Cost Minimization while Satisfying Hard/Soft Timing Constraints for Heterogeneous Embedded Systems

MEIKANG QIU
University of New Orleans
and
EDWIN H. -M. SHA
University of Texas at Dallas

In high-level synthesis for real-time embedded systems using heterogeneous functional units (FUs), it is critical to select the best FU type for each task. However, some tasks may not have fixed execution times. This article models each varied execution time as a probabilistic random variable and solves *heterogeneous assignment with probability* (HAP) problem. The solution of the HAP problem assigns a proper FU type to each task such that the total cost is minimized while the timing constraint is satisfied with a guaranteed confidence probability. The solutions to the HAP problem are useful for both hard real-time and soft real-time systems. Optimal algorithms are proposed to find the optimal solutions for the HAP problem when the input is a tree or a simple path. Two other algorithms, one is optimal and the other is near-optimal heuristic, are proposed to solve the general problem. The experiments show that our algorithms can effectively reduce the total cost while satisfying timing constraints with guaranteed confidence probabilities. For example, our algorithms achieve an average reduction of 33.0% on total cost with 0.90 confidence probability satisfying timing constraints compared with the previous work using worst-case scenario.

Categories and Subject Descriptors: C.3 [**Computer System Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Embedded Systems, heterogeneous, high-level synthesis, real-time

This work was supported in part by NSF CCR-0309461, NSF IIS-0513669, HK CERG B-Q60B, and NSFC 60728206.

Authors' addresses: M. Qiu, Department of Electrical and Computer Engineering, University of New Orleans, New Orleans, LA 70148; email: mqiu@uno.edu; E. H. -M. Sha, Dept. of Computer Science, University of Texas at Dallas, Richardson, TX 75083; email: edsha@utdallas.edu.

DOI 10.1145/1497561.1497568 http://doi.acm.org/10.1145/1497561.1497568

ACM Reference Format:
Qiu, M. and Sha, E. H.-M. 2009. Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems. ACM Trans. Des. Autom. Elect. Syst., 14, 2, Article 25 (March 2009), 30 pages, DOI = 10.1145/1497561.1497568 http://doi.acm.org/10.1145/1497561.1497568

1. INTRODUCTION

It is a critical issue to do high-level synthesis for special-purpose architectures of real-time embedded systems to satisfy the time and cost requirements [Wang and Parhi 1995; Foster 1994; Lester 1993; Wolfe 1996; Chao and Sha 1995]. The cost of embedded systems may relate to power, reliability, and so on, and consists of both hardware and software portions [Dogan and Özgüner 2002; Srinivasan and Jha 1999; Shatz et al. 1992; He et al. 2003]. The systems become more and more complicated in two aspects.

First, in many systems, such as heterogeneous parallel DSP systems [Hou and Shin 1997; Banino et al. 2004], the same types of operations can be processed by heterogeneous FUs with different costs. Many MPSoCs are heterogeneous, with multiple types of CPUs, irregular memory hierarchies, and irregular communication. Heterogeneity allows architects to support necessary operations while eliminating the costs of unnecessary features [Wolf 2006], which is extremely critical for cost-sensitive real-time embedded systems. Hence, an important problem arises: how to assign a proper FU type to each operation of a DSP application such that the requirements can be met and the total cost can be minimized while satisfying timing constraints with a guaranteed confidence probability [Shao et al. 2005].

Second, some tasks (operations or instructions) may not have fixed execution times. Such tasks usually contain conditional instructions and/or operations that could have different execution times for different inputs [Tongsima et al. 2000; Zhou et al. 2001; Hua et al. 2003a, Hua et al. 2003b; Hua and Qu 2003]. Although many static assignment techniques can thoroughly check for the best assignment for dependent tasks, existing methods are not able to deal with such uncertainty. Therefore, either worst-case or average-case computation times for these tasks are usually assumed. Such assumptions, however, may not be applicable for real-time systems and may result in an ineffective task assignment. Using a probabilistic approach, we can obtain solutions that can not only be used for hard real-time systems, but also provide more choices at smaller total costs while satisfying timing constraints with guaranteed confidence probabilities.

This article presents high-level synthesis algorithms which operate in probabilistic environments to solve the *heterogeneous assignment with probability* (HAP) problem. In the HAP problem, we model the execution time of a task as a random variable [Zadeh 1996]. For heterogeneous systems, each FU type has different cost, representing hardware cost, size, reliability, and so on. The faster one has a higher cost while the slower one has a lower cost. This article will introduce the concept of *Probabilistic Data Flow Graph* (PDFG) (definition is in the System Model section). Each node of the PDFG represents a task or an operation. The direction between nodes of the PDFG represents

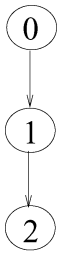
task-implementation order. The nodes can be executed either in sequential or parallel order depending on the structure of the graph.

In this article, we show how to assign a proper FU type to each node (representing a task) of a PDFG such that the total cost is minimized while satisfying the timing constraint with a guaranteed confidence probability. In other words, we can guarantee that the total execution time of the PDFG is less than or equal to the timing constraint with a probability greater than or equal to P.

We give an example to illustrate the HAP problem. Heterogeneous assignment of special purpose architectures for real-time DSP applications has become a common and critical step in the design flow in order to satisfy the requirements of stringent timing constraints. DSP applications need special high-speed functional units (FUs) such as ALUs and multipliers to perform addition and multiplication operations [Shao et al. 2005]. For a digital camera chip, there are five major parts in the chip [Vahid and Givargis 2002]: (1) the *JPEG codec* compresses and decompresses an image using the JPEG compression standard, enabling compact storage of images in the limited memory of the camera; (2) the *memory controller* controls access to a memory chip also found in the camera; (3) the *display control* circuits control the display of images on the camera's liquid-crystal display device; (4) the *multiplier/accumulator* circuit performs a particular frequently executed multiply/accumulate computation faster than the microcontroller could; and (5) at the heart of the system is the *microcontroller*, which is a programmable processor that controls the activities of all the other circuits.

We focus on the *multiplier/accumulator* circuit, which performs a single function repeatedly and has soft real-time constraints. Due to conditional instructions and environmental change, such as temperature change on the chip, the same tasks may have varied execution times even with the same FU. In this article, we will show that with heterogeneous FUs available, significant costs could be saved by satisfying timing constraints with some probabilities, as opposed to using worst-case times. Assume that the FU type library provides two types of FUs (such as ALU, multiplier, etc.), R_1 and R_2, for us to select from. Assume there are enough FUs available for each node. Each node needs only on type of FU. FUs can be reused by different nodes at different steps, and there is no FU-sharing conflict (i.e., one task cannot use a FU that is being used by another task). The nodes of the PDFG represent task implementation order, which can be executed in either sequence or parallel order.

For demonstration purposes, assume that there is an input PDFG on the multiplier/accumulator circuit of the digital camera chip. The PDFG is a simple path with three nodes which is shown in Figure 1(a). Each node represents a task, such as an instruction or an operation. The execution times (T), probabilities (P), and costs (C) of each node for different FU types are shown in Figure 1(b). The probabilities at which these events occur can be obtained by building a historic table and using statistical profiling [Tia et al. 1995; Zhou et al. 2001; Kalavade and Moghe 1998]. We think that statistical profiling can be used to determine probabilities in many applications in the embedded systems area, especially for applications with loops. If different inputs have completed



(a)

Nodes	R1			R2		
	T	P	C	T	P	C
0	1	0.9	10	2	0.7	4
	3	0.1	10	4	0.3	4
1	1	0.8	8	3	0.6	3
	4	0.2	8	5	0.4	3
2	1	0.9	9	2	0.8	5
	4	0.1	9	6	0.2	5

(b)

Nodes	R1			R2		
	T	P	C	T	P	C
0	1	0.9	10	2	0.7	4
	3	1.0	10	4	1.0	4
1	1	0.8	8	3	0.6	3
	4	1.0	8	5	1.0	3
2	1	0.9	9	2	0.8	5
	4	1.0	9	6	1.0	5

(c)

Fig. 1. (a) A given simple path; (b) the times, probabilities, and costs of its node for different FU types; (c) the time cumulative distribution functions (CDFs) and costs of its node for different FU types.

different characteristics, we will adjust this scenario with dynamic scheduling. This will be our future work.

Each node can select one of the two different FU types. The execution time (T) of each FU type is modeled as a random variable. For example, node 0 can choose one of the two types: R_1 or R_2. When choosing R_1, node 0 will be finished in 1 time unit with probability 0.9 and will be finished in 3 time units with probability 0.1. In other words, node 0 can guarantee being finished in 3 time units with 100% probability. Hence, we care about the time *cumulative distribution function* (CDF) $F(t)$, which gives accumulated probability for $T \leq t$. For example, the CDF of node 0 at time unit 1 is 0.9; and the CDF of node 0 at time unit 3 is 1.0. Figure 1(c) shows the time CDFs and costs of each node for different FU types.

A solution to the HAP problem with timing constraint 10 can be found as follows: We assign FU types 2, 2, and 1 for nodes 0, 1, and 2, respectively. Let T_0, T_1, and T_2 be the random variables representing the execution times of nodes 0, 1, and 2. From Figure 1(c), we get: $Pr(T_0 \leq 4) = 1.0$, $Pr(T_1 \leq 5) = 1.0$, and $Pr(T_2 \leq 1) = 0.9$. Hence, we obtain minimum total cost 16 with 0.9 probability satisfying the timing constraint 10. The total cost is computed by adding the costs of all nodes together and the probability corresponding to the total cost is computed by multiplying the probabilities of all nodes based on the basic properties of probability and cost of a PDFG.

In Figure 1(b), if we use the worst-case execution time as a fixed execution time for each node, then the assignment problem becomes the *hard heterogeneous assignment* (hard HA) problem, which is related to the hard real-time problem. The hard HA problem is the worst-case scenario of the *heterogeneous assignment with probability* (HAP) problem. For example, in the hard HA problem, when choosing type R_1, node 0 has only one execution time 3. When choosing type R_2, node 0 has one execution time 4. With certain timing constraints, there might not be a solution for the hard HA problem. However, for soft real-time applications, it is desirable to find an assignment that guarantees the total execution time to be less than or equal to the timing constraint with certain confidence probability.

For example, in Figure 1, under timing constraint 10, we cannot find a solution to the hard HA problem. But we can obtain minimum system cost 16 with probability 0.9 satisfying the timing constraint 10. Also, the cost obtained from the worst-case scenario is always larger than or equal to the cost from the probabilistic scenario. For example, under timing constraint 11, the minimum cost is 27 for the hard HA problem. While in the HAP problem, with confidence probability 0.9 satisfying the timing constraint, we get the minimum cost of 16, which gives 40.7% improvement.

It is known that the hard HA problem is NP-complete [Shao et al. 2005]. Since the HAP problem is NP harder than the hard HA problem, the HAP problem is also NP-complete. In this article, two polynomial time algorithms are proposed to optimally solve the HAP problem when the given PDFG is a tree or a simple path, and two other algorithms are proposed to solve the general problem.

There has been a lot of research on allocating applications in heterogeneous distributed systems [Beaumont et al. 2002; Hou and Shin 1997; Banino et al. 2004; Ramamritham et al. 1990; Bettati and Liu 1992; Beaumont et al. 2004; Beaumont et al. 2003a, 2003b]. Incorporating reliability cost into heterogeneous distributed systems, the reliability-driven assignment problem has been studied by researchers [Dogan and Özgüner 2002; Srinivasan and Jha 1999; Shatz et al. 1992]. In these works, allocations are performed based on a fixed architecture. However, when performing assignments in architecture synthesis, no fixed architectures are available. Most previous work on the synthesis of special-purpose architectures for real-time DSP applications focuses on the architectures that use only homogeneous FUs; that is, the same type of operations will be processed by the same types of FUs [Paulin and Knight 1989; Ito et al. 1998; McFarland et al. 1990; De Man et al. 1990; Parhi and Messerschmitt 1991; Hwang et al. 1991; Gebotys and Elmasry 1993; Chao and Sha 1997; Chang et al. 1996].

Our work is related to the work of the following authors: [Tongsima et al. 2000; Ito and Parhi 1995; Shao et al. 2005; Li et al. 1993]. Although the ILP model from Ito and Parhi [1995] can obtain an optimal solution for the heterogeneous assignment problem, it is a NP-hard problem to solve the ILP model. Therefore, the ILP model may take a very long time to get results, even when a given *data flow graph* (DFG) is not very big. In this article, the algorithm solving the hard HA problem in Ito and Parhi [1995] is called the *hard HA*

ILP algorithm in general. Shao et al. [2005] proposed two optimal algorithms for the hard HA problem when the given input is a tree or simple path, and three heuristic algorithms for the general, hard HA problem. But they did not consider a varied execution time situation. Also, their solutions are not optimal for the general problem.

Probabilistic retiming (PR) was proposed in Tongsima et al. [2000] and Passos et al. [1994]. For a system without resource constraints, PR can be applied to optimize the input graph, that is, reduce the length of the longest path of the graph such that the probability of the longest path computation time being less than or equal to the given timing constraint, L is greater than or equal to a given confidence probability P. Since the execution times of the nodes can be either fixed or varied, a probability model is employed to represent the execution time of the tasks. But PR does not model the hard HA problem, which focuses on how to obtain the best assignment from different FU types.

Our contributions are as follows:

—When the given PDFG is a tree or a simple path, the results of our algorithms, *Path_Assign* and *Tree_Assign*, cover the results of the optimal *hard HA ILP algorithm* [Ito and Parhi 1995] using the worst-case scenario of our algorithms, and our algorithms produce results for soft real-time systems.

—For the general problem, that is, when the given input is a *directed acyclic graph* (DAG), our optimal algorithm, *DAG_Opt*, gives the optimal solution and covers the results of the *hard HA ILP algorithm* [Ito and Parhi 1995] using the worst-case scenario of our algorithms. Our heuristic algorithm, *DAG_Heu*, gives near optimal solutions efficiently.

—Our algorithms are able to give solutions and provide more choices of smaller total costs with guaranteed confidence probabilities that satisfy timing constraints. While the *hard HA ILP algorithm* [Ito and Parhi 1995] may not find a solution with certain timing constraints.

—Our algorithms are practical and quick. In practice, when the number of multiparent nodes and multichild nodes in the given input graph is small and the timing constraint is polynomial to the size of PDFG, our algorithms become polynomial. The running times of these algorithms are very small, and our experiments always finished in a very short time.

We conducted experiments on a set of benchmarks and compared our algorithms with the *hard HA ILP algorithm* [Ito and Parhi 1995]. Experiments show that when the input PDFG is a tree or a simple path, the results of our algorithms have an average (from 6 different benchmarks under different set-up parameters, and the same for the other cases) a 32.5% improvement with 0.9 confidence probability satisfying timing constraints, compared with the results for the hard HA problem. With 0.8 confidence probability in satisfying timing constraints, the average improvement is 38.2%; and with 0.7 confidence probability satisfying timing constraints, the improvement is 40.6%. When the input PDFG is a DAG, both our optimal and near-optimal algorithms have significant improvement on total cost reduction compared with the cost for hard real-time. On average, our algorithms give a cost reduction of 33.5% with 0.9 confidence

probability satisfying timing constraints, and a cost reduction of 45.3% and 48.9% with, respectively, 0.8 and 0.7 confidence probability satisfying timing constraints.

The remainder of this article is organized as follows: In the next section, we give the basic definitions and models used in the rest of the article. Examples of the HAP problem when the input is a simple path or a tree are given in Section 3, The algorithms for the HAP problem are presented in Section 4. Experimental results and concluding remarks are provided in Section 5 and Section 6, respectively.

2. SYSTEM MODEL

We use *probabilistic data-flow graph (PDFG)* to model an application of embedded systems. A ***PDFG G*** $= \langle V, E, T, R \rangle$ is a *directed acyclic graph* (DAG), where $V = \langle v_1, v_2, \cdots, v_N \rangle$ is the set of nodes, and $E \subseteq V \times V$ is the edge set that defines the precedence relations among nodes in V. In practice, many architectures consist of different types of FUs. Assume there are maximum M different FU types in a FU set $\mathrm{R} = \{R_1, R_2, \cdots, R_M\}$. For each FU type, there are maximum K execution time variations T, although each node may have a different number of FU types and execution time variations.

An assignment for a PDFG G is to assign a FU type to each node. Define an *assignment A* to be a function from domain V to range R, where V is the node set and R is FU type set. For a node $v \in V$, $A(v)$ gives the selected type of node v. For example, in Figure 1(a), assigning FU types 2, 2, and 1 for nodes 0, 1, and 2, respectively, we obtain minimum total cost 16 with 0.9 probability satisfying the timing constraint 10. That is, $A(0) = 2$, $A(1) = 2$, and $A(2) = 1$.

In a PDFG G, each varied execution time T is modeled as a probabilistic random variable. $\mathbf{T_{R_j}(v)}$ $(1 \leq j \leq M)$ represents the execution times of each node $v \in V$ for FU type j, and $\mathbf{P_{R_j}(v)}$ $(1 \leq j \leq M)$ represents the corresponding probability function. And $\mathbf{C_{R_j}(v)}$ $(1 \leq j \leq M)$ is used to represent the cost of each node $v \in V$ for FU type j, which is a fixed value. For instance, in Figure 1(a), $T_1(0) = 1, 3$; $T_2(0) = 2, 4$. Correspondingly, $P_1(0) = 0.9, 0.1$; $P_2(0) = 0.7, 0.3$. And $C_1(0) = 10$, $C_2(0) = 4$.

Given an assignment A of a PDFG G, we define the *system total cost under assignment A*, denoted $\mathbf{C_A(G)}$, to be the summation of costs of all nodes, that is, $C_A(G) = \sum_{v \in V} C_{A(v)}(v)$. In this article we call $C_A(G)$ the *total cost*, in brief. For example, in Figure 1(a), under assignment 2, 2, and 1 for nodes 0, 1, and 2, respectively, the costs of nodes 0, 1, and 2 are $C_2(0) = 4$, $C_2(1) = 3$, and $C_1(2) = 9$. Hence, the total cost of the graph G is $C_A(G) = C_2(0) + C_2(1) + C_1(2)$, that is, $C_A(G) = 16$.

For the input PDFG G, given an assignment A, assume that $\mathbf{T_A(G)}$ stands for the *execution time of graph G under assignment A*. $T_A(G)$ can be gotten from the longest path p in G. The new variable $T_A(G) = \max_{\forall p} T_{A(v)}(p)$, where $T_{A(v)}(p) = \sum_{v \in p} T_{A(v)}(v)$, is also a random variable. In Figure 1(a), there is only one path. Under assignment 2, 2, and 1 for nodes 0, 1, and 2, $T_A(G) = T_{A(v)}(p) = T_2(0) + T_2(1) + T_1(2)$. Since $T_2(0)$, $T_2(1)$, and $T_1(2)$ are all random variables, then $T_A(G)$ is also a random variable.

The *minimum total cost C with confidence probability P under timing constraint L* is defined as $C = \min_A C_A(G)$, where probability of $(T_A(G) \leq L) \geq P$. Probability of ($T_A(G) \leq L$) is computed by multiplying the probabilities of all nodes together while satisfying $T_A(G) \leq L$. That is, $P_A(G) = \prod_{v \in V} P_{A(v)}(v)$.

In Figure 1(a), under assignment 2, 2, and 1 for nodes 0, 1, and 2, $P_2(0) = Pr(T_2(0) \leq 4) = 1.0$, $P_2(1) = Pr(T_2(1) \leq 5) = 1.0$, and $P_1(2) = Pr(T_1(2) \leq 1) = 0.9$. Hence, $P_A(G) = \prod_{v \in V} P_{A(v)}(v) = 0.9$. With confidence probability P, we can guarantee that the total execution time of the graph G is less than or equal to the timing constraint L with a probability greater than or equal to P. For each timing constraint L, our algorithm will output a serial of (probability, cost) pairs (P, C).

$T_{R_j}(v)$ is either a discrete random variable or a continuous random variable. We define **F** to be the *cumulative distribution function* of the random variable $T_{R_j}(v)$ (abbreviated as *CDF*), where $F(t) = P(T_{R_j}(v) < t)$. When $T_{R_j}(v)$ is a discrete random variable, the CDF $F(t)$ is the sum of all the probabilities associating with the execution times that are less than or equal to t. Figure 1(c) gives the time CDFs of each node for different FU types. If $T_{R_j}(v)$ is a continuous random variable, then it has a *probability density function (PDF)* . If we assume the PDF is f, then $F(t) = \int_0^t f(s)ds$. Function F is nondecreasing, and $F(-\infty) = 0$, $F(\infty) = 1$.

We define the *heterogeneous assignment with probability (HAP)* problem as follows: Given M different FU types: $R_1, R_2, \cdots, R_M$, a PDFG $G = \langle V, E \rangle$ where $V = \langle v_1, v_2, \cdots, v_N \rangle$, $T_{R_j}(v)$, $P_{R_j}(v)$, $C_{R_j}(v)$ for each node $v \in V$ executed on each FU type j, and a timing constraint L, find an assignment for G that gives the *minimum total cost C with confidence probability P under timing constraint L*. In Figure 1(a), a solution to the HAP problem with timing constraint 11 can be found as follows. Assigning FU types 2, 2, and 2 for nodes 0, 1, and 2, respectively, we obtain minimum total cost 12 with 0.8 probability under the timing constraint 11.

3. EXAMPLES

Here we give two different types of examples to illustrate the *heterogeneous assignment probability* (HAP) problem. One is a simple path, and the other is a tree for the given input PDFG.

3.1 Simple Path

For the example in Figure 1(a), each node has two different FU types to choose from, and is executed on them with probabilistic times. In many applications, a real-time system does not always have hard deadlines. The execution time can be smaller than the hard deadline time with certain probabilities. So the hard deadline time is the worst-case of the varied smaller time cases. If we consider these time variations, we can achieve a better minimum cost with satisfying confidence probabilities.

Different FU types have different costs, which can be any cost such as hardware cost, energy consumption or reliability costs. A node may run slower, but with less energy consumption or reliability cost when executed on one type of

Table I. Minimum Total Costs (with computed confidence probabilities under various timing constraints for a simple path)

T	(P , C)	(P , C)	(P , C)	(P , C)	(P , C)	(P , C)
3	0.65, 27					
4	0.50, 21	0.58, 23	0.65, 27			
5	0.45, 17	0.50, 21	0.58, 23	0.72, 27		
6	0.38, 16	0.45, 17	0.72, 21	0.81, 27		
7	0.34, 12	0.38, 16	0.64, 17	0.72, 21	0.81, 22	
8	0.34, 12	0.63, 16	0.64, 17	0.72, 18	0.81, 22	0.90, 27
9	0.56, 12	0.63, 16	0.64, 17	0.72, 18	0.90, 21	
10	0.56, 12	0.90, 16				
11	0.80, 12	0.90, 16	**1.00, 27**			
12	0.80, 12	0.90, 16	**1.00, 21**			
13	0.80, 12	**1.00, 16**				
14	0.80, 12	**1.00, 16**				
15	**1.00, 12**					

FU rather than on another [Shao et al. 2005]. But for the same FU type, a node may have varied execution times, while there is not too much difference in cost. In this article, we assume the cost of a FU type is fixed for a node executed on it while the execution time is a random variable. When the cost is related to energy consumption, it is clear that the total energy consumption is the summation of the energy cost of each node. Also, when the cost is related to reliability, the total reliability cost is the summation of the reliability cost of all nodes. We compute the reliability cost using the same model as Srinivasan and Jha [1999]. From the conclusions of the papers by Srinivasan and Jha [1999] and Shao et al. [2005], we know that in order to maximize the reliability of a system, we need to find an assignment such that the timing constraint is satisfied and the summation of the reliability costs of all nodes is minimized.

For this simple path with three nodes, we can obtain the minimum total cost with computed confidence probabilities under various timing constraints. The results generated by our algorithms are shown in Table I. The entries with probability equal to 1 (see the entries in boldface) actually give the results to the hard HA problem, which show the worst-case scenario of the HAP problem. For each row of the table, the C in each (P, C) pair gives the minimum total cost with confidence probability P under timing constraint T. For example, when $T = 3$, with pair (0.65, 27), we can achieve minimum total cost 27 with confidence probability 0.65 under timing constraint 3.

Compared to the optimal results of the hard HA problem for a simple path using a worst-case scenario, Table I provides more information, more selections and decisions, no matter whether the system is hard or soft real-time. In Table I, we have the output of our algorithm from timing constraint 3 to 15, while the optimal results of the hard HA problem for a simple path only has 5 entries (in boldface) from timing constraints 11 to 15.

For a soft real-time system, some nodes of PDFG have smaller probabilistic execution times compared with the hard deadline time. We can achieve much smaller cost than the cost of worst-case with guaranteed confidence probability. For example, under timing constraint 11, we can select the pair (0.90, 16) which guarantees achieving minimum cost 16 with 0.90 confidence, satisfying the

Table II. With Timing Constraint 11, the Assignments of Types for Each Node with Different (Probability, Cost) Pairs

	Node id	Type id	T	Prob.	Cost
Assign	0	2	4	1.00	4
Assign	1	2	5	1.00	3
Assign	2	2	2	0.80	5
Total			11	0.80	12
Assign	0	2	4	1.00	4
Assign	1	2	5	1.00	3
Assign	2	1	1	0.90	9
Total			11	0.90	16
Assign	0	1	3	1.00	10
Assign	1	1	4	1.00	8
Assign	2	1	4	1.00	9
Total			11	1.00	27

Table III. Given an Assignment for a Simple Path, the (Probability, Cost) Pairs under Different Timing Constraints

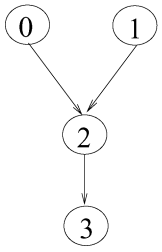
T	6, 7	8 , 9	10 - 12	13
(P , C)	(0.38, 16)	(0.63, 16)	(0.90, 16)	(1.00, 16)

timing constraint 11. It achieves 40.7% reduction in cost compared with cost 27, the result obtained by the algorithms using worst-case scenario of the HAP problem. In many situations, this assignment is good enough for users. We can also select the pair (0.80, 12), which has provable confidence probability 0.8, satisfying the timing constraint 11, while the cost 12 is only 55.5% of the cost of the worst-case, 27. The assignments for each pair of (0.80, 12), (0.90, 16), and (1.00, 27) under the timing constraint 11 are shown in Table II.

Given an assignment, we can get a minimum total cost under every timing constraint. But the probability of achieving this minimum total cost may not be same. For example, if the assignments for nodes 0, 1, and 2 are types 2, 2, and 1, respectively, then the total cost is 12. But the probabilities vary from 0.38 to 1.00. The probabilities under different timing constraints are shown in Table III.

3.2 Tree

Here we give an example of the HAP problem when the input is a tree. The PDFG graph is shown in Figure 2(a), which is a tree with four nodes. The times, costs, and probabilities are shown in Figure 2(b). For example, node 1 can choose one of the two types: R_1 or R_2. When choosing R_1, node 1 will be finished in 1 time unit with probability 0.9, and will finish in time units 3 with probability 0.1. The cost of type R_1 is 10. When node 1 chooses type R_2, it will be finished within 2 time units with probability 0.7, and will be finished in 4 time units with probability 0.3. The cost of type R_2 is 4. Node 0 has two types of FUs to execute on. But for each type, node 0 has fixed execution time. Figure 2(c) shows the time cumulative distribution functions (CDFs) and costs of each node for different FU types.



(a)

Nodes	R1			R2		
	T	P	C	T	P	C
0	1	0.1	8	3	0.3	3
	2	0.9	8	6	0.7	3
1	1	0.9	9	2	0.8	5
	4	0.1	9	6	0.2	5
2	1	0.9	10	2	0.7	4
	3	0.1	10	4	0.3	4
3	1	1.0	9	2	1.0	5

(b)

Nodes	R1			R2		
	T	P	C	T	P	C
0	1	0.1	8	3	0.3	3
	2	1.0	8	6	1.0	3
1	1	0.9	9	2	0.8	5
	4	1.0	9	6	1.0	5
2	1	0.9	10	2	0.7	4
	3	1.0	10	4	1.0	4
3	1	1.0	9	2	1.0	5

(c)

Fig. 2. (a) A given tree; (b) the time, probabilities, and costs of its nodes for different FU types; (c) the time cumulative distribution functions (CDFs) and costs of its node for different FU types.

Table IV. Minimum Total Costs with Computed Confidence Probabilities under Various Timing Constraints for a Tree

T	(P , C)	(P , C)	(P , C)	(P , C)	(P , C)	(P , C)
3	0.08, 36					
4	0.06, 30	0.72, 32	0.81, 36			
5	0.56, 26	0.72, 28	0.81, 32			
6	0.17, 21	0.56, 22	0.63, 26	0.72, 28	0.81, 32	0.90, 36
7	0.17, 17	0.19, 21	0.56, 22	0.80, 26	0.90, 30	
8	0.17, 17	0.24, 21	0.80, 22	0.90, 26	**1.00, 36**	
9	0.24, 17	0.70, 21	0.80, 22	0.90, 23	**1.00, 30**	
10	0.70, 17	0.80, 22	0.90, 23	**1.00, 26**		
11	0.70, 17	**1.00, 21**				
12	**1.00, 17**					

The minimum total costs with computed confidence probabilities under various timing constraints for a tree are shown in Table IV. For example, pair (0.08, 36) with timing constraint 3 means that we can get minimal cost 36 with guaranteed probability 0.08 while satisfying timing constraint 3. The assignment is (node 0: R_1; node 1: R_1; node 3: R_1; node 4: R_1). We multiply all the probabilities and get confidence probability 0.08. We add all the costs together and get total cost 36. All the corresponding execution times are 1. Since nodes 2 and 3 are executed in parallel, the total execution time of the PDFG is 3. The entries with probability equal to 1 (see the entries in boldface) actually give the results to the hard HA problem which shows the worst-case scenario of the HAP problem.

From the two examples above, we can see that the probabilistic approach to the heterogeneous assignment problem has great advantages: It provides guaranteed confidence probabilities to reduce the total costs of systems under different timing constraints. It is suitable to both hard and soft real-time systems. We will give related algorithms and experiments in later sections.

4. THE ALGORITHMS FOR THE HAP PROBLEM

In this section, we propose two algorithms to achieve the optimal solution for the HAP problem when the input PDFG is a simple path or a tree. For the

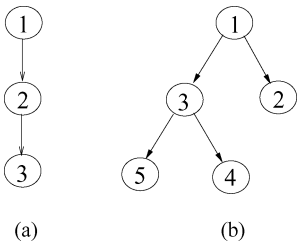


Fig. 3. (a) A simple path with three nodes; (b) a tree with five nodes.

general problem, that is, the given input is a DAG, we propose one near-optimal heuristic algorithm and one optimal algorithm to solve the HAP problem.

4.1 Definitions and Lemma

To solve the HAP problem, we use dynamic programming method. For example, Figure 3(a) shows a simple path with nodes $v_1 \rightarrow v_2 \rightarrow v_3$. Figure 3(b) shows a tree with sequence: $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_5$.
Given the timing constraint L, a PDFG G, and an assignment A, we first give several definitions, as follows:

—$\mathbf{G^i}$: The subgraph rooted at node v_i, containing all the nodes reached by node v_i. In our algorithm, each step will add one node which becomes the root of its subgraph. For example, in Figure 3(b), G^3 is the tree containing nodes 1, 2, and 3.
—$\mathbf{C_A(G^i)}$ and $\mathbf{T_A(G^i)}$: The total cost and total execution time of G^i under assignment A. In our algorithm, each step will achieve the minimum total cost of G^i with computed confidence probabilities under various timing constraints.
—In our algorithm, table $D_{i,j}$ will be built. Each entry of table $D_{i,j}$ will store a linked list of (probability, cost) pairs sorted by probability in ascending order. Here we define the **(Probability, Cost) pair ($\mathbf{P_{i,j}}$, $\mathbf{C_{i,j}}$)** as follows: $C_{i,j}$ is the minimum cost of $C_A(G^i)$ computed by all assignments A satisfying $T_A(G^i) \leq j$ with probability that is greater than or equal to $P_{i,j}$.

In every step in our algorithm, one more node will be included for consideration. The information of this node is stored in local table $B_{i,j}$, which is similar to table $D_{i,j}$. A local table only stores data of probabilities and consumptions of a node itself. Table $B_{i,j}$ is the local table storing only the information of node v_i. In more detail, $B_{i,j}$ is a local table of linked lists that stores pair $(p_{i,j}, c_{i,j})$ sorted by $p_{i,j}$ in an ascending order; $c_{i,j}$ is the cost only for node v_i at time j, and $p_{i,j}$ is the corresponding probability. The building procedures of $B_{i,j}$ are as follows. First, sort the execution time variations in an ascending order. Then, accumulate the probabilities of the same type. Finally, let $L_{i,j}$ be the linked list in each entry of $B_{i,j}$, insert $L_{i,j}$ into $L_{i,j+1}$ while redundant pairs are canceled out based on Algorithm 4.1. For example, node 0 in Figure 1(b) has the following (T: P, C) pairs: (1: 0.9, 10), (3: 0.1, 10) for type R_1, and (2: 0.7, 4), (4: 0.3, 4) for type R_2. After sorting and accumulating, we get (1: 0.9, 10), (2: 0.7, 4), (3: 1.0, 10), and (4: 1.0, 4). We obtain Table V after the insertion.

Table V. An Example of Local Table, $B_{0,j}$

$Time$	1	2	3	4
(P_i, C_i)	(0.9, 10)	(0.7, 4) (0.9, 10)	(0.7, 4) (1.0, 10)	(1.0, 4)

Algorithm 4.1. Redundant-pair removal algorithm.

Require: A list of $(P^k_{i,j}, C^k_{i,j})$
Ensure: A redundant-pair-free list
1: Sort the list by $P_{i,j}$ in an ascending order such that $P^k_{i,j} \leq P^{k+1}_{i,j}$.
2: From the beginning to the end of the list,
3: **for** each two neighboring pairs $(P^k_{i,j}, C^k_{i,j})$ and $(P^{k+1}_{i,j}, C^{k+1}_{i,j})$
4: **if** $P^k_{i,j} = P^{k+1}_{i,j}$
5: **if** $C^k_{i,j} \geq C^{k+1}_{i,j}$
6: cancel the pair $P^k_{i,j}, C^k_{i,j}$
7: **else**
8: cancel the pair $P^{k+1}_{i,j}, C^{k+1}_{i,j}$
9: **end if**
10: **else**
11: **if** $C^k_{i,j} \geq C^{k+1}_{i,j}$
12: cancel the pair $(P^k_{i,j}, C^k_{i,j})$
13: **end if**
14: **end if**
15: **end for**

In this article, we introduce the ***operator "⊕"***. For two (Probability, Cost) pairs H_1 and H_2, if H_1 is $(P^1_{i,j}, C^1_{i,j})$, and H_2 is $(P^2_{i,j}, C^2_{i,j})$, then, after the $\oplus$ operation between H_1 and H_2, we get pair (P', C'), where $P' = P^1_{i,j} * P^2_{i,j}$ and $C' = C^1_{i,j} + C^2_{i,j}$. We denote this operation $\mathbf{H_1} \oplus \mathbf{H_2}$. This is the key operation of our algorithms. The meaning is that when two task nodes add together, the total cost is computed by adding the costs of all nodes together, and the probability corresponding to the total cost is computed by multiplying the probabilities of all nodes based on the basic properties of the probability and cost of a PDFG. For two independent events A and B, $P(A \cup B) = P(A) * P(B)$ and $C(A \cup B) = C(A) + C(B)$.

In our algorithm, $D_{i,j}$ is the table in which each entry has a link list that stores pair $(P_{i,j}, C_{i,j})$. Here, i represents a node number, and j represents time. For example, a link list can be (0.1, 2)→(0.3, 3)→(0.8, 6)→(1.0, 12). Usually, there are redundant pairs in a link list. We give the redundant-pair removal algorithm in Algorithm 4.1.

For example, we have a list with pairs $(0.1, 2) \rightarrow (0.3, 3) \rightarrow (0.5, 3) \rightarrow (0.3, 4)$, we remove the redundant-pair as follows: First, sort the list according to $P_{i,j}$ in an ascending order. This list becomes $(0.1, 2) \rightarrow (0.3, 3) \rightarrow (0.3, 4) \rightarrow (0.5, 3)$. Second, cancel redundant pairs. Comparing (0.1, 2) and (0.3, 3), we keep both. For the two pairs (0.3, 3) and (0.3, 4), we cancel pair (0.3, 4) since the cost 4 is bigger than 3 in pair (0.3, 3). Comparing (0.3, 3) and (0.5, 3), we cancel (0.3, 3)

Algorithm 4.2. *Path_Assign* algorithm: optimal algorithm for the HAP problem when the input is a simple path.

Require: M different types of FUs, a simple path, and the timing constraint L.
Ensure: An optimal assignment for the simple path

(1) Build a local table $B_{i,j}$ for each node of PDFG.
(2) 1: let $D_{1,j} = B_{1,j}$
2: **for** each node $v_i, i > 1$ **do**
3: **for** each time j **do**
4: **for** each time k in $B_{i,k}$ **do**
5: **if** $D_{i-1,j-k}! = NULL$ **then**
6: $D_{i,j} = D_{i-1,j-k} \oplus B_{i,k}$
7: **else**
8: continue
9: **end if**
10: **end for**
11: insert $D_{i,j-1}$ to $D_{i,j}$ and remove redundant pairs using Algorithm 4.1.
12: **end for**
13: **end for**
(3) return $D_{N,j}$

since $0.3 < 0.5$ while $3 \geq 3$. The probability 0.3 is already covered by probability 0.5 while the costs are the same. There is no information lost in redundant-pair removal.

Using Algorithm 4.1, redundant-pair $(P_{i,j}, C_{i,j})$ whenever we find conflicting pairs in a list during a computation. After the $\oplus$ operation and redundant-pair removal, the list $(P_{i,j}, C_{i,j})$ has the following properties:

LEMMA 4.1. *For any $(P^1_{i,j}, C^1_{i,j})$ and $(P^2_{i,j}, C^2_{i,j})$ in the same list:*

(1) $P^1_{i,j} \neq P^2_{i,j}$ *and* $C^1_{i,j} \neq C^2_{i,j}$.
(2) $P^1_{i,j} < P^2_{i,j}$ *if and only if* $C^1_{i,j} < C^2_{i,j}$.

For two linked lists L_1 and L_2, the operation $\mathbf{L_1} \oplus \mathbf{L_2}$ is implemented as follows: First, implement the $\oplus$ operation on all possible combinations of two pairs from different linked lists. Then insert the new pairs into a new linked list and remove redundant pairs using Algorithm 4.1.

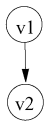
4.2 An Optimal Algorithm for a Simple Path

An optimal algorithm, *Path_Assign*, is proposed in the following. It can give the optimal solution for the HAP problem when the given PDFG is a simple path. Assume the node sequence of the simple path in the HAP problem is $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_N$. For example, in Figure 3(a), the node sequence of the simple path is $v_1 \rightarrow v_2 \rightarrow v_3$.

In algorithm *Path_Assign*, first build a local table $B_{i,j}$ for each node. Next, in step 2 of the algorithm, when $i = 1$, there is only one node. We set the initial value, and let $D_{1,j} = B_{1,j}$. Then, using the dynamic programming method, build the table $D_{i,j}$. For each node v_i under each time j, we try all the times k $(1 \leq k \leq j)$ in table $B_{i,j}$. We use "$\oplus$" on the two tables $B_{i,k}$ and $D_{i-1,j-k}$. Since

node	R1			R2		
	T	P	C	T	P	C
v1	1	0.3	3	—	—	—
	3	0.7	3	—	—	—
v2	1	0.3	3	2	0.4	1
	3	0.7	3	4	0.6	1

(a)



(b)

B Table

T	1	2	3	4
v1	(0.3, 3)	(0.3, 3)	(1.0, 3)	—
v2	(0.3, 3)	(0.4, 1)	(0.4, 1) (1, 3)	(1, 1)

(c)

D Table

1	2	3
(0.3, 3)	(0.3, 3)	(1.0, 3)
—	(0.09, 6)	(0.12, 4)

(d)

Fig. 4. (a) Initial parameters; (b) a PDFG; (c) the corresponding B table; (d) part of the corresponding D table.

$k+(j-k)=j$, the total time for nodes from v_1 to v_i is j. We use the $\oplus$ operation to add the costs of two tables together and multiply the probabilities of two tables with each other. Finally, we use Algorithm 4.1 to cancel the conflicting (Probability, Cost) pairs. The new cost in each pair obtained in table $D_{i,j}$ is the cost of current node v_i at time k plus the cost in each pair obtained in $D_{i-1,j-k}$. Since we have used Algorithm 4.1 to cancel redundant pairs, the cost of each pair in $D_{i,j}$ is the minimum total cost for graph G^i with confidence probability $P_{i,j}$ under timing constraint j.

For example, for the PDFG shown in Figure 4(b), the initial parameters are shown in Figure 4(a). We compute the corresponding B table of node v_1 and v_2. For node v_2, after sorting and accumulating, we get (T: P, C) pairs: (1: 0.3, 3), (2: 0.4, 1), (3: 0.4, 1), (3: 1, 3), and (4: 1.0, 1). The results are shown in Figure 4(c). Figure 4(d) shows the corresponding $D_{i,j}$ table. For instance, computing $D_{2,3}$ entry, for $v_1 \rightarrow v_2$ path and buffer size $j=3$. Buffer size $j=t(v_1)+t(v_2)=3$. Using algorithm *Path_Assign*, we have two cases. Case 1: $3=2+1$. Then $D_{2,3}=D_{1,2}\oplus B_{2,1}$, (0.3, 3) $\oplus$ (0.4, 1) = (0.12, 4). Case 2: $3=1+2$. Then $D_{2,3}=D_{1,1}\oplus B_{2,2}$. We get (0.3, 3) $\oplus$ (0.3, 3) = (0.09, 6). Since (0.09, 6) is inferior to (0.12, 4), it can be removed. Hence, we (0.12, 4) into put the $D_{2,3}$ entry.

The cost in $D_{N,j}$ is the minimum total cost with computed confidence probability under timing constraint j. Given the timing constraint L, the minimum total cost for the graph G is the cost in $D_{N,L}$. Theorem 4.2 follows:

THEOREM 4.2. *For each pair $(P_{i,j}, C_{i,j})$ in $D_{i,j}$ $(1 \le i \le N)$ obtained by algorithm* Path_Assign*, $C_{i,j}$ is the minimum total cost for graph G^i with confidence probability $P_{i,j}$ under timing constraint j.*

PROOF. By induction. *Basic step:* When $i=1$, there is only one node and $D_{1,j}=B_{1,j}$. Thus, when $i=1$, Theorem 4.2 is true. *Induction step:* We need

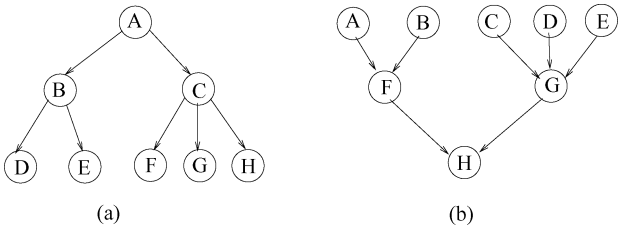


Fig. 5. (a) Tree case 1; (b) tree case 2.

to show that for $i \geq 1$, if for each pair $(P_{i,j}, C_{i,j})$ in $D_{i,j}$, $C_{i,j}$ is the minimum total cost for graph G^i with confidence probability $P_{i,j}$ under timing constraint j, then for each pair $(P_{i+1,j}, C_{i+1,j})$ in $D_{i+1,j}$, $C_{i+1,j}$ is the minimum total cost for graph G^{i+1} with confidence probability $P_{i+1,j}$ under timing constraint j. In step 2 of the algorithm, since $j = k + (j - k)$ for each k in $B_{i+1,j}$, we try all the possibilities to obtain j. Then we use the $\oplus$ operator to add the costs of the two tables and multiply the probabilities of two tables. Finally, we use Algorithm 4.1 to cancel the conflicting (Probability, Cost) pairs. The new cost in each pair obtained in table $D_{i+1,j}$ is the cost of current node $i+1$ at time k plus the cost in each pair obtained in $D_{i,j-k}$. Since we used Algorithm 4.1 to cancel redundant pairs, the cost of each pair in $D_{i+1,j}$ is the minimum total cost for graph G^{i+1} with confidence probability $P_{i+1,j}$ under timing constraint j. Thus, Theorem 4.2 is true for any i $(1 \leq i \leq N)$. □

From Theorem 4.2, we know that $D_{N,L}$ records the minimum total cost of the whole path with corresponding confidence probabilities under the timing constraint L. We can record the corresponding FU type assignment of each node when computing the minimum total cost in step 2 in the algorithm *Path_Assign*. Using this information, we can get an optimal assignment by tracing how to reach $D_{N,L}$.

It takes $O(M * K)$ to compute one value of $D_{i,j}$, where M is the maximum number of FU types, and K is the maximum number of execution time variations for each node. Thus, the complexity of the algorithm *Path_Assign* is $O(|V| * L * M * K)$, where $|V|$ is the number of nodes and L is the given timing constraint. Usually, the execution time of each node is upper-bounded by a constant. So L equals $O(|V|^c)$ (c is a constant). In this case, *Path_Assign* is a polynomial algorithm.

4.3 An Optimal Algorithm for a Tree

In this section we propose an optimal algorithm, *Tree_Assign*, to produce the optimal solution to the HAP problem when the input PDFG 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. We start from the longest leaf node towards the root node in the implementation of the algorithm. For example, both $\{D, E, B\}$ and $\{E, D, B\}$ are sequences for the given tree in Figure 5(a). When we begin to process a node, the processing of all of its child nodes has already been finished in the algorithm. So there is no difference between $\{D, E, B\}$ and $\{E, D, B\}$ in

Algorithm 4.3. *Tree_Assign* algorithm: optimal algorithm for the HAP problem when the input is a tree.

Require: M different types of FUs, a tree, and the timing constraint L.
Ensure: An optimal assignment for the tree

(1) $|V| \leftarrow N$, where $|V|$ is the number of nodes.
Topological sorting all the nodes from the longest leaf to the root. Assume the sequence of the tree is $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_N$, v_N is the root.

(2) Let $D_{1,j} = B_{1,j}$.
Assume $D'_{i,j}$ is the table that stored minimum total cost with computed confidence probabilities under the timing constraint j for the subtree rooted on v_i except v_i.
Nodes $v_{i_1}, v_{i_2}, \cdots, v_{i_w}$ are all child nodes of node v_i.
w is the number of child nodes of node v_i, then

$$D'_{i,j} = \begin{cases} (0,0) & \text{if } w = 0 \\ D_{i_1,j} & \text{if } w = 1 \\ D_{i_1,j} \oplus D_{i_2,j} \oplus \cdots \oplus D_{i_w,j} & \text{if } w \geq 1 \end{cases} \quad (1)$$

Then, for each k in $B_{i,k}$.

$$D_{i,j} = D'_{i,j-k} \oplus B_{i,k} \quad (2)$$

(3) return $D_{N,j}$

terms of our algorithm. In Figure 5(a), we can merge two children first as one node, then we get the simple path. Hence we can get the optimal assignment. If we start from the root, we have two paths. It is possible there are two optimal assignments for the root node and other nodes are common to both paths. For example, in Figure 5(a), if we use start from the root toward a leaf to execute the dynamic program, the best assignments for routes $A \rightarrow B \rightarrow D$ and $A \rightarrow B \rightarrow E$ may have different assignments at node A. We can verify our approach on the tree case 2 in Figure 5(b). The pseudo-polynomial algorithm for trees is shown in Algorithm 4.3.

In *Tree_Assign* algorithm, we first implement topological sorting of all the nodes from the longest leaf to the root. Assume total N nodes, and let the root be v_N. When $w = 1$, node v_i has only one child v_{i_1}. By using the *Path_Assign* algorithm, we get $D_{i,j} = (P_{i,j}, C_{i,j})$, where $P_{i,j} = P_{i_1,j-k} * P_{i,k}$, and $C_{i,j} = C_{i_1,j-k} + C_{i,k}$, by using the operator "$\oplus$". If $w \geq 1$, node v_i has multiple children. We merge all the children of node v_i into one pseudo child. Then we can use the *Path_Assign* algorithm to get the final solution. The merging procedures are as follows. At the same time j, sum up the costs of all children and multiply the probabilities of all children. For instance, if node v_i has two child nodes v_{i_1} and v_{i_2}, then $D'_{i,j} = (P'_{i,j}, C'_{i,j})$, where $P'_{i,j} = P_{i_1,j} * P_{i_2,j}$ and $C'_{i,j} = C_{i_1,j} + C_{i_2,j}$. After merging all children into one pseudo child, we can continue to implement the *Path_Assign* algorithm to get the final solution for the tree.

The execution of $D_{i-1,j}$ for each child node of v_i has finished before executing $D_{i,j}$. From Equation (1), $D'_{i,j}$ gets the summation of the minimum total cost of all child nodes of v_i because they can be executed simultaneously within time j. From Equation (2), the minimum total cost is selected from all possible costs caused by adding v_i to graph G^i. Therefore, for each pair $(P_{i,j}, C_{i,j})$ in $D_{i,j}$,

$C_{i,j}$ is the minimum total cost for the graph G^i with confidence probability $P_{i,j}$ under timing constraint j.

Algorithm *Tree_Assign* gives the optimal solution when the given PDFG is a tree. Theorem 4.3 follows:

THEOREM 4.3. *For each pair $(P_{i,j}, C_{i,j})$ in $D_{i,j}$ $(1 \leq i \leq N)$ obtained by algorithm* Tree_Assign*, $C_{i,j}$ is the minimum total cost for the graph G^i with confidence probability $P_{i,j}$ under timing constraint j.*

PROOF. By induction. *Basic step:* When $i = 1$, there is only one node and $D_{1,j} = B_{1,j}$. Thus, when $i = 1$, Theorem 4.3 is true. *Induction step:* We need to show that for $i \geq 1$, if for each pair $(P_{i,j}, C_{i,j})$ in $D_{i,j}$, $C_{i,j}$ is the minimum total cost for the graph G^i with confidence probability $P_{i,j}$ under timing constraint j, then for each pair $(P_{i+1,j}, C_{i+1,j})$ in $D_{i+1,j}$, $C_{i+1,j}$ is the total system cost for the graph G^{i+1} with confidence probability $P_{i+1,j}$ under timing constraint j. The execution of $D_{i,j}$ for each child node of v_{i+1} was finished before executing $D_{i+1,j}$. From Equation (1), $D'_{i+1,j}$ gets the summation of the minimum total cost of all child nodes of v_{i+1} because they can be executed simultaneously within time j. From Equation (2), the minimum total cost is selected from all possible costs caused by adding v_{i+1} to graph G^{i+1}. So for each pair $(P_{i+1,j}, C_{i+1,j})$ in $D_{i+1,j}$, $C_{i+1,j}$ is the minimum total cost for the graph G^{i+1} with confidence probability $P_{i+1,j}$ under timing constraint j. Therefore, Theorem 4.3 is true for any i $(1 \leq i \leq N)$. □

The complexity of algorithm *Tree_Assign* is $O(|V| * L * M * K)$, where $|V|$ is the number of nodes, L is the given timing constraint, M is the maximum number of FU types for each node, and K is the maximum number of execution time variations for each node. When L equals $O(|V|^c)$ (c is a constant), which is the general case in practice, algorithm *Tree_Assign* is polynomial.

4.4 A Near-Optimal Heuristic Algorithm for DAG

In this and the next sections, we propose two algorithms to solve the general case of the HAP problem, that is, the given input is a *directed acyclic graph* (DAG). Since the general problem of the HAP problem is NP-complete, one of the two algorithms we propose here is a near-optimal heuristic algorithm, and the other is an optimal one. In many cases, the near-optimal heuristic algorithm will give us the same results as the those for the optimal algorithm. The optimal algorithm is suitable for cases when the given PDFG has a small number of multiparent and multichild nodes.

We give an example of DAG. The input PDFG is shown in Figure 6(a), which has five nodes. The time, probabilities, and costs of each node are shown in Figure 6(b). Node 4 is a multichild node which has three children: 0, 2, and 3. Node 0 is a multiparent node and has two parents: 2 and 4. Figure 6(c) shows the time-cumulative distribution functions (CDFs) and costs of each node for different FU types.

We give the near-optimal heuristic algorithm (*DAG_Heu*) for the HAP problem when the given PDFG is a DAG, which is shown in Algorithm 4.4.

Algorithm 4.4. The *DAG_Heu* algorithm.

Require: M different types of FUs, a DAG, and the timing constraint L.
Ensure: A near-optimal heuristic assignment for the DAG

(1) $SEQ \leftarrow$ Sequence obtained by topological sorting all the nodes.
(2) $t_{mp} \leftarrow$ the number of multiparent nodes;
$t_{mc} \leftarrow$ the number of multichild nodes;
If $t_{mp} < t_{mc}$, use bottom-up approach;
else, use top down approach.
(3) If bottom-up approach, use the following algorithm;
If top-down approach, just reverse the sequence.
$|V| \leftarrow N$, where $|V|$ is the number of nodes.
(4) $SEQ \leftarrow \{v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_N\}$, in bottom-up fashion;
$D_{1,j} \leftarrow B_{1,j}$;
$D'_{i,j} \leftarrow$ the table that stored MIN(C) with $Prob.(T \leq j) \geq P$ for the subgraph rooted on v_i except v_i;
$v_{i_1}, v_{i_2}, \cdots, v_{i_w} \leftarrow$ all child nodes of node v_i;
$w \leftarrow$ the number of child nodes of node v_i.

$$D'_{i,j} = \begin{cases} (0,0) & \text{if } w = 0 \\ D_{i_1,j} & \text{if } w = 1 \\ D_{i_1,j} \oplus \cdots \oplus D_{i_w,j} & \text{if } w > 1 \end{cases} \quad (3)$$

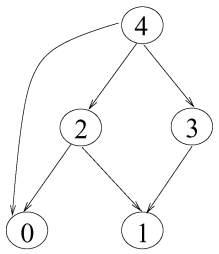
(5) Computing $D_{i_1,j} \oplus D_{i_2,j}$:
$G' \leftarrow$ the union of all nodes in the graphs rooted at nodes v_{i_1} and v_{i_2};
Travel all the graphs rooted at nodes v_{i_1} and v_{i_2};
If a node is a common node, then use a selection function to choose the FU type of a node.
(6) For each k in $B_{i,k}$.

$$D_{i,j} = D'_{i,j-k} \oplus B_{i,k} \quad (4)$$

(7) Then use Algorithm 4.1 to remove redundant pairs;
$D_{N,j} \leftarrow$ a table of MIN(C) with $Prob.(T \leq j) \geq P$;
Output $D_{N,L}$.

In algorithm *DAG_Heu*, if using a bottom-up approach, for each sequence node use the simple path algorithm to get the dynamic table of a parent node. If using the top-down approach, reverse the sequence and use the same algorithm. For example, in Figure 6(a), there are two multichild nodes: 2 and 4, and there are also two multiparent nodes, that is, nodes 0 and 1. Hence, we can use either approach.

In algorithm *DAG_Heu*, we also have to solve the problem of common nodes, that is, one node appears in two or more graphs that are rooted by the child nodes of node v_i. In Equation (3), even if there are common nodes, we must not count the same node twice. That is, the cost is just added once, and the probability is multiplied once. For example, in Figure 7, the two children of node v_i are v_{i_1} and v_{i_2}. The two subgraphs rooted at nodes v_{i_1} and v_{i_2} have common areas (the shaded area in Figure 7). The nodes in this common area are called *common nodes*. When we compute the cost and probability of node v_i,



(a)

Nodes	R1			R2		
	T	P	C	T	P	C
4	1	0.8	9	3	0.9	3
	2	0.2	9	4	0.1	3
3	1	1.0	8	2	1.0	6
2	1	1.0	8	2	1.0	6
1	1	0.7	8	2	0.9	4
	3	0.3	8	4	0.1	4
0	1	0.9	10	3	0.8	3
	2	0.1	10	4	0.2	3

(b)

Nodes	R1			R2		
	T	P	C	T	P	C
4	1	0.8	9	3	0.9	3
	2	1.0	9	4	1.0	3
3	1	1.0	8	2	1.0	6
2	1	1.0	8	2	1.0	6
1	1	0.7	8	2	0.9	4
	3	1.0	8	4	1.0	4
0	1	0.9	10	3	0.8	3
	2	1.0	10	4	1.0	3

(c)

Fig. 6. (a) A given DAG; (b) the time, probabilities and costs of its nodes for different FU types; (c) the time cumulative distribution functions (CDFs) and costs of its nodes for different FU types.

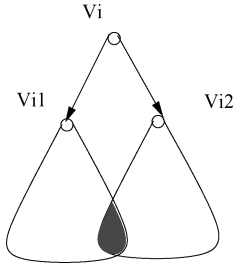


Fig. 7. Problem of common nodes.

we can only count these common nodes once. If a common node has conflicting FU type selection, then we need to define a selection function to decide which FU type should be chosen for the common node. For example, we can select the FU type that has a smaller execution time as the FU type of a common node.

Due to the problem of common nodes, algorithm *DAG_Heu* is not optimal. The reason is that an assignment conflict for a common node may exist, and algorithm *DAG_Heu* cannot solve this problem. For example, in Figure 6, we focus on the subgraph composed of nodes 1, 2, 3, and 4. Using the bottom-up approach, there will be two paths from node 1 to node 4. Path a is $1 \rightarrow 2 \rightarrow 4$, and path b is $1 \rightarrow 3 \rightarrow 4$. Hence, node 1 is a common node for both paths while node 4 is the root. It is possible, under a timing constraint, that the best assignment

for path a gives node 1 assignment as FU type 1, while the best assignment for path b gives node 1 assignment as FU type 2. This kind of assignment conflict cannot be solved by algorithm *DAG_Heu*. Hence, *DAG_Heu* is not an optimal algorithm, although it is very efficient in practice.

From algorithm *DAG_Heu*, we know $D_{N,L}$ records the minimum total cost of the whole path within the timing constraint L. We can record the corresponding FU type assignment of each node when computing the minimum system cost in the algorithm *DAG_heu*. Using this information, we can get an optimal assignment by tracing how to reach $D_{N,L}$.

It takes at most $O(|V|)$ to compute common nodes for each node in the algorithm *DAG_heu*, where $|V|$ is the number of nodes. Thus, the complexity of the algorithm *DAG_heu* is $O(|V|^2 * L * M * K)$, where L is the given timing constraint, M is the maximum number of FU types for each node, and K is the maximum number of execution time variations for each node. Usually, the execution time of each node is upper bounded by a constant, that is, L equals $O(|V|^c)$ (c is a constant). In this case, *DAG_heu* is a polynomial algorithm.

4.5 An Optimal Algorithm for DAG

In this section we give the optimal algorithm (*DAG_Opt*) for the HAP problem when the given PDFG is a DAG. In *DAG_Opt*, we exhaust all the possible assignments of multiparent or multichild nodes. Without loss of generality, assume we use the bottom-up approach. If the total number of nodes with a multiparent is t, and there are maximum K variations for the execution times of all nodes, then we give each of these t nodes a fixed assignment. We exhaust all of the K^t possible fixed assignments by algorithm *DAG_Heu* without using the selection function, since there is no assignment conflict for a common node.

Algorithm *DAG_Opt* gives the optimal solution when the given PDFG is a DAG, which is shown in Algorithm 4.5. Theorem 4.4 and Theorem 4.5 follow from this.

THEOREM 4.4. *In each possible fixed assignment, for each pair ($P_{i,j}$, $C_{i,j}$) in $D_{i,j}$ ($1 \leq i \leq N$) obtained by algorithm* DAG_Opt*, $C_{i,j}$ is the minimum total cost for the graph G^i with confidence probability $P_{i,j}$ under timing constraint j.*

PROOF. By induction. *Basic step:* When $i = 1$, there is only one node and $D_{1,j} = B_{1,j}$. Thus, when $i = 1$, Theorem 4.4 is true. *Induction step:* We need to show that for $i \geq 1$, if for each pair $(P_{i,j}, C_{i,j})$ in $D_{i,j}$, $C_{i,j}$ is the minimum total cost of the graph G^i, then for each pair $(P_{i+1,j}, C_{i+1,j})$ in $D_{i+1,j}$, $C_{i+1,j}$ is the total cost of the graph G^{i+1} with confidence probability $P_{i+1,j}$ under timing constraint j. According to the bottom-up approach (for the top-down approach, just reverse the sequence), the execution of $D_{i,j}$ for each child node of v_{i+1} was finished before executing $D_{i+1,j}$. From Equation (5), $D'_{i+1,j}$ gets the summation of the minimum total cost of all child nodes of v_{i+1} because they can be executed simultaneously within time j. We avoid the repeat counting of the common nodes. Hence, each node in the graph rooted by node v_{i+1} is counted only once. From Equation (6), the minimum total cost is selected from all possible costs caused by adding v_{i+1} into the subgraph rooted on v_{i+1}. Hence, for each pair

Algorithm 4.5. *DAG_Opt* algorithm.

Require: M different types of FUs, a DAG, and the timing constraint L.
Ensure: An optimal assignment for the DAG

(1) $SEQ \leftarrow$ Sequence obtained by topological sorting of all the nodes.
(2) $t_{mp} \leftarrow$ the number of multiparent nodes;
$t_{mc} \leftarrow$ the number of multichild nodes;
If $t_{mp} < t_{mc}$, use bottom-up approach;
else, use top-down approach.
(3) If bottom-up approach, use the following algorithm;
If top-down approach, just reverse the sequence.
$|V| \leftarrow N$, where $|V|$ is the number of nodes.
(4) If the total number of nodes with multiparent is t, and there are maximum K variations for the execution data loads of all nodes, then we will give each of these t nodes a fixed assignment.
(5) For each of the K^t possible fixed assignments,
$SEQ \leftarrow \{v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_N\}$, in bottom-up fashion;
$D_{1,j} \leftarrow B_{1,j}$;
$D'_{i,j} \leftarrow$ the table that stored MIN(C) with $Prob.(T \leq j) \geq P$ for the subgraph rooted on v_i except v_i;
$v_{i_1}, v_{i_2}, \cdots, v_{i_w} \leftarrow$ all child nodes of node v_i;
$w \leftarrow$ the number of child nodes of node v_i.

$$D'_{i,j} = \begin{cases} (0,0) & \text{if } w = 0 \\ D_{i_1,j} & \text{if } w = 1 \\ D_{i_1,j} \oplus \cdots \oplus D_{i_w,j} & \text{if } w > 1 \end{cases} \tag{5}$$

(6) Computing $D_{i_1,j} \oplus D_{i_2,j}$:
$G' \leftarrow$ the union of all nodes in the graphs rooted at nodes v_{i_1} and v_{i_2};
Travel all the graphs rooted at nodes v_{i_1} and v_{i_2};

(7) For each k in $B_{i,k}$.

$$D_{i,j} = D'_{i,j-k} \oplus B_{i,k} \tag{6}$$

(8) For each possible fixed assignment, we get a $D_{N,j}$. Merge the (P, C) pairs in all the possible $D_{N,j}$ together, and sort them in ascending sequence according to P.
(9) Then use Algorithm 4.1 to remove redundant pairs;
$D_{N,j} \leftarrow$ a table of MIN(C) with $Prob.(T \leq j) \geq P$;
Output $D_{N,L}$.

$(P_{i+1,j}, C_{i+1,j})$ in $D_{i+1,j}$, $C_{i+1,j}$ is the total cost of the graph G^{i+1} with confidence probability $P_{i+1,j}$ under timing constraint j. Therefore, Theorem 4.4 is true for any i $(1 \leq i \leq N)$. □

THEOREM 4.5. *For each pair $(P_{i,j}, C_{i,j})$ in $D_{N,j}$ $(1 \leq j \leq L)$ obtained by algorithm* DAG_Opt*, $C_{i,j}$ is the minimum total cost for the given DAG G with confidence probability $P_{i,j}$ under timing constraint j.*

PROOF. According to Theorem 4.4, in each possible fixed assignment, for each pair $(P_{i,j}, C_{i,j})$ in $D_{i,j}$ that we obtained, $C_{i+1,j}$ is the total cost of the graph G^{i+1} with confidence probability $P_{i+1,j}$ under timing constraint j. In step (4)

Table VI. Minimum Total Costs with Computed Confidence Probabilities under Various Timing Constraints for a DAG

T	(P , C)	(P , C)	(P , C)	(P , C)	(P , C)	(P , C)
3	0.50, 43					
4	0.72, 39					
5	0.58, 30	0.72, 35	0.90, 39			
6	0.58, 28	0.72, 30	0.80, 32	0.81, 33	0.90, 35	**1.00, 43**
7	0.65, 24	0.80, 28	0.81, 29	0.90, 30	**1.00, 32**	
8	0.65, 22	0.81, 24	0.90, 26	**1.00, 28**		
9	0.90, 22	**1.00, 26**				
10	**1.00, 22**					

of the algorithm *DAG_Opt*, we try all the possible fixed assignments, combine them into a new row $D_{N,j}$ in a dynamic table, and remove redundant pairs using Algorithm 4.1. Hence, for each pair $(P_{i,j}, C_{i,j})$ in $D_{N,j}$ $(1 \leq j \leq L)$ obtained by algorithm *DAG_Opt*, $C_{i,j}$ is the minimum total cost for the given DAG G with confidence probability $P_{i,j}$ under timing constraint j. □

In algorithm *DAG_Opt*, there are K^t loops and each loop needs $O(|V|^2 * L * M * K)$ running times. The complexity of algorithm *DAG_Opt* is $O(K^t * |V|^2 * L * M * K)$, where t is the total number of nodes with multiparent (or multichild) in the bottom-up approach (or top-down approach), $|V|$ is the number of nodes, L is the given timing constraint, M is the maximum number of FU types for each node, and K is the maximum number of execution time variations for each node. Algorithm *DAG_Opt* is exponential, hence it can not be applied to a graph with large amounts of multiparent and multichild nodes.

For Figure 6, the minimum total costs with computed confidence probabilities under the timing constraint are shown in Table VI. The entries with probability equal to 1 (see the entries in boldface) actually give the results to the hard HA problem, which shows the worst-case scenario of the HAP problem. In this example, the algorithm *DAG_Heu* gives the same results as those of the algorithm *DAG_Opt*. Actually, experiments show that although algorithm *DAG_Heu* is only near-optimal, it can give the same results as those given by the optimal algorithm in most cases.

5. EXPERIMENTS

This section presents the experimental results of our algorithms. We conducted experiments on a set of benchmarks including a 4-stage lattice filter, an 8-stage lattice filter, a voltera filter, a differential equation solver, a RLS-languerre lattice filter, and an elliptic filter. The PDFG for the first three filters are trees and those for the others are DAGs. The basic information about the benchmarks is shown in Table VII, in which a multiparent node is a node with more than one parent and a multichild node is a node with more than one child.

M different FU types, $R_1, \ldots, R_M$, are used in the system, in which an FU with type R_1 is the quickest with the highest cost and an FU with type R_M is the slowest with the lowest cost. For example, in a computation task in a digital camera chip: Let $M = 3$. We perform a computation based on the benchmark voltera, which has 27 task nodes. At each node, we perform add, sub, shift or

Table VII. Basic Information for Benchmarks

Benchmarks	PDFG	# of nodes	# of multi-parent	# of multi-child
voltera	Tree	27		
4-lat-iir	Tree	26		
8-lat-iir	Tree	42		
Diff. Eq.	DAG	11	3	1
RLS-lagu.	DAG	19	6	3
elliptic	DAG	34	8	5

Table VIII. Minimum Total Costs with Computed Confidence Probabilities under Various Timing Constraints for Voltera Filter

Voltera Filter							
TC	0.7		0.8		0.9		1.0
	cost	%	cost	%	cost	%	cost
62	7896		×		×		×
80	7166		7169		7847		×
100	5366	31.5	5369	31.4	6047	22.8	**7827**
125	5347	31.7	5352	31.6	5843	25.3	**7820**
150	4032	43.8	4066	43.6	4747	32.8	**7169**
175	1604	66.2	2247	52.7	2947	37.9	**4747**
200	1587	66.3	1618	65.6	2318	50.7	**4704**
225	1580	46.4	1593	45.9	1647	44.1	**2947**
250	1580	31.9	1582	31.8	1604	30.8	**2318**
273	1580	4.1	1580	4.1	1580	4.1	**1647**
274	1580		1580		1580		**1580**
Ave. Redu.(%)		40.2		38.3		31.0	

multiplication operations. For instance, at node 1, perform the add operation. There are 3 ALUs (ALU A, B, and C) available for this add operation. At node 2 perform the multiplication operation, there are three multipliers (multiplier 1, 2, and 3) available for this multiply operation. Node 3 is a shift operation, and there are three shifters available to implement this operation.

The distribution of execution times of each node is Gaussian. The probabilities are obtained by building a historic table and using statistical profiling. For each benchmark, the first timing constraint we used is the minimum execution time. The experiments were performed on a Dell PC with a P4 2.1 G processor and 512 MB memory running Red Hat Linux 7.3. In our experiments, the instructions (represented by nodes) were obtained from TI TMS320C 6000 Instruction Set. We first obtained the linear assembly code based on TI C6000 for various benchmarks and then we modeled them as the PDFG.

The experiments on the voltera filter, the 4-stage lattice filter, and the 8-stage lattice filter finished in less than one second, in which we compared our algorithms with the *hard HA ILP algorithm* [Ito and Parhi 1995]. The experimental results for these filters are shown in Tables VIII to X. In each table, column TC represents the given timing constraint. The minimum total costs obtained from different algorithms, *Tree_Assign* and the optimal *hard HA ILP algorithm* [Ito and Parhi 1995], are shown in each entry. Columns 1.0, 0.9, 0.8, and 0.7 show that the confidence probability is 1.0, 0.9, 0.8, and 0.7, respectively. Algorithm *Tree_Assign* covers all the probability columns, while the

Table IX. Minimum Total Costs with Computed Confidence Probabilities under Various Timing Constraints for 4-Stage Lattice Filter

4-stage Lattice IIR Filter							
TC	0.7		0.8		0.9		1.0
	cost	%	cost	%	cost	%	cost
81	3462		×		×		×
100	3452		3472		×		×
125	3452		2290		3525		×
150	1881		2257		2690		×
166	1858	47.3	2250	36.2	2290	35.0	**3525**
175	1853	46.8	1890	45.7	1890	45.7	**3481**
200	1325	50.4	1325	50.4	1462	45.3	**2672**
226	1259	15.5	1259	15.5	1450	2.7	**1490**
227	1259		1259		1259		**1259**
Ave. Redu.(%)		40.0		36.9		32.4	

Table X. Minimum Total Costs with Computed Confidence Probabilities under Various Timing Constraints for 8-Stage Lattice Filter

8-stage Lattice IIR Filter							
TC	0.7		0.8		0.9		1.0
	cost	%	cost	%	cost	%	cost
94	4543		×		×		×
100	4499		5039		×		×
125	1870		2375		4539		×
144	1863	66.4	1863	66.4	2380	57.1	**5543**
150	1820	66.9	1849	66.3	2362	57.0	**5495**
175	795	67.4	951	61.0	1339	45.1	**2439**
200	732	43.5	732	43.5	775	40.2	**1295**
225	595	29.1	638	23.8	639	23.8	**839**
250	532	12.9	540	11.6	540	11.6	**611**
277	506	4.9	511	4.0	511	4.0	**532**
278	506		506		506		**506**
Ave. Redu.(%)		41.6		39.5		34.1	

optimal *hard HA ILP algorithm* [Ito and Parhi 1995] only includes column 1.0, which is in boldface. For example, in Table VIII, under the timing constraint 100, the entry under column 1.0 is 7827, which is the minimum total cost for the hard HA problem. The entry under column 0.9 is 6047, which means we can achieve minimum total cost 6047 with confidence probability 0.9 under timing constraint 100. From the information provided in the structure of the linked list in each entry of the dynamic table, we are able to trace how to get a satisfactory assignment.

Column % shows the percentage of reductions in the total cost, compared to the results of the algorithm, with those obtained by the optimal *hard HA ILP algorithm* [Ito and Parhi 1995]. The average percentage reduction is shown in the last row: Ave. Redu(%) for Tables VIII to X. The entry with "×" means no solution is available. In Table VIII, under timing constraint 80, the optimal *hard HA ILP algorithm* [Ito and Parhi 1995] cannot find a solution. However, we can find solution 7847 with 0.9 probability which guarantees that the total execution time of the PDFG is less than or equal to the timing constraint 80.

Table XI. Minimum Total Costs with Computed Confidence Probabilities under Various Timing Constraints for Differential Equation Solver

Diff. Eq. Solver							
TC	0.7		0.8		0.9		1.0
	cost	%	cost	%	cost	%	cost
35	5320		×		×		×
40	5130		5230		×		×
50	5020		5040		5100		×
60	3330	38.6	3430	36.7	3830	29.3	**5420**
70	2730	49.7	2830	47.9	3230	40.5	**5430**
80	2590	47.1	2620	46.5	3020	38.4	**4900**
90	2520	48.4	2540	48.0	3100	36.5	**4880**
100	1910	38.2	1930	37.5	2690	12.9	**3090**
117	1970	32.5	1970	32.5	1970	32.5	**2920**
118	1970		1970		1970		**1970**
Ave. Redu.(%)		42.4		41.5		31.7	

Table XII. The Minimum Total Costs with Computed Confidence Probabilities Under Various Timing Constraints for the RLS-Laguerre Filter

RLS-Laguerre filter							
TC	0.7		0.8		0.9		1.0
	cost	%	cost	%	cost	%	cost
49	7803		×		×		×
60	7790		7791		7793		×
70	7082		7087		7787		×
80	5403	30.6	5991	23.0	5993	23.0	**7780**
100	3969	48.9	4669	39.9	5380	30.8	**7769**
125	2165	59.8	2269	58.0	4664	13.6	**5390**
150	1564	66.4	2264	49.3	2864	38.6	**4667**
175	1564	66.5	1564	66.5	2264	51.5	**4664**
205	1564	30.9	1564	30.9	1564	30.9	**2264**
206	1564		1564		1564		**1564**
Ave. Redu.(%)		50.5		44.6		31.4	

Through the experimental results, we find that our algorithms have better performance compared with the optimal *hard HA ILP algorithm* [Ito and Parhi 1995]. On average, algorithm *Tree_Assign* gives a cost reduction of 32.5% with 0.9 confidence probability in satisfying timing constraints, and a cost reduction of 38.2% and 40.6% with 0.8 and 0.7 confidence probability, respectively, in satisfying timing constraints.

We also conducted experiments on a different equation solver, a RLS-Laguerre filter, and an elliptic filter; these are DAGs. The different equation solver has 1 multichild node and 3 multiparent nodes. Using a top-down approach, we exhaust all 3 possible fixed assignments. The RLS-Laguerre filter has 3 multichild nodes and 6 multiparent nodes. Using a top-down approach, we implement all $3^3 = 27$ possibilities. The elliptic filter has 5 multichild nodes and 8 multiparent nodes. There are total $3^5 = 243$ possibilities by the top-down approach. The experimental results for these filters are shown in Tables XI to XIII. Column % shows the percentage of reduction on system costs, compared to the results for soft real-time with those for hard real-time. The average

Table XIII. Minimum Total Costs with Computed Confidence Probabilities under Various Timing Constraints for Elliptic Filter

Elliptic Filter							
TC	0.7		0.8		0.9		1.0
	cost	%	cost	%	cost	%	cost
120	6025		×		×		×
130	5685		5722		×		×
140	4685		4722		7524		×
150	4661		4680		5721		×
157	2585	65.5	4681	37.6	5681	24.2	**7496**
160	2604	65.3	2641	64.8	5703	24.1	**7511**
170	1983	73.4	2571	65.5	4382	41.2	**7449**
180	1900	70.7	1933	70.2	2612	59.7	**6482**
190	1850	57.4	1872	56.9	2533	41.7	**4344**
200	1816	57.8	1823	57.6	1933	55.1	**4301**
210	1803	57.6	1807	57.5	1881	55.8	**4257**
220	1798	30.3	1798	30.3	1828	29.1	**2579**
230	1796	7.1	1796	7.1	1822	5.7	**1933**
231	1796		1796		1796		**1796**
Ave. Redu.(%)		53.9		49.7		37.4	

percentage reduction is shown in the last row: Ave. Redu(%) in all these three tables. The entry with "×" means no solutions is available. Under timing constraint 50 in Table XI, there is no solution for hard real-time. However, we can find solution 5100 with 0.9 probability which guarantees that the total execution time of the PDFG is less than or equal to the timing constraint 50.

The experimental results show that our algorithm can greatly reduce the total cost while having a guaranteed confidence probability. On average, algorithm *DAG_Opt* gives a cost reduction of 33.5% with 0.9 confidence probability under timing constraints, and a cost reduction of 45.3% and 48.9% with, respectively, 0.8 and 0.7 confidence probabilities satisfying timing constraints. The experiments using *DAG_Heu* on these benchmarks are finished within several seconds, and the experiments using *DAG_Opt* on these benchmarks are finished within several minutes.

The experimental results are the same for the experiments on these three DAG benchmarks using algorithm *DAG_Heu* or *DAG_Opt*. When the number of multiparent and multichild nodes is large, we can use algorithm *DAG_Heu*, which will give good results in most cases.

We also unfold all the benchmarks 10 times, 15 times, and 20 times, which means that the number of nodes increases 10 times, 15 times, and 20 times. The largest benchmark, the 8-stage lattice filter originally has only 42 nodes. After unfolding 20 times, the benchmark has 840 nodes. We test the scalability of our algorithms. When the input graph is a simple path or a tree, our optimal algorithms *Path_Assign* and *Tree_Assign* run very fast (less than one minute), while the ILP algorithm takes a much larger time to get results. When the unfolding factor is 15, it cannot get the result even after a day for the 8-stage lattice filter. When the input graph is a DAG, our algorithm *DAG_Heu* runs very fast. It only takes several minutes to finish a 20 times unfolded benchmark.

The advantages of our algorithms over the *hard HAILP algorithm* [Ito and Parhi 1995] are summarized as follows. First, our algorithms are efficient and provide an overview of all possible variations of minimum costs compared with the the worst-case scenario generated by the *hard HAILP algorithm* [Ito and Parhi 1995]. More information and choices are provided by our algorithms. Second, it is possible to greatly reduce the system's total cost while having a very high confidence probability under different timing constraints. Third, given an assignment, we are able to get a minimum total cost with different confidence probabilities under each timing constraint. Finally, our algorithms are very quick and practical.

6. CONCLUSION

This article proposed a probabilistic approach to high-level synthesis of special-purpose architectures for real-time embedded systems using heterogeneous functional units with probabilistic execution times. For the *heterogeneous assignment with probability* (HAP) problem, algorithms *Path_Assign* and *Tree_Assign*, were proposed to give optimal solutions when the input graphs are a simple path and a tree, respectively. Two other algorithms, one is optimal and the other is near-optimal heuristic, were proposed to solve the general problem. Experiments show that our algorithms achieved significant energy-saving and provided more design choices to achieve a minimum total cost, while the timing constraint was satisfied with a guaranteed confidence probability. Our algorithms are useful for both hard and soft real-time systems.

REFERENCES

Banino, C., Beaumont, O., Carter, L., Ferrante, J., Legrand, A., and Robert, Y. 2004. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Trans. Parall. Distrib. Syst. 15*, 4, 319–330.

Beaumont, O., Legrand, A., Marchal, L., and Robert, Y. 2004. Pipelining broadcasts on heterogeneous platforms. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2004)*, IEEE.

Beaumont, O., Legrand, A., Marchal, L., and Robert, Y. 2003a. Scheduling strategies for mixed data and task parallelism on heterogeneous clusters. *Parall. Process. Lett. 13*, 2, 225–244.

Beaumont, O., Legrand, A., and Robert, Y. 2003b. The master-slave paradigm with heterogeneous processors. *IEEE Trans. Parall. Distrib. Syst. 14*, 9, 897–908.

Beaumont, O., Legrand, A., and Robert, Y. 2002. Static scheduling strategies for heterogeneous systems. *Comput. Informatics 21*, 413–430.

Bettati, R. and Liu, J. W.-S. 1992. End-to-end scheduling to meet deadlines in distributed systems. In *Proceedings of the International Conference. on Distributed Computing Systems*. 452–459.

Chang, Y.-N., Wang, C.-Y., and Parhi, K. K. 1996. Loop-list scheduling for heterogeneous functional units. In Proceedings of the *6th Great Lakes Symposium on Very Large-Scale Integration*. 2–7.

Chao, L.-F. and Sha, E. H. -M. 1997. Scheduling dataflow graphs via retiming and unfolding. *IEEE Trans. Parall. Distrib. Syst. 8*, 1259–1267.

Chao, L. -F. and Sha, E. H. -M. 1995. Static scheduling for synthesis of dsp algorithms on various models. *J.VLSI Signal Process. Syst. 10*, 207–223.

De Man, H., Catthoor, F., Goossens, G., Vanhoof, J., Van Meerbergen, S. N., and Huisken, J. A. 1990. Architecture-driven synthesis techniques for VLSI implementation of DSP algorithms. *Proc. IEEE 78*, 2, 319–335.

DOGAN, A. AND ÖZGÜNER, F. 2002. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Trans. Parall. Distrib. Syst. 13*, 308–323.

FOSTER, I. 1994. *Designing and Building Parallel Program: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley.

GEBOTYS, C. H. AND ELMASRY, M. 1993. Global optimization approach for architectural synthesis. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 12*, 1266–1278.

HE, Y., SHAO, Z., XIAO, B., ZHUGE, Q., AND SHA, E. H. -M. 2003. Reliability driven task scheduling for tightly coupled heterogeneous systems. In *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*.

HOU, C. -J. AND SHIN, K. G. 1997. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. *IEEE Trans. Computers* 46, 1338–1356.

HUA, S. AND QU, G. 2003. Approaching the maximum energy saving on embedded systems with multiple voltages. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. 26–29.

HUA, S., QU, G., AND BHATTACHARYYA, S. S. 2003a. Energy reduction techniques for multimedia applications with tolerance to deadline misses. In *Proceedings of the Design Automation Conference (DAC'03)*. 131–136.

HUA, S., QU, G., AND BHATTACHARYYA, S. S. 2003b. Exploring the probabilistic design space of multimedia systems. In *Proceedings of the IEEE International Workshop on Rapid System Prototyping*. 233–240.

HWANG, C. -T., LEE, J. -H., AND HSU, Y. -C. 1991. A formal approach to the scheduling problem in high level synthesis. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 10*, 464–475.

ITO, K., LUCKE, L., AND PARHI, K. 1998. ILP-based cost-optimal DSP synthesis with module selection and data format conversion. *IEEE Trans. VLSI Syst. 6*, 582–594.

ITO, K. AND PARHI, K. 1995. Register minimization in cost-optimal synthesis of dsp architecture. In *Proceedings of the IEEE VLSI Signal Processing Workshop*.

KALAVADE, A. AND MOGHE, P. 1998. A tool for performance estimation of networked embedded end-systems. In *Proceedings of the IEEE/ACM Design Automation Conference*. 257–262.

PARHI, K. AND MESSERSCHMITT, D. G. 1991. Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding. *IEEE Trans. Computers 40*, 178–195.

LESTER, B. P. 1993. *The Art of Parallel Programming*. Prentice Hall, Englewood Cliffs, NJ.

LI, W. N., LIM, A., AGARWAL, P., AND SAHNI, S. 1993. On the circuit implementation problem. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 12*, 1147–1156.

MCFARLAND, M. C., PARKER, A. C., AND CAMPOSANO, R. 1990. The high-level synthesis of digital systems. *Proc. IEEE 78*, 301–318.

PASSOS, N. L., SHA, E. H. -M., AND BASS, S. C. 1994. Loop pipelining for scheduling multi- dimensional systems via rotation. In *Proceedings of the 31st Design Automation Conference* .485–490.

PAULIN, P. G. AND KNIGHT, J. P. 1989. Force-directed scheduling for the behavioral synthesis of ASICs. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 8*, 661–679.

RAMAMRITHAM, K., STANKOVIC, J. A., AND SHIAH, P. -F. 1990. Efficient scheduling algorithms for real-time multiprocessor systems. *IEEE Trans. Parall. Distrib. Syst*. 1, 184–194.

SHAO, Z., ZHUGE, Q., XUE, C., AND SHA, E. H. -M. 2005. Efficient assignment and scheduling for heterogeneous dsp systems. *IEEE Trans. Parall. Distrib. Syst. 16*, 516–525.

SHATZ, S. M., WANG, J. -P., AND GOTO, M. 1992. Task allocation for maximizing reliability of distributed computer systems. *IEEE Trans. Computers 41*, 1156–1168.

SRINIVASAN, S. AND JHA, N. K. 1999. Safety and reliability driven task allocation in distributed systems. *IEEE Trans. Parall. Distrib. Syst. 10*, 238–251.

TIA, T., DENG, Z., SHANKAR, M., STORCH, M., SUN, J., WU, L.-C., AND LIU, J. -S. 1995. Prob- abilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*. 164–173.

TONGSIMA, S., SHA, E. H. -M., CHANTRAPORNCHAI, C., SURMA, D., AND PASSOSE, N. 2000. Probabilistic loop scheduling for applications with uncertain execution time. *IEEE Trans. Computers 49*, 65–80.

VAHID, F. AND GIVARGIS, T. 2002. *Embedded System Design, A Unified Hardware/Software Introduction*. Wiley, New York.

WANG, C. -Y. AND PARHI, K. K. 1995. High-level synthesis using concurrent transformations, scheduling, and allocation. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 14*, 274–295.
WOLF, W. 2006. Design challenges in multiprocessor Systems-On-Chip. In *From Model-Driven Design to Resource Management for Distributed Embedded Systems*, B. Kleinjohann et al. Eds., vol. 225, Springer, 1-8.
WOLFE, M. E. 1996. *High Performance Compilers for Parallel Computing*. Addison-Wesley.
ZADEH, L. A. 1996. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets Syst. 1*, 3–28.
ZHOU, T., HU, X., AND SHA, E. H. -M. 2001. Estimating probabilistic timing performance for real-time embedded systems. *IEEE Trans. VLSI Syst. 9*, 6, 833–844.

Received May 2008; revised November 2008; accepted December 2008