Efficient Assignment with Guaranteed Probability for Heterogeneous Parallel DSP *

Meikang Qiu[1] Chun Xue[1] Zili Shao[2] Qingfeng Zhuge[1] Meilin Liu[1] Edwin H.-M. Sha[1]
[1]Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75083, USA
email: {mxq012100, cxx016000,
qfzhuge, mxl024100, edsha}@utdallas.edu

[2]Department of Computing
Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
email: cszshao@comp.polyu.edu.hk

Abstract

In real-time digital signal processing (DSP) architectures using heterogeneous functional units (FUs), it is critical to select the best FU for each task. However, some tasks may not have fixed execution times. This paper models each varied execution time as a probabilistic random variable and solves h*eterogeneous 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. Two algorithms, one is optimal and the other is heuristic, are proposed to solve the general problem. The experiments show that our algorithms can effectively reduce the total cost with guaranteed confidence probabilities satisfying timing constraints. For example, our algorithms achieve an average reduction of 33.5% on total cost with 90% confidence probability satisfying timing constraints compared with the previous work using worst-case scenario.*

1 Introduction

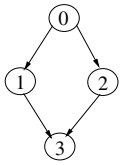
In DSP applications, some tasks may not have fixed execution time. Such tasks usually contain conditional instructions and/or operations that could have different execution times for different inputs [9]. Although many static assignment techniques can thoroughly check for the best assignment for dependent 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 the real-time systems and may result in an inefficient task assignment. Although the problem becomes very complicate when using probabilistic approach, we can obtain the solution that not only cover the results for hard real-time systems, but also provide more choices of smaller total costs with guaranteed confidence probabilities satisfying timing constraints.

In heterogeneous parallel DSP systems, same type of operations can be processed by heterogeneous FUs with different costs, which may relate to power, reliability, etc. [5, 7]. Therefore, 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 with a guaranteed confidence probability satisfying timing constraints.

This paper presents assignment and optimization 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 [10]. For heterogeneous systems, each FU has different cost, representing hardware cost, size, reliability, etc.. Faster one has higher cost while slower one has lower cost. This paper shows how to assign a proper FU type to each node of a *Data Flow Graph* (DFG) such that the total cost is minimized while the timing constraint is satisfied with a guaranteed confidence probability. With confidence probability P, we can guarantee that the total execution time of the DFG is less than or equal to the timing constraint with a probability that is greater than or equal to P.

We show an example to illustrate the HAP problem. Assume that we can select FUs from a FU library that provides two types of FUs: R_1, R_2. An exemplary DFG is shown in Figure 1(a), which is a *directed acyclic graph* (DAG) with 4 nodes. Node 0 is a multi-child node, which has two children: 1 and 2. Node 3 is a multi-parent node, and has two parents: 1 and 2. The execution times (T), probabilities (P), and costs (C) of each node for different FU types are shown in Figure 1(b). 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; and the probabilities may come from the statistical profiling. For example, node 0 can choose one of the two types: R_1 or R_2. When choosing R_1,

*This work is partially supported by TI University Program, NSF EIA-0103709, Texas ARP 009741-0028-2001, NSF CCR-0309461, NSF IIS-0513669, and Microsoft, USA.



(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	0.8	6	3	0.6	2
	4	0.2	6	5	0.4	2
3	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	0.8	6	3	0.6	2
	4	1.0	6	5	1.0	2
3	1	0.9	9	2	0.8	5
	4	1.0	9	6	1.0	5

(c)

Figure 1. (a) A given DAG. (b) The times, 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

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 to be 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 $T \leq t$. 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 11 can be found as follows: We assign FU types 2, 2, 2, and 1 for nodes 0, 1, 2, and 3, respectively. Let T_0, T_1, T_2, and T_3 be the random variables representing the execution times of nodes 0, 1, 2, and 3. From Figure 1(c), we get: $Pr(T_0 \leq 4) = 1.0$, $Pr(T_1 \leq 5) = 1.0$, $Pr(T_2 \leq 5) = 1.0$, and $Pr(T_3 \leq 1) = 0.9$. Hence, we obtain minimum total cost 18 with 0.9 probability satisfying the timing constraint 11. 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.

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. The hard HA problem is the worst-case scenario of the *heterogeneous assignment with probability* (HAP) problem, and is easy to compute. 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 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. Solving HAP problem is a complicate task, but the result we achieved is deserve the efforts we spend for it.

For example, in Figure 1, under timing constraint 11, we cannot find a solution to the hard HA problem. But we can obtain minimum system cost 18 with probability 0.9 satisfying the timing constraint 11. Also, the cost obtained from 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 33 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 18, which gives 45.5% improvement.

It is known that the hard HA problem is NP-complete [7]. Since the HAP problem is NP harder than the hard HA problem, the HAP problem is also NP-complete. In this paper, we propose two algorithms, one is heuristic, one is optimal, to solve the general problem.

There have been a lot of research efforts on allocating applications in heterogeneous distributed systems [1]. Incorporating reliability cost into heterogeneous distributed systems, the reliability driven assignment problem has been studied in [3, 4]. In these work, allocation are performed based on a fixed architecture. However, when performing assignment and scheduling 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 only use homogeneous FUs, that is, same type of operations will be processed by same type of FUs [2, 5, 6].

Our work is related to the work in [7, 9]. *Probabilistic retiming* (PR) had been proposed in [9]. But PR does not model the hard HA problem which focuses on how to obtain the best assignment from different FU types. In [7], the authors 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 do not consider varied execution time situation. Also, for the general problem, their solutions are not optimal. In this paper, the algorithms solving the hard HA problem are called *hard HA algorithms* in general. When considering probabilistic approach, we find the problem become very difficult. But the efficient algorithms we propose here can improve the results significantly.

Our contributions are listed as the following: **First**, when the given DFG is a DAG, our optimal algorithm, *DAG_Opt*, gives the optimal solution and covers the results of the *hard HA algorithms* using worst-case scenario of our algorithms. Our heuristic algorithm, *DAG_Heu*, gives near optimal solutions efficiently. **Second**, our algorithms are able to give solutions and provide more choices of smaller total cost with guaranteed confidence probabilities satisfying timing constraints. While the *hard HA algorithms* may not find solution with certain timing constraints. **Third**, our algorithms are practical and quick. In practice, when the number of multi-parent nodes and multi-child nodes in the given input graph is small, and the timing constraint is polynomial to the size of DFG, our algorithms become polynomial. The running times of these algorithms are very small and our experiments always finished in very short time.

We conduct experiments on a set of benchmarks, and compare our algorithms with the *hard HA algorithms*. Experiments show that the results of our algorithms has an average 33.5% improvement on cost reduction with confidence probability 0.9 satisfying timing constraints compared with the results to the hard HA problem. With 0.8 confidence probability satisfying timing constraints, the average improvement is 43.0% and with 0.7 confidence probability satisfying timing constraints, the improvement is 46.6%.

The remainder of this paper is organized as follows: In the next section, we give the basic definitions and models used in the rest of the paper. Example of the HAP problem when the input is a DAG is given in Section 3, The algorithms for 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 *Data-Flow Graph (DFG)* to model a DSP application. A ***DFG G*** $= \langle V, E \rangle$ is a *directed acyclic graph* (DAG), where $V = \langle v_1, v_2, \cdots, v_N \rangle$ is the set of nodes, $E \subseteq V \times V$ 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 R=$\{R_1, R_2, \cdots, R_M\}$. An assignment for a DFG 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, 2, and 1 for nodes 0, 1, 2, and 3, respectively, we obtain minimum total cost 18 with 0.9 probability satisfying the timing constraint 11. That is, $A(0) = 2$, $A(1) = 2$, $A(2) = 2$, and $A(3) = 1$.

In a DFG G, each varied execution time 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 $C_1(0) = 10$; $C_2(0) = 4$.

We define the ***heterogeneous assignment with probability (HAP)*** problem as follows: Given M different FU types: R_1,R_2,$\cdots$,R_M, a DFG $G = \langle V, E \rangle$ 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 under timing constraint 11 can be found as follows. Assigning FU types 2, 2, 2, and 1 for nodes 0, 1, 2, and 3, respectively, we obtain minimum total cost 18 with 0.9 probability under the timing constraint 11.

3 Example

In this section, we continue the example of Figure 1 to illustrate the *heterogeneous assignment probability* (HAP) problem. In Figure 1(a), each node has two different FU types to choose from, and is executed on them with probabilistic times. In DSP applications, a real-time system does not always has hard deadline time. 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 under timing constraints.

T	(P , C)	(P , C)	(P , C)	(P , C)	(P , C)
3	0.52, 33				
4	0.40, 27	0.46, 29	0.52, 33		
5	0.36, 23	0.40, 27	0.46, 29	0.58, 33	
6	0.26, 20	0.30, 22	0.36, 23	0.58, 27	0.81 33
7	0.20, 14	0.27, 18	0.38, 22	0.51, 23	0.81, 24
8	0.20, 14	0.63, 18	0.72, 20	0.81, 24	0.90, 33
9	0.56, 14	0.63, 18	0.72, 20	0.90, 24	
10	0.56, 14	0.90, 18			
11	0.80, 14	0.90, 18	**1.00, 33**		
12	0.80, 14	0.90, 18	**1.00, 24**		
13	0.80, 14	**1.00, 18**			
14	0.80, 14	**1.00, 20**			
15	**1.00, 14**				

Table 1. Minimum total costs with computed confidence probabilities under various timing constraints for a DAG

Different FU types have different costs, which can be any cost such as hardware cost, energy consumption or reliability cost. A node may run slower but with less energy consumption or reliability cost when executed on one type of FUs than on another [7]. But for the same FU type, a node may have varied execution times while there is not too much difference in cost. In this paper, 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 energy cost of each node. Also, when the cost is related to reliability, the total reliability cost is the summation of reliability cost of all nodes. We compute the reliability cost using the same model as in [8]. From the conclusion of papers [8], 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 reliability costs of all nodes is minimized.

For this DAG with four 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 1. The entries with probability that is 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 (0.52, 33), we can

achieve minimum total cost 33 with confidence probability 0.52 under timing constraint 3.

Assign	0	2	4	1.0000	4
Assign	1	2	5	1.0000	3
Assign	2	2	5	1.0000	2
Assign	3	1	1	0.9000	9
Total			18	0.9000	18
Assign	0	1	3	1.0000	10
Assign	1	1	4	1.0000	8
Assign	2	1	4	1.0000	6
Assign	3	1	4	1.0000	9
Total			11	1.0000	33

Table 2. With timing constraint 11, the assignments of types for each node with different (Probability, Cost) pairs.

Comparing with optimal results of the hard HA problem for a DFG using worst-case scenario, Table 1 provides more information and more selections, no matter whether the system is hard or soft real-time. In Table 1, we have the output of our algorithm from timing constraint 3 to 15, while the optimal results of the hard HA problem for a DAG only has 5 entries (in boldface) from timing constraint 11 to 15.

For a soft real-time system, some nodes of DFG 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, 18) which guarantees to achieve minimum cost 18 with 90 % confidence satisfying the timing constraint 11. It achieves 45.5% reduction in cost compared with the cost 33, 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 also can select the pair (0.80, 14) which has provable confidence probability 0.8 satisfying the timing constraint 11, while the cost 14 is only 42.4% of the cost of worst-case, 33. The assignments for each pair of (0.9,18) and (1.0,33) under the timing constraint 11 are shown in Table 2.

From above the example, we can see that the probabilistic approach to heterogeneous assignment problem has great advantages: It provides the possibility to reduce the total cost of systems with guaranteed confidence probabilities 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 solve the general case of the HAP problem, that is, the given input is a *directed acyclic graph* (DAG).

4.1 Definitions and Lemma

To solve the HAP problem, we use dynamic programming method traveling the graph in bottom up fashion. For the easiness of explanation, we will index the nodes based on bottom up sequence. Define a *root* node to be a node without any parent and a *leaf* node to be a node without any child. In multi-child case, we use bottom up approach; And in multi-parent case, we use top down approach.

Given the timing constraint L, a DFG G, and an assignment A, we first give several definitions as follows: 1).$\mathbf{G^i}$: The sub-graph 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 sub-graph. 2).$\mathbf{C_A}(\mathbf{G^i})$ and $\mathbf{T_A}(\mathbf{G^i})$: The total cost and total execution time of G^i under the assignment A. In our algorithm, each step will achieve the minimum total cost of G^i with computed confidence probabilities under various timing constraints. 3).In our algorithm, table $D_{i,j}$ will be built. Each entry of table $D_{i,j}$ will store a link list of (Probability, Cost) pairs sorted by probability in ascending order. Here we define the **(Probability, Cost) pair ($\mathbf{C_{i,j}}$, $\mathbf{P_{i,j}}$)** 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 $\geq P_{i,j}$.

We introduce the ***operator* "$\oplus$"** in this paper. 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 as "$\mathbf{H_1} \oplus \mathbf{H_2}$".

In our algorithm, $D_{i,j}$ is the table in which each entry has a link list that store 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 as following:

Algorithm 4.1 redundant-pair removal algorithm

Input: A list of ($P^k_{i,j}$, $C^k_{i,j}$)
Output: 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 ($P^{k+1}_{i,j}$, $C^{k+1}_{i,j}$) **do**
4: **if** $P^k_{i,j} = P^{k+1}_{i,j}$ **then**
5: **if** $C^k_{i,j} \geq C^{k+1}_{i,j}$ **then**
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** $P^k_{i,j} \geq P^{k+1}_{i,j}$ **then**
12: cancel the pair ($P^{k+1}_{i,j}$, $C^{k+1}_{i,j}$)
13: **end if**
14: **end if**
15: **end for**

For example, we have a list with pairs (0.1, 2) → (0.3, 3) → (0.5, 3) → (0.3, 4), we do the redundant-pair removal as following: First, sort the list according $P_{i,j}$ in an ascending order. This list becomes to (0.1, 2) → (0.3, 3) → (0.3, 4) → (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) since $0.3 < 0.5$ while $3 \geq 3$. There is no information lost in redundant-pair removal

In summary, we can use the following Lemma to cancel redundant pairs.

Lemma 4.1 *Given ($P^1_{i,j}$, $C^1_{i,j}$) and ($P^2_{i,j}$, $C^2_{i,j}$) in the same list:*

1. *1. If $P^1_{i,j} = P^2_{i,j}$, then the pair with minimum $C_{i,j}$ is selected to be kept.*
2. *2. If $P^1_{i,j} < P^2_{i,j}$ and $C^1_{i,j} \geq C^2_{i,j}$, then $C^2_{i,j}$ is selected to be kept.*

Using Lemma 4.1, we can cancel many 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 of ($P_{i,j}$, $C_{i,j}$) has the following properties:

Lemma 4.2 *For any ($P^1_{i,j}$, $C^1_{i,j}$) and ($P^2_{i,j}$, $C^2_{i,j}$) in the same list:*

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

Since the link list is in an ascending order by probabilities, if $P^1_{i,j} < P^2_{i,j}$, while $C^1_{i,j} \geq C^2_{i,j}$, based on the definitions, we can guarantee with $P^1_{i,j}$ to find smaller energy consumption $C^2_{i,j}$. Hence ($P^2_{i,j}$, $C^2_{i,j}$) has already covered ($P^1_{i,j}$, $C^1_{i,j}$). We can cancel the pair ($P^1_{i,j}$, $C^1_{i,j}$). For example, we have two pairs: (0.1, 3) and (0.6, 2). Since (0.6, 2) is better than (0.1, 3), in other words, (0.6, 2) covers (0.1, 3), we can cancel (0.1, 3) and will not lose useful information. We can prove the vice versa is true similarly. When $P^1_{i,j} = P^2_{i,j}$, smaller C is selected.

In every step in our algorithm, one more node will be included for consideration. The information of this node is stored in local table $E_{i,j}$, which is similar to table $D_{i,j}$. A local table only store information, such as probabilities and costs, of a node itself. Table $E_{i,j}$ is the local table only storing the information of node v_i. In more detail, $E_{i,j}$ is a local table of link lists that store 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 $E_{i,j}$ are as follows. First, sort the execution time variations in an ascending order. Then, accumulate the probabilities of same type. Finally, let $L_{i,j}$ be the link list in each entry of $E_{i,j}$, insert $L_{i,j}$ into $L_{i,j+1}$ while redundant pairs canceled out based on Lemma 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 3 after the insertion.

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

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

Table 3. An example of local table, $E_{0,j}$

4.2 A Heuristic Algorithm for DAG

The general problem of the HAP problem is NP-complete problem, we propose a near optimal heuristic algorithm in this subsection and a optimial one in next subsection. In many cases, the near optimal heuristic algorithm will give us the same results as the results of the optimal algorithm. The optimal algorithm is suitable for the cases when the given DFG has small number of multi-parent and multi-child nodes.

The DAG_Heu Algorithm

Input: M different types of FUs, a DAG, and the timing constraint L.

Output: An heuristic assignment for the DAG

1. Topological sort all the nodes, and get a sequence A. Count the number of multi-parent nodes p and the number of multi-child nodes q.
   if $p < q$ **then**
   use bottom up approach
   else
   use top down approach
   end if
2. For bottom up approach, use the following algorithm. For top down approach, just reverse the sequence. Assume the sequence after topological sorting is $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_N$, in bottom up fashion. Let $D_{1,j} = E_{1,j}$. Assume $D^{'}_{i,j}$ is the table that stored minimum total cost with computed confidence probabilities under the timing constraint j for the sub-graph rooted on v_i except v_i. Nodes $v_{i_1}, v_{i_2}, \cdots, v_{i_R}$ are all child nodes of node v_i and R is the number of child nodes of node v_i, then

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

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

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

   //In equation (1), $D_{i_1,j} \oplus D_{i_2,j}$ is computed as follows. let $G^{'}$ be 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 a in $G^{'}$ appears for the first time, we add the cost of a and multiply the probability of a to $D^{'}_{i,j}$. If a

appears more than once, that is, a is a common node, then use a selection function to choose the type of a. For instance, the selection function can be defined as selecting the type that has a smaller execution time. Therefore, we can compute $D_{i,j}^{'}$ by using "$\oplus$" on all child nodes of node v_i when $R \geq 1$. //

3. Return $D_{N,j}$.

In algorithm *DAG_Heu*, if using bottom up approach, for each sequence node, use the simple path algorithm to get the dynamic table of parent node. If using top down approach, reverse the sequence and use the same algorithm. For example, in Figure 1(a), there are one multi-child nodes: node 0, and there are also one multi-parent nodes, that is, node 3. Hence, we can use either approach.

In algorithm *DAG_Heu*, we have to solve the problem of common nodes, that is, one node appears in two or more graphs that rooted at the child nodes of node v_i. In equation (1), 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. 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 as the type that has a smaller execution time.

Due to the problem of common nodes, algorithm *DAG_Heu* is not an optimal algorithm. The reason is that an assignment conflict for a common node maybe exist, while algorithm *DAG_Heu* cannot solve this problem. For example, in Figure 1, using bottom up approach, there will be two paths from node 3 to node 0. Path a is $3 \rightarrow 1 \rightarrow 0$, and path b is $3 \rightarrow 2 \rightarrow 0$. Hence, node 3 is a common node for both paths while node 0 is the root. It is possible, under a timing constraint, the best assignment for path a gives node 3 assignment as FU type 1, while the best assignment for path b gives node 3 assignment as FU type 2. This kind of assignment conflict can not 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 step 2 of the algorithm *DAG_heu*. Using these 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. Usually, the execution time of each node is upper bounded by a constant. Then L equals $O(|V|^c)$ (c is a constant). In this case, *DAG_heu* is polynomial.

4.3 An Optimal Algorithm for DAG

In this subsection, we give the optimal algorithm (*DAG_Opt*) for the HAP problem when the given DFG is a DAG. In *DAG_Opt*, we exhaust all the possible assignments of multi-parent or multi-child nodes. Without loss of generality, assume we using bottom up approach. If the total number of nodes with multi-parent is t, and there are maximum K variations for the execution times of all nodes, then we will give each of these t nodes a fixed assignment. We will exhausted 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.

The DAG_Opt Algorithm

Input: M different types of FUs, a DAG, and the timing constraint L.

Output: An optimal assignment for the DAG

1. Topological sort all the nodes, and get a sequence A. Count the number of multi-parent nodes p and the number of multi-child nodes q.
   if $p < q$ **then**
   use bottom up approach
   else
   use top down approach
   end if

2. For bottom up approach, use the following algorithm. For top down approach, just reverse the sequence. If the total number of nodes with multi-parent is t, and there are maximum K variations for the execution times of all nodes, then we will give each of these t nodes a fixed assignment. For each of the K^t possible fixed assignments, we do as follows.

3. Assume the sequence after topological sorting is $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_N$, in bottom up fashion. Let $D_{1,j} = E_{1,j}$. Assume $D_{i,j}^{'}$ is the table that stored minimum total cost with computed confidence probabilities under the timing constraint j for the sub-graph rooted on v_i except v_i. Nodes $v_{i_1}, v_{i_2}, \cdots, v_{i_R}$ are all child nodes of node v_i and R is the number of child nodes of node v_i, then

$$D_{i,j}^{'} = \begin{cases} (0,0) & \text{if } R = 0 \\ D_{i_1,j} & \text{if } R = 1 \\ D_{i_1,j} \oplus \cdots \oplus D_{i_R,j} & \text{if } R \geq 1 \end{cases} \quad (3)$$

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

$$D_{i,j} = D_{i,j-k}^{'} \oplus E_{i,k} \quad (4)$$

   //In equation (3), $D_{i_1,j} \oplus D_{i_2,j}$ is computed as follows. let $G^{'}$ be 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}. For each node a in $G^{'}$, we add the cost of a and multiply the probability of a to $D_{i,j}^{'}$ for only once, because each node can only have one assignment and there is no assignment conflict.//

4. For each possible fixed assignment, we get a $D_{N,j}$. Merge the (Probability, Cost) pairs in all the possible $D_{N,j}$ together, and sort them in ascending sequence

according probability. Then use the Lemma 4.1 to remove redundant pairs. The final $D_{N,j}$ we get is the table in which each entry has the minimum cost with a guaranteed confidence probability under the timing constraint j.

Algorithm *DAG_Opt* gives the optimal solution when the given DFG is a DAG. In the following, we give the Theorem 4.1 and Theorem 4.2 about this. Due to the space limitation, we will not give proofs for them in this paper.

Theorem 4.1 *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.*

Theorem 4.2 *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.*

According to Theorem 4.1, in each possible fixed assignment, for each pair ($P_{i,j}$, $C_{i,j}$) in $D_{i,j}$ 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) of the algorithm *DAG_Opt*, we try all the possible fixed assignments, combine them together into a new row $D_{N,j}$ in dynamic table, and remove redundant pairs using the Lemma 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 time. The complexity of *Algorithm DAG_Opt* is $O(K^t * |V|^2 * L * M * K)$, where t is the total number of nodes with multi-parent (or multi-child) in 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 variation for each node. Algorithm *DAG_Opt* is exponential, hence it can not be applied to a graph with large amounts of multi-parent and multi-child nodes.

In this example of Figure 1, the algorithm *DAG_Heu* gives the same results as those of the algorithm *DAG_Opt*. Actually, experiments shown that although algorithm *DAG_Heu* is only near-optimal, it can give same results as those given by the optimal algorithm in most cases.

5 Experiments

This section presents the experimental results of our algorithms. We conduct experiments on a set of benchmarks including voltera filter, 4-stage lattice filter, 8-stage lattice filter, differential equation solver, RLS-languerre lattice filter, and elliptic filter. Among them, the DFG for first three filters are trees and those for the others are DAGs. The basic information about the benchmarks is shown in Table 4. Three different FU types, R_1, R_2, and R_3, are used in the system, in which a FU with type R_1 is the quickest with the highest cost and a FU with type R_3 is the slowest with the lowest cost. The execution times, probabilities, and costs for each node are randomly assigned. For each benchmark, the first timing constraint we use is the minimum execution time. The experiments are performed on a Dell PC with a P4 2.1 G processor and 512 MB memory running Red Hat Linux 7.3.

Benchmarks	DFG	# 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 4. The basic information for the Benchmarks

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	

Table 5. The minimum total costs with computed confidence probabilities under various timing constraints for voltera filter.

The experiments on voltera filter, 4-stage lattice filter, and 8-stage lattice filter are finished in less than one second, in which we compare our algorithms with the *hard HA algorithms*. The experimental results for voltera filter are shown in Table 5. In each table, column "TC" represents the given timing constraint. The minimum total costs obtained from different algorithms: *DAG_Opt* and the optimal *hard HA algorithms* [7], are presented in each entry. Columns "1.0", "0.9", "0.8", and "0.7" represent that the confidence probability is 1.0, 0.9, 0.8, and 0.7, respectively. Algorithm *DAG_Opt* covers all the probability columns, while the optimal *hard HA algorithm* [7] only include the column "1.0", which is in boldface. For example, in Table 5, under the timing constraint 100, the entry under "1.0" is 7827, which is the minimum total cost for the hard HA problem. The entry under "0.9" is 6047, which means we can achieve minimum total cost 6047 with confidence probability 0.9 under timing constraints. From the information provided in the structure of the link list in each entry of the dynamic table, we are able to trace how to get the satisfied assignment.

Column "%" shows the percentage of reduction on the total cost, compared the results of algorithm with those ob-

tained by the optimal *hard HA algorithms* [7]. The average percentage reduction is shown in the last row "Ave. Redu(%)" of all Tables 5. The entry with "×" means no solution available. In Table 5, under timing constraint 80, the optimal *hard HA algorithms* can not find a solution. However, we can find solution 7847 with probability 0.9 that guarantees the total execution time of the DFG is less than or equal to the timing constraint 80.

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	

Table 6. The minimum total costs with computed confidence probabilities under various timing constraints for rls-laguerre filter.

We also conduct experiments on RLS-Laguerre filter, elliptic filter, and Different equation solver , which are DAGs. RLS-Laguerre filter has 3 multi-child nodes and 6 multi-parent nodes. Using top-down approach, we implemented all $3^3 = 27$ possibilities. Elliptic filter has 5 multi-child nodes and 8 multi-parent nodes. There are total $3^5 = 243$ possibilities by top-down approach. Table 6 shows the experimental results for RLS-Laguerre filter. Column "%" shows the percentage of reduction on system cost, compared the results for soft real-time with those for hard real-time. The average percentage reduction is shown in the last row "Ave. Redu(%)" of all these two tables. The entry with "×" means no solution available. Under timing constraint 70 in Table 6, there is no solution for hard real-time. However, we can find solution 7787 with probability 0.9 that guarantees the total execution time of the DFG is less than or equal to the timing constraint 70.

The experimental results show that our algorithms can greatly reduce the total cost while have a guaranteed confidence probability satisfying timing constraints. On average, algorithm *DAG_Opt* gives a cost reduction of 33.5% with confidence probability 0.9 under timing constraints, and a cost reduction of 43.0% and 46.6% with 0.8 and 0.7 confidence probabilities satisfying timing constraints, respectively. 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. Algorithm *DAG_Heu* gives good results in most cases.

The advantages of our algorithms over the *hard HA algorithms* are summarized as follows. First, our algorithms are efficient and provide overview of all possible variations of minimum costs comparing with the the worst-case scenario generated by the *hard HA algorithms*. Second, it is possible to greatly reduce the system total cost while have a very high confidence probability under different timing constraints. Finally, our algorithms are very quick and practical.

6 Conclusion

This paper proposed a probability approach for real-time digital signal processing (DSP) applications to assign and optimize special purpose architectures using heterogeneous functional units with probabilistic execution time. The systems become very complicate when considering probability in execution time. For the *heterogeneous assignment with probability* (HAP) problem, Two algorithms, one is optimal and the other is heuristic, were proposed to solve the general problem. Experiments showed that our algorithms provide more design choices to achieve minimum total cost while the timing constraint is satisfied with a guaranteed confidence probability. Our algorithms are useful for both hard and soft real-time systems.

References

[1] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Trans. Parallel Distributed Systems*, 15(4):319–330, 2004.

[2] Y.-N. Chang, C.-Y. Wang, and K. K. Parhi. Loop-list scheduling for heterogeneous functional units. In *6th Great Lakes Symposium on VLSI*, pages 2–7, Mar. 1996.

[3] A. Dogan and F. Özgüner. Matching and scheduling algorithms for minimizing execution time and failure probability of applications in heterogeneous computing. *IEEE Trans. on Parallel and Distributed Systems*, 13:308–323, Mar. 2002.

[4] Y. He, Z. Shao, B. Xiao, Q. Zhuge, and E. H.-M. Sha. Reliability driven task scheduling for tightly coupled heterogeneous systems. In *Proc. of IASTED International Conference on Parallel and Distributed Computing and Systems*, Nov. 2003.

[5] K. Ito, L. Lucke, and K. Parhi. Ilp-based cost-optimal dsp synthesis with module selection and data format conversion. *IEEE Trans. on VLSI Systems*, 6:582–594, Dec. 1998.

[6] W. N. Li, A. Lim, P. Agarwal, and S. Sahni. On the circuit implementation problem. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 12:1147–1156, Aug. 1993.

[7] Z. Shao, Q. Zhuge, C. Xue, and E. H.-M. Sha. Efficient assignment and scheduling for heterogeneous dsp systems. *IEEE Trans. on Parallel and Distributed Systems*, 16:516–525, Jun. 2005.

[8] S. Srinivasan and N. K. Jha. Safety and reliability driven task allocation in distributed systems. *IEEE Trans. on Parallel and Distributed Systems*, 10:238–251, Mar. 1999.

[9] S. Tongsima, E. H.-M. Sha, C. Chantrapornchai, D. Surma, and N. Passose. Probabilistic loop scheduling for applications with uncertain execution time. *IEEE Trans. on Computers*, 49:65–80, Jan. 2000.

[10] T. Zhou, X. Hu, and E. H.-M. Sha. Estimating probabilistic timing performance for real-time embedded systems. *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, 9(6):833–844, Dec. 2001.