2}, & \tau_i \in \mathcal{H}. \end{cases} \tag{6}$$

Definition 3 Given a real-time task τ_i, the *Earliest Start Time* of task τ_i, denoted by $EST(\tau_i)$, is defined as

$$EST(\tau_i) = \begin{cases} RLS_i & \mathrm{Pre}(\tau_i) = \emptyset \\ \max\{RLS_i, \max_{\tau_j \in \mathrm{Pre}(\tau_i)}\{EST(\tau_i) + ET(\tau_j) + c_{i,j}\}\} & \mathrm{Pre}(\tau_i) \neq \emptyset \end{cases} \tag{7}$$

where $\mathrm{Pre}(\tau_i)$ is the set of the direct predecessors of task τ_i.

Definition 4 Given a real-time task τ_i, the *Last Finish Time* of task τ_i, denoted by $LFT(\tau_i)$, is defined as

$$LFT(\tau_i) = \begin{cases} DDL_i & Suc(\tau_i) = \emptyset \\ \min\{DDL_i, \min_{\tau_j \in Suc(\tau_i)}\{LFT(\tau_j) - ET(\tau_j) - c_{i,j}\}\} & Suc(\tau_i) \neq \emptyset \end{cases} \tag{8}$$

Definition 5 During scheduling, given a real-time task τ_i, the *Scheduling Urgency* of task τ_i at the current time CT, denoted by $SU(\tau_i, CT)$, is defined as

$$SU(\tau_i, CT) = \frac{ET(\tau_i)}{LFT(\tau_i) - \max\{EST(\tau_i), CT\}} \tag{9}$$

$SU(\tau_i, CT) = 1$ means that if τ_i must be executed at the current time. Otherwise, at least one successor will miss its deadline.

4.2 ESHCS

In this section, we introduce our efficient software–hardware co-scheduling strategy (ESHCS), which includes two phases: allocation phase and scheduling phase.

In the allocation phase, we obtain the software and hardware allocating solution efficiently. The goal is to minimize the makespan within the hardware resource constraints. There are two factors that affect the makespan: the task parallelism rate and the software–hardware speedup. Software–hardware speedup is the

acceleration ratio of software computing cost and hardware computing cost. It is a great indicator for maximizing the time benefits of hardware acceleration. Meanwhile, to improve the efficiency of the algorithm, we design a heuristic strategy that combines linear programming and greedy strategy. This algorithm achieves time minimization by increasing the parallelism rate of tasks and the software-hardware speedup.

In the scheduling phase, we obtain the scheduling sequence based on the software–hardware allocation result. The goal is to minimize the makespan and to meet the deadline of tasks. In this phase, the strategies for software and hardware tasks are different. We dynamically measure the priorities of software tasks. The software task with the highest real-time priority will be assigned to the suitable software cores. Hardware tasks are executed in parallel in the FPGA.

4.2.1 Task allocation phase

The linear programming model guarantees the generation of the optimal solution in NP-hard problem. But it yields the shortest makespan at the cost of exponential computation cost. In contrast, heuristic methods usually generate sub-optimal solutions in a polynomial-time. We make a tradeoff between computation cost and quality of results. We propose a heuristic allocation method combining the linear programming and greedy strategy.

In our methods, we analyze the topology of the task graph and cluster the tasks into groups $\mathcal{P}$ based on their dependencies. In each group, the tasks are independent of each other, and they can be executed simultaneously in FPGA. We calculate the sum of software–hardware speedup of each group. Software–hardware speedup of task τ_i is defined as follows.

$$\text{speedup}_i = \frac{w_{i,1}}{w_{i,2}} \tag{10}$$

We use $\mathcal{P}_{\max}$ to represent the task group with the largest total speedup. We assume the tasks in $\mathcal{P}_{\max}$ are all hardware tasks. There will be two cases: the total hardware consumption of this group is less than the hardware constraints or is over the hardware constraints. In the first case, we assign the tasks of $\mathcal{P}_{\max}$ to the FPGA and find the next group with the largest total speedup. Repeat the same step until there are no groups left or case two occurs. In the second case, we use the linear programming strategy to choose the hardware tasks from $\mathcal{P}_{\max}$. Finally, if there are some remaining hardware resources, a greedy strategy is applied to the remaining tasks: assigning the tasks with the maximum speedup to the FPGA until the hardware resource constraints are satisfied.

The main allocation procedure is described as pseudo-code in Algorithm 1. The hardware task choosing strategy is described as Algorithm 2.

Algorithm 1 Main Partitioning Procedure

Input: V: Set of the tasks
Output: $\mathcal{H}$: the sets of hardware tasks

1: $i = 1$
2: $\mathcal{H} = \emptyset$
3: set the entry tasks as the first task group $\mathcal{P}_1$
4: **while** all tasks in $\mathcal{P}_i$ are the exit tasks **do**
5: $i = i + 1$
6: put the direct successors of the tasks in $\mathcal{P}_1$ in a new group $\mathcal{P}_i$
7: when $e(i, j) \in E$ and $\tau_i, \tau_j \in \mathcal{P}_i$, delete τ_j
8: **end while**
9: Hardware Task Choosing
10: **while** exist task $\tau_i \notin \mathcal{H}$ whose $LUT_i < LUT_{con} - LUT_{csp}$ **do**
11: Hardware Task Choosing
12: **end while**

Algorithm 2 Hardware Task Choosing

Input: V: Set of the tasks
$\mathcal{GP}$: the set of the task groups $\mathcal{P}$
Output: $\mathcal{H}$: the sets of hardware tasks

1: $\mathcal{H}$ is empty.
2: LUT_{remain} always equal to $LUT_{con} - \sum_{\tau_i \in \mathcal{H}} LUT_i$
3: **while** $\mathcal{GP}$ is not empty **do**
4: find the group $\mathcal{P}_{max}$ in $\mathcal{GP}$
5: **if** $\sum_{\tau_i \in \mathcal{H} \cup \mathcal{P}_{max}} LUT_i > LUT_{con}$ **then**
6: $\mathcal{T} = \mathcal{P}_{max} - \mathcal{H} \cap \mathcal{P}_{max}$
7: using linear programing to $\mathcal{T}$
8: obtain $\mathcal{HT}$
9: $\mathcal{H} = \mathcal{H} \cup \mathcal{HT}$
10: break
11: **end if**
12: $\mathcal{H} = \mathcal{H} \cap \mathcal{P}_{max}$
13: $\mathcal{GP} = \mathcal{GP} - \mathcal{P}_{max}$
14: **end while**
15: **while** exist $\tau_i \notin \mathcal{H}, LUT_i <= LUT_{remain}$ **do**
16: put all tasks τ_i in a set $\mathcal{T}$, if $LUT_i <= LUT_{remain}$
17: find task $\tau_i \in \mathcal{T}$ with the largest speedup
18: $\mathcal{H} = \mathcal{H} + \tau_i$
19: **end while**

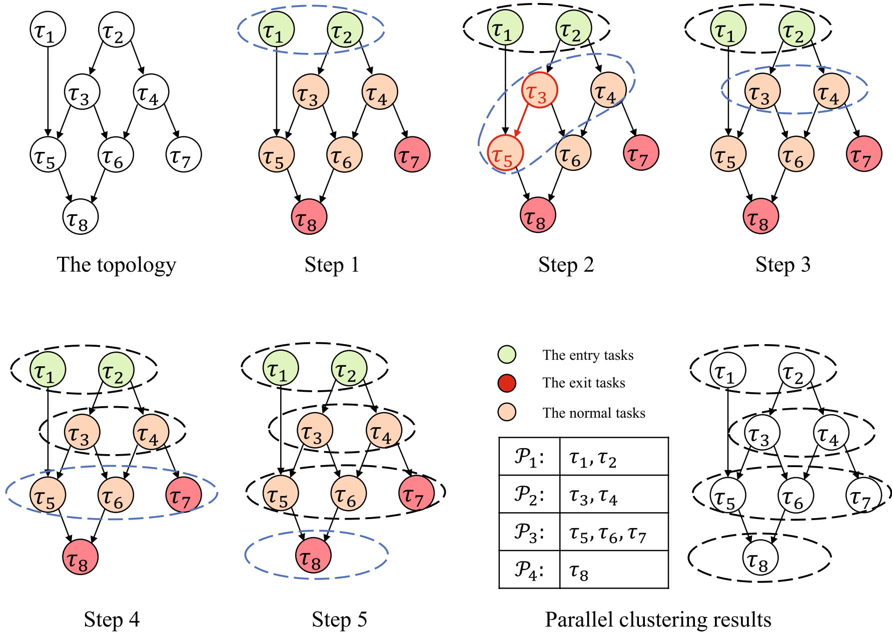
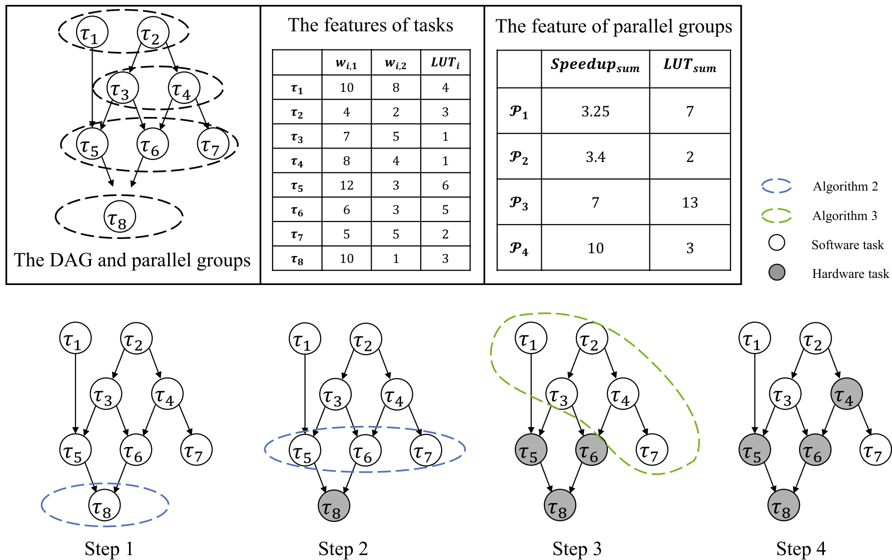


Fig. 2 An example of task grouping



	$w_{i,1}$	$w_{i,2}$	LUT_i
τ_1	10	8	4
τ_2	4	2	3
τ_3	7	5	1
τ_4	8	4	1
τ_5	12	3	6
τ_6	6	3	5
τ_7	5	5	2
τ_8	10	1	3

	$Speedup_{sum}$	LUT_{sum}
$\mathcal{P}_1$	3.25	7
$\mathcal{P}_2$	3.4	2
$\mathcal{P}_3$	7	13
$\mathcal{P}_4$	10	3

Fig. 3 An example of allocation when $LUT_{con} = 15$

We give an example of the allocation procedure in Figs. 2, 3. Figure 2 shows the division of the task groups. In step 1, the entry tasks τ_1 and τ_2 are initialized as the first group. And then we find all the direct successors of the first group. Of these, the task τ_5 is the direct successor of τ_3 so we delete the task τ_5 and get the second group. Similarly, we obtain the third group. The procedure stops when all the tasks in the group are exit tasks. Figure 3 shows the allocation when $LUT_{con} = 15$. In step 1, τ_8 is assigned to the FPGA. Because $\mathcal{P}_4$ has the largest $\mathrm{Speedup}_{\mathrm{sum}}$. The group $\mathcal{P}_3$ has the second-largest $\mathrm{Speedup}_{\mathrm{sum}}$. But the LUT_{sum} of it is larger than the remaining hardware resource. We using linear programming: choosing τ_5 and τ_6 to hardware. At this point, one hardware resource remains. We use a greedy strategy to assign τ_4 to the FPGA.

4.2.2 Task scheduling phase

In the scheduling phase, if all predecessors of task τ_i and all the communications task τ_i have been completed, task τ_i is a ready task. We schedule when a new ready task arrives or a software core is idle. We apply the different strategies to the software and hardware task.

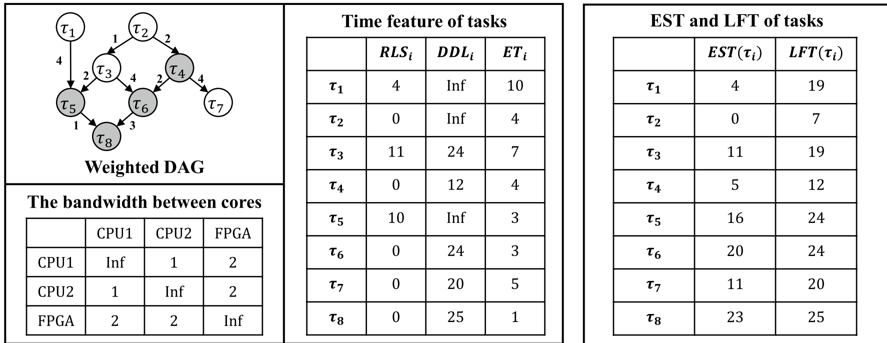
For the software tasks, we design a priority-driven approach based on the *Successor Parallel Rank* and *Scheduling Urgency*. The ready software task with the highest priority is assigned to the idle CPU first. When $SU(\tau_i, CT) = 1$, we assign the task τ_i to the software cores first without priority comparing to maximize the guarantee that the successor can meet the deadline.

The priority of software task τ_i is calculated by:

$$PRI(\tau_i) = \begin{cases} SU(\tau_i, CT) & Suc(\tau_i) = \emptyset \\ SU(\tau_i, CT) + SPR(\tau_i) \times \max_{\tau_j \in Suc(\tau_i)} PRI(\tau_j) & Suc(\tau_i) \neq \emptyset \end{cases} \tag{11}$$

In the priority calculation, tasks with higher SU at the current time are given higher priority to maximize the chance that the tasks will all finish before the deadline; tasks with more successors are given higher priority so that more successors can be released early to improve parallelism; and tasks with higher successors are given higher priority.

For the ready hardware task τ_i, we directly assign it to the FPGA for execution. The main scheduling procedure is described as pseudo-code in Algorithm 3.



The bandwidth between cores

	CPU1	CPU2	FPGA
CPU1	Inf	1	2
CPU2	1	Inf	2
FPGA	2	2	Inf

Time feature of tasks

	RLS_i	DDL_i	ET_i
τ_1	4	Inf	10
τ_2	0	Inf	4
τ_3	11	24	7
τ_4	0	12	4
τ_5	10	Inf	3
τ_6	0	24	3
τ_7	0	20	5
τ_8	0	25	1

EST and LFT of tasks

	$EST(\tau_i)$	$LFT(\tau_i)$
τ_1	4	19
τ_2	0	7
τ_3	11	19
τ_4	5	12
τ_5	16	24
τ_6	20	24
τ_7	11	20
τ_8	23	25

Fig. 4 Data information for scheduling

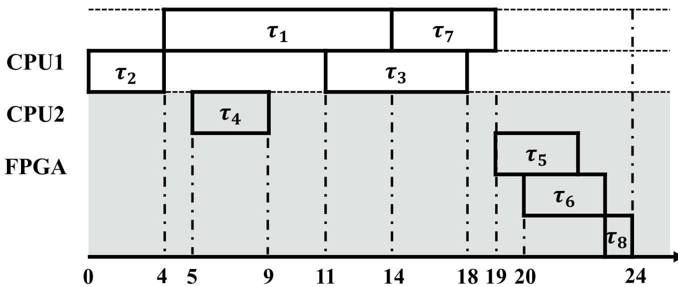


Fig. 5 Final scheduling solution using ESHCS

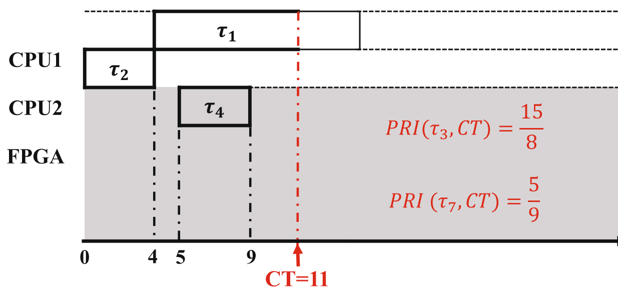


Fig. 6 Map of tasks execution at 11, scheduling τ_3 or τ_7 to CPU2

Algorithm 3 Main Scheduling Procedure

Input: $\mathcal{S}$: the set of software partitioned tasks
$\mathcal{H}$: the set of hardware partitioned tasks
CT: the current time of system

1: start the system
2: $\mathcal{R} = \emptyset$
3: **while** system running **do**
4: put the ready tasks in the set $\mathcal{R}$
5: **if** $\mathcal{R} \neq \emptyset$ **then**
6: $\mathcal{HR} = \mathcal{H} \cap \mathcal{R}$
7: $\mathcal{SR} = \mathcal{S} \cap \mathcal{R}$
8: tasks in $\mathcal{HR}$ to hardware
9: **if** exit idle CPUs **then**
10: obtain the current time CT
11: the priorities of the tasks in $\mathcal{SR}$
12: assign the task with the highest priority to idle CPU
13: **end if**
14: **end if**
15: **end while**

We give the scheduling solution obtained by ESHCS in Fig. 5, which is equal to 24. The EST and LFT of tasks in Fig. 4 is calculated using Eqs. 7 and 8. And we give an example of priority calculation and comparison during the system running time in Fig. 6. When $CT = 11$, the task τ_3 and τ_3 are released. Since $PRI(\tau_3, 11) > PRI(\tau_7, 11)$, the task τ_3 is assigned to CPU2 for execution first, and the task τ_7 will be executed when CPU1 is idle.

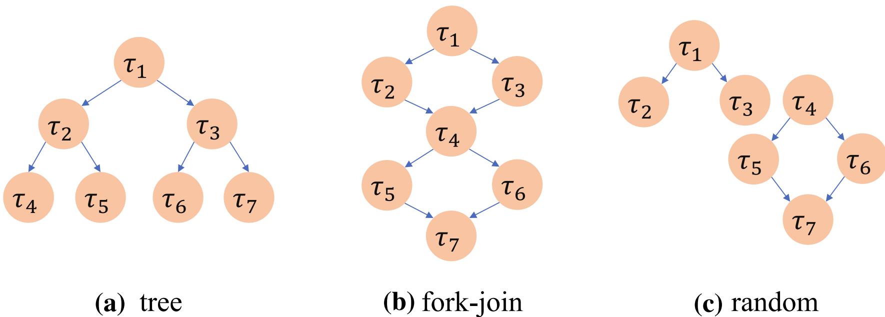


Fig. 7 Examples of different simulation structures

5 Experiments

In the comparative evaluation, we consider two types of applications as the workload for performance experiments: simulation applications generated by the sample generator [29] and the graphs represented the numerical real-world problems [16]. All the experiments were run on a computer with 8 Intel i9-9900k@4.8Ghz processors with 16GB DDR4 memory. The operating system is a 64-bits Windows 10 pro. We used MATLAB 2019b to implement and test the algorithms.

The metrics used for performance evaluation include *Speedup* and *Running Time*. *Speedup* evaluate the solution from a makespan optimization perspective, which presents the effectiveness of the scheduling solution. For a given application, the value of *Speedup* is the ratio of sequential execution time to the makespan. The sequential execution time is computed by assigning all tasks to a single processor. For the same application, if the makespan is minimized, it results in a larger *Speedup*.

In addition to evaluating the performance of the solutions, we also pay attention to the efficiency of the algorithm itself. *Running Time* is the computation cost of obtaining the scheduling solution. Among the algorithms that give equal *Speedup* values, the one with the smallest *Running Time* is the most practical implementation.

In the remainder of this section, we first introduce the samples generator used in the experiments in Sect. 5.1. And then, the comparison results among our scheduling algorithm ESHCS and the related work are analyzed in Sects. 5.2 and 5.3.

5.1 Sample generator

We use the sample generator proposed in [29] to simulate real-time application. This generator generates the samples from two aspects, the DAG and task attributes.

Table 1 Information of five datasets

	n	$\overline{\sum_{i=1}^{n} \mathrm{LUT}_i}$
Dataset1	8	61.1333
Dataset2	16	128.8667
Dataset3	24	189.2000
Dataset4	32	254.6667
Dataset5	40	316.6667

Table 2 Comparison of scheduling performance between ESHCS and ReTPA

	Dataset1		Dataset2		Dataset3		Dataset4		Dataset5	
	$\overline{S_{\mathrm{up}}}$	$\overline{R_{\mathrm{time}}}$	$\overline{S_{\mathrm{up}}}$	$\overline{R_{\mathrm{time}}}$	$\overline{S_{\mathrm{up}}}$	$\overline{R_{\mathrm{time}}}$	$\overline{S_{\mathrm{up}}}$	$\overline{R_{\mathrm{time}}}$	$\overline{S_{\mathrm{up}}}$	$\overline{R_{\mathrm{time}}}$
ReTPA	2.1951	0.0092	2.6135	0.0187	3.7441	0.0244	3.8164	0.0376	3.9436	0.0491
ESHCS	2.241	0.0051	2.5758	0.0121	3.4846	0.0193	3.8278	0.0337	3.8332	0.0444
Improvement (%)	2.09	79.29	− 1.44	54.66	− 6.93	26.7	0.3	11.54	− 2.8	10.73

Table 3 Five considered algorithms

Approach	Allocation	Schedule
ESHCS	Heuristic	Priority-driven
LHEFT	ILP	Priority-driven
GHETS	Greedy	Priority-driven
GAA-A	Sequential	Genetic
MGAA	Genetic	Genetic

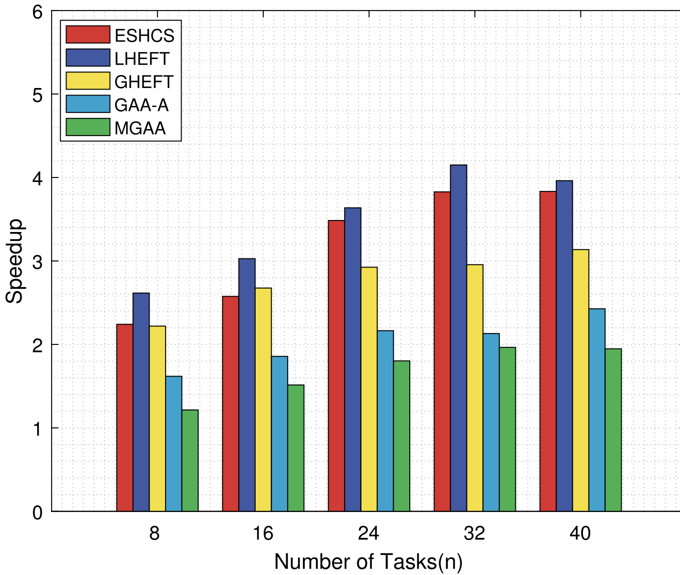


Fig. 8 *Speedup* of algorithms with respect to application scale

DAG generator

The DAG generator has two input parameters: the number of tasks n and the type of DAG. We generate DAG with three different structures for experiments including tree, fork-join, and random. Examples of these three DAG structures are shown in Fig. 7.

Task attribute generator
We generate five attributes for each task including release time, deadline, software execution time, hardware execution time, and hardware resource. Referring to [16], the values of attributes are generated according to the following rules:

- The task must be theoretically schedulable. The difference between the release time and deadline is not less than the hardware execution time, i.e., $DDL_i - RLS_i \geq w_{i,2}$.

- The hardware execution time $w_{i,2}$ is randomly generated from a uniform distribution, and the software execution time $w_{i,1}$ is α times longer than that of the hardware execution time $w_{i,2}$, which follows the random uniform distribution from 2 to 8.
- Hardware resource LUT_i is generated by a random parameter β and the ratio of software execution time $w_{i,1}$ to hardware execution time $w_{i,2}$ as $LUT_i = \alpha \times \frac{w_{i,1}}{w_{i,2}}$, where α follows the Gaussian distribution $N(\mu, \sigma^2)$ with $\mu = 3$ and $\sigma = 1$.

For the experiments, we generate five datasets. The value of DAG parameters are assigned from the corresponding sets given: $SET_n = \{8, 16, 24, 32, 40\}$ and $SET_{DAGtype}$ = {tree, fork-join, and random}. Each dataset includes three different types of samples, and 50 random samples are selected for each type. We list the statistical information of five datasets in Table 1. v is the number of tasks and $\sum_{i=1}^{n} LUT_i$ represents the average value of total hardware resource. It is noted that we are not using the parameter, communication to computation ratio (CCR), in this work. The value of CCR is set to 0.1.

5.2 Comparative experiment on simulation application

In this subsection, we design three different experiments using the simulated applications. First, we compare ESHCS with our previous algorithm ReTPA. Then, we test our algorithm against other algorithms. Finally, we explored the impact of resource size on the performance of scheduling algorithms.

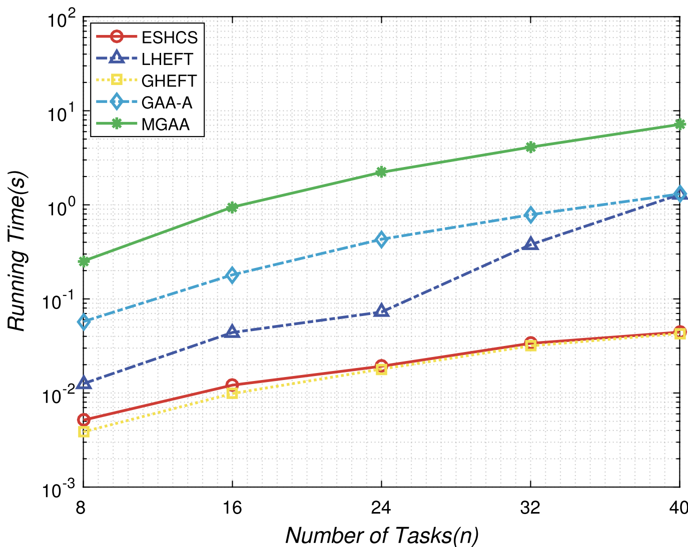


Fig. 9 *Running Time* of algorithms with respect to application scale

5.2.1 Comparative experiment with ReTPA

We compare ESHCS with our proposed algorithm RePTA [29]. ESHCS is a further performance-balanced scheduling algorithm based on RePTA. The results are in Table 2. We can see that *Speedup* of ReTPA and ESHCS differ minimally in all five datasets, but ESHCS shows a significant improvement in *Running Time*. The mean improvements are the reduction of 1.76% and the improvement of 36.59%, respectively. ESHCS achieves our goal of significantly reducing the computation cost of the algorithm by sacrificing a very small amount of the makespan speedup.

5.2.2 Comparative experiment with other algorithms

In this subsection, we analyze the performance of ESHCS and other five algorithms. The structures of the five algorithms are compared in Table 3, where 'Allocation' presents the allocation strategy used in the algorithms and 'Schedule' presents the scheduling strategy. The allocation strategy of the GAA is sequential, i.e., task T_i is assigned to the processor $p = (i-1)[ns] + 1$, where $p \leq ns + 1$ and $p = ns + 1$ represents FPGA. Moreover, the parameters of GAA-A and MAGG in our experiments are: *Popsize* $= 10$, number of generations $NG = 10$, number of threads $Nb = 6$.

This experiment compares the performance of the algorithms with respect to various application scales. The performance ranking of *Speedup* is {LHEFT, ESHCS, GHEFT, GAA-A, MGAA} in Fig. 8. It should be noted that each ranking in this paper starts with the best algorithm and ends with the worst one. ESHCS and LHEFT provide the highest speed-up solutions, far better than HEFT with greedy

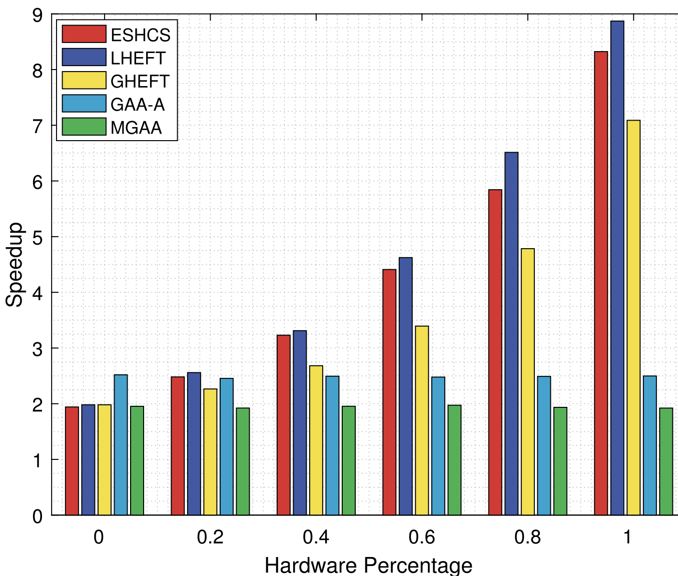


Fig. 10 *Speedup* of algorithms with respect to hardware resources

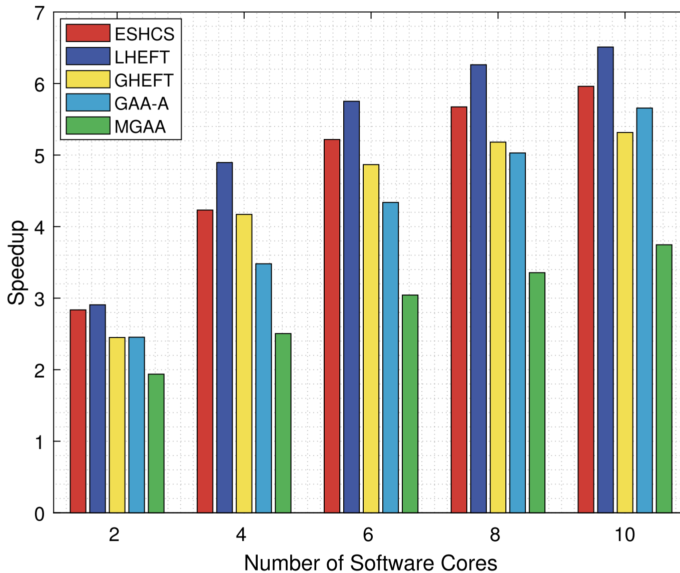


Fig. 11 *Speedup* of algorithms with respect to software cores

strategy and the other two genetic algorithms. The average speedup of LHEFT of all datasets is the best, better than the ESHCS algorithm by 8.21%, the GHEFT algorithm by 20.01%, the GAA-A algorithm by 41.37%, and the MGAA algorithm by 51.44%. The reason why the genetic algorithms do not find the best solution as expected is that they consider the hardware as a special software core, so the number of tasks allocated to the hardware is about $n/(ns + 1)$. The iterations are more about trying a better scheduling strategy or allocating better groups of tasks to the hardware. There is not a tendency to divide more tasks into hardware to improve parallel efficiency, which causes the algorithm to easily fall into a local optimal. This shortcoming can also be seen in the parameter experiments in Sect. 5.2.3. *Speedup* of the genetic algorithms improve significantly with increasing software scale, but the size of hardware resources has almost no effect on the performance of genetic algorithms.

The performance ranking of *Running Time* is {GHEFT, ESHCS, LHEFT, GAA-A, MGAA} in Fig. 9. In general, the computation cost of all algorithms increases with the application scale. *Running Time* of LHEFT increases fastest, due to the computational complexity of linear programming. The computational complexity of the greedy strategy is $\mathcal{O}(n)$ so that GHEFT provides the best performance in terms of computation cost. The time performance of ESHCS is second only to GHEFT and remains at the millisecond level. The average *Running Time* of ESHCS and GHEFT is 0.0229 s and 0.0213 s.

In these experiments, LHEFT and ESHCS outperform the other algorithms for any application scale in terms of *Speedup*. But the average *Running Time* of LHEFT is 14.67 times larger than ESHCS. The *Running Time* of GHEFT outperforms the other algorithms but its *Speedup* is much worse than LHEFT and ESHCS. For the

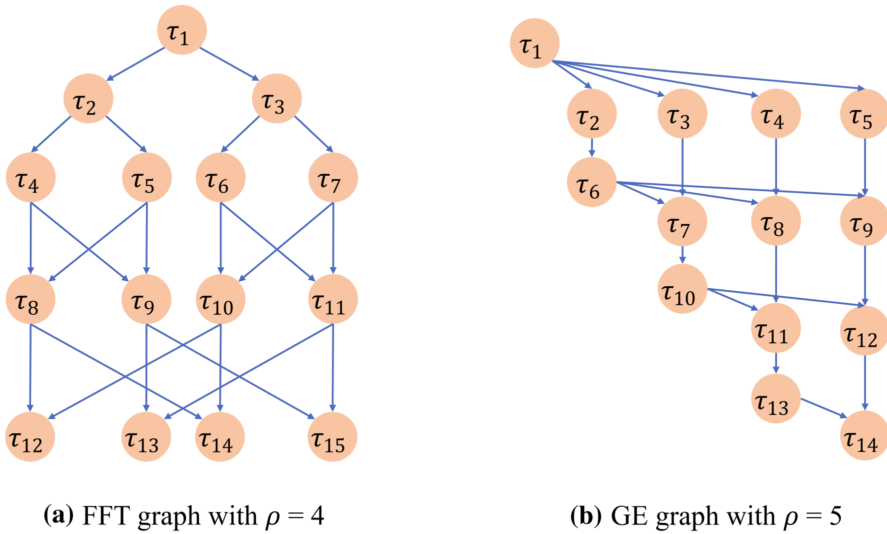


Fig. 12 Example of real parallel structures

Table 4 Scheduling performance in fast Fourier transform

ρ	v	ESHCS		LHEFT		GHEFT		GAA-A		MGAA	
		$\overline{S}_{\text{up}}$	$\overline{R}_{\text{time}}$	$\overline{S}_{\text{up}}$	$\overline{R}_{\text{time}}$	$\overline{S}_{\text{up}}$	$\overline{R}_{\text{time}}$	$\overline{S}_{\text{up}}$	$\overline{R}_{\text{time}}$	$\overline{S}_{\text{up}}$	$\overline{R}_{\text{time}}$
2	5	1.3345	0.0092	**1.7123**	0.0130	1.3721	**0.0019**	1.6593	0.0341	1.3021	0.1442
4	15	3.1378	0.0196	**3.1479**	0.0237	2.2454	**0.0115**	2.8295	0.2722	1.8402	1.1535
8	39	**4.9916**	0.0410	4.9708	0.0445	2.7368	**0.0402**	3.3725	1.3073	2.2462	7.0349
16	95	**6.2070**	**0.1932**	6.1744	0.2163	2.8771	0.2061	3.8293	7.2177	2.8071	39.860
32	223	**6.5439**	2.6021	6.5345	3.6223	2.9213	**1.1357**	4.0269	37.527	3.2933	218.23

Bold data represents the best performance among five algorithms

two genetic algorithms, GAA-A and MGAA have neither the best nor the worst average *Speedup* performance, and they spend too much time searching for the solution. In conclusion, ESHCS obtains the best scheduling solution in a reasonable computation cost.

5.2.3 Parameter experiment

This experiment investigates the impact of resource size on scheduling performance. The resources of hardware-accelerated multicore system includes software cores and hardware resource. The value of parameters are given: $SET_{\text{softcore number}} = \{2, 4, 6, 8, 10\}$ and $SET_{\text{hardware percentage}} = \{0, 0.2, 0.4, 0.6, 0.8, 1\}$, where the number of software cores represents the scale of the software resources on the platform and hardware percentage is the ratio of the hardware resource constraint

Table 5 Scheduling performance in Gaussian elimination

ρ	v	ESHCS		LHEFT		GHEFT		GAA-A		MGAA	
		$\overline{S_{up}}$	$\overline{R_{time}}$	$\overline{S_{up}}$	$\overline{R_{time}}$	$\overline{S_{up}}$	$\overline{R_{time}}$	$\overline{S_{up}}$	$\overline{R_{time}}$	$\overline{S_{up}}$	$\overline{R_{time}}$
5	14	2.3329	0.0288	**2.4771**	0.0748	1.6413	**0.0232**	2.3730	0.5001	1.7602	2.0213
7	27	3.1309	0.0529	**3.2563**	0.0890	1.9759	**0.0434**	2.6703	1.5207	2.2158	7.2676
9	44	3.6846	0.1070	**3.7682**	0.1671	2.0569	**0.0977**	2.9162	4.2832	2.6663	21.9262
11	65	**4.4073**	0.2118	4.3969	0.2698	2.2091	**0.2061**	3.1069	13.1982	2.9026	44.5618
13	90	**5.1886**	**0.3594**	5.0715	0.4200	2.3541	0.3660	3.3590	35.2279	2.9425	81.9448

Bold data represents the best performance among five algorithms

to the hardware resource overhead required by all tasks. As described in Table 1, $\sum_{i=1}^{n} LUT_i$ is the average hardware resource overhead for each dataset which use as a baseline. $\sum_{i=1}^{n} LUT_i \times SET_{\text{hardware percentage}}$ are the hardware constraints we set in the experiment. For the experiments, we select the fifth datasets with $n = 40$, where $\sum_{i=1}^{n} LUT_i = 316.6667$. Figures 10 and 11 show the results of these two parameter experiments.

The first experiment, in Fig 10, compares *Speedup* of the algorithms in different hardware scales. The increase in hardware resources leads to a dramatic increase in LHEFT, ESHCS and GHEFT. In contrast, the trends of MGAA and GAA-A appear to be flat. This indicates that LHEFT, ESHCS and GHEFT are better adapted to hardware-accelerated multicore systems. The best performance is achieved by LHEFT and ESHCS with a slight difference, whose *Speedup* is up to 8.8712 and 8.3227 when the hardware percentage reaches 1.

The second experiment, in Fig 11, demonstrates the impact of the software scale on scheduling performance. Overall, the increase in software scale also contributes to the increase in *Speedup*. But the trend is smaller than the one brought by hardware percentage. The trends of algorithms {LHEFT, ESHCS, GHEFT, GAA-A, MGAA} increase as the number of software cores increases, and the *Speedup* stabilizes after the number of software cores reaches 6. This indicates that software cores have a limited effect on application acceleration. This is a reflection of the fact that hardware acceleration is the future of high performance computing. In addition, how to minimize the active software cores and achieve efficient scheduling is another interesting future research topic [14, 15].

5.3 Comparative experiment on real world application

Except for simulation task graph generation, we select two real parallel applications for our experiments: Fast Fourier transform (FFT) and Gaussian elimination (GE). The number of tasks in FFT with ρ is $v = (2 \times \rho - 1) + \rho \times \lg \rho$, where $\rho = 2^n$. The number of tasks in GE with ρ is $v = \frac{\rho^2+\rho-2}{2}$. Figure 12 shows the examples of FFT and

GE applications. In Fig. 12, FFT is generated with the parameter $\rho = 4$, and GE is generated with the parameter $\rho = 5$.

Fast Fourier Transform

In this experiment, we compare the performance of algorithms using FFT. The application scale varies ρ from 2 to 32 with the number of tasks $v = \{5, 13, 39, 95, 223\}$. Table 4 shows the scheduling performance in terms of *Speedup* $\overline{S_{\text{up}}}$ and *Running Time* $\overline{R_{time}}$. Bold data represents the best performance among five algorithms.

It is seen that ESHCS has an advantage in *Speedup* for three datasets as well as LHEFT provides a similar performance. ESHCS performs better than LHEFT in *Running Time*. It slightly performs less than GHEFT in the small scales. The allocation algorithm integrating linear programming and greedy algorithms effectively reduces the computational cost of ESHCS. The scheduling performances of genetic-based algorithms, GAA-A and MGAA, in FFT are better than theirs in GE, but there is still a gap with the other three methods. In summary, ESHCS has the best performance in *Speedup* in most cases, while GHEFT outperforms in *Running Time* in most cases. ESHCS shows the best performance in large-scale applications.

Gaussian Elimination

In this experiment, we compare the performance of algorithms using GE. The application scale varies ρ from 5 to 13 with the number of tasks $v = \{14, 27, 44, 65, 90\}$. Table 5 shows the scheduling performance in *Speedup* $\overline{S_{up}}$ and *Running Time* $\overline{R_{time}}$.

Running Time of LHEFT, ESHCS and GHEFT have increased with the expansion of GE scale. They have remained at the millimeter level. In contrast, the running time of GAA-A and MGAA is nearly multiple times larger than the other three algorithms. The expansion of search space in the genetic algorithm brings a large scheduling cost, especially in large-scale applications. Although GHEFT shows a slight disadvantage in *Running Time*, the scheduling efficiency of ESHCS far outperforms it with an average improvement of 43.22%.

Among all five algorithms, ESHCS and GHEFT achieve the best performance in the *Running Time* and the gap is very small. In contrast, the *Speedup* of ESHCS is better than GHEFT by 44.82%. The above results suggest that for GE, ESHCS obtains the best scheduling solution in a reasonable computation time.

6 Conclusion

In this paper, we propose ESHCS for scheduling real-time applications in the multicore system integrated with hardware acceleration. The goal of this algorithm is to minimize the makespan subject to hardware resource constraints while meeting the

real-time requirements of tasks. Our strategy consists of two phases: heuristic allocation and list-based scheduling. In the experiments, we test our algorithm in both simulation applications and real applications. The experiments confirm that ESHCS greatly improves the running time by sacrificing slight speedup performance compared to ReTPA. ESHCS also shows a significant enhancement compared to other advanced algorithms LHEFT, GHEFT, GAA-A and MGAA. Furthermore, we plan to extend this work to periodic real-time tasks. Moreover, how to reduce the number of active processors while achieving the best solutions is another interesting future research topic.

Acknowledgements The authors would like to thank the editors and anonymous reviewers for their constructive comments. The authors would like to thank Professor Patrice Quinton from Inria, Rennes, France, for his valuable suggestions. Thanks for the supports from him and the Inria team TARAN. This work was also supported in part by the National Key Research and Development Project of China (2018YFB2101300), the Dean's Fund of Engineering Research Center of Software/Hardware Co-design Technology and Application, Ministry of Education (East China Normal University), Shanghai Collaborative Innovation Center for Trusted Industrial Internet Software, and Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

Data availability The datasets generated during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of interest We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

References

1. Zhang C, Li P, Sun G, Guan Y, Xiao B, Cong J (2015) Optimizing fpga-based accelerator design for deep convolutional neural networks. In: Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, February 22-24, 2015, pp. 161–170. https://doi.org/10.1145/2684746.2689060
2. Chen H, Madaminov S, Ferdman M, Milder P.A (2020) Fpga-accelerated samplesort for large data sets. In: FPGA '20: The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, February 23–25, 2020, pp. 222–232. https://doi.org/10.1145/3373087.3375304
3. Li J, Chi Y, Cong J (2020) Heterohalide: From image processing DSL to efficient FPGA acceleration. In: FPGA '20: The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Seaside, CA, USA, February 23-25, 2020, pp. 51–57. https://doi.org/10.1145/3373087.3375320
4. Gupta R.K, De Micheli G (1992) System-level synthesis using re-programmable components. In: [1992] Proceedings The European Conference on Design Automation, pp. 2–7
5. Lee E, Seshia S (2015) Introduction to embedded systems: a cyber-physical systems approach
6. Kadri AA, Labadi K, Kacem I (2015) An integrated petri net and ga-based approach for performance optimisation of bicycle sharing systems. Eur J Ind Eng 9(5):638
7. Kao C (2020) Resource and performance tradeoff for task scheduling of parallel reconfigurable architectures. J Circuits Syst Comput 29(2):2050029–1205002914. https://doi.org/10.1142/S0218126620500292
8. Bhuiyan A, Liu D, Khan A, Saifullah A, Guan N, Guo Z (2020) Energy-efficient parallel real-time scheduling on clustered multi-core. IEEE Trans Parallel Distrib Syst 31(9):2097–2111. https://doi.org/10.1109/TPDS.2020.2985701

9. Kumar N, Mayank J, Mondal A (2020) Reliability aware energy optimized scheduling of non-preemptive periodic real-time tasks on heterogeneous multiprocessor system. IEEE Trans Parallel Distrib Syst 31(4):871–885. https://doi.org/10.1109/TPDS.2019.2950251
10. Deng Z, Yan Z, Huang H, Shen H (2020) Energy-aware task scheduling on heterogeneous computing systems with time constraint. IEEE Access 8:23936–23950. https://doi.org/10.1109/ACCESS.2020.2970166
11. Moulik S, Chaudhary R, Das Z, Sarkar A (2020) EA-HRT: an energy-aware scheduler for heterogeneous real-time systems. In: 25th Asia and South Pacific Design Automation Conference, ASP-DAC 2020, Beijing, China, January 13–16, 2020, pp. 500–505. https://doi.org/10.1109/ASP-DAC47756.2020.9045240
12. Li T, Zhang T, Yu G, Song J, Fan J (2019) Minimizing temperature and energy of real-time applications with precedence constraints on heterogeneous mpsoc systems. J Syst Archit 98:79–91. https://doi.org/10.1016/j.sysarc.2019.07.001
13. Thammawichai M, Kerrigan EC (2018) Energy-efficient real-time scheduling for two-type heterogeneous multiprocessors. Real-Time Syst 54(1):132–165. https://doi.org/10.1007/s11241-017-9291-6
14. Cho H, Kim C, Sun J, Easwaran A, Park J, Choi B (2020) Scheduling parallel real-time tasks on the minimum number of processors. IEEE Trans Parallel Distrib Syst 31(1):171–186. https://doi.org/10.1109/TPDS.2019.2929048
15. Nelissen G, Berten V, Goossens J, Milojevic D (2012) Techniques optimizing the number of processors to schedule multi-threaded tasks. In: 24th Euromicro Conference on Real-Time Systems, ECRTS 2012, Pisa, Italy, July 11–13, 2012, pp. 321–330. https://doi.org/10.1109/ECRTS.2012.37
16. Topcuoglu H, Hariri S, Wu M (2002) Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans Parallel Distributed Syst 13(3):260–274. https://doi.org/10.1109/71.993206
17. Ilavarasan E, Thambidurai P, Mahilmannan R (2005) Performance effective task scheduling algorithm for heterogeneous computing system. In: 4th International Symposium on Parallel and Distributed Computing (ISPDC 2005), 4-6 July 2005, Lille, France, pp. 28–38. https://doi.org/10.1109/ISPDC.2005.39
18. Masood A, Munir E.U, Rafique M.M, Khan S.U (2015) HETS: heterogeneous edge and task scheduling algorithm for heterogeneous computing systems. In: 17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICESS 2015, New York, NY, USA, August 24–26, 2015, pp. 1865–1870. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.295
19. Du P, Sun Z, Zhang H, Ma, H (2019) Feature-aware task scheduling on CPU-FPGA heterogeneous platforms. In: 21st IEEE International Conference on High Performance Computing and Communications; 17th IEEE International Conference on Smart City; 5th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2019, Zhangjiajie, China, August 10–12, 2019, pp. 534–541. https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00084
20. Ahmad I, Kwok, Y (1994) A new approach to scheduling parallel programs using task duplication. In: Proceedings of the 1994 International Conference on Parallel Processing, North Carolina State University, NC, USA, August 15–19, 1994. Volume II: Software, pp. 47–51. https://doi.org/10.1109/ICPP.1994.37
21. Chung Y, Ranka S (1992) Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. In: Proceedings Supercomputing '92, Minneapolis, MN, USA, November 16–20, 1992, pp. 512–521. https://doi.org/10.1109/SUPERC.1992.236653
22. Tsuchiya T, Osada T, Kikuno T (1998) Genetics-based multiprocessor scheduling using task duplication. Microprocess Microsyst 22(3–4):197–207. https://doi.org/10.1016/S0141-9331(98)00079-9
23. Li G, Chen D, Daming W, Zhang D (2003) Task clustering and scheduling to multiprocessors with duplication. In: 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), 22–26 April 2003, Nice, France, CD-ROM/Abstracts Proceedings, p. 6. https://doi.org/10.1109/IPDPS.2003.1213079
24. Hu W, Gan Y, Wen Y, Lv X, Wang Y, Zeng X, Qiu M (2020) An improved heterogeneous dynamic list schedule algorithm. In: Algorithms and Architectures for Parallel Processing - 20th International Conference, ICA3PP 2020, New York City, NY, USA, October 2-4, 2020, Proceedings, Part I, pp. 159–173. https://doi.org/10.1007/978-3-030-60245-1_11

25. Orr M, Sinnen O (2020) Integrating task duplication in optimal task scheduling with communication delays. IEEE Trans Parallel Distrib Syst 31(10):2277–2288. https://doi.org/10.1109/TPDS.2020.2989767
26. Quan Z, Wang Z, Ye T, Guo S (2020) Task scheduling for energy consumption constrained parallel applications on heterogeneous computing systems. IEEE Trans Parallel Distrib Syst 31(5):1165–1182. https://doi.org/10.1109/TPDS.2019.2959533
27. Ahmad S.G, Munir E.U, Nisar M.W (2012) PEGA: A performance effective genetic algorithm for task scheduling in heterogeneous systems. In: 14th IEEE International Conference on High Performance Computing and Communication & 9th IEEE International Conference on Embedded Software and Systems, HPCC-ICESS 2012, Liverpool, United Kingdom, June 25–27, 2012, pp. 1082–1087. https://doi.org/10.1109/HPCC.2012.158
28. Aba MA, Zaourar L, Munier A (2020) Efficient algorithm for scheduling parallel applications on hybrid multicore machines with communications delays and energy constraint. Concurr Comput Pract Exp. https://doi.org/10.1002/cpe.5573
29. Xu J, Li K, Chen Y (2022) Real-time task scheduling for fpga-based multicore systems with communication delay. Microprocess Microsyst 90:104468. https://doi.org/10.1016/j.micpro.2022.104468
30. Zhu Z, Zhang J, Zhao J, Cao J, Zhao D, Jia G, Meng Q (2019) A hardware and software task-scheduling framework based on CPU+FPGA heterogeneous architecture in edge computing. IEEE Access 7:148975–148988. https://doi.org/10.1109/ACCESS.2019.2943179
31. Rodríguez A, Navarro AG, Asenjo R, Corbera F, Tejero RG, Gracia DS, Núñez-Yáñez JL (2019) Exploring heterogeneous scheduling for edge computing with CPU and FPGA mpsocs. J Syst Archit 98:27–40. https://doi.org/10.1016/j.sysarc.2019.06.006
32. Zhang T, Liu G, Yue Q, Zhao X, Hu M (2019) Using firework algorithm for multi-objective hardware/software partitioning. IEEE Access 7:3712–3721. https://doi.org/10.1109/ACCESS.2018.2886430
33. Abdallah F, Tanougast C, Kacem I, Diou C, Singer D (2019) Genetic algorithms for scheduling in a CPU/FPGA architecture with heterogeneous communication delays. Comput Ind Eng. https://doi.org/10.1016/j.cie.2019.106006
34. Purnaprajna M, Reformat M, Pedrycz W (2007) Genetic algorithms for hardware-software partitioning and optimal resource allocation. J Syst Archit 53(7):339–354. https://doi.org/10.1016/j.sysarc.2006.10.012
35. Jiang Q, Xu J, Chen Y (2021) A genetic algorithm for scheduling in heterogeneous multicore system integrated with FPGA. In: 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), New York City, NY, USA, September 30-Oct. 3, 2021, pp. 594–602. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00087
36. Dai G, Shan Y, Chen F, Wang Y, Wang K, Yang H (2014) Online scheduling for FPGA computation in the cloud. In: 2014 International Conference on Field-Programmable Technology, FPT 2014, Shanghai, China, December 10–12, 2014, pp. 330–333. https://doi.org/10.1109/FPT.2014.7082811
37. Shi W, Wu J, Jiang G, Lam S (2020) Multiple-choice hardware/software partitioning for tree task-graph on mpsoc. Comput J 63(5):688–700. https://doi.org/10.1093/comjnl/bxy140
38. Hao C, Chen D (2021) Software/hardware co-design for multi-modal multi-task learning in autonomous systems. In: 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS), pp. 1–5. https://doi.org/10.1109/AICAS51828.2021.9458577
39. Hagras T, Atef A, Mahdy YB (2021) Greening duplication-based dependent-tasks scheduling on heterogeneous large-scale computing platforms. J Grid Comput 19(1):13. https://doi.org/10.1007/s10723-021-09554-2
40. Bertolino M (2021) Efficient scheduling of applications onto cloud fpgas. (ordonnancement efficace des applications sur cloud fpgas). PhD thesis, Polytechnic Institute of Paris, France. https://tel.archives-ouvertes.fr/tel-03276708
41. Shi H, Chen Y, Xu J (2021) An efficient scheduling algorithm for distributed heterogeneous systems with task duplication allowed. In: 2021 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom), New York City, NY, USA, September 30-Oct. 3, 2021, pp. 578–587. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00085

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.