J Sign Process Syst (2015) 81:359–375
DOI 10.1007/s11265-014-0958-0

Reliability-Guaranteed Task Assignment and Scheduling for Heterogeneous Multiprocessors Considering Timing Constraint

Juan Yi · Qingfeng Zhuge · Jingtong Hu · Shouzhen Gu · Mingwen Qin · Edwin H.-M. Sha

Received: 11 February 2014 / Revised: 3 September 2014 / Accepted: 12 October 2014 / Published online: 1 November 2014

Abstract Heterogeneous multiprocessors have become the mainstream computing platforms nowadays and are increasingly employed for critical applications. Inherently, heterogeneous systems are more complex than homogeneous systems. The added complexity increases the potential of system failures. This paper addresses this problem by proposing a reliability-guaranteed task assignment and scheduling approach for heterogeneous multiprocessors considering timing constraint. We propose a two-phase approach to solve this problem. In the first phase, we determine assignments for heterogeneous multiprocessors such that both reliability requirement and timing constraint can be satisfied with the minimum total system cost. Efficient algorithms are proposed to produce optimal solutions for simple-path and tree-structured task graphs in polynomial time. When the input graph is a directed acyclic graph (DAG), the task assignment problem is NP-Complete. In this situation, we first develop an Integer Linear Programming (ILP) formulation to generate optimal solutions. Then, we propose a polynomial-time heuristic algorithm to find near optimal solutions. In the second phase, based on the assignments obtained in the first phase, we propose a minimum resource scheduling algorithm to generate a schedule and a feasible configuration that uses as little resource as possible. Experimental results show that the proposed algorithms and the ILP formulation can effectively reduce the total cost compared with the previous work.

Keywords Task assignment · Scheduling · Heterogeneous · Multiprocessor · System reliability

J. Yi · Q. Zhuge (✉) · S. Gu · E. H.-M. Sha
College of Computer Science, Chongqing University, Chongqing, People's Republic of China
e-mail: qfzhuge@cqu.edu.cn

J. Yi
e-mail: wsjenni@gmail.com

S. Gu
e-mail: shannon1229@cqu.edu.cn

E. H.-M. Sha
e-mail: edwinsha@gmail.com

J. Hu
School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK, 74078, USA
e-mail: jthu@okstate.edu

M. Qin
Department of Electrical Engineering, The Hong Kong Polytechnic University, Hung Hom, Hong Kong
e-mail: owen.qin@connect.polyu.hk

1 Introduction

Multiprocessors have become the mainstream computing platforms nowadays [21]. Heterogeneous multiprocessor systems, which are often implemented on platforms comprising of multiple heterogeneous processing units such as general-purpose central processing units (CPU), digital signal processors (DSP), and application-specific integrated circuits (ASIC), provide both high performance and high flexibility to allow efficient execution of a wide range of applications [13, 18, 31]. One example of such heterogeneous architecture is the IBM Cell processor, which consists of a Power processor and eight synergistic processors [23].

In heterogeneous multiprocessor systems, applications are normally divided into multiple small tasks executed on different processing elements (PEs) concurrently to satisfy requirements on performance, reliability, etc. Different PEs typically have distinct speed/reliability characteristics for different tasks. Therefore, how to assign a proper PE such that the requirements can be met and the total system cost can be minimized arises as an important problem.

At the same time, the reliability of today's high-performance integrated circuits (ICs) has become a serious concern [4, 30]. As shown in [1], the failure rates for electronic products can be quite high within warranty and the main reason is traced to excessive stress on the processors. For heterogeneous multiprocessor system, if the processor failures are not taken into consideration during the task mapping process, some processors might age much faster than the others and become the reliability bottleneck for the system, thus significantly reducing the system's service life.

There have been a lot of research efforts on allocating and scheduling applications in heterogeneous systems [3, 9, 10, 15, 19, 27]. Some prior works [3, 19] perform assignment and scheduling on independent tasks. Incorporating reliability cost into heterogeneous distributed systems, the reliability driven assignment and scheduling problem has been studied in [9, 10, 15, 27]. In these works, they perform task assignment and scheduling with the goal of maximizing system reliability under certain constraints. None of them, however, consider system reliability as a constraint that has to be satisfied as in the problem we study in this paper.

This paper targets efficient techniques for task assignment and scheduling with guaranteed reliability and timing constraint. The goal is to assign each task to a processor and generate a schedule and configuration such that the total system cost for finishing all the tasks is minimized while both the reliability and timing constraints are satisfied. Here, a configuration means which PE types and how many PEs for each type should be selected in a system. The system cost can be energy cost or other costs depending on the optimization goal.

Algorithms for different types of input task graphs are proposed in this paper. When the given task graph is a simple path or a tree, we propose two polynomial-time algorithms, *Path_Assign* (for path) and *Tree_Assign* (for tree), to generate optimal solutions. The complexity of these two algorithms is proportional to polynomial degrees of timing constraint, and the size of the input task graph. The general problem of task assignment in a heterogeneous environment has been shown to be NP-complete [16]. For the general problem, that is, when the given task graph is a *directed acyclic graph* (DAG), we design an integer linear programming (ILP) to obtain optimal solutions. The computation time of the ILP model, however, grows exponentially as the size of input increases. Therefore, we propose a heuristic algorithm, *DAG_Heu*, which gives near optimal solutions in polynomial time. Then, based on the obtained assignment, a minimum resource scheduling algorithm is proposed to generate a schedule and a configuration.

We conducted experiments on a set of benchmarks and compared our algorithms with the *greedy algorithm* [20] and the *Hybrid Particle Swarm Optimization (HPSO) algorithm* [34]. Experimental results show that when the input task graph is a tree or a simple path, the proposed algorithms reduce the cost by 42.4 % on average compared with the greedy algorithm; the proposed algorithms reduce the cost by 37.1 % on average when compared with the HPSO algorithm. When the input task graph is a DAG, the proposed algorithm, *DAG_Heu*, has significant improvement on total system cost reduction compared with the greedy algorithm and the HPSO algorithm. On average, the proposed algorithm reduces cost by 38.8 % and 31.1 % compared with the greedy algorithm and the HPSO algorithm, respectively. The experiments also show that the proposed heuristic algorithm always generates solutions close to the optimal solutions obtained by the ILP model.

The main contributions of this paper include:

- We propose polynomial-time algorithms, *Path_Assign* algorithm and *Tree_Assign* algorithm, to optimally solve the heterogeneous task assignment problem when the given DFG is a simple path or a tree.
- We formulate an ILP model for the general heterogeneous task assignment problem to obtain an optimal solution.
- We propose an efficient heuristic algorithm, *DAG_Heu*, for the general task assignment problem to produce near optimal solutions when the given input is a *directed acyclic graph* (DAG).
- We propose a minimum resource scheduling algorithm to generate a schedule and a configuration based on the obtained assignment.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 presents the system model and gives the formal definition of our problem. Section 4 uses an example to illustrate the motivation of this paper. Section 5 gives two optimal algorithms, *Path_Assign*, *Tree_Assign*, for the problem when the input graph is a path

or a tree. The ILP model of the problem is presented in Section 6. A heuristic algorithm for large-size problems, *DAG_Heu*, is presented in Section 7. Experimental results and concluding remarks are provided in Section 9 and Section 10, respectively. A brief description of the refined Hybrid Particle Swarm Optimization Algorithm (HPSO) is presented in Appendix.

2 Related Work

In this section, we briefly discuss previous related research in the areas of heterogeneous task assignment and scheduling problem.

The general problem of optimally mapping tasks to processors in a heterogeneous environment is shown to be NP-complete [16]. Therefore, only small-sized instances of the problem can be solved optimally using exact algorithms [3, 8, 17, 29, 33]. Chen et al. [8] propose a state-space reduction technique, branch-and-bound-with-underestimates, solving the task assignment problem in a distributed computing system to minimize the task turnaround time, which gives the optimal solutions. Kafil et al. [17] propose two algorithms based on the A* technique, which give optimal solutions for task assignment in heterogeneous distributed computing systems. Both [8] and [17], however, do not consider the timing constraint and the reliability requirement. [33] use dynamic programming technique obtaining optimal task assignment for tree-structure task model on heterogeneous multicore embedded systems considering timing constraint. However, they do not consider system reliability and their technique can only apply to tree-structure task model.

For large scale instances, most researchers have focused on developing heuristic algorithms that yield near-optimal solutions within a reasonable computation time. Heuristic algorithms are useful in applications where an optimal solution is not obtainable within a critical time limit. They are also applicable to large-size problems. Numerous constructive heuristics are reported for heterogeneous task assignment problem [2, 5, 6, 10, 14, 15, 25, 34]. Ahmad et al. [2] propose a technique based on the problem-space genetic algorithm (PSGA) for the static task assignment problem in both homogeneous and heterogeneous distributed computing systems. Bultan et al. [6] propose a mapping heuristic based on the Mean Field Annealing (MFA) algorithm. Hwang et al. [15] propose a greedy algorithm addressing the $k-$DTA problem to maximizing the system reliability. Salcedo et al. [25] introduce a novel formulation of TAP, in which each processor is limited in the number of task it can handle, due to the so-called resource constraint. They propose two hybrid meta-heuristic approaches. Both hybrid approaches use a Hopfield neural network to solve the problem's constraints, mixed with a genetic algorithm (GA) and a simulated annealing for improving the quality of the solutions found. However, these works either do not consider the timing constraint and the system reliability requirement, or the task model is different from what we are addressing.

For heterogeneous multiprocessor system, lots of works focus on task assignment under various computation models, computing system characteristics, objectives [3, 7, 12, 14, 31]. Chang et al. [7] propose an algorithm ETAHM, which is based on ant colony optimization algorithm, to allocate tasks on a target multiprocessor system. It mixes task scheduling, mapping, and Dynamic Voltage Scaling (DVS) utilization in one phase. Hsu et al. [14] target a synthesis problem for heterogeneous multiprocessor systems to schedule a set of periodic real-time tasks under a given energy consumption constraint. The objective is to minimize the processor cost of the entire system under the given timing and energy consumption constraints. Although [14] considers timing constraint, it does not consider the reliability requirement. Also, the task considered in [14] is independent and periodic, which is different from our task graph model.

From above works, we know that all existing work, either they do not consider the reliability requirement or the task model is different from what we addressing. In this paper, we consider task assignment problem with timing constraint and reliability requirement in heterogeneous multiprocessor systems. So the techniques proposed by all previous works cannot be directly applied for our problem.

Cell-library binding (technology mapping) is the task of transforming an unbound logic network into a bound network, i.e., into an interconnection of components that are instances of elements of a given library. Optimal tree covering can be computed by dynamic programming. When the objective is to minimize the total area of the bound network, the covering algorithm determines the matches of the locally rooted subtrees with the pattern trees by choosing the best labeling among all possible matches. When considering delay as the objective, the covering algorithm find minimum-delay cover to minimize the maximum path delay [22]. In both cases, there is no timing constraint, nor reliability constraint. On the contrary, the proposed *Path_Assign/Tree_Assign* algorithm address the problem of task assignment for heterogeneous multiprocessor system with guaranteed reliability and timing constraint. Although both the covering algorithm and *Tree_Assign* are based on dynamic programming and traverse the graph

in a bottom-up fashion, they are different because the the specific requirements and constraints presented in our problem.

3 Models and Problem Definition

In this section, we first describe the targeting heterogeneous system model and task model. We then present the reliability model. Finally, the problem addressed in this paper is defined.

3.1 System Model and Task Model

This paper targets heterogeneous multiprocessor system, which often comprises a general-purpose processor along with one or more DSP, FPGA, and/or ASIC. We consider there are maximum M different types of heterogeneous processing elements (PEs), or processors, $P = \{p_1, p_2, ..., p_M\}$.

We use *Data Flow Graph* (DFG) to model tasks to be executed on the heterogeneous system. A DFG $G = \langle V, E \rangle$ is a *directed acyclic graph* (DAG), where $V = \langle v_1, v_2, ..., v_N \rangle$ is the set of nodes, and $E \subseteq V \times V$ is the edge set that defines the precedence relations among nodes in V. Each node $v \in V$ represents a task to be executed. All resources operate non-preemptively, meaning a task could not transfer execution from one PE to another once it gets started.

The processing capabilities of different types of PEs might be different. Hence, a task has different execution times and costs when it is assigned to different PE types. $T(v_i, p_j)$ and $C(v_i, p_j)$ are used to denote the execution time and the cost required to complete execution of task v_i on one PE of p_j, respectively, when v_i executes alone and has all the resources that it requires. For instance, in Fig. 1a, $T(v_1, p_3) = 4$, $C(v_1, p_3) = 5$. If task v_i cannot execute on processor p_j, both $T(v_i, p_j)$ and $C(v_i, p_j)$ are infinity.

3.2 Reliability Model

The reliability model used here is the same as those used in [10, 15, 27, 28].

For heterogeneous multiprocessor systems, system reliability is defined as the probability that the system will not fail during the time of executing a DFG G. In order for a DFG G assigned to the processors in a heterogeneous multiprocessor system to run successfully, each processor must be operational during the period of application execution.

We use the widely accepted reliability model of Shatz and Wang [28], where each hardware component is fail-silent and is characterized by a constant failure rate per time unit f such that its reliability during the interval of time t is $e^{-f \cdot t}$ (that is, the occurrences of the failures follow a constant-parameter Poisson law). In real cases, the failure rate of each component is closely related to its physical parameters, such as the size and the material of the component. The data generated from operating life test sampling is the principal method used by the industry for estimating the failure rate of a semiconductor device in field service. Since reliability data can be accumulated from a number of different life tests with several different failure mechanisms, such as electromigration, Oxide Defects and Corrosion, a comprehensive failure rate can be calculated based on the failure mechanisms [32].

Based on the reliability model of Shatz and Wang [28], the reliability of PE p_j in time interval t is $(1 - f_j)^t$, where f_j is the failure rate of PE p_j. The reliability of the heterogeneous multiprocessor system during the execution a DFG under a task assignment X, $Pr(G, X)$, is defined as the probability that G can run successfully on the system under assignment X:

$$Pr(G, X) = \prod_{v_i \in V, p_j \in P} (1 - f_j)^{T(v_i, p_j) X_{i,j}} \tag{1}$$

Here, $X_{i,j}$ is a binary variable where $X_{i,j} = 1$ if task v_i is assigned to PE p_j; otherwise, it equals 0. When f_j is small, $1 - f_j \approx e^{-f_j}$. Therefore, $Pr(G, X) \approx \prod (e^{-f_j T(v_i, p_j) X_{i,j}})$.

In our system, reliability requirement serves as one of the constraints that must be satisfied. Suppose the system reliability $Pr(G, X)$ must be greater than or equal to R ($0 < R < 1$), that is:

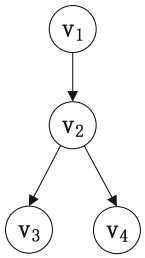
$$\prod_{v_i \in V, p_j \in P} \left(e^{-f_j T(v_i, p_j) X_{i,j}} \right) = e^{-\sum_{v_i \in V} \sum_{p_j \in P} (f_j T(v_i, p_j) X_{i,j})} \geq R \tag{2}$$

We could deduce the following inequality from Eq. 2.

$$\sum_{v_i \in V} \sum_{p_j \in P} (f_j T(v_i, p_j) X_{i,j}) \leq -ln R \tag{3}$$

In this paper, we define an accumulated failure rate function $F(v_i, p_j) = f_j \times T(v_i, p_j)$. Therefore, a task has different accumulated failure rates when it is assigned to different PE types. For example, in Fig. 1a, $F(v_1, p_2) = 4$, $F(v_1, p_3) = 6$. These accumulated failure rates are normalized values to bring the entire probability distributions of adjusted values into alignment.

We assume that the failure of a component in the system follows a Poisson process and each component is accordingly associated with a constant failure rate. In real world the failure rate of a component may not be constant. We can always use accumulated failure rate $F(v_i, p_j)$ to model



(a)

Nodes	p1			p2			p3		
	T	F	C	T	F	C	T	F	C
v1	2	6	7	2	4	10	4	2	5
v2	2	6	6	1	2	11	2	1	7
v3	1	3	8	1	2	11	2	1	10
v4	1	3	10	3	6	8	2	1	11

(b)

Figure 1 **a** a given tree; **b** the time, accumulated failure rate, and execution cost of its node for different PE types.

it. It should be noted that modeling the failure of a component by a Poisson process may not always coincide with the actual failure dynamic of the resource. However, it is shown experimentally in [24] that such an assumption can still result in reasonably useful mathematical models. For mathematical tractability, failures of components are assumed to be statistically independent. In addition, once a component has failed, it is assumed that it remains in the failed state for the remainder of the execution of the application.

3.3 Problem Definition

For a given DFG $G = \langle V, E\rangle$, define a task assignment function $A : V \rightarrow P$, where task $v_i \in V$ is assigned to PE $p_i \in P$. For example, $A(v_1) = p_2$ means that task v_1 is assigned to PE p_2. Given an assignment A of a DFG G, we define the *total system cost*, denoted by $C_A(G)$, to be the summation of execution cost of all nodes. That is, $C_A(G) = \sum_{v\in V} C(v, A(v))$. Similarly, define the *accumulated system failure rate*, $F_A(G)$, to be the summation of accumulated failure rate of all nodes according to our analysis and assumptions. That is, $F_A(G) = \sum_{v\in V} F(v, A(v))$.

For an input graph DFG, under an assignment A, define the *execution time of graph* G, denoted by $T_A(G)$, to be the completion time of a critical path Pa in G, $T_A(G) = \max_{\forall Pa} T_A(Pa)$, where $T_A(Pa) = \sum_{v\in Pa} T(v, A(v))$.

The problem considered in this paper is defined as follows:

Definition 1 Task Assignment and Scheduling for Heterogeneous Multiprocessor System with Guaranteed Reliability and Timing Constraint (TASHM) problem.

Given a set P of available different PE types: $P = \{p_1, p_2, ..., p_M\}$, a DFG $G = \langle V, E\rangle$, execution time $T(v_i, p_j)$, accumulated failure rate $F(v_i, p_j)$, cost $C(v_i, p_j)$ for task v_i executed on PE p_j, a timing constraint L, an accumulated system failure rate constraint W, find an assignment A for G that has the minimum total cost $C_A(G)$ under timing constraint L and accumulated failure rate constraint W.

4 Motivational Example

In this section, we present an example of the TASHM problem to show the main idea of the proposed algorithms.

There are three PE types, p_1, p_2, p_3. We assume that each PE type has enough processors. Figure 1a shows the task graph, which is a tree with four nodes. Figure 1b gives the corresponding execution times, accumulated failure rates, and execution costs. For example, v_1 can choose one of the three types: p_1, p_2 or p_3. When choosing p_1, it will be finished in 2 time units and the accumulated failure rate is 6. The corresponding execution cost of type p_1 is 7. When node v_1 is assigned to type p_2, it will be finished in 2 time units with accumulated failure rate 4 and execution cost 10.

Table 1 gives five different assignments. The timing constraint considered in this example is $L = 7$, and the accumulated failure rate constraint is $F = 9$. For assignment "1", all tasks are assigned to PE p_1. With the total execution cost 31, the completion time and accumulated failure rate of the DFG G is 5 and 18 respectively. The accumulated failure rate constraint is not satisfied in assignment "1". For assignment "2", all four tasks are assigned to PE p_3 with total cost 33, completion time 8 and accumulated failure rate 5. Although assignment "2" achieves the lowest accumulated failure rates, the timing constraint is not satisfied and cannot be applied.

Assignment "3" and assignment "4" meet both timing constraint and the accumulated failure rate constraint, however, they do not achieve the minimum execution cost. Assignment "5", which can be obtained by the proposed algorithm, is an optimal assignment which achieves the minimum execution cost. The total execution cost is being reduced by 37.5 % compared with the result generated by assignment "3".

Table 1 Five assignments for the task graph in Fig. 1.

Num.	Assignment	Completion time	Accumulated failure rate	Total cost	Reduc. (vs. assign.3)
1	$v_1, v_2, v_3, v_4 \rightarrow p_1$	5	18 $\star$	31	Invalid
2	$v_1, v_2, v_3, v_4 \rightarrow p_3$	8 $\star$	5	33	Invalid
3	$v_1, v_2, v_3 \rightarrow p_2$; $v_4 \rightarrow p_3$;	5	9	43	–
4	$v_1 \rightarrow p_2$; $v_2, v_4 \rightarrow p_3$; $v_3 \rightarrow p_1$;	6	9	36	16.3 %
5	$v_1, v_2 \rightarrow p_3$; $v_3, v_4 \rightarrow p_1$	7	9	30	30.2 %

From the above five different assignments, we can see that task assignment has big influence on the total system cost. Moreover, some improper assignment can even end up with the situation where the execution cannot meet the deadline or the reliability requirement. And the TASHM problem is NP-Complete [26]. Therefore, there is a strong need to devise efficient techniques to solve the TASHM problem during the design space exploration process.

For Assignment "5", two different schedules with corresponding configuration are shown in Fig. 2. The configuration in Fig. 2a uses four PEs while the configuration in Fig. 2b uses three PEs. The schedule in Fig. 2b can be generated by the minimum resource scheduling algorithm in Section 8, in which the configuration achieves the minimal resource for assignment "5".

5 Optimal Algorithms for Paths and Trees

In order to find an optimal assignment, in this section, we present two algorithms, *Path_Assign* and *Tree_Assign*, to solve the TASHM problem when the input graph is a path or tree. We also prove the optimality of the proposed algorithms.

5.1 An Optimal Algorithm for Simple Paths

$$CT[G_i][l][w] = \begin{cases} \min_{1 \le m \le M}\{CT[G_{i-1}][l - T(v_i, p_m)][w - F(v_i, p_m)] + C(v_i, p_m) \\ \quad if\, l - T(v_i, p_m) \ge 0, w - F(v_i, p_m) \ge 0, CT[G_{i-1}][l - T(v_i, p_m)][w - F(v_i, p_m)] \ge 0\} \\ no\ feasible\ solution, \text{otherwise.} \end{cases} \quad (4)$$

In this subsection, we propose an efficient algorithm, *Path_Assign*, to produce an optimal solution for the TASHM problem when the input DFG is a path. *Path_Assign* algorithm is based on dynamic programming.

Assume the node sequence of the path is $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_N$. Let G_i be the path from v_1 to v_i. In algorithm *Path_Assign*, we build table $CT[G_i]$ and table $PRO[v_i]$ for each node $v_i \in V$. Table $CT[G_i]$, which has $L \times W$ entries, represent the minimum system cost for G_i. Table entry $CT[G_i][l][w]$, where $1 \le l \le L$ and $1 \le w \le W$, is the minimum system cost for G_i when the following three conditions are satisfied: all the nodes including v_i on G_i are optimally assigned; the total execution time of G_i is less than or equal to l (a positive integer, $1 \le l \le L$); and the accumulated failure rate of G_i is less than or equal to w (a positive integer, $1 \le w \le W$).

We record the corresponding selected PEs at table $PRO[v_i]$ when calculating $CT[G_i][l][w]$. $PRO[v_i][l][w]$ is the PE type assigned to node v_i when achieving $CT[G_i][l][w]$. Using table $PRO[v_i]$, the assignments for all the tasks can be obtained.

According to the definition of $CT[G_i][l][w]$, the recursive function of *Path_Assign* algorithm can be constructed as shown in Eq. 4. $CT[G_i][l][w]$ is the summation of the cost of node v_i and the cost of subgraph G_{i-1}. Because the timing constraint and the accumulated failure rate constraint have to be satisfied at the same time, $CT[G_i][l][w]$ can be obtained only if $l - T(v_i, p_m) \ge 0$, $w - F(v_i, p_m) \ge 0$, $CT[G_{i-1}][l - T(v_i, p_m)][w - F(v_i, p_m)] \ge 0$, which

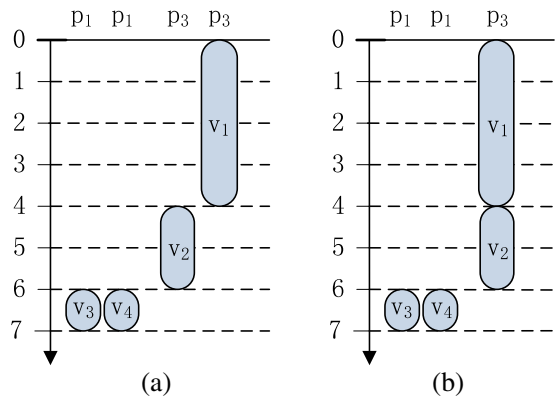


Figure 2 Two schedules corresponding to assignment "5".

means there has to be an optimal assignment for the subgraph G_{i-1} under $l - T(v_i, p_m)$ and $w - F(v_i, p_m)$ before the execution of node v_i. Otherwise, there is no feasible solution.

According to the recursive function, *Path_Assign* algorithm is presented in Algorithm 5.1. Assume the node sequence of the path is $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_N$. We compute table $CT[G_i]$ for each node v_i starting from node v_1. Meanwhile, use $PRO[v_i][l][w]$ to record the PE types assigned to node v_i. By tracing back table $PRO[v_i]$, an optimal assignment $\{A(v_1), A(v_2), \cdots, A(v_N)\}$ for G can be obtained, and the minimum system cost of G is $CT[G_N][L][W]$.

It takes $O(M)$ to compute $CT[G_i][l][w]$, where M is the maximum number of PE types. Thus, the complexity of the algorithm *Path_Assign* is $O(M * W * L * N)$, where N is the number of nodes, L is the given timing constraint, and W is the accumulated system failure rate constraint. The complexity of algorithm *Path_Assign* is polynomial to timing constraint L, accumulated system failure rate constraint W and the input graph size.

5.2 An Optimal Algorithm for Trees

In this subsection, we present the *Tree_Assign* algorithm to produce an optimal solution for the TASHM problem when the input DFG is a tree.

Define a *root* node to be a node without any parent and a *leaf* node to be a node without any child. A *postordering* for a tree is a linear ordering of all its nodes such that, if there is an edge $u \rightarrow v$ in the tree, then v appears before u in the ordering. Postordering is used to guarantee that, when we begin to process a node, the processing of all its child nodes has already been finished in the algorithm.

$$CT[G_i][l][w] = \begin{cases} \min_{1 \le m \le M}\{CT[G'_i][l - T(v_i, p_m)][w - F(v_i, p_m)] + C(v_i, p_m) \\ \quad if\ l - T(v_i, p_m) \ge 0, w - F(v_i, p_m) \ge 0, CT[G'_i][l - T(v_i, p_m)][w - F(v_i, p_m)] \ge 0\} \\ no\ feasible\ solution, \text{otherwise.} \end{cases} \tag{5}$$

$$CT[G_i \oplus G_j][l][w] = \min_{1 \le r \le w} \{CT[G_i][l][w - r] + CT[G_j][l][r]\} \tag{6}$$

Let G_i be the subtree rooted at node v_i and G'_i be the subtree rooted at node v_i except v_i. For example, in Fig. 1a, G_2 is the tree containing nodes v_2, v_3, v_4, and G'_2 is the trees containing nodes v_3, v_4. Define $CT[G_i]$, which has $L \times W$ entries, to be the minimum total cost table for executing G_i. Table entry $CT[G_i][l][w]$ $(1 \le l \le L, 1 \le w \le W)$, is the minimum total cost for executing G_i under timing constraint l and accumulated failure rate constraint w. A matrix $PRO[v_i]$, which has $L \times W$ entries, is used to record the corresponding selected PEs for node v_i. $PRO[v_i][l][w]$ $(1 \le l \le L, 1 \le w \le W)$ records the selected PEs for node v_i under l and w, respectively. $CT[G_i][l][w]$ and $PRO[v_i][l][w]$ is an one-to-one correspondence.

$$CT[G'_i] = \begin{cases} 0 & if\ h = 0 \\ CT[G_{i_1}] & if\ h = 1 \\ CT[G_{i1}] \oplus CT[G_{i2}] \oplus \cdots \oplus CT[G_{ih}] & if\ h > 1 \end{cases} \tag{7}$$

The recursive function is shown in Eq. 5. The execution cost of G_i is the summation of the execution cost of G'_i and the cost of execute node v_i.

The main idea of *Tree_Assign* algorithm is constructing the table $CT[G_i]$ and table $PRO[v_i]$ for each node one by one following the sequence of the postordering of the

Algorithm 5.1 *Path_Assign* algorithm

Input: A DFG $G =< V, E, P >$, where P is a set of M type of PEs, a timing constraint L, and an accumulated failure rate constraint W.
Output: An optimal assignment for G.

```
for i ← 1 to N do
    for l ← 1 to L do
        for w ← 1 to W do
            Compute CT[G_i][l][w] by Equation (4), and record values of PRO[v_i][l][w] during
            the computation of CT[G_i][l][w];
        end for
    end for
end for
{A(v_1), A(v_2), ..., A(v_N)} ← an optimal assignment for G obtained by tracing how to
reach CT[G_N][L][W] using the table PRO[v_i] of each node v_i.
```

Algorithm 5.2 *Tree_Assign* algorithm: optimal algorithm for the TASHM problem when the input is a tree

Input: A DFG $G =< V, E, P >$, where P is a set of M types of PE, a timing constraint L, and an accumulated failure rate constraint W.
Output: An optimal assignment for G.
1: $\{v_1, v_2, \cdots, v_N\} \leftarrow$ Postorder G.
2: **for** $i \leftarrow$ 1 *to* N **do**
3: Compute $CT[G'_i]$ by Equation (7) using Algorithm 5.3.
4: **for** $l \leftarrow$ 1 *to* L **do**
5: **for** $w \leftarrow$ 1 *to* W **do**
6: Compute $CT[G_i][l][w]$ by Equation (5), and keep the record of assignment for node v_i at time l and accumulated failure rate w as $PRO[v_i][l][w]$.
7: **end for**
8: **end for**
9: **end for**
10: $\{A(v_1), A(v_2), \cdots, A(v_N)\} \leftarrow$ backtrace table $CT[G_i]$ and $PRO[v_i]$ to obtain the final assignment.

task graph. We first implement topological sorting of all the nodes from *leaf* to the *root* (line 1). For each node v_i in the sorting sequence, compute table $CT[G'_i]$ first, which is the minimum total cost for G'_i under various constraints (line 3). We compute $CT[G'_i]$ according to different situations by Eq. 7 using Algorithm 5.3. In Eq. 7, h is the number of the child nodes of task v_i. After that, we build the matrix $CT[G_i]$ by Eq. 5 and matrix $PRO[v_i]$ for node v_i (line 4–9).

Here, we introduce the operator "$\oplus$" to illustrate the procedure of merging all the child nodes of node v into one pseudo node. For two graph G_i and graph G_j, where both node v_i and node v_j are the child nodes of a certain node v, after the $\oplus$ operation between $CT[G_i]$ and $CT[G_j]$ by Algorithm 5.3, we get table $CT[G_i \oplus G_j]$ and table $FA[G_j]$, where $G_i \oplus G_j$ is the forest containing tree G_i and tree G_j. For example, in Fig. 1a, $G_3 \oplus G_4$ is the forest containing node v_3 and node v_4. $G_3 \oplus G_4$ is G'_2 in this case. We denote this operation $CT[G_i] \oplus CT[G_j]$. This is the key operation of our algorithm.

$CT[G_i \oplus G_j][l][w]$ computed by Eq. 6 is the minimum total cost for $G_i \oplus G_j$ under timing constraint l and accumulated failure rate constraint w. Table $FA[G_j]$ is constructed during the computation of $CT[G_i \oplus G_j][l][w]$, and is used to obtain an optimal assignments by tracing how to reach $CT[G_N][L][W]$. $FA[G_j][l][w]$ is the accumulated failure rate of G_j when we achieve the value of $CT[G_i \oplus G_j][l][w]$ by Eq. 6.

From Eq. 7, $CT[G'_i]$ gets the minimum total cost of G'_i. From Eq. 5, the minimum total cost of G_i is selected from all possible costs caused by adding v_i to G'_i. Therefore, for each value $CT[G_i][l][w]$, it is the minimum total cost for graph G_i under timing constraint l and accumulated failure rate constraint w.

To better illustrate the Tree_Assign algorithm, we use the task graph in Fig. 1a with timing constraint 7 and accumulated system failure rate constraint 9 as an example. $CT[G_i]$ and $PRO[v_i]$ for this example is shown in Fig. 3. All of those table are obtained by the proposed algorithm. Algorithm *Tree_Assign* first constructs table $CT[G_i]$ and table $PRO[v_i]$ for node v_3 and node v_4. Then $CT[G'_2]$ and $FA[G_4]$ is been built before computing $CT[G_2]$ and $PRO[v_2]$. $CT[G_i]$ and $PRO[v_i]$ for node v_1 and node v_2 are built next. For the DFG shown in Fig. 1a, the time, accumulated failure rate and execution cost for each task are shown in Fig. 1b. According to the table $CT[G_1]$ in Fig. 3a, the minimum total system cost of the tree is 30, which is the value of $CT[G_1][7][9]$. According to $PRO[v_1][7][9] = 3$, which means node v_1 is assigned to PE p_3, we trace back to $PRO[v_2][3][7]$ since $7 - T(v_1, p_3) = 3$ and $9 - F(v_1, p_3) = 7$. From $PRO[v_2][3][7] = 3$, we know v_2 is assigned to PE p_3. Using table $CT[G'_2]$ and $FA[G_4]$, we

Algorithm 5.3 *Merg* algorithm

Input: $CT[G_i]$ and $CT[G_j]$.
Output: The merged table $CT[G_i \oplus G_j]$ and table $FA[G_j]$.
1: **for** $l \leftarrow$ 1 *to* L **do**
2: **for** $w \leftarrow$ 1 *to* W **do**
3: Compute $CT[G_i \oplus G_j][l][w]$ by Equation (6).
4: Assume r is the value where we get the minimum $CT[G_i][l][w-r] + CT[G_j][l][r]$ in Equation (6). Then,
$FA[G_j][l][w] \leftarrow r$.
5: **end for**
6: **end for**

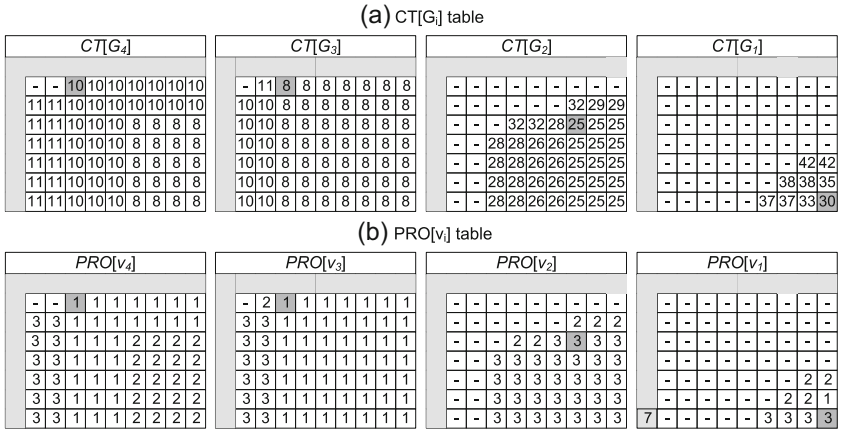


Figure 3 Table $CT[G_i]$ and table $PRO[v_i]$ for Fig. 1a under timing constraint 7 and accumulated failure rate constraint 9.

trace back to $PRO[v_4][1][3] = 1$ and $PRO[v_3][1][3] = 1$. Finally, we get an optimal assignment, $A(v_1) = p_3$, $A(v_2) = p_3$, $A(v_3) = p_1$, $A(v_4) = p_1$. In Fig. 3, we mark these numbers with gray color.

It takes $O(M)$ to compute $CT[G_i][l][w]$, where M is the maximum number of PE types. The complexity of Algorithm 5.2 is $O(M * W^2 * L * N)$, where N is the number of nodes, L is the given timing constraint, W is the given accumulated failure rate constraint.

6 An ILP Model for the TASHM Problem

The TASHM problem is NP-Complete [26]. In order to achieve optimal results, in this section, we present an ILP formulation for the TASHM problem.

We first list the notations used for ILP model in Table 2. We assume that there have enough PEs for each type.

Constraint 1: Each node only can be executed at one PE at any time.

$$\sum_{p_m \in P} X_{i,m} = 1 \qquad \forall v_i \in V \tag{8}$$

Constraint 2: Each node must follow the precedence relationship defined by the DFG. That is, node v_j can be executed only after all its ancestors have finished.

$$S(v_i) + \sum_{p_m \in P} [X_{i,m} \times T(v_j, p_m)] \leq S(v_j) \qquad \forall (v_i, v_j) \in E \tag{9}$$

Constraint 3: For those nodes without any parent, the earliest execution time is zero.

$$\sum_{p_m \in P} [X_{i,m} \times T(v_i, p_m)] \leq S(v_i) \qquad \forall v_i \in V \tag{10}$$

Constraint 4: All the nodes must be finished in time L to meet the given timing constraint L.

$$S(v_i) \leq L \qquad \forall v_i \in V \tag{11}$$

Constraint 5: The accumulated system failure rate of this DFG, which is the summation of accumulated failure rate of all nodes, must be less than or equal to W.

$$\sum_{p_m \in P} \sum_{v_i \in V} [X_{i,m} \times F(v_i, p_m)] \leq W \tag{12}$$

Objective function: We want to minimize the total system cost with timing constraint and accumulated failure

Table 2 Notations used in ILP model.

Notation	Definition
$X_{i,m}$	a binary variable used to determine which PE v_i is assigned to.
$S(v_i)$	the earliest finish time of node v_i.
$T(v_i, p_m)$	the execution time of node v_i when it is assigned to PE p_m.
$C(v_i, p_m)$	the execution cost of node v_i when it is assigned to PE p_m.
$F(v_i, p_m)$	the accumulated failure rate of node v_i when it is assigned to PE p_m.

rate constraint. Total system cost is the summation of execution cost of all nodes.

$$\min \sum_{p_m \in P} \sum_{v_i \in V} [X_{i,m} \times C(v_i, p_m)] \tag{13}$$

We can always obtain optimal assignments using ILP method. However, it will not find an optimal assignment in an acceptable time when the node number of DFG is large. ILP is only suitable for small cases. For large cases, we propose a polynomial-time near-optimal algorithm in Section 7.

7 A Heuristic Algorithm for the General TASHM Problem

In Section 6, we presented an ILP formulation for the TASHM problem. However, ILP take exponential time to finish. In this section, we propose a polynomial-time algorithm, *DAG_Heu*, to attain near-optimal results when the input to the TASHM problem is a general *directed acyclic graph* (DAG).

We devise a novel method to dynamically adjust the reduction range of the completion time or the accumulated failure rate during the optimization process by using a parameter $ratio(v)$. Only one task that is the most cost-effective among all tasks is chose to change its PE type in each iteration.

The basic idea of *DAG_Heu* is to first assign each node with the best cost PE type, then iteratively change the PE type of the node such that the timing constraint and accumulated failure rate constraint can be satisfied with minimum system cost increased.

The detail of DAG_Heu is shown in Algorithm 7.1. The DAG_Heu algorithm first sorts the PE types for each task in descending order according to execution cost (line 2). Then, the best cost type $y_i[M]$ is assigned to task v_i (line 3). We use a variable $Mobility(v_i)$ to record the number of times node v_i can still change. The initial value of $Mobility(v_i)$ is $M-1$ (line 4), because each node can change its PE type from current assigned type $y_i[M]$ to all other types. We then compute the completion time T_G and the accumulated failure rate F_G of the graph based on the current assigned PEs for DFG G. Next, if $T_G > L$ or $F_G > W$, we reduce them iteratively by changing the type of a node which is selected among all the nodes.

$$Co(T) = \frac{T_G - L}{(T_G - L) + (F_G - W)} \tag{14}$$

$$Co(F) = \frac{F_G - W}{(T_G - L) + (F_G - W)} \tag{15}$$

In each iteration of reducing the completion time and the accumulated failure rate process, we select a node v_i to change its PE based on its ratio, denoted by $ratio(v_i)$. $ratio(v_i)$ is used to represent the average decreased time and accumulated failure rate per increased cost unit for node v_i. We compute the value of $ratio(v_i)$ for each node $v_i \in V$ by Eq. 16.

Because both the completion time and the accumulated failure rate of DFG G have to be taken into consideration

Algorithm 7.1 *DAG_Heu* algorithm: heuristic algorithm for the general TASHM problem when the input is a DAG

Input: A DFG $G =< V, E, P >$, where P is a set of M PE types, a timing constraint L, and an accumulated failure rate constraint W.
Output: A near optimal assignment for DFG G.
1: **for** $i \leftarrow$ 1 *to* N **do**
2: $\{y_i[1], y_i[2], \cdots, y_i[M]\} \leftarrow$ sort the PE types for task v_i in descending sequence according to execution cost.
3: $A(v_i) \leftarrow y_i[M]$.
4: $Mobility(v_i) \leftarrow M - 1$.
5: **end for**
6: Compute the completion time of G, denoted by T_G, and the accumulated failure rate of G, denoted by F_G.
7: **while** $T_G > L$ or $F_G > W$ **do**
8: Compute $Co(T)$ and $Co(F)$ by Equation (14) and Equation (15).
9: **for** $i \leftarrow$ 1 *to* N **do**
10: According to $Mobility(v_i)$, compute $ratio(v_i)$ for node v_i by Equation (16).
11: Update $ToType(v_i)$ to $y_i[m]$ where v_i gets the maximum ratio by Equation (16).
12: **end for**
13: Find node v_i which has the maximum value of $ratio(v_i)$, then
$A(v_i) \leftarrow ToType(v_i)$,
Update $Mobility(v_i)$.
14: Compute the completion time T_G and the accumulated system failure rate F_G of the graph based on the new assignment for G.
15: **end while**
16: $\{A(v_1), A(v_2), \cdots, A(v_N)\}$.

during the reducing process to satisfy the timing constraint L and accumulated failure rate constraint W, we use two proportional coefficients, $Co(T)$ and $Co(F)$, to represent the corresponding proportions. $Co(T)$ and $Co(F)$, calculated by Eqs. 14 and 15, denote the proportion of excess execution time and the proportion of excess accumulated failure rate of DFG G, respectively. More specifically, if $T_G - L$ is bigger than $F_G - W$, $Co(T)$ is bigger than $Co(F)$. Thus, the reduction of the execution time is faster than the reduction of the accumulated failure rate by Eq. 16. We change those two proportional coefficients dynamically in each iteration. $Co(T)$ and $Co(F)$ remain the same for each node within one iteration.

We use three variables, $ChT(v_i, y_i[m])$, $ChF(v_i, y_i[m])$, and $IncreasedCost(v_i, y_i[m])$, to record changed execution time, changed accumulated failure rate, and changed system cost of G, respectively, when the assigned PE for node v_i is changed from its current type to type $y_i[m]$. Since the PE type changing only happens from type p_i with lower execution cost to PE p_j with higher execution cost for each node, the total system cost is always getting larger when changing the PE type of each node. Hence, $IncreasedCost(v_i, y_i[m])$ represents the increased cost and is a positive number. $ChF(v_i, y_i[m])$ is the value of $F(v_i, y_i[m]) - F(v_i, y_i[c])$.

Let $T_A(G)$ be the execution time of G with original assignment. Let $T'_A(G)$ be the execution time of G with the assigned PE type for v_i is changed to $y_i[m]$. Then, $ChT(v_i, y_i[m]) = T_A(G) - T'_A(G)$. $ChT(v_i, y_i[m])$ can be positive, negative, and zero according to different situations. By using coefficient $Co(T)$, we do not need to consider wether $ChT(v_i, y_i[m])$ is positive or negative. If it is negative and $Co(T)$ is larger than $Co(F)$, the value of $ratio(v_i)$ for PE $y_i[m]$ will be small.

Table 3 Benchmarks Information.

Benchmark	DFG	# of nodes
kbasic_path	Path	42
2IIR	Tree	16
4-Lat-IIR	Tree	26
Voltera	Tree	27
8-Lat-IIR	Tree	42
kbasic_tree	Tree	92
2-deq	DAG	22
3-deq	DAG	33
elf	DAG	34
2-rls-lat	DAG	38
3-elf	DAG	102
20-4Lat-IIR	DAG	520
kbasic_DAG	DAG	583

We use the variable $ToType(v_i)$ to record the corresponding PE $y_i[m]$ where node v_i gets the maximum ratio by Eq. 16 when its type is changed to $y_i[m]$. After calculating all the ratio for each node, we pick a node v_i which has the maximum value of $ratio(v_i)$, and change its PE type to $ToType(v_i)$. Then, update the value of $Mobility(v_i)$. Meanwhile, compute the execution time T_G and the accumulated failure rate F_G of the graph based on the new assigned PEs for DFG G. If both the timing constraint and accumulated failure rate constraint are satisfied, that is, $T_G \leq L$ and $F_G \leq W$, we stop the *while* loop, and the current assignments for each node is the final result.

$$ratio(v_i) = \max_{1 \leq m \leq Mobility(v_i)} \left\{ \frac{Co(T) * ChT(v_i, y_i[m]) + Co(F) * ChF(v_i, y_i[m])}{IncreasedCost(v_i, y_i[m])} \right\} \tag{16}$$

Given a DFG G, it takes $O(|E|)$ to find a critical path, where $|E|$ is the number of edges. In the DAG_Heu algorithm, in each iteration, we have to calculate the execution time of DFG G every time when calculating ratio for each node, thus, it takes at most $O(|E| * M)$ to calculate ratios for one node, where M is the number PE types. And it takes at most $O(N * |E| * M)$ to calculate ratios for all nodes. The algorithm iterates at most $N * M$ times, since each type of a node is only assigned one time. Therefore, the time complexity of the DAG_Heu algorithm is $O(N^2 * |E| * M^2)$. Considering M is a constant, the DAG_Heu algorithm's complexity is $O(N^2 * |E|)$.

8 The Minimum Resource Scheduling and Configuration

In this section, we propose minimum resource scheduling algorithms, which take two major steps, to generate a schedule and a configuration. Firstly, the *Lower_Bound algorithm* is proposed to produce an initial configuration with low bound resource. Then, we devise the *Min_Scheduling algorithm* which refines the initial configuration and generates a schedule to satisfy the timing constraint and the accumulated failure rate constraint.

Given a DFG with PE assignments and a timing constraint L, *Lower_Bound algorithm* obtains the lower bound for each PE type. In the algorithm, it counts the number of the same type of PEs in each control step in the As Soon As Possible (ASAP) and As Late As Possible (ALAP) schedules, respectively. We denote the number of PE type p_i in step l for ASAP and ALAP schedules as $num\_S(l,i)$ and $num\_L(l,i)$, respectively. The lower bound for PE type p_i for ASAP schedule is $\max_{1\le l\le L}\{\frac{\sum_{l\le j\le L} num\_S(j,i)}{L-l+1}\}$. For ASAP schedule, the corresponding lower bound is $\max_{1\le l\le L}\{\frac{\sum_{1\le j\le l} num\_L(j,i)}{l}\}$. The lower bound for each PE type is the maximum value of these two schedules.

Using the lower bound of each PE obtained with *Lower_Bound algorithm* as an initial configuration, *Algorithm Min_Scheduling* is proposed to generate a schedule to use as little resource as possible. In the algorithm, it first compute $ALAP(v)$ for each node $v \in V$, where $ALAP(v)$ is the schedule step of node v in the ALAP schedule. Then, we use a revised list scheduling to perform scheduling. Specifically, in each step,

Table 4 Experimental results for benchmarks of trees with various constraints.

				Path_Assign/Tree_Assign			
Benchmark	L/W(10^{-5})	Greedy cost	HPSO Cost	Cost	%(Greedy)	%(HPSO)	Configuration
kbasic_path	130/160	545	521	395	27.5 %	24.2 %	1, 1, 1, 0
	160/190	504	484	326	35.3 %	32.6 %	1, 1, 1, 0
	190/220	444	422	267	39.9 %	36.7 %	1, 1, 1, 0
	220/250	399	384	216	45.9 %	43.8 %	0, 1, 1, 0
	250/280	347	318	174	49.9 %	45.3 %	0, 1, 1, 1
2IIR	15/40	196	194	158	19.4 %	18.6 %	2, 0, 4, 0
	20/50	166	155	109	34.3 %	29.7 %	1, 0, 5, 1
	25/60	163	144	85	47.9 %	41.0 %	1, 1, 5, 1
	30/70	145	106	65	55.2 %	38.7 %	1, 1, 5, 2
	35/80	133	90	50	62.4 %	44.4 %	1, 1, 3, 2
4-Lat-IIR	20/60	319	306	242	24.1 %	20.9 %	3, 0, 4, 0
	30/80	263	256	149	43.3 %	41.8 %	1, 0, 4, 2
	40/100	225	216	112	50.2 %	48.1 %	1, 2, 4, 1
	50/120	196	149	85	56.6 %	43.0 %	1, 1, 5, 3
	60/140	159	127	68	57.2 %	46.5 %	1, 1, 3, 4
Voltera	25/60	332	310	276	16.9 %	11.0 %	2, 1, 3, 0
	35/90	279	238	154	44.8 %	35.3 %	1, 1, 5, 1
	45/120	237	188	119	49.8 %	36.7 %	1, 2, 3, 3
	55/150	185	176	96	48.1 %	45.5 %	1, 1, 4, 3
	65/180	139	131	81	41.7 %	38.2 %	1, 1, 2, 4
8-Lat-IRR	30/90	506	514	445	16.9 %	11.0 %	3, 0, 2, 0
	40/120	452	464	294	35.0 %	36.6 %	2, 0, 4, 2
	50/150	394	340	222	43.7 %	34.7 %	1, 2, 5, 1
	60/180	362	316	178	50.8 %	43.7 %	1, 1, 5, 2
	70/210	317	301	150	52.7 %	50.2 %	1, 1, 5, 3
kbasic_tree	60/250	1047	1005	703	32.9 %	30.0 %	3, 1, 6, 0
	70/300	922	840	525	43.1 %	37.5 %	3, 3, 7, 2
	80/350	796	787	407	48.9 %	48.3 %	2, 3, 4, 0
	90/400	675	620	327	51.6 %	47.3 %	1, 1, 7, 4
	100/450	556	542	272	51.1 %	49.8 %	1, 1, 5, 3
Average reduction					42.4 %	37.1 %	

we first schedule all nodes that have reached the deadline with additional resources if necessary to guarantee that no task misses the deadline. Then schedule all other nodes, as many as possible, without increasing resources. The strategy of our scheduling algorithm is to maintain the minimal resources as long as the timing constraint is satisfied. *Lower_Bound* and *Min_Scheduling* algorithms both take $O(N + |E|)$ to get results, where N is the number of nodes and $|E|$ is the number of edges in a DFG.

9 Experiment

In this section, we first present the experimental setup of our simulation. Then, we provide the detailed experimental results for each benchmark.

9.1 Experimental Setups

We conduct experiments with the proposed algorithms on a set of benchmarks including 2IIR, 4-Stage Lattice Filter, Voltera Filter, 8-Stage Lattice Filter, kbasic_path, kbasic_tree, 2-deq, 3-deq, elf, 2-RSL-Languerre Lattice Filter, 3-elf, 20-4Stage Lattice Filter, and kbasic_DAG. The task graphs, kbasic_path, kbasic_tree, and kbasic_DAG, are generated using TGFF as explained in [11]. The rest benchmarks are from DSPstone benchmarks [35]. The basic information about the benchmarks including their structures and the number of nodes is shown in Table 3. The first benchmark is a path, which is used to test the *Path_Assign* algorithm. The next five benchmarks are trees, which are used for testing the *Tree_Assign* algorithm. The rest eight benchmarks are DAGs using for testing *DAG_Heu* algorithm. Four different PE types, p_1, p_2, p_3, p_4, are used in the system. A PE type has the highest capability may not so reliable and have a higher failure rate than other types.

The execution time of each task of the task graph is uniformly distributed between 1 and 10 min, where the execution times of a given task are different on different types of PE. The failure rates of processors are assumed to be uniformly distributed between 10^{-4} and 10^{-3} failures/h [24]. Based on the value of execution time of each task and the failure rate of processors, we calculate the accumulated failure rate of execution of each task, which is a random variable in the range of $[0.16 \times 10^{-5}, 0.17 \times 10^{-3}]$. Execution cost of each task on different PE types is simulated by uniformly distributing the values on a range of [1 14]. In the simulation environment, we generally assign higher cost for PEs with higher performance and/or better reliability, which is true for most of the cases in real world.

The proposed algorithms are compared with the *Greedy algorithm* [20] and the *HPSO algorithm* [34]. For HPSO algorithm, the results are averaged over 20 trials. The parameter values that control HPSO algorithm are given as follows: (a) c1 = c2 = 2.05. (b) Population size = 80.

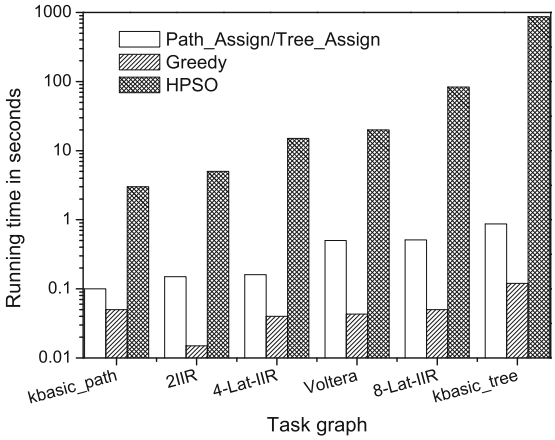


Figure 4 The running time comparison of different techniques for the tree structure benchmarks.

9.2 Experimental Results

The experimental results for tree benchmarks, 2IIR, 4-Stage Lattice Filter, Voltera Filter, 8-Stage Lattice Filter, kbasic_path, kbasic_tree, are shown in Table 4.

In the table, Column "L/W(10^{-5})" presents the given timing constraint and the accumulated failure rate constraint, respectively. The minimum total costs obtained from different algorithms, *the greedy algorithm* and *HPSO algorithm*, are shown in each entry. Column "Cost" under "Path_assign/Tree_Assign" shows the total system costs obtained by *Path_Assign* algorithm or *Tree_Assign* algorithm. Column "%(Greedy)" under "Path_Assign/Tree_Assign" shows the percentage of reduction on system cost obtained by our algorithms, compared with the Greedy algorithm. Column "%(HPSO)" under "Path_Assign/Tree_Assign" shows the percentage of reduction on system cost obtained by our algorithms, compared with the HPSO algorithm. Column "Configuration" shows a feasible configuration corresponding to the assignment obtained by "Path_Assign/Tree_Assign", in which "$x1, x2, x3, x4$" means that $x1$ PEs with type P_1, $x2$ PEs with type P_2, $x3$ PEs with type P_3, and $x4$ PEs with type p_4 are used. The row "Average Reduction" in Table 4 shows that, compared with the greedy algorithm and the HPSO algorithm, the proposed optimal algorithms averagely reduce the total system cost by 42.4 % and 37.1 %, respectively.

Figure 4 shows the running time for each technique running different paths and trees. The running time of each task graph for each technique is the mean time over the five given timing constraint and the accumulated failure

rate constraint in Table 4. We can see that the proposed Path_Assign/Tree_Assign algorithm takes less than 1 second for each cases. And the HPSO algorithm takes much longer time compared to Path_Assign/Tree_Assign.

The experimental results for those benchmarks that are DAGs are shown in Table 5. Those benchmarks include 2-deq, 3-deq, elf, 2-RSL-Languerre Lattice Filter, 3-elf, 20-4Stage Lattice Filter, kbasic_DAG.

In Table 5, Column "Cost" under "DAG_Heu" shows the execution cost obtained by DAG_Heu algorithm. Column "%(Greedy)" under "DAG_Heu" shows the total system cost reduction rate generated by DAG_Heu algorithm, compared with the Greedy algorithm. Column "%(HPSO)" under "DAG_Heu" shows the total system cost reduction rate generated by DAG_Heu algorithm, compared with the HPSO algorithm. The configurations corresponding to the assignment obtained by "DAG_Heu" are shown in Column "Configuration" under "DAG_Heu". Column "Cost" under "ILP" shows execution cost obtained by ILP method. Column "%(DAG_Heu)" shows the total system cost reduction rate generated by ILP algorithm, compared with the results obtained by DAG_Heu algorithm. Each "×" in Table 5 indicates that the running time of the corresponding experiment is longer than ten hours.

Experimental results in Table 5 shows that *DAG_Heu* algorithm outperform both the Greedy algorithm and the HPSO algorithm. It averagely reduces the total system cost by 38.8 % and 31.8 % compared to these two algorithms. The mapping results obtained with *DAG_Heu* algorithm is very close to the optimal results generated with ILP model. The total cost obtained by the ILP model has only a reduction of 4.7 % on average compared with results obtained with DAG_Heu algorithm. The mapping results obtained with HPSO algorithm has a relatively large gap from the optimal results. It could be as high as 50.7 % (for the benchmark kbasic_DAG, which is a task graph with 583 nodes). And the results become worse as the number of tasks increases.

Figure 5 shows the running time for each technique running general DAGs. The running time of each task graph for each technique is the mean time over the five given timing constraint and the accumulated failure rate constraint in Table 5. We can see that DAG_Heu algorithm can always generate a solution efficiently for all the benchmarks. And the running time for DAG_Heu algorithm is much shorter than that for ILP model and HPSO algorithm. For example, for the benchmark of 2-rls-lat which has 38 tasks, the running time for DAG_Heu is 0.04s. However, ILP formulation needs 391s and HPSO algorithm generates result with 48.5s. The computation time of the ILP method grows exponentially with increasing size of benchmarks. It takes a long time to get results even for a medium-sized DFG with more than 100 nodes.

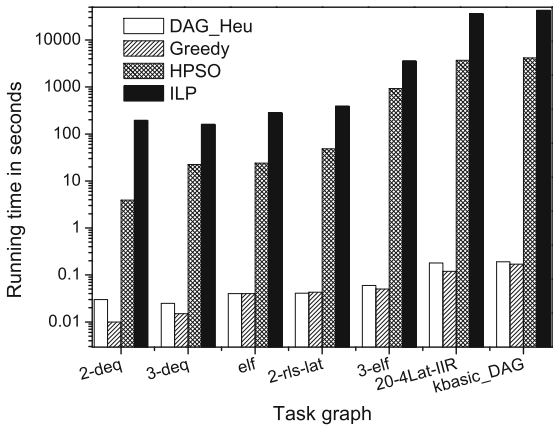


Figure 5 The running time comparison of different techniques for the general DAGs.

For example, the ILP model takes 1651 seconds to compute the assignment result for 3-elf with timing constraint 180 and accumulated failure rate constraint 340, while DAG_Heu algorithm takes less than 0.2 second to generate a near optimal solution. Also, for the 20-4Stage Lattice Filter and kbasic_DAG, the ILP method cannot obtain an optimal solution within 10 hours. The HPSO algorithm iteratively searches for a better mapping in a much slower pace than the DAG_Heu. The population size and parameter used by the HPSO algorithm is large in order to find better mapping. To achieve a final mapping results, it takes from a few seconds when running 2-deq to more than one hour for both 20-4Stage Lattice Filter and kbasic_DAG. Through the analysis of these experimental results, we can conclude that DAG_Heu algorithm outperform the state-of-the-art in two aspects. Firstly, the mapping results obtained with DAG_Heu algorithm is quite close to optimal results generated using ILP formulation owing to a novel well-designed method to dynamically adjust the reduction range of the completion time or the accumulated failure rate during the optimization process. Secondly, the running time of DAG_Heu algorithm is much lower than that for HPSO and ILP formulation. While DFGs become too large for the ILP model to solve, the proposed algorithms can still efficiently give results.

10 Conclusion

In this paper, we proposed a two-phase approach for the problem of Task Assignment and Scheduling with Guaranteed Reliability and Timing Constraint (TASHM). In the first phase, we solved the heterogeneous assignment problem, i.e., how to assign proper PE types to applications such that the total cost can be minimized while the timing constraint and the accumulated failure rate are satisfied.

Table 5 Experimental results for benchmarks of DAGs with various constraints.

				DAG_Heu				ILP	
Benchmark	L/W(10^{-5})	Greedy cost	HPSO cost	Cost	%(Greedy)	%(HPSO)	Configuration	Cost	%(DAG_Heu)
2-deq	20/50	251	231	216	13.9 %	6.5 %	2, 0, 3, 0	216	0.0 %
	25/60	235	228	185	21.3 %	18.9 %	2, 1, 4, 0	174	5.9 %
	30/70	223	192	169	24.2 %	12.0 %	2, 0, 3, 0	139	17.8 %
	35/80	199	176	117	41.2 %	33.5 %	1, 1, 5, 0	115	1.7 %
	40/90	192	157	105	45.3 %	33.1 %	1, 1, 4, 1	100	4.8 %
3-deq	25/70	379	366	348	8.2 %	4.9 %	1, 1, 4, 1	348	0.0 %
	35/90	348	334	276	20.7 %	17.4 %	1, 1, 4, 1	251	9.1 %
	45/110	317	288	204	35.6 %	29.2 %	1, 1, 4, 1	188	7.8 %
	55/130	289	249	164	43.3 %	34.1 %	1, 1, 3, 2	150	8.5 %
	65/150	263	203	135	48.7 %	33.5 %	0, 2, 4, 1	123	8.9 %
elf	50/80	381	376	315	17.3 %	16.2 %	1, 0, 3, 0	314	0.3 %
	60/100	348	289	218	37.4 %	24.6 %	2, 0, 3, 0	218	0.0 %
	70/120	322	261	164	49.1 %	37.2 %	1, 1, 4, 0	159	3.0 %
	80/140	296	210	139	53.0 %	33.8 %	1, 2, 4, 0	132	5.0 %
	90/160	263	181	119	54.8 %	34.3 %	1, 1, 4, 1	112	5.9 %
2-rls-lat	30/80	439	419	404	8.0 %	3.6 %	3, 0, 2, 0	403	0.2 %
	40/100	403	369	302	25.1 %	18.2 %	3, 0, 5, 0	301	0.3 %
	50/120	372	329	224	39.8 %	31.9 %	1, 1, 6, 0	224	0.0 %
	60/140	339	321	169	50.1 %	47.4 %	1, 1, 6, 0	169	0.0 %
	70/160	316	256	156	50.6 %	39.1 %	2, 2, 3, 1	145	7.1 %
3-elf	160/320	1032	866	687	33.4 %	20.7 %	2, 0, 4, 0	613	10.8 %
	180/340	1001	865	587	41.4 %	32.1 %	2, 2, 4, 0	545	7.2 %
	200/360	956	815	523	45.3 %	35.8 %	2, 1, 4, 0	483	7.6 %
	220/380	932	714	448	51.9 %	37.3 %	1, 2, 4, 1	434	3.1 %
	240/400	899	663	405	54.9 %	38.9 %	2, 1, 4, 1	391	3.5 %
20-4Lat-IIR	300/1200	5883	5802	5118	13.0 %	11.8 %	8, 3, 5, 0	×	×
	400/1500	5452	5124	3411	37.4 %	33.4 %	8, 3, 2, 2	×	×
	500/1800	5220	4783	2399	54.0 %	49.8 %	3, 7, 5, 2	×	×
	600/2100	4397	4329	1800	59.1 %	58.4 %	3, 7, 0, 9	×	×
	700/2400	3935	3602	1705	56.7 %	52.7 %	4, 2, 4, 5	×	×
kbasic_DAG	400/1500	7344	7137	5108	30.4 %	28.4 %	9, 4, 5, 0	×	×
	500/1800	6594	5940	3884	41.1 %	34.6 %	7, 0, 5, 3	×	×
	600/2100	5837	5625	3124	46.5 %	44.5 %	0, 3, 9, 4	×	×
	700/2400	5087	4935	2526	50.3 %	48.8 %	7, 4, 7, 2	×	×
	800/2700	4345	4108	2026	53.4 %	50.7 %	3, 3, 7, 5	×	×
Average reduction					38.8 %	31.1 %			4.7 %

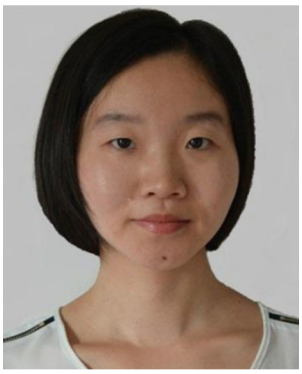
Path_Assign algorithm and *Tree_Assign* algorithm were proposed to give optimal solutions when the input graph is a simple path and a tree. For general TASHM problem, we devised an ILP model to obtain optimal solution. However, the computation time of the ILP model grows exponentially as the size of the input graph increases. Therefore, we proposed a heuristic algorithm, *DAG_Heu*, for general problems, which can produce near-optimal solution in polynomial time. In the second phase, we proposed a minimum resource scheduling algorithm to generate a schedule and a feasible configuration that uses as few resources as possible. The experiments show that the proposed algorithms can effectively reduce the total cost compared with the existing techniques.

Acknowledgements This work is partially supported by Chongqing High-Tech Research Program csct2012ggC40005, National 863 Program 2013AA013202, NSFC 61173014, NSFC 61472052

References

1. Nvidia provides second quarter fiscal 2009 business update. http://www.nvidia.com/object/io_1215037160521.html.
2. Ahmad, I., & Dhodhi, M. (2006). Task assignment using a problem-space genetic algorithm. *Concurrency: Practice and Experience*, *7*(5), 411–428.
3. Andersson, B., Raravi, G., Bletsas, K. (2010). Assigning real-time tasks on heterogeneous multiprocessors with two unrelated types of processors. In *2010 IEEE 31st Real-Time Systems Symposium (RTSS)* (pp. 239–248).
4. Borkar, S. (2005). Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *IEEE Microbiology*, *25*(6), 10–16.
5. Braun, T.D., Siegel, H.J., Beck, N., Bölöni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., et al. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, *61*(6), 810–837.
6. Bultan, T., & Aykanat, C. (1992). A new mapping heuristic based on mean field annealing. *Journal of Parallel and Distributed Computing*, *4*, 292–305.
7. Chang, P., Wu, I., Shann, J., Chung, C. (2008). Etahm : An energy-aware task allocation algorithm for heterogeneous multiprocessor. In *45th ACM/IEEE Design Automation Conference, 2008* (pp. 776–779).
8. Chen, G., & Yur, J. (1990). A branch-and-bound-with-underestimates algorithm for the task assignment problem with precedence constraint. In *Proceedings of 10th International Conference on Distributed Computing Systems, 1990* (pp. 494–501).
9. Chiu, C., Hsu, C., Yeh, Y. (2006). A genetic algorithm for reliability-oriented task assignment with k duplications in distributed systems. *IEEE Transactions on Reliability*, *55*(1), 105–117.
10. Chiu, C., Yeh, Y., Chou, J. (2002). A fast algorithm for reliability-oriented task assignment in a distributed system. *Computer Communications*, *25*(17), 1622–1630.
11. Dick, R., Rhodes, D., Wolf, W. (1998). Tgff: Task graphs for free. In *Proceedings of the 6th International Workshop on Hardware/Software Codesign* (pp. 97–101).
12. Funk, S., & Baruah, S. (2005). Task assignment on uniform heterogeneous multiprocessors. In *Proceedings of the 17th Euromicro Conference on Real-Time Systems, 2005, (ECRTS 2005)* (pp. 219–226). IEEE.
13. Goh, L.K., Veeravalli, B., Viswanathan, S. (2009). Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, *20*(1), 1–12.
14. Hsu, H., Chen, J., Kuo, T. (2006). Multiprocessor synthesis for periodic hard real-time tasks under a given energy constraint. In *Proceedings of the Conference on Design, Automation and Test in Europe* (pp. 1061–1066).
15. Hwang, G., & Tseng, S. (1993). A heuristic task assignment algorithm to maximize reliability of a distributed system. *IEEE Transactions on Reliability*, *42*(1), 408–415.
16. Ibarra, O.H., & Kim, C.E. (1977). Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM (JACM)*, *24*(2), 280–289.
17. Kafil, M., & Ahmad, I. (1998). Optimal task assignment in heterogeneous distributed computing systems. *IEEE Concurrency*, *6*(3), 42–50.
18. Kumar, R., Tullsen, D.M., Jouppi, N.P., Ranganathan, P. (2005). Heterogeneous chip multiprocessors. *Computer*, *38*(11), 32–38.
19. Li, D., & Wu, J. (2012). Energy-aware scheduling for frame-based tasks on heterogeneous multiprocessor platforms. In *41st International Conference on Parallel Processing (ICPP), 2012* (pp. 430–439). IEEE.
20. Li, W., Lim, A., Agrawal, P., Sahni, S. (1993). On the circuit implementation problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *12*(8), 1147–1156.
21. Luk, C.K., Hong, S., Kim, H. (2009). Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In *42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009, MICRO-42* (pp. 45–55). IEEE.
22. Micheli, G.D. (1994). *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education.
23. Pham, D., Asano, S., Bolliger, M., Day, M.N., Hofstee, H.P., Johns, C., Kahle, J., Kameyama, A., Keaty, J., Masubuchi, Y., et al. (2005). The design and implementation of a first-generation cell processor. In *2005 IEEE International Solid-State Circuits Conference, Digest of Technical Papers, 2005, ISSCC* (pp. 184–592). IEEE.
24. Plank, J.S., & Elwasif, W.R. (1998). Experimental assessment of workstation failures and their impact on checkpointing systems. In *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing, 1998, Digest of Papers* (pp. 48–57) . IEEE.
25. Salcedo-Sanz, S., Xu, Y., Yao, X. (2006). Hybrid meta-heuristics algorithms for task assignment in heterogeneous computing systems. *Computers & operations research*, *33*(3), 820–835.
26. Shao, Z., Zhuge, Q., Xue, C., Sha, E.M. (2005). Efficient assignment and scheduling for heterogeneous dsp systems. *IEEE Transactions on Parallel and Distributed Systems*, *16*(6), 516–525.
27. Shatz, S., Wang, J., Goto, M. (1992). Task allocation for maximizing reliability of distributed computer systems. *IEEE Transactions on Computers*, *41*(9), 1156–1168.
28. Shatz, S.M., & Wang, J.P. (1989). Models and algorithms for reliability-oriented task-allocation in redundant distributed-computer systems. *IEEE Transactions on Reliability*, *38*(1), 16–27.
29. Shen, C.C., & Tsai, W.H. (1985). A graph matching approach to optimal task assignment in distributed computing systems using a minimax criterion. *IEEE Transactions on Computers*, *100*(3), 197–203.
30. Srinivasan, J., Adve, S.V., Bose, P., Rivers, J.A. (2004). The case for lifetime reliability-aware microprocessors. In *ACM SIGARCH Computer Architecture News*, (Vol. 32, p. 276). IEEE Computer Society.
31. Sun, F., Jha, N.K., Ravi, S., Raghunathan, A. (2005). Synthesis of application-specific heterogeneous multiprocessor architectures using extensible processors. In *18th International Conference on VLSI Design, 2005* (pp. 551–556). IEEE.
32. Vigrass, W.J. (2010). *Calculation of semiconductor failure rates*. Harris Semiconductor.
33. Wang, L., Liu, J., Hu, J., Zhuge, Q., Sha, E. (2012). Optimal assignment for tree-structure task graph on heterogeneous multicore systems considering time constraint. In *2012 IEEE 6th International Symposium on Embedded Multicore Socs (MCSoC)* (pp. 121–127).

34. Yin, P., Yu, S., Wang, P., Wang, Y. (2006). A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems. *Computer Standards & Interfaces*, *28*(4), 441–450.
35. Zivojnovic, V., Velarde, J., Schlager, C., Meyr, H. (1994). Dspstone: A dsp-oriented benchmarking methodology. In *Proceedings of the International Conference on Signal Processing and Technology*.

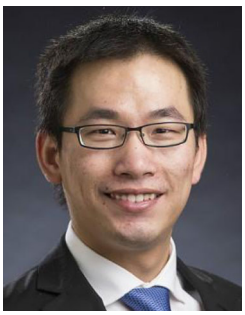


Juan Yi received the B.E. degree from the School of Software Engineering at Chongqing University, Chongqing, China, in 2010 and is currently pursuing the Ph.D degree from the Department of Computer Science at the same university.

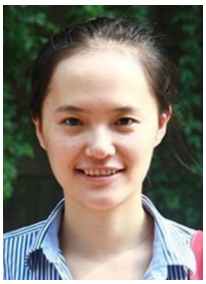
Her current research interests include temperature modeling and optimization, and reliability analysis of multi-core system.

Qingfeng Zhuge received her Ph.D. from the Department of Computer Science at the University of Texas at Dallas in 2003. She obtained her BS and MS degrees in Electronics Engineering from Fudan University, Shanghai, China. She is currently a professor at Chongqing University, China.

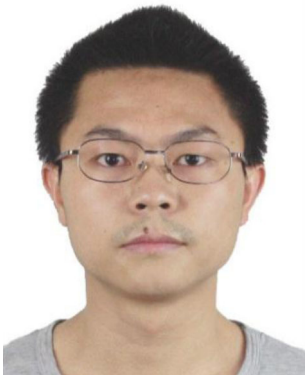
She received Best Ph.D. Dissertation Award in 2003. She has published more than 90 research articles in premier journals and conferences. Her research interests include parallel architectures, embedded systems, supply-chain management, real-time systems, optimization algorithms, compilers, and scheduling.



Jingtong Hu received his B.E. degree from School of Computer Science and Technology, Shandong University, China in 2007 and M.S. and Ph.D. degree in Computer Science from the University of Texas at Dallas, TX, USA in June 2010 and Aug. 2013, respectively. He is currently an Assistant Professor in the School of Electrical and Computer Engineering at Oklahoma State University, OK, USA. His research interests include high performance and low power embedded systems, wireless sensor network, and non-volatile memory.



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



Mingwen Qin is currently working towards the Ph.D degree in the Department of Electrical Engineering at Hong Kong Polytechnic University, Hong Kong.

He received the B.E. and M.Eng. degree from the School of Software Engineering at Chongqing University, China, in 2008 and 2011, respectively.

His current research interests lie in the areas of planning, operation and optimization in power systems and smart grid.

Edwin H.-M. Sha received Ph.D. degree from the Department of Computer Science, Princeton University, USA in 1992.

From August 1992 to August 2000, he was with the Department of Computer Science and Engineering at University of Notre Dame, USA. Since 2000, he has been a tenured full professor in the Department of Computer Science at the University of Texas at Dallas. Since 2012, he served as the Dean of College of Computer Science at Chongqing University, China.

He has published more than 300 research papers in refereed conferences and journals. He received Teaching Award, Microsoft Trustworthy Computing Curriculum Award, NSF CAREER Award, and NSFC Overseas Distinguished Young Scholar Award, Chang Jiang Honorary Chair Professorship and China Thousand-Talent Program.