Assignment and Scheduling of Real-time DSP Applications for Heterogeneous Functional Units *

Zili Shao, Qingfeng Zhuge, Yi He, Chun Xue, Meilin Liu, Edwin H.-M. Sha
Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75083, USA

Abstract

In high level synthesis for real-time digital signal processing (DSP) architectures using heterogeneous functional units (FUs), an important problem is how to assign a proper FU type to each operation of a DSP application and generate a schedule in such a way that all requirements can be met and the total cost can be minimized. In this paper, we propose a two-phase approach to solve this problem. In the first phase, we solve heterogeneous assignment problem, *i.e., how to assign a proper FU type to a DSP application such that the total cost can be minimized while the timing constraint is satisfied. In the second phase, based on the assignments obtained from the first phase, we propose* a minimum resource scheduling algorithm *to generate a schedule and a feasible configuration that uses as little resource as possible. We prove* heterogeneous assignment problem *is NP-complete and propose several algorithms to solve it. The experiments show that* Algorithm DFG_Assign_Repeat *is the best that gives a reduction of 25.7% on total cost compared with the previous work.*

1 Introduction

To satisfy the requirements of high sample rates and low power consumption for real-time digital signal processing (DSP) applications, high-level synthesis of special purpose architectures has become a common and critical step in the design flow [3–6, 11, 12, 14, 15, 19]. Although a lot of research efforts have been put into this field, most previous work on the synthesis of special purpose architectures for real-time DSP applications focuses on the architectures that only use homogeneous FUs (same type of operations will be processed by same type of FUs) [4–6, 14, 15, 19]. With more and more different types of functional units (FUs) available, same type of operations can be processed by heterogeneous FUs with different costs, where the cost may relate to power, reliability, or monetary cost. Therefore, an important problem arises: how to assign a proper FU type to each operation of a DSP application and generate a schedule in such a way that the requirements can be met and the total cost can be minimized.

A lot of techniques have been developed on allocating and scheduling applications in heterogeneous distributed systems [1, 2, 10, 16]. Incorporating reliability cost into heterogeneous distributed systems, the reliability driven assignment and scheduling problem has been studied in [7, 9, 17, 18]. However, since allocation and scheduling are performed based on a fixed architecture in these work, they can not be directly applied to solve this problem. In this paper, we propose a two-phase approach to solve the problem. In the first phase, we solve *heterogeneous assignment problem*, i.e., given the types of heterogeneous FUs, a Data-Flow Graph (DFG) in which each node has different execution times and costs for different FU types, and a timing constraint, how to assign a proper FU type to each node such that the total cost can be minimized while the timing constraint is satisfied. In the second phase, based on the assignments obtained from the first phase, we propose *a minimum resource scheduling algorithm* to generate a schedule and a feasible configuration that uses as little resource as possible. Here, a configuration means which types and how many for each type should be selected in a system. Both *heterogeneous assignment problem* and *scheduling* are difficult problems. It is well known that the scheduling with resource constraints is NP-complete [8]. We will show *heterogeneous assignment problem* is also NP-complete in Section 4.

Our work is closely related to the work in [3, 11–13]. In [11, 12], Ito et. al. first propose a ILP (Integer Linear Programming) model for the assignment problem considering heterogeneous functional units. While this ILP model can generate an optimal solution for *heterogeneous assignment problem*, the exponential run time of the algorithm limits its applicability. In [3], Chang et. al. propose a heuristic approach for *heterogeneous assignment problem*. This approach can produce a solution with one or two orders of magnitude less time compared with the previous ILP model; however, this simple heuristic approach may not produce the good result. In the circuit design field, Li et. al. in [13] study the problem of selecting an implemen-

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

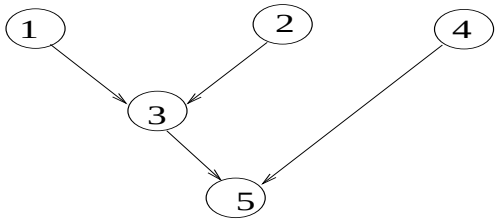
tation of each circuit module from a cell library. The problem is shown to be NP-hard. A pseudo-polynomial time algorithm on the series-parallel circuits and heuristics for general circuits are proposed for basic circuit implementation problem. Basic circuit implementation problem is only a special case of *heterogeneous assignment problem* in which each node must have the same execution time; therefore, their solutions can not be applied to solve *heterogeneous assignment problem*. And the circuit implementation problem doesn't need to consider the scheduling problem. Hence, we propose a more general and effective approach in this paper. To solve *heterogeneous assignment problem*, we generalize the approach used in [13] and make it capable of dealing with any DFG. To solve *scheduling and configuration problem*, we propose *a minimum resource scheduling algorithm* to generate a schedule and a feasible configuration that use as little resource as possible.

We first prove *heterogeneous assignment problem* is NP-complete and propose several practical algorithms. When the given DFG is a simple path or a tree, we propose two algorithms, *Path_Assign* (for simple path) and *Tree_Assign* (for tree), to produce the optimal solution. These two algorithms are efficient in practice, though rigorously speaking they are pseudo polynomial because the complexities are related to the value of the maximum execution time of nodes. But this value is usually not large or can be normalized to be small. To solve the general problem, Algorithm *DFG_Assign_Once* and *DFG_Assign_Repeat*, are proposed based on *Algorithm Tree_Assign*. Then, based on the obtained assignment, a minimum resource scheduling algorithm is proposed to generate a schedule and a configuration. We experiment with our algorithms on a set of benchmarks. We compare our algorithms with the greedy algorithm in [13]. The experimental results show *DFG_Assign_Once* and *DFG_Assign_Repeat* have better performance compared with the greedy algorithm. On average, *DFG_Assign_Once* gives a reduction of 25% and *DFG_Assign_Repeat* gives a reduction of 25.7% on system cost compared with the greedy algorithm. When the given DFG is a tree, *DFG_Assign_Once* and *DFG_Assign_Repeat* both give the optimal solution. *DFG_Assign_Repeat* is recommended to be used because it performs best especially when the input graph is large.

The remainder of this paper is organized as follows: In the next section, motivational examples are given. In Section 3, we give the basic definitions and concepts used in the rest of the paper. In Section 4, we prove *heterogeneous assignment problem* is NP-complete. The algorithms for *heterogeneous assignment problem* are presented in Section 5. The minimum resource scheduling algorithm are presented in Section 6. Experimental results and concluding remarks are provided in Section 7 and Section 8 respectively.

2 Motivational Examples

Assume we can select FUs from a FU library that provides three types of FUs: P_1, P_2, P_3. An exemplary DFG is shown in Figure 1(a). The execution times and costs of each node for different FU types are shown in Figure 1(b). In Figure 1(b), column "T_i" presents the execution time, and column "C_i" presents the execution cost for each FU type P_i.



(a)

Nodes	P1		P2		P3	
	T_1	C_1	T_2	C_2	T_3	C_3
1	1	9	2	4	3	1
2	1	10	2	6	3	1
3	1	9	2	3	3	1
4	1	7	2	2	6	1
5	1	8	2	5	3	1

(b)

Figure 1. A given DFG and the execution times and costs of its nodes for different FU types.

The execution cost can be any cost such as energy consumption or reliability cost. A node executed on one type of FUs may run slower but with less energy consumption or reliability cost than it executed on another type. 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. When the execution cost is related to reliability, the total reliability cost is still the summation of reliability costs of all nodes. We compute the reliability cost using the same model as in [18]. Define the reliability of a system as the probability that the system will not fail during the time of executing a DFG. Consider a heterogeneous system with M FU types, $\{P_1, P_2, \cdots, P_M\}$, and a DFG containing N nodes, $\{u_1, u_2, \cdots, u_N\}$. Let $t_j(i)$ be the execution time of node u_i for type P_j. Let f_j be the failure rate of type P_j. Then the *reliability cost* of node u_i for type P_j is defined as $t_j(i) \times f_j$. Let x_{ij} be a binary number that denotes whether type P_j is assigned to node u_i or not (it equals 1 if P_j is assigned to u_i; otherwise it equals 0). The probability of a system not to fail during the time of processing a DFG, is:

$$Pr = \prod_{1 \le j \le M, 1 \le i \le N} (1 - f_j)^{x_{ij} t_j(i)}.$$

From this equation, we know $Pr \approx \prod(e^{-f_j x_{ij} t_j(i)})$ when f_j is small. Thus, in order to maximize Pr, we need to minimize $\sum(f_j x_{ij} t_j(i))$. In other words, we need to find an assignment such that the timing constraint is satisfied and the summation of reliability costs of all nodes is minimized in order to maximize the reliability of a system.

Nodes	P1	P2	P3
1		√ (4)	
2		√ (6)	
3		√ (3)	
4		√ (2)	
5		√ (5)	

(a)

Nodes	P1	P2	P3
1			√ (1)
2			√ (1)
3		√ (3)	
4		√ (2)	
5	√ (8)		

(b)

Figure 2. (a) Assignment 1 with cost 20. (b) Assignment 2 with cost 15.

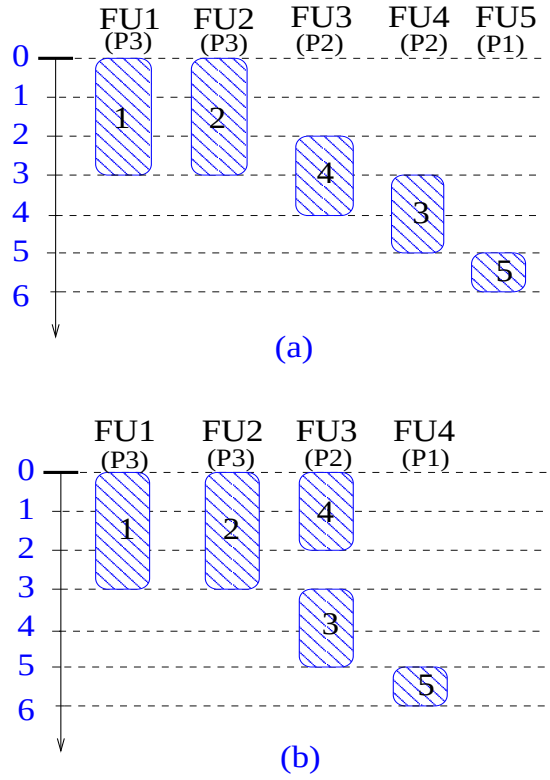


Figure 3. Two schedules corresponding to Assignment 2.

Assume the given costs are energy consumption and the timing constraint is 6 time units in this example. For the given DFG in Figure 1(a) and the time cost table in Figure 1(b), two different assignments are shown in Figure 2. In Figure 2, if a FU type is assigned to a node, "√" is put into the right location and the value in the parentheses beside "√" is the corresponding execution cost. The total execution cost for Assignment 1 is 20. The total cost for Assignment 2 is 15 and this is an optimal solution, which is 25% less than that for Assignment 1. Our assignment algorithm in Section 5.2 achieves the optimal solution for this example.

For Assignment 2, two different schedules with corresponding configuration are shown in Figure 3. The configuration in Figure 3(a) uses 5 FUs while the configuration in Figure 3(b) uses 4 FUs. The schedule in Figure 3(b) is generated by the minimum resource scheduling algorithm in Section 6, in which the configuration achieves the minimal resource for Assignment 2.

3 Definitions

In our work, Data-Flow Graph (DAG) is used to model a DSP application. A DFG $G = \langle V, E, d \rangle$ is a node-weighted directed graph, where $V = \langle u_1, u_2, \cdots, u_N \rangle$ is the set of nodes, $E \subseteq V \times V$ is the edge set that defines the precedence relations among nodes in V, and $d(e)$ represents the number of delays for an edge e. In our case, a DFG can contain cycles. The intra-iteration precedence relation is represented by the edge without delay and the inter-iteration precedence relation is represented by the edge with delays. The *cycle period* of a DFG corresponds to the minimum schedule length of one iteration of the loop when there are no resource constraints. A *static* schedule of a cyclic DFG is a repeated pattern of an execution of the corresponding loop. A static schedule must obey the precedence relations of the *directed acyclic graph (DAG)* portion of a DFG. In this paper, we consider the DAG part in assignment and scheduling that is obtained by removing all edges with delays from a DFG.

A special purpose architecture consists of different types of FUs. Assume there are M different FU types in a FU library, $P_1, P_2, \cdots, P_M$. $T(u_i)$ is used to represent the execution times of each node $u_i \in V$ for different FU types: $T(u_i)$=$\{t_1(i), t_2(i), \cdots, t_M(i)\}$ where $t_j(i)$ denotes the execution time of u_i for type P_j. $C(u_i)$ is used to represent the execution costs of each node $u_i \in V$ for different FU types: $C(u_i)$=$\{c_1(i), c_2(i), \cdots, c_M(i)\}$ where $c_j(i)$ denotes the execution cost of u_i for type P_j. An assignment for a DFG is to assign a FU type to each node. Given an assignment of a DFG, we define *the system cost* to be the summation of execution costs of all nodes because it is easy to explain and useful as we described in Section 2. Please note that our algorithms presented later will still work with straightforward revisions to deal with any func-

tion that computes the total cost such as $\sum_i c_j(i)^2$ as long as the function satisfies "associativity" property.

We define *heterogeneous assignment problem* as follows:
Given M different FU types: P_1,P_2,$\cdots$,P_M, a DFG $G = \langle V, E, d\rangle$ where V=$\langle u_1$,u_2,$\cdots$,$u_N\rangle$, $T(u_i) = \{t_1(i), t_2(i), \cdots, t_M(i)\}$ and $C(u_i)$=$\{c_1(i), c_2(i), \cdots, c_M(i)\}$ for each node $u_i \in V$, and a timing constraint L, *find an assignment for* G *such that the system cost is minimized within* L.

4 NP-Completeness

In this section, we prove *heterogeneous assignment problem* is NP-complete. If there is no timing constraints, the FU type with the minimum cost can be assigned to every node to minimize the system cost. So the problem is trivial. When adding a timing constraint, the problem becomes NP-complete.

In order to prove Theorem 4.1, we first define a decision problem (*DP*) for *heterogeneous assignment problem*.

DP: Given a positive integer C, a positive integer L, the number of resource types M and a DFG $G = \langle V, E, d\rangle$, is there an assignment for G with the M resource types such that the execution time of $G \leq L$ and the system cost of $G \leq C$?

Theorem 4.1. *The decision problem of heterogeneous assignment problem is NP Complete.*

In the proof, we will transform *0-1 Knapsack Problem* to our problem by setting G to be a simple path: $u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_N$ and $M = 2$. *0-1 Knapsack Problem* is defined as follows.

0-1 Knapsack Problem: Given a set of items $I = \{1, 2, \cdots, N\}$ in which for each item $i \in I$, $Q_i \in Z^+$ is its value and $W_i \in Z^+$ is its weight, and two given positive integers H and W, is there a subset S of I such that $\sum_{i\in S} P_i \geq H$ and $\sum_{i\in S} W_i \leq W$?

Proof. It is obvious DP belongs to NP. Assume $I = \{1, 2, \cdots, N\}$ is an instance of *0-1 Knapsack Problem*. Set $M = 2$. Construct a simple path $G = \langle V, E\rangle$ as follows. $V = \langle u_1, u_2, \cdots, u_N\rangle$ where u_i corresponds to item i in I. Add $e(u_i \rightarrow u_{i+1})$ into E for $1 \leq i \leq (N-1)$. Let Q_{max} be the maximum of Q_i in I. For each node $u_i \in V$, $T(u_i) = \{t_1(i), t_2(i)\}$ is obtained by $t_1(i) = 0$ and $t_2(i) = W_i$, and $C(u_i) = \{c_1(i), c_2(i)\}$ is obtained by $c_1(i) = Q_{max}$ and $c_2(i) = Q_{max} - Q_i$. Let $C = N * Q_{max} - H$ and $L = W$. Then, an instance of the knapsack problem can be transformed correctly.

Since *0-1 Knapsack Problem* is NP-complete and the reduction can be done in polynomial time. *DP* is NP-complete. □

5 The Algorithms for Heterogeneous Assignment Problem

In Section 4, we prove heterogeneous assignment problem is NP-complete. In this section, several algorithms are proposed to solve this problem. When the given DFG is a simple path or a tree, two algorithms are presented to give the optimal solution. These algorithms are efficient in practice though rigorously speaking they are pseudo polynomial. To solve the general problem, two algorithms are proposed.

5.1 An Efficient Algorithm for Simple Paths

An efficient algorithm, *Path_Assign*, is proposed in this section. It can give the optimal solution for *heterogeneous assignment problem* when the given DFG is a simple path. Assume the simple path in *heterogeneous assignment problem* is $u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_N$, *Path_Assign* is shown in Figure 4.

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

1. Associate an array $X_i[1, 2, \cdots, L]$ to each node $u_i \in V$ and let $X_i[j]$ store the minimum system cost of the path from u_1 to u_i with total execution time $\leq j$. For $1 \leq j \leq L$, $X_0[j] = 0$.
2. For $i = 1$ to N, compute $X_i[j]$ $(j = 1, 2, \cdots, L)$ by:

$$X_i[j] = \begin{cases} \min_{1\leq k\leq M}\{X_{i-1}[j - t_k(i)] + c_k(i) \\ \qquad \text{if } j - t_k(i) \geq T_{min}(i-1)\}, \\ \text{No feasible solution} \quad \text{Otherwise} \end{cases} \quad (1)$$

where, $T_{min}(i-1)$ is the minimum time needed to process the path from u_1 to u_{i-1} and $T_{min}(0) = 0$.
3. $X_N[L]$ is the minimum system cost and the assignment can be obtained by tracing how to reach to $X_N[L]$.

Figure 4. Algorithm Path_Assign.

By induction, we can prove the optimal solution is obtained by *Algorithm Path_Assign* and $X_N[L]$ records the minimum system cost of the whole path within the time constraint L. We can record the corresponding FU type assignment of each node when computing the minimum system cost in Step 2 in *Algorithm Path_Assign*. Using this information, we can get an optimal assignment by tracing how to reach $X_N[L]$. An example is shown in Figure 5.

A simple path $u_1 \rightarrow u_2 \rightarrow u3$ is shown in Figure 5(a). Assume there are 2 FU types, P_1 and P_2, in the system. The execution times and execution costs of nodes for different FU types are shown in Figure 5(b). When the timing constraint is 7 time units, the computation procedure using *Algorithm Path_Assign* is shown in Figure 5(c). In Figure 5(c), the FU type assignment for each node is recorded under Xi[j]. The minimum system cost is 8 which is shown

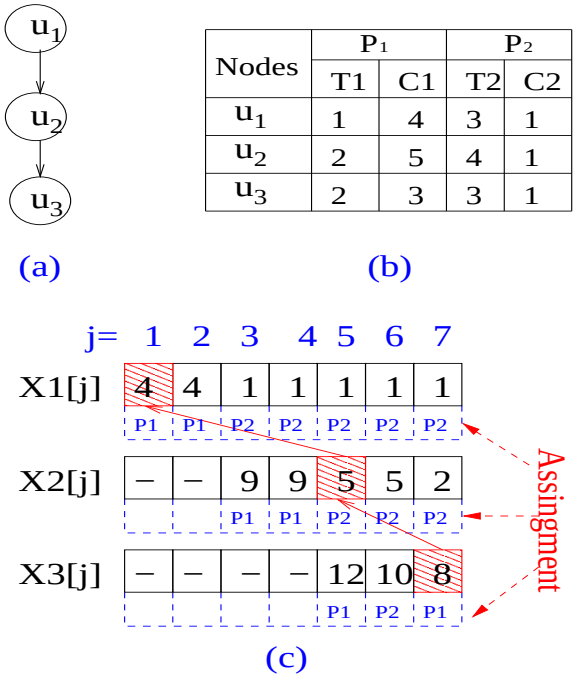


Figure 5. An example for a simple path with 2 FU types.

in $X_3[7]$, and the assignment is $P_1 \rightarrow u_1, P_2 \rightarrow u_2$, and $P_1 \rightarrow u_3$ which is obtained by tracing how to reach $X_3[7]$. Starting from $X_3[7]$, we know P_1 is assigned to u_3 and its execution time $t_1(3) = 2$ which is shown in Figure 5(b). Then we can get the index for $X_2[j]$ by subtracting $t_1(3)$ from L: $L - t_1(3) = 7 - 2 = 5$. So we get to location $X_2[5]$, from which we can see P_2 is assigned to u_2 and its execution time $t_2(2) = 4$. In the same way, we can find out P_1 is assigned to u_1.

It takes $O(M)$ to compute one value of $X_i[j]$ where M is the number of FU types. Thus, the complexity of *Algorithm Path_Assign* is $O(|V| * L * M)$, 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 $O(|V|^k)$ (k is a constant). In this case, *Path_Assign* is polynomial.

5.2 An Efficient Algorithm For Trees

In this section, we propose an efficient algorithm, *Tree_Assign*, to produce the optimal solution for *heterogeneous assignment problem* when the given DFG is a tree.

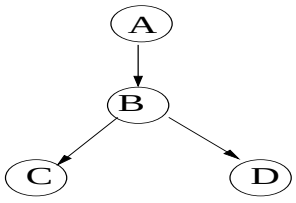


Figure 6. A given tree.

Define a *root* node to be a node without any parent and a *leaf* node to be a node without any child. A post-ordering for a tree is a linear ordering of all its nodes such that if there is an edge $u \rightarrow v$ in the tree, then v appears before u in the ordering. For example, $\{C, D, B, A\}$ is a post-ordering for the given tree in Figure 6. The pseudo polynomial algorithm for trees, *Tree_Assign*, is shown in Figure 7.

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

1. Add a pseudo node u_{N+1} in V and set all 0's for its execution times and execution costs. For each *root* node $u \in V$, add an edge $e(u_{N+1} \rightarrow u)$ into E. Then u_{N+1} is only root node in G.
2. Post-order G and let $V = \{v_1, v_2, \cdots, v_N, v_{N+1}\}$ be the post-ordering node set. Without loss of generality, for each node $v_i \in V$, let $T(v_i) = \{t_1(i), t_2(i), t_3(i), \cdots, t_M(i)\}$ where $t_j(i)$ is the execution time of node v_i for FU type P_j; and $C(v_i) = \{c_1(i), c_2(i), \cdots, c_M(i)\}$ where $c_j(i)$ is the execution cost of node v_i for FU type P_j.
3. Associate an array $X_i[1, \cdots, L]$ to each node $v_i \in V$ and $X_i[j]$ stores the minimum system cost of the subtree rooted on v_i with total execution time $\leq j$. $X_0[j] = 0$ for $j = 1, 2, \cdots, L$.
4. For $i = 1$ to $N + 1$, compute $X_i[j]$ ($j = 1, 2, \cdots, L$) as follows:

$$X_i[j] = \begin{cases} \min_{1 \leq k \leq M}\{X_{i'}[j - t_k(i)] + c_k(i) \\ \quad \text{if } j - t_k(i) \geq T_{min}(v_i)\}, \\ \text{No feasible solution} \quad \text{Otherwise} \end{cases} \tag{2}$$

where: $T_{min}(v_i)$ is the minimum time needed to process the subtree rooted on v_i except v_i. v_i' is a pseudo node. Assume that $v_{i_1}, v_{i_2}, \cdots, v_{i_R}$ are all child nodes of node v_i and R is the number of child nodes of node u_i, then $X_{i'}[j]$ $(j = 1, 2, \cdots, L)$ is calculated as follows:

$$X_{i'}[j] = \begin{cases} 0 & \text{if } R = 0 \\ X_{i_1}[j] & \text{if } R = 1 \\ \sum_{1 \leq h \leq R} X_{i_h}[j] & \text{if } R \geq 1 \end{cases} \tag{3}$$

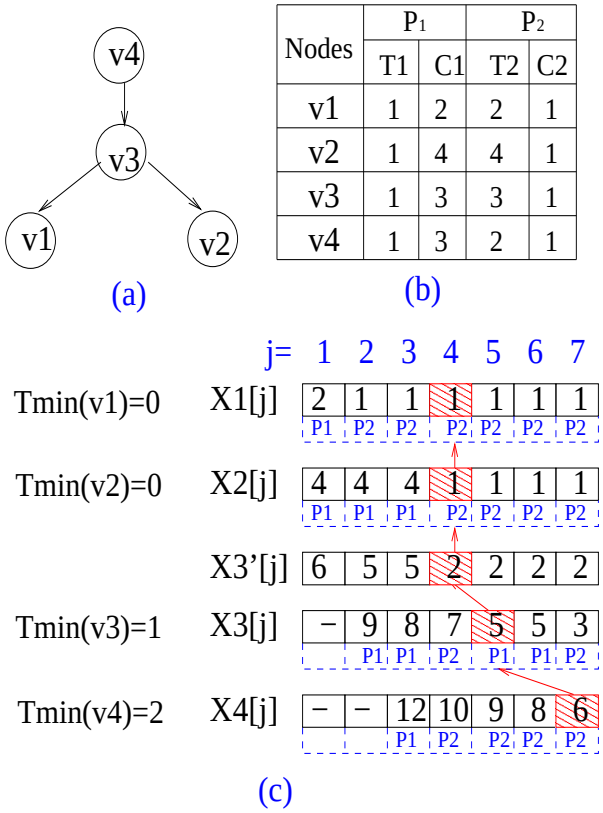
5. $X_{N+1}[L]$ is the minimum system cost for G and the corresponding assignment can be obtained by tracing how to reach to $X_{N+1}[L]$.

Figure 7. Algorithm Tree_Assign.

Following the post-ordering in Step 2, we can get $T_{min}(v_i)$ for each node $v_i \in V$ by setting the minimum execution time for each node and computing the longest path from any leaf node to v_i. In equation 2, basically, we select the minimum system cost from all possible system costs caused by adding v_i into the subtree. By induction, we can prove that $X_i[j]$ $(1 \leq i \leq N)$ obtained by *Algorithm Tree_Assign* is the minimum system cost of the subtree rooted on v_i with total execution time $\leq j$.

From equation 3, $X_{i'}[j] = 0$ if v_i has no child node and $X_{i'}[j] = X_{i_1}[j]$ if v_i has only one child node v_{i_1}. Thus, we

don't really need to compute $X_{i'}[j]$ in these cases. When there is only one root node in a tree, we don't need to add a pseudo root node u_{N+1}. Using these simplified methods, an example is shown in Figure 8 for the given tree in Figure 6.



Nodes	P_1		P_2	
	T1	C1	T2	C2
v1	1	2	2	1
v2	1	4	4	1
v3	1	3	3	1
v4	1	3	2	1

Figure 8. An example for a tree with 2 FU types.

Assume that there are 2 different FU types, P_1 and P_2. The post-ordering node set of the given tree and the corresponding execution times and execution costs are shown in Figure 8(a) and Figure 8(b) respectively. The computation procedure using *Algorithm Tree_Assign* is shown in Figure 8(c) when the timing constraint is 7 time units. In Figure 8(c), the FU type assignments are recorded below $X_i[j]$. Node v_3 has two child nodes, v_1 and v_2, so a pseudo node $v_{3'}$ is added and $X_{3'}[j] = X_1[j] + X_2[j]$. When computing $X_i[j]$, all possible system costs caused by adding v_i are computed and the minimum cost is selected. For example, to compute $X_4[7]$, we first get $t_1(4) = 1, c_1(4) = 3, t_2(4) = 2, c_2(4) = 1$ from the last row in Figure 8(b). Then try all possible system costs: if P_1 is assigned to v_4, $t_1(4) = 1$ and $c_1(4) = 3$, then the system cost is $X_3[7-1] + c_1(4) = 5 + 3 = 8$; if P_2 is assigned to v_4, $t_2(4) = 2$ and $c_2(4) = 1$, then the system cost becomes $X_3[7-2] + c_2[4] = 5 + 1 = 6$. So the minimum cost $X_4[7] = 6$ and the assignment is $P_2 \rightarrow v_4$. The assignment for the DFG is $P_2 \rightarrow v_1, P_2 \rightarrow v_2, P_1 \rightarrow v_3$, and $P_2 \rightarrow v_4$, which is obtained by tracing how to reach $X_4[7]$ using similar method in Section 5.1.

The complexity of *Algorithm Tree_Assign* is $O(|V| * L * M)$, where $|V|$ is the number of nodes, L is the given time constraint, and M is the number of FU types. When L equals $O(|V|^k)$ (k is a constant) which is the general case in practice, *Algorithm Tree_Assign* is polynomial.

5.3 The Algorithms for DFGs

In this section, we propose two heuristics to solve the general *heterogeneous assignment problem* based on Algorithm Tree_Assign. The basic idea is to get a tree from a given DFG and then use *Algorithm Tree_Assign* to solve it. We first show some notations and definitions used in the algorithms.

As defined in Section 5.2, for a DFG, a *leaf* node is a node without any child and a *root* node is a node without any parent. We define a *critical path* to be a path from a *root* node to a *leaf* node. A *common* node is a node located on more than one *critical* path. For example, in the DFG shown in Figure 9, node A and B are *root* nodes; node E and F are *leaf* nodes; there are four critical paths: $A \rightarrow C \rightarrow D \rightarrow E$, $A \rightarrow C \rightarrow D \rightarrow F$, $B \rightarrow C \rightarrow D \rightarrow E$, and $B \rightarrow C \rightarrow D \rightarrow F$; and node C and D are *common* nodes. To be a legal assignment for a DFG, the execution time for any *critical path* should be less than or equal to the given timing constraint.

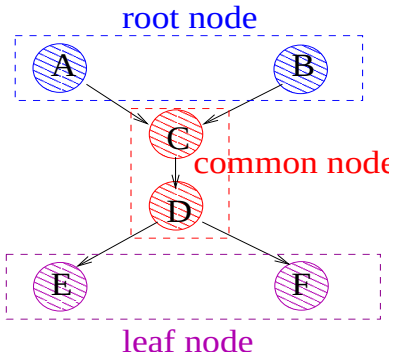


Figure 9. A DFG and its root, leaf and common nodes.

In order to use *Algorithm Tree_Assign* to solve a general problem, we need to obtain a tree from a DFG and this tree should contain all critical paths of the DFG. So the solution for the tree can satisfy the timing constraint for the DFG. A *critical path tree* of a DFG is a tree that is extracted from the DFG and contains all of its *critical paths*. *Algorithm DFG_Expand* (Figure 10) is designed to obtain a *critical path tree* for a given DFG.

Algorithm DFG_Expand duplicates the common nodes with multiple parent nodes in a DFG from the bottom up based on the post-ordering. For each common node v with multiple parent nodes, the subtree rooted by v is duplicated and after duplication, each parent node is connected to one copy of v such that each copy of v is on a unique path. Therefore, all nodes that have been went through are in a

Input: a DFG G.
Output: A tree $G_{TR} = \langle V_{TR}, E_{TR} \rangle$.

1. $G_{TR} = G$.
2. Post-order all nodes in G_{TR} such that if there is an edge $u \rightarrow v \in V_{TR}$, node v appears before u in the order.
3. Go through all nodes of G_{TR} by post-ordering. On visiting each node $v \in V$, if v has k $(k > 1)$ parent nodes, $u_1, u_2, \cdots, u_k$, do as follows:
•Duplicate $(k-1)$ copies of the subtree rooted by v into G_{TR}.
•For $i = 1, 2, \cdots, k-1$, remove the edge $u_i \rightarrow v$ from E_{TR} and add one edge from u_i to one of copies of v into E_{TR} such that there is only one incoming edge for each copy of v.

Figure 10. Algorithm DFG_Expand.

unique path. For the DFG shown in Figure 9, a *critical path tree* obtained by *Algorithm DFG_Expand* is shown in Figure 11(a).

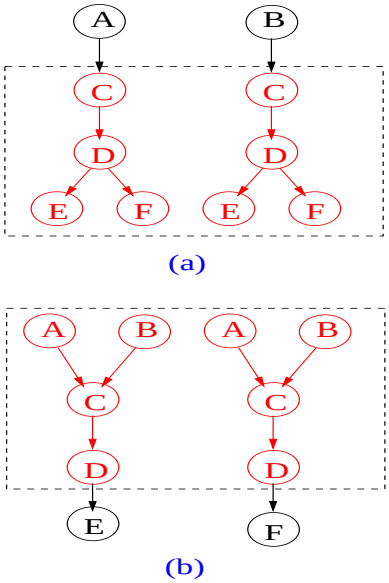


Figure 11. Two critical path trees for the DFG in Figure 9.

Another way to obtain a *critical path tree* for a DFG is to duplicate the subtree connecting to a common node with multiple child nodes and connect each copy of the common node to one child node from the top down. One example using this method is shown in Figure 11(b). By running *Algorithm DFG_Expand* on the transpose of a given DFG, we can get this kind of *critical path tree* with the reversed directions of edges.

After obtaining a *critical path tree* from a DFG, we can use *Algorithm Tree_Assign* to get the optimal solution. In this solution, the different copies of a duplicated node may have different assignments in the *critical path tree*. So we need to select an assignment. In our first heuristic, *Algorithm DFG_Assign_Once*, we select the assignment with minimum execution time as the assignment of a duplicated node. *Algorithm DFG_Assign_Once* is shown in Figure 12.

Input: M different types of FUs, a DFG, and the timing constraint L.
Output: An assignment for the DFG.

1. Let G^T be the transpose of G. Let G_{TR} and G^T_{TR} be the *critical path trees* obtained by running *Algorithm DFG_Expand* on G and G^T respectively. If $|V(G_{TR})| < |V(G^T_{TR})|$, then $G_{TREE} = G_{TR}$; otherwise $G_{TREE} = G^T_{TR}$, where $|V(G)|$ denotes the number of nodes of G.
2. Run *Algorithm Tree_Assign* on G_{TREE}.
3. For each node $v \in V$, if it has only one copy in G_{TREE}, set its assignment as the assignment in G_{TREE}; otherwise, set its assignment as the assignment with minimum execution time among all assignments of its copies in G_{TREE}.

Figure 12. Algorithm DFG_Assign_Once.

Using *Algorithm DFG_Assign_Once*, the less nodes a critical path tree for a DFG has, the closer result to the optimal solution we can obtain. So we choose a critical path tree with less nodes between G_{TREE} and G^T_{TREE}, where G_{TREE} and G'_{TREE} are critical path trees obtained from G and G^T respectively. We select the assignment with minimum execution time as the assignment of a duplicated node in such a way that this assignment can satisfy the timing constraint for a DFG.

In *Algorithm DFG_Assign_Once*, when we fix the assignment of a duplicated node, we may reduce the execution costs of other nodes because all its copies now use the minimum execution time. So in our second heuristic, *Algorithm DFG_Assign_Repeat*, we repeatedly run *Algorithm Tree_Assign* and fix the assignment of one duplicated node in each step. *Algorithm DFG_Assign_Repeat* is show in Figure 13.

In *Algorithm DFG_Assign_Repeat*, in each step, we fix the execution time and cost of one duplicated node based on the assignment obtained by Algorithm Tree_Assign. Since the more copies a node has, the more paths in the critical path tree it can influence. Thus, we sort the duplicated nodes by the number of copies and fix the node with greatest number of copies first. In such a way, when the assignment of a duplicated node is fixed, we can reduce the cost of other nodes as much as possible.

6 The Minimum Resource Scheduling and Configuration

In this section, we propose minimum resource scheduling algorithms to generate a schedule and a configuration. We first propose *Algorithm Lower_Bound_RC* that

Input: M different types of FUs, a DFG, and the timing constraint L.
Output: An assignment for the DFG.

1. For each node $u_i \in V$, associate a boolean variable, $Mark(u_i)$, and set $Mark(u_i) = False$.
2. Let G^T be the transpose of G. Let G_{TR} and G^T_{TR} be the *critical path trees* obtained by running *Algorithm DFG_Expand* on G and G^T respectively. If $|V(G_{TR})| < |V(G^T_{TR})|$, then $G_{TREE} = G_{TR}$; otherwise $G_{TREE} = G^T_{TR}$, where $|V(G)|$ denotes the number of nodes of G.
3. Let set V_{CN} record nodes in G that have more than one copies in G_{TREE}. Sort V_{CN} by the number of copies such that $V_{CN} = \langle v_1, v_2, \cdots, v_n \rangle$ in which $Copy\_Num(v_1) \geq Copy\_Num(v_2) \geq \cdots \geq Copy\_Num(v_n)$ where $Copy\_Num(v_i)$ denotes the number of copies of v_i.
4. Run *Algorithm Tree_Assign* on G_{TREE}.
5. for each node $v_i \in V_{CN}$ ($i = 1$ to $|V_{CN}|$), do:
• Set the assignment of v_i as the assignment with minimum execution time among all assignments of its copies in G_{TREE}. Set $Mark(v_i) = True$.
• Fix its execution time and execution cost in G_{TREE} based on this assignment. Run *Algorithm Tree_Assign* on G_{TREE}.
6. For each node $u_i \in V$, if $Mark(v) = False$, set its assignment as the assignment of its copy in G_{TREE}.

Figure 13. Algorithm DFG_Assign_Repeat.

produces an initial configuration with low bound resource. Then we propose *Algorithm Min_RC_Scheduling* that refine the initial configuration and generate a schedule to satisfy the timing constraint.

Algorithm Lower_Bound_RC is shown in Figure 14. In the algorithm, it counts the total number of each FU type in each control step in the ASAP and ALAP schedule, respectively. Then the lower bound for each FU type is obtained by the maximum value that is selected from the average resource needed in each time period.

Using the lower bound of each FU as an initial configuration, *Algorithm Min_RC_Scheduling* (Figure 15) is proposed to generate a schedule that satisfies the timing constraint and get the finial configuration. In the algorithm, we first compute ALAP(v) for each node v, where ALAP(v) is the schedule step of v in the ALAP schedule. Then we use a revised list scheduling to perform scheduling. In each scheduling step, we first schedule all nodes that have reached to the deadline with additional resource if necessary and then schedule all other nodes as many as possible without increasing resource.

7 Experiments

In this section, we experiment with our algorithms on a set of benchmarks including 4-stage lattice filter, 8-stage lattice filter, voltera filter, differential equation solver, elliptic filter and RLS-laguerre lattice filter. Among them, the graphs for first three filters are trees and those for the others

Input: A DFG with FU assignments and timing constraint L.
Output: Lower bound for each FU type

1. Schedule the DFG by ASAP and ALAP scheduling, respectively.
2. $N_{ASAP}[i][j] \leftarrow$ the total number of nodes with FU type j and scheduled in step i in the ASAP schedule.
3. $N_{ALAP}[i][j] \leftarrow$ the total number of nodes with FU type j and scheduled in step i in the ALAP schedule.
4. for each FU type j,
$LB_{ASAP}[j] \leftarrow \max\{\frac{N_{ASAP}[1][j]}{1}, \frac{N_{ASAP}[1][j]+N_{ASAP}[2][j]}{2}, \cdots, \frac{\sum_{1 \leq k \leq L} N_{ASAP}[k][j]}{L}\}$
5. for each FU type j,
$LB_{ALAP}[j] \leftarrow \max\{\frac{N_{ALAP}[L][j]}{1}, \frac{N_{ALAP}[L][j]+N_{ALAP}[L-1][j]}{2}, \cdots, \frac{\sum_{1 \leq k \leq L} N_{ALAP}[L-k+1][j]}{L}\}$
6. for each FU type j, its lower bound: $LB[j] \leftarrow \max\{LB_{ASAP}[j], LB_{ALAP}[j]\}$.

Figure 14. Algorithm Lower_Bound_RC.

Input: A DFG with FU assignments, timing constraint L, and an initial configuration.
Output: A schedule and a configuration.

1. For each node v in DFG, compute ALAP(v) that is the schedule step of v in the ALAP schedule.
2. $S \leftarrow 1$;
3. Do{
• Ready_List $\leftarrow$ all ready nodes;
• For each node $v \in$ Ready_List, if ALAP(v)==S, schedule v in step S with additional resource if necessary;
• For each node $v \in$ Ready_list, schedule node v without increasing current resource or schedule length;
• Update Ready_List and $S \leftarrow S + 1$;
} Until $S > L$;

Figure 15. Algorithm Min_RC_Scheduling.

are DFGs. For the DFGs, differential equation solver and RLS-laguerre filter have three duplicated nodes and elliptic filter has 19 duplicated nodes, in which a duplicated node is a node that has more than one copy in G_{Tree}. Three different FU types, P_1, P_2, P_3, are used in the system, in which a FU with type P_1 is the quickest with the highest cost and a FU with type P_3 is the slowest with the lowest cost. The execution costs and times for each node are randomly assigned. For each benchmark, the first time constraint we use is the minimum execution time. We compare our algorithms with the greedy algorithm in [13].

The experimental results for 4-stage lattice filter, 8-stage lattice file, and voltera filter, are shown in Table 1. In the table, Column "TC" presents the given timing constraint. The system costs obtained from four different algorithms: *the greedy algorithm*, *Algorithm Tree_Assign*, *Algorithm DFG_Assign_Once*, and *Algorithm DFG_Assign_Repeat*, are presented in Column "Greedy",

TC	Greedy (from [13])	Tree_Assign, DFG_Assign_Once DFG_Assign_Repeat		
	cost	cost	%	Configuration
4-stage Lattice Filter				
31	286	**191**	**33.2%**	3 P_1 2 P_2 8 P_3
35	210	**159**	**24.3%**	1 P_1 4 P_2 7 P_3
40	201	**149**	**25.9%**	2 P_1 2 P_2 8 P_3
45	192	**140**	**27.1%**	2 P_1 1 P_2 5 P_3
50	189	**133**	**29.6%**	1 P_1 2 P_2 4 P_3
55	184	**126**	**31.5%**	1 P_1 2 P_2 4 P_3
Average Reduction (%)		–	**28.6**	–
8-stage Lattice Filter				
55	490	**325**	**33.7%**	2 P_1 2 P_2 9 P_3
60	369	**291**	**21.1%**	1 P_1 3 P_2 9 P_3
65	350	**273**	**22.0%**	2 P_1 2 P_2 8 P_3
70	340	**252**	**25.9%**	1 P_1 3 P_2 8 P_3
75	329	**237**	**28.0%**	1 P_1 2 P_2 8 P_3
80	323	**226**	**30.0%**	1 P_1 2 P_2 8 P_3
Average Reduction (%)		–	**26.8**	–
Voltera Filter				
37	324	**243**	**25.0%**	4 P_1 3 P_2 5 P_3
40	324	**214**	**34.0%**	3 P_1 2 P_2 7 P_3
45	233	**191**	**18.0%**	1 P_1 4 P_2 5 P_3
50	219	**175**	**20.1%**	3 P_1 2 P_2 7 P_3
55	211	**155**	**26.5%**	1 P_1 3 P_2 6 P_3
60	209	**145**	**30.6%**	1 P_1 2 P_2 6 P_3
Average Reduction (%)		–	**25.7**	–

Table 1. Comparison of the system costs for 4-stage lattice filter, 8-stage lattice filter, and voltera filter when the timing constraint varies.

TC	Greedy (from [13])	DFG_Assign_Once		DFG_Assign_Repeat		
	cost	cost	%	cost	%	Configuration
Differential Equation Solver						
21	132	**111**	**15.9**	**111**	**15.9**	2 P_1 1 P_2 2 P_3
25	128	**101**	**21.1**	**97**	**24.2**	2 P_1 3 P_2 3 P_3
30	91	**81**	**11.0**	**81**	**11.0**	2 P_1 2 P_2 2 P_3
35	74	**60**	**18.9**	**60**	**18.9**	2 P_1 2 P_2 2 P_3
40	74	**45**	**39.2**	**45**	**39.2**	1 P_1 2 P_2 3 P_3
45	74	**40**	**45.9**	**40**	**45.9**	0 P_1 1 P_2 4 P_3
Average Reduction (%)		**25.3**	–	**25.9**	–	
RLS-laguerre Lattice Filter						
23	204	**157**	**23.0**	**155**	**24.0**	1 P_1 3 P_2 5 P_3
25	188	**141**	**25.0**	**139**	**26.1**	1 P_1 3 P_2 6 P_3
30	156	**117**	**25.0**	**117**	**25.0**	1 P_1 1 P_2 6 P_3
35	137	**122**	**10.9**	**104**	**24.1**	1 P_1 1 P_2 6 P_3
40	112	**101**	**9.8**	**98**	**12.5**	1 P_1 1 P_2 6 P_3
45	98	**93**	**5.1**	**92**	**6.1**	0 P_1 1 P_2 4 P_3
Average Reduction (%)		**16.5**	–	**19.6**	–	
Elliptic Filter						
57	399	**378**	**5.3**	**318**	**20.3**	3 P_1 3 P_2 3 P_3
60	395	**350**	**11.4**	**288**	**27.1**	2 P_1 3 P_2 4 P_3
65	371	**309**	**16.7**	**247**	**33.4**	2 P_1 3 P_2 3 P_3
70	328	**278**	**15.2**	**229**	**30.2**	2 P_1 4 P_2 3 P_3
75	296	**249**	**15.9**	**224**	**24.3**	2 P_1 3 P_2 3 P_3
80	293	**236**	**19.5**	**208**	**29.0**	2 P_1 3 P_2 3 P_3
Average Reduction (%)		**14.0**	–	**27.4**	–	

Table 2. Comparison of the system costs for differential equation solver, RLS-laguerre lattice filter and elliptic filter when the timing constraint varies.

Column "Tree_Assign", Column "DFG_Assign_Once", and Column"DFG_Assign_Repeat", respectively. Column "Configuration" shows a feasible configuration corresponding to the assignment obtained by *Algorithm Tree_Assign*, in which "x_1 P_1 x_2 P_2 x_3 P_3" means x_1 FUs with P_1, x_2 FUs with P_2, and x_3 FUs with P_3 are used. *The greedy algorithm* is implemented based on the idea in [13]. Since these benchmarks are represented by trees, the optimal system costs are obtained by *Algorithm Tree_Assign*. The experimental results show *Algorithm DFG_Assign_Once* and *Algorithm DFG_Assign_Repeat* give the same results as *Algorithm Tree_Assign*. Column "%" shows the percentage of reduction on system cost, comparing the optimal results with those obtained by *the greedy algorithm*. The average percentage reduction is shown in the last row of each benchmark.

The experimental results for differential equation solver, RLS-laguerre lattice filter and elliptic filter, are shown in Table 2. In the table, Column "cost" presents the system cost obtained from 3 different algorithms: *the greedy algorithm* (Field "Greedy"), *Algorithm DFG_Assign_Once*(Field "DFG_Assign_Once"), and *Algorithm DFG_Assign_Repeat* (Field "DFG_Assign_Repeat"). Column "%" under "DFG_Assign_Once" and "DFG_Assign_Repeat" presents the percentage of reduction on system cost compared to *the greedy algorithm*. Column "Configuration" under "DFG_Assign_Repeat" shows the configuration corresponding to the assignment obtained by *Algorithm DFG_Assign_Repeat*.

Through the experimental results from Table 1-2, we found that our heuristics based on the optimal solution for trees have better performance compared with *the greedy algorithm*. On average, *Algorithm DFG_Assign_Once* gives a reduction of 22.8% and *Algorithm DFG_Assign_Repeat* gives a reduction of 25.7% on system cost comapred to *the greedy algorithm*. When the given DFG is a tree, both *Algorithm DFG_Assign_Once* and *Algorithm DFG_Assign_Repeat* give the optimal solution. In Differential equation solver and RLS-laguerre lattice filter, the number of duplicated nodes is small, so *Algorithm DFG_Assign_Once* and *Algorithm DFG_Assign_Repeat* give the similar results. In elliptic filter, the number of duplicated nodes is relatively big, so *Algorithm DFG_Assign_Repeat* gives better results than *Algorithm DFG_Assign_Once*.

8 Conclusion

We have proposed a two-phase approach for real-time digital signal processing (DSP) applications to perform high-level synthesis of special purpose architectures using heterogeneous functional units. In the first phase, we

solved *heterogeneous assignment problem*, i.e., how to assign a proper FU type to a DSP application such that the total cost can be minimized while the timing constraint is satisfied. In the second phase, we proposed *a minimum resource scheduling algorithm* to generate a schedule and a feasible configuration that uses as little resource as possible. We proved *heterogeneous assignment problem* is NP-complete. When the given DFG is a simple path or a tree, we presented two efficient algorithms, *Algorithm Path_Assign* and *Algorithm Tree_Assign*, to give the optimal solutions respectively. To solve the general problem, two algorithms, *Algorithm DFG_Assign_Once* and *Algorithm DFG_Assign_Repeat*, are proposed. *Algorithm DFG_Assign_Repeat* is the best especially when the input graph is large.

References

[1] R. Bettati and J. W.-S. Liu. End-to-end scheduling to meet deadlines in distributed systems. In *Proc. of the International Conf. on Distributed Computing Systems*, pages 452–459, June 1992.

[2] B. M. Carlson and L. W. Dowdy. Static processor allocation in a soft real-time multiprocessor environment. *IEEE Trans. on Parallel and Distributed Systems*, 5:316–320, March 1994.

[3] 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, March 1996.

[4] L.-F. Chao, A. LaPaugh, and E. H.-M. Sha. Rotation scheduling: A loop pipelining algorithm. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16:229–239, March 1997.

[5] L.-F. Chao and E. H.-M. Sha. Static scheduling for synthesis of dsp algorithms on various models. *Journal of VLSI Signal Processing Systems*, 10:207–223, 1995.

[6] L.-F. Chao and E. H.-M. Sha. Scheduling data-flow graphs via retiming and unfolding. *IEEE Trans. on Parallel and Distributed Systems*, 8:1259–1267, Dec. 1997.

[7] 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, March 2002.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[9] Y. He, Z. Shao, B. Xiao, Q. Zhuge, and E. Sha. Reliability driven task scheduling for tightly coupled heterogeneous systems. In *Proc. of IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 465–470, Nov. 2003.

[10] C.-J. Hou and K. G. Shin. Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems. In *IEEE Trans. on Computers*, volume 46, pages 1338–1356, Dec. 1997.

[11] 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.

[12] K. Ito and K. Parhi. Register minimization in cost-optimal synthesis of dsp architecture. In *Proc. of the IEEE VLSI Signal Processing Workshop*, Oct. 1995.

[13] 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.

[14] M. C. McFarland, A. C. Parker, and R. Camposano. The high-level synthesis of digital systems. *Proceedings of the IEEE*, 78:301–318, Feb. 1990.

[15] P. G. Paulin and J. P. Knight. Force-directed scheduling for the behavioral synthesis of asic's. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 8:661–679, June 1989.

[16] K. Ramamritham, J. A. Stankovic, and P.-F. Shiah. Efficient scheduling algorithms for real-time multiprocessor systems. In *IEEE Trans. on Parallel and Distributed Systems*, volume 1, pages 184–194, Apr. 1990.

[17] S. M. Shatz, J.-P. Wang, and M. Goto. Task allocation for maximizing reliability of distributed computer systems. *IEEE Trans. on Computers*, 41:1156–1168, Sept. 1992.

[18] 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, March 1999.

[19] C.-Y. Wang and K. K. Parhi. High-level synthesis using concurrent transformations, scheduling, and allocation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 14:274–295, March 1995.