Proceedings of the 17th IASTED International Conference
Parallel and Distributed Computing and Systems
November 14-16, 2005, Phoenix, AZ, USA

OPTIMAL ASSIGNMENT WITH GUARANTEED CONFIDENCE PROBABILITY FOR TREES ON HETEROGENEOUS DSP SYSTEMS

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

Zili Shao
Department of Computing
Hong kong Polytechnic University
Hung Hom, Kowloon, Hongkong
email: cszshao@comp.polyu.edu.hk

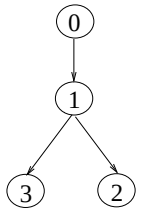
ABSTRACT
In real-time digital signal processing (DSP) architectures using heterogeneous functional units (FUs), it is critical to select the best FU for each task. However, some tasks may not have fixed execution times. This paper models each varied execution time as a probabilistic random variable and solves *heterogeneous assignment with probability* (HAP) problem. The solutions to the HAP problem are useful for both hard real time and soft real time systems. We propose optimal algorithms for the HAP problem when the input is a tree or a simple path. The experiments show that our algorithms can effectively obtain the optimal solutions to simple paths and trees. For example, with our algorithms, we can obtain an average reduction of 32.5% on total cost with 90% confidence probability compared with the previous work using worst-case scenario.

KEY WORDS
Assignment, probability, heterogeneous, real time, DSP, synthesis

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 [1]. In heterogeneous parallel DSP system, same type of operations can be processed by heterogeneous FUs with different costs, which may relate to power, reliability, etc. [2]. 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 [3].

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. For heterogeneous systems, each FU has different cost. 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. A confidence probability P means that we can guarantee that the total execution time of the DFG is less than or equal to the timing constraint with a probability greater than or equal to P.



(a)

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

(b)

Figure 1. (a) A given tree. (b) The times, probabilities, and costs of its nodes for different FU types.

Here 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 tree with 4 nodes. The execution times (T), probabilities (P), and costs (C) of each node for different FU types are shown in Figure 1(b). Each node has two FU types to choose from, and execute each type of FU with probabilistic execution times. Here, execution time (T) is a random variable. For example, node 1 can choose one of the two types: R_1 or R_2. When choosing R_1, node 1 has the probability of 0.9 to be finished in 1 time unit and the probability of 0.1 to be finished in 3 time units. In other words, node 1 can guarantee to finish the task in 3 time units with 100% probability. Hence, we care about the time *cumulative distribution function* (CDF) $F(t)$, which gives accumulated probability for $T \leq t$.

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 to 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. For example, in the hard HA problem, when choosing type R_1, node 1 has only one execution time 3. When choosing type R_2, node 0 has one execution time 4. With certain timing constraint, there is no solution for the hard HA problem. However, for soft real time applications, it is desirable to find assignment that guarantees the total execution time

to be less than or equal to timing constraint. Also, the cost obtained from hard HA algorithms is always larger than or equal to the cost from HAP algorithms. Based on [3], we already know the NP-completeness of the hard HA problem. Since the HAP problem is NP harder than the hard HA problem, the HAP problem is NP-complete too.

There have been a lot of research efforts on allocating applications in heterogeneous distributed systems [4]. Incorporating reliability cost into heterogeneous distributed systems, the reliability driven assignment problem has been studied in [5]. 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]. Our work is related to the work in [1, 3]. *Probabilistic retiming* (PR) had been proposed by Tongsima et al. in [1]. But PR does not model the hard HA problem, which is focused on how to obtain the best assignment from several different function units. Shao et al. [3] propose several efficient and optimal algorithms for the hard HA problem. But they do not consider varied execution time situation. In this paper, the optimal algorithms solving the hard HA problem [5] are called the *optimal hard HA algorithms* in general.

Our contributions are listed as the following: First, the results of our algorithms, *Path_Assign* and *Tree_Assign*, cover the results of the *optimal hard HA algorithms*, which is equivalent to the results using worst-case scenario of our algorithms. Second, our algorithms are able to give solutions and provide more choices with guaranteed confidence probabilities and smaller total costs. Yet, the *optimal hard HA algorithms* may not find solution with certain timing constraints. Third, our algorithms is practical and quick. In practice, when the timing constraint is a polynomial to the size of DFG, our algorithms become polynomial. The running time of these algorithm is very small and our experiments always finished in seconds.

We conduct experiments on a set of benchmarks, and compare our algorithms with the *optimal hard HA algorithms*. Experiments show that the results of our algorithms has an average 32.5% improvement with 0.9 confidence probability compared with the results to the hard HA problem. With 0.8 confidence probability, the average improvement is 38.2% and with 0.7 the improvement is 40.6%.

The remainder of this paper is organized as follows: In the next section, examples of the HAP problem when the input is a tree are given. In Section 3, we give the basic definitions and models used in the rest of the paper. 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 Examples

For the example in Figure 1(a), each node has two FU types for choose from and will execute 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 better minimum cost with provable confidences.

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**					

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

For this tree with four nodes, we can obtain the minimum total cost with different timing constraints and different confidence probability levels. The minimum cost table generated with our algorithms is shown in Table 1. 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.

Comparing with the optimal hard HA algorithms using worst-case scenario, Table 1 provides more information, more selections and decisions, 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 12, yet the output of *optimal hard HA algorithms* only has 5 entries, which are in boldface, from timing constraint 8 to 12. We don't know other informations.

	Node id	Type id	T	Prob.	Cost
Assign	0	2	2	1.0000	5
Assign	1	2	4	1.0000	4
Assign	2	2	2	0.8000	5
Assign	3	1	2	1.0000	8
Total			8	0.8000	22
Assign	0	2	2	1.0000	5
Assign	1	2	4	1.0000	4
Assign	2	1	1	0.9000	9
Assign	3	1	2	1.0000	8
Total			8	0.9000	26
Assign	0	1	1	1.0000	9
Assign	1	1	3	1.0000	10
Assign	2	1	4	1.0000	9
Assign	3	1	4	1.0000	8
Total			8	1.0000	36

Table 2. With timing constraint 8, the assignments of types for each node with different (Probability, Cost) pairs.

For a soft real time system, some nodes of DFG have smaller probabilistic execution times comparing with the hard deadline time. We can achieve much smaller cost than the cost of worst-case with guaranteed confidence probability. For example, with timing constraint 8, we can select

the pair (0.90, 26), which guarantee us to achieve minimum cost 26 with 90 % confidence, which is 27.8% less than the cost of worst-case, 36. In many situations, this is good enough for user to adopt this assignment. And we also can select the pair (0.80, 22), which has provable confidence probability 0.8, yet the cost is only 38.9% of the cost of worst-case, 36. The assignments for each pair of (0.8,22), (0.9,26), and (1.0,36) with timing constraint 8 are shown in Table 2.

T	**7, 8**	**9**	**10, 11**	**12**
(P , C)	(0.17, 17)	(0.24, 17)	(0.70, 17)	(1.00, 17)

Table 3. Given an assignment for the tree, the (Probability, Cost) pairs at different times.

Given an assignment, we can get a minimum total cost with every timing constraint. But the probability to achieve this minimum total cost may not same. For example, if the assignments for node 0, 1, 2, and 3 are all type 2, then the total cost is 17. But the probabilities vary from 0.17 to 1.00. The probabilities at different time are shown in Table 3.

3 System Model

Data-Flow Graph (DFG) is broadly used 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 $R=\{R_1, R_2, \cdots, R_M\}$. For each FU type, there are maximum K execution time variations, although each node may have different number of FU types and execution time variations. 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 selected type of node v. In a DFG G, each varied execution time is modeled as a probabilistic random variable, $\mathbf{T_{R_j}(v)}$, $1 \le j \le M$, representing the execution times of each node $v \in V$ for FU type j, and $\mathbf{P_{R_j}(v)}$, $1 \le j \le M$, representing the corresponding probability function. And $\mathbf{C_{R_j}(v)}$, $1 \le j \le M$, is used to represent the cost of each node $v \in V$ for FU type j, which is a fixed value. Given an assignment A of a DFG G, we define the ***system total cost under assignment A***, denoted as $\mathbf{C_A(G)}$, to be the summation of costs, $C_{A(v)}(v), v \in V$, of all nodes, that is, $C_A(G) = \sum_{v \in V} C_{A(v)}(v)$. In this paper we call $C_A(G)$ as ***total cost*** in brief.

For the input DFG G, given an assignment A, assume that $\mathbf{T_A(G)}$ stands for the ***execution time of graph G under assignment A***. $\mathbf{T_A(G)}$ 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. 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 $(T_A(G) \le L) \ge P$. 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. If $T_{R_j}(v)$ is a continuous random variable, then it has a ***probability density function (PDF)*** . If 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 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*.

4 The Algorithms for HAP problem

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

4.1 Definitions and Lemma

To solve the HAP problem, we use dynamic programming method, which will travel the graph in bottom up fashion. For the easiness of explaination, we will index the nodes based on bottom up sequence. Given the timing constraint L, a DFG G, and an assignment A, we first give several definitions as follows: $\mathbf{G^i}$ is 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. $\mathbf{C_A(G^i)}$ and $\mathbf{T_A(G^i)}$ are the total cost and total execution time of G^i under the assignment A. 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 $T_A(G^i) \le j$ with probability $\ge P_{i,j}$.

In our algorithm, $D_{i,j}$ is the table in which each entry has a link list that store pair ($P_{i,j}$, $C_{i,j}$) sorted by $P_{i,j}$ in an ascending order. 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 can use the following Lemma to cancel redundant pairs.

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

Since the link list is in an ascending order by probabilities, if $P_{i_1,j} \le P_{i_2,j}$, yet $C^1_{i,j} > C^2_{i,j}$, based on the

definitions, we can guarantee with $P^1_{i,j}$ to find smaller cost $C^2_{i,j}$. 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. The vice versa is same.

For 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 $D_{i,j}$. $E_{i,j}$ is a table of link lists that store pair ($p_{i,j}$, $c_{i,j}$) sorted by $p_{i,j}$ in an ascending order. Here, $c_{i,j}$ 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 1 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 4 after the insertion.

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 4. An example of local table, $E_{1,j}$

Here, 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 as "$\mathbf{H_1} \oplus \mathbf{H_2}$". Simlilarly, 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 pairs using Lemma 4.1.

4.2 An Optimal Algorithm for Simple Path

An optimal algorithm, *Path_Assign*, is proposed below. It can give the optimal solution for the HAP problem when the given DFG is a simple path. For a simple path, we can use either bottom up or top down approach, since there is no multi-parents or multi-children node. Without loss of generality, we use bottom up approach, that is, starting from child to parent. Assume the node sequence of the simple path in the HAP problem is $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_N$, indexed in a bottom up fashion.

Theorem 4.1. *For each pair ($P_{i,j}$, $C_{i,j}$) in $D_{i,j}$ ($1 \leq i \leq N$) 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.*

Algorithm 4.1 optimal algorithm for the HAP problem when the given input is a simple path. (Path_Assign)

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

Output: An optimal assignment for the simple path

1. Build a local table $E_{i,j}$ for each node of DFG.
2. Starting from node v_1, let $D_{1,j} = E_{1,j}$. We build the dynamic table $D_{i,j}$ step by step. For every node v_i, $i > 1$, under each timing constraint j, we will compute the entry $D_{i,j}$. The procedures are as follows: First, for each time k in $E_{i,k}$, whenever $D_{i-1,j-k} != NULL$, we compute $D_{i,j} = D_{i-1,j-k} \oplus E_{i,k}$. Then, after computing all the k in $E_{i,k}$, we will insert $D_{i,j-1}$ to $D_{i,j}$ and remove redundant pairs using Lemma 4.1
3. The cost in $D_{N,j}$ is the minimum total cost with computed confidence probability under timing constraint j, and the assignment can be obtained by tracing how to reach $D_{N,j}$. Given the timing constraint L, the minimum total cost is $D_{N,L}$.

Proof. By induction. **Basic Step:** When $i = 1$, there is only one node and $D_{1,j} = E_{1,j}$. Thus, when $i = 1$, Theorem 4.1 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 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 $E_{i+1,j}$, we try all the possibilities to obtain j. Then we use $\oplus$ operation to add the costs of two tables and multiply the probabilities of two tables. Finally, we use Lemma 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 have used Lemma 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 4.1 is true for any i ($1 \leq i \leq N$). □

From Theorem 4.1, 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 in the algorithm *Path_Assign*. Using these 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 polynomial.

4.3 An Optimal Algorithm For Tree

In this section, we propose an optimal algorithm, *Tree_Assign*, to produce the optimal solution to the HAP problem when the input DFG is a tree. In multi-children case, we use bottom up approach. The reason we use bottom up approach is that we can merge two children first as one node, then we get the simple path. Hence we can get the optimal assignment. If using top down approach, we have two paths. It is possible there are two optimal assignments for the root node and other nodes are common to both paths. The pseudo polynomial algorithm for trees is shown in *Tree_Assign*.

Algorithm 4.2 optimal algorithm for the HAP problem when the given input is a tree. (Tree_Assign)

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

Output: An optimal assignment for the tree

1. **if** the tree is multi-children type **then**
   using bottom up approach (leaf to root)
   else
   using top down approach (root to leaf)
   end if
2. For bottom up approach, use following algorithm.
   For top down approach, just reverse the tree.
   Assume the sequence of the tree 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 subtree 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

$$D'_{i,j} = \begin{cases} (0,0) & \text{if } R = 0 \\ D_{i_1,j} & \text{if } R = 1 \\ \sum_{1 \le h \le R} D_{i_h,j} & \text{if } R \ge 1 \end{cases} \quad (1)$$

   Here the $\sum$ is executed by $\oplus$ operator.
   Then, for each k in $E_{i,k}$.

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

In *Tree_Assign* algorithm, when $R = 1$, node v_i has only one child v_{i_1}. By using *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 $R \ge 1$, node v_i has multiple children. We merge all the children of node v_i into one pseudo child. Then we can use *Path_Assign* algorithm to get final solution. The merging procedures are as follows. At the same time j, sum up the costs of all children and multiple 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 implement *Path_Assign* algorithm to get the final solution for the tree.

In the following, we prove *Algorithm Tree_Assign* gives the optimal solution when the given DFG is a tree.

Theorem 4.2. *For each pair ($P_{i,j}$, $C_{i,j}$) in $D_{i,j}$ ($1 \le i \le N$) 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 $D_{1,j} = E_{1,j}$. Thus, when $i = 1$, Theorem 4.2 is true. **Induction Step:** We need to show that for $i \ge 1$, 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. According to the bottom up approach in multi-children tree (for top down approach, just reverse the tree), the execution of $D_{i,j}$ for each child node of v_{i+1} has been 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} into the 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 4.2 is true for any i ($1 \le i \le N$). □

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 variation 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.

5 Experiments

This section presents the experimental results of our algorithms. We conduct experiments on a set of benchmarks including 4-stage lattice filter, 8-stage lattice filter, and voltera filter. The DFG of all these three filters are trees. 4-stage lattice filter DFG has 26 nodes; 8-stage lattice filter DFG has 42 nodes, and there are 27 nodes in voltera filter DFG.

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. We compare our algorithms with the *optimal hard HA algorithms* in [3]. 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. All experiments are finished in less than one second.

The experimental results for voltera filter, 4-stage lattice filter, and 8-stage lattice filter, are shown in Table 5-

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. Comparison of the total costs for voltera filter between the HAP and hard HA problems.

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 6. Comparison of the total costs for 4-stage lattice filter between the HAP and hard HA problems.

7. 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 algorithms*, are presented in each entry. Columns "1.0","0.9", "0.8", and "0.7", etc., represent that the confidence probability is 1.0, 0.9, 0.8, and 0.7 respectively. Algorithm *Tree_Assign* covers all the probability columns, yet the *optimal hard HA algorithms* only include the column "1.0", which is in boldface. For example, for the first row, the timing constraint 51, the entry under "1.0" is 237, which is the minimum total cost for the hard HA problem. The entry under "0.9" is 229, which means we have 90 % confidence to achieve minimum total cost 229.

Column "%" shows the percentage of reduction on the total cost, comparing the results of algorithm with those obtained by the *optimal hard HA algorithms*. The average percentage reduction is shown in the last row "Ave. Redu(%)" of all Tables 5-7. The entry with "×" means no solution available. With timing constraint 80 in Table 5, the *optimal hard HA algorithms* can not find a solution. However, we can find solution 7874 with 0.9 probability that guarantees the total execution time of the DFG are less than or equal to the timing constraint 80.

Through the experimental results, we found that our algorithms have much better performance compared with the *optimal hard HA algorithms*. On average, algorithm *Tree_Assign* gives a cost reduction of 32.5% with 0.9 confidence probability, and a cost reduction of 38.2% and 40.6% with 0.8 and 0.7 confidence probability respectively.

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	

Table 7. Comparison of the total costs for 8-stage lattice filter between the HAP and hard HA problems.

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. For *heterogeneous assignment with probability* (HAP) problem, we presented two optimal algorithms, *Path_Assign* and *Tree_Assign*, to give the optimal solutions to the HAP problem when the input is a simple path or a tree respectively. 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] S. Tongsima, Edwin H.-M. Sha, C. Chantrapornchai, D. Surma, and N. Passose, "Probabilistic loop scheduling for applications with uncertain execution time," *IEEE Trans. on Computers*, vol. 49, pp. 65–80, Jan. 2000.

[2] 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*, vol. 6, pp. 582–594, Dec. 1998.

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

[4] 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*, vol. 15, no. 4, pp. 319–330, 2004.

[5] Y. He, Z. Shao, B. Xiao, Q. Zhuge, and Edwin 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.