Efficient Assignment and Scheduling for Heterogeneous DSP Systems *

Zili Shao Qingfeng Zhuge Xue Chun Edwin H.-M. Sha

Department of Computer Science

University of Texas at Dallas

Richardson, Texas 75083, USA

Abstract

This paper addresses high level synthesis for real-time digital signal processing (DSP) architectures using heterogeneous functional units (FUs). For such special purpose architecture synthesis, 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.

We propose a two-phase approach to solve this 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 (may relate to power, reliability, etc.) 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 in the first phase, we propose *a minimum resource scheduling algorithm* to generate a schedule and a feasible configuration that uses as little resource as possible.

We prove *heterogeneous assignment problem* is NP-complete. Efficient algorithms are proposed to find an optimal solution when the given DFG is a simple path or a tree. Three other algorithms are proposed to solve the general problem. The experiments show that our algorithms can effectively reduce the total cost compared with the previous work.

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

1 Introduction

High-level synthesis of special purpose architectures for real-time digital signal processing (DSP) applications has become a common and critical step in the design flow in order to satisfy the requirements of high sample rates or low power consumption [1–14]. DSP applications that process signals by digital means need special high-speed functional units (FUs) like adders and multipliers to perform addition and multiplication operations. With more and more different types of FUs available, same type of operations can be processed by heterogeneous FUs with different costs, where the cost may relate to power, reliability, etc. 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. After this fundamental problem is solved, we can further consider the optimization for other structures such as registers, ports, buses, etc., based on the obtained architecture. It is not practical to solve this problem by trying all combinations since the run time will increase exponentially with the length of the input. For example, if an application has 100 operations and there are 10 different types FUs available, it needs 10^{100} steps to try all combinations. Hence, a more efficient algorithm needs to be developed.

In this paper, we propose a two-phase approach to solve this 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 in the first phase, we propose *a minimum resource scheduling algorithm* to generate a schedule and a feasible configuration that uses as little resource as possible. Here, a configuration means which FU types and how many FUs 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 [15]. We will show *heterogeneous assignment problem* is also NP-complete in Section 4.

There have been a lot of research efforts on allocating and scheduling applications in heterogeneous distributed systems [16–26]. Incorporating reliability cost into heterogeneous distributed systems, the reliability driven assignment and scheduling problem has been studied in [27–30]. In these work, allocation and scheduling are performed based on a fixed architecture. However, when performing as-

signment 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 (same type of operations will be processed by same type of FUs) [1–8, 10, 12, 13]. In [9, 14], 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 [11], Chang et. al. propose a heuristic approach for the *heterogeneous assignment problem*. This approach can produce a solution with one or two orders of magnitude less time compared with the previous ILP model. This approach, however, may not produce a good result in terms of the total cost, since the resource configuration is fixed in the early design phase. In the circuit design field, Li et. al. in [31] study the problem of selecting an implementation 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 the *heterogeneous assignment problem* in which each node must have the same execution time; therefore, their solutions can not be applied to solve the *heterogeneous assignment problem*. And the circuit implementation problem doesn't need to consider the scheduling problem. Our work is related to the work in [11, 31]. To solve *heterogeneous assignment problem*, we propose several efficient algorithms to obtain optimal (when given DFG is path or tree) or near-optimal solution (for general problem). 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.

In the paper, 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 an 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, three algorithms, *DFG_Assign_Once*, *DFG_Assign_Repeat*, and *DFG_Assign_CP*, are proposed. Algorithm *DFG_Assign_Once* and *DFG_Assign_Repeat* are based on *Algorithm Tree_Assign*; *DFG_Assign_CP* di-

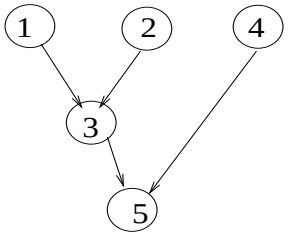
rectly works on DFGs. Then, based on the obtained assignment, a minimum resource scheduling algorithm is proposed to generate a schedule and a configuration.

We experiment with our algorithms on a set of benchmarks, and compare our algorithms with the greedy algorithm in [31] and the ILP model in [9]. The experimental results show that our algorithms have better performance compared with the greedy algorithm. On average, *DFG_Assign_Once*, *DFG_Assign_Repeat*, and *DFG_Assign_CP*, give reductions of 25%, 27.4%, and 25.8%, respectively, on system cost compared with the greedy algorithm. Our algorithms give near-optimal solutions with much less time compared with the ILP model. While DFGs become too big for the ILP model to solve, our algorithms can still efficiently give results. *DFG_Assign_CP* is recommended to be used for solving the general problem, since it gives close results with less time compared with *DFG_Assign_Once* and *DFG_Assign_Repeat*.

The remainder of this paper is organized as follows: In the next section, examples are given. In Section 3, we give the basic definitions and models 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 Example

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 may run

slower but with less energy consumption or reliability cost when executed on one type of FUs than on another. 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 execution 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 [28]. 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 [29]. 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.

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, "$\surd$" is put into the right location and the value in the parentheses beside "$\surd$" 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

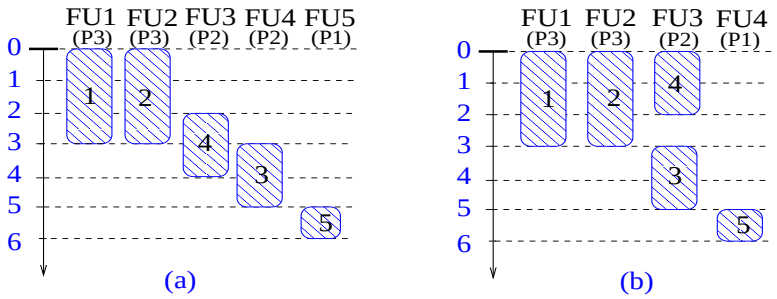


Figure 3: Two schedules corresponding to Assignment 2.

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 System Model

In our work, Data-Flow Graph (DFG) 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. A DFG may contain cycles to model a DSP application with loops. 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. Given an edge, $e = u \rightarrow v$, $d(e)$ means the data used as inputs in node v are generated by node u $d(e)$ iteration before. A *static* schedule of a cyclic DFG is a repeated pattern of an execution of the corresponding loop. And a static schedule must obey the precedence relations of the *directed acyclic graph (DAG)* portion of a DFG that is obtained by removing all edges with delays from the DFG. In this paper, the DAG part of a DFG is considered when we do assignment and scheduling.

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 function that computes the total cost such as $\sum_i c_j(i)^2$ as long as the function satisfies "associativity" property. This model can deal with the case when a FU type can only compute a subset of tasks. In such circumstances, we can set the execution time of a task as "infinite" if it can not be assigned to a FU type, where "infinite" means a value greater than the given timing constraint. Therefore, this node can not be assigned to this FU type in any assignment because it does not satisfy the timing constraint.

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 $I = \{I_1, I_2, \cdots, I_N\}$ in which for each item $I_i \in I$, $Q_i \in Z^+$ 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 $I = \{I_1, I_2, \cdots, I_N\}$ is an instance of *0-1 Knapsack Problem*. Set $M = 2$. Construct a simple path $G = \langle V, E, d \rangle$ 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 and set $d(e(u_i \rightarrow u_{i+1})) = 0$ 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, three other 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.

Theorem 5.1. $X_i[j]$ *(*$1 \leq i \leq N$*) obtained by* Algorithm Path_Assign *is the minimum system cost of the path from* u_1 *to* u_i *with total execution time* $\leq j$.

Proof. By induction. **Basic Step:** When $i = 1$, $X_0[j] = 0$ and $T_{min}(0) = 0$, so $X_1[j] = \min_{1 \leq k \leq M}\{c_k(1)$ if $j \geq t_k(1)\}$. Thus, When $i = 1$, Theorem 5.1 is true. **Induction Step:** We need to show that for $i \geq 1$,

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

Output: An optimal assignment for the simple path.

Algorithm:

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:

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

   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.

if $X_i[j]$ is the minimum system cost of the path from u_1 to u_i, then $X_{i+1}[j]$ is the minimum system cost of the path from u_1 to u_{i+1}. It is obviously true from equation 1. Thus, Theorem 5.1 is true for any i ($1 \leq i \leq N$). □

From Theorem 5.1, we know $X_N[L]$ 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 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

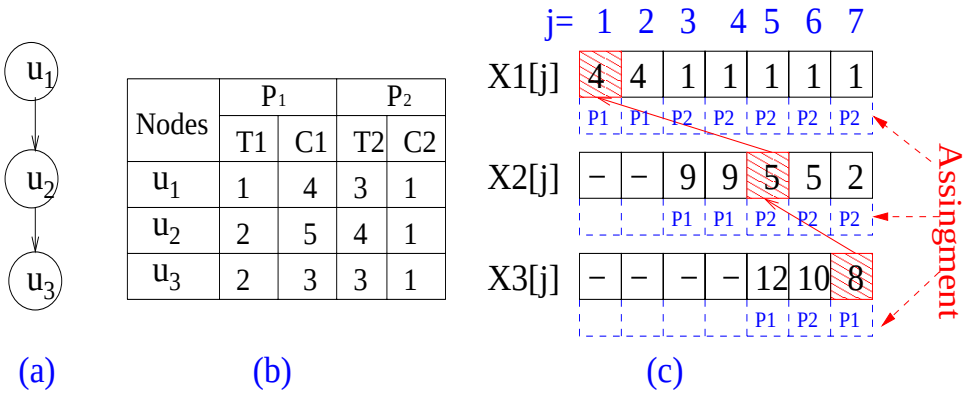


Figure 5: An example for a simple path with 2 FU types.

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.

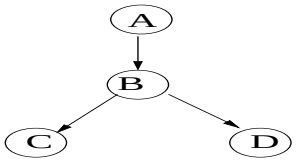


Figure 6: A given tree.

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, both $\{C, D, B, A\}$ and $\{D, C, B, A\}$ are post-ordering for the given tree in Figure 6. Here, sequences do not matter as long as post-ordering is

followed, since post-ordering is used to guarantee that, when we begin to process a node, the processing of all of its child nodes has already been finished in the algorithm. So, there is no difference between $\{C, D, B, A\}$ and $\{D, C, B, A\}$ in terms of our algorithm. The pseudo polynomial algorithm for trees, *Tree_Assign*, is shown in Figure 7.

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

Theorem 5.2. $X_i[j]$ *(*$1 \leq i \leq N$*) obtained by* Algorithm Tree_Assign *is the minimum system cost of the subtree rooted on* v_i *with total execution time* $\leq j$.

Proof. By induction. **Basic Step:** When $i = 1$, because the computation of $X_i[j]$ follows the post-ordering, v_1 must be a leaf node. Thus, $X_{1'}[j] = 0$. $X_0[j] = 0$ and $T_{min}(v_1) = 0$, so $X_1[j] = \min_{1 \leq k \leq M}\{c_k(1) \text{ if } j \geq t_k(1)\}$. Thus, When $i = 1$, Theorem 5.2 is true. **Induction Step:** We need to show that for $i \geq 1$, if $X_i[j]$ is the minimum system cost of the subtree rooted on v_i, then $X_{i+1}[j]$ is the minimum system cost of the subtree rooted on v_{i+1}. According to the post-ordering, the computation of $X_i[j]$ for each child node of v_{i+1} has been finished before computing $X_{i+1}[j]$. From equation 3, $X_{(i+1)'}[j]$ gets the summation of the minimum system cost of all child nodes of v_{i+1} because they can be allocated and executed simultaneously within time j. From equation 2, the minimum system cost is selected from all possible system costs caused by adding v_{i+1} into the subtree rooted on v_{i+1}. So $X_{i+1}[j]$ is the minimum system cost of the subtree rooted on v_{i+1}. Therefore, Theorem 5.2 is true for any i $(1 \leq i \leq N + 1)$. □

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.

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

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

Output: An optimal assignment for the tree.

Algorithm:

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:

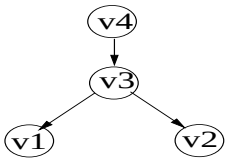
$$X_i[j] = \begin{cases} \min_{1 \leq k \leq M}\{X_{i'}[j - t_k(i)] + c_k(i) \\ \qquad \text{if } j - t_k(i) \geq T_{min}(v_i)\}, \\ \text{No feasible solution} \quad \text{Otherwise} \end{cases} \tag{2}$$

   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 $X_{N+1}[L]$.

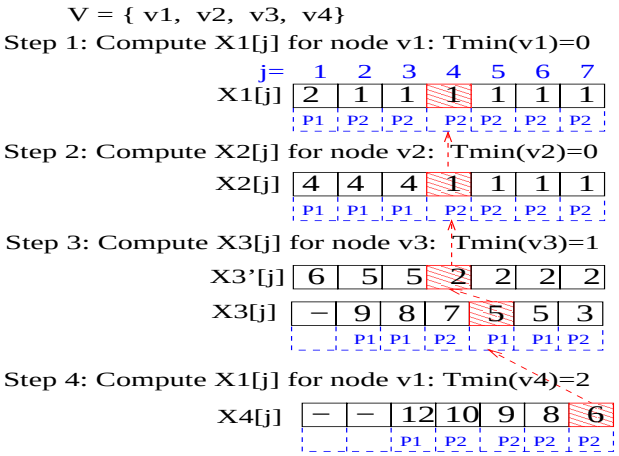
Figure 7: Algorithm Tree_Assign.



(a)

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

(b)



(c)

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

timing constraint is 7 time units. We process node v_1, v_2, v_3, and v_4 one by one following the order in V. In step 1, we compute $X_1[j]$ for node v_1. Since v_1 has no child nodes, $T_{min}(v_1) = 0$ (the minimum time needed to process the subtree rooted on v_1 except v_1 is 0) and $X_{1'}[j] = 0$ (from Equation 3). From the row corresponding to v_1 in Figure 8(b), we get $c_1(1) = 2, t_1(1) = 1, c_2(1) = 1, t_2(1) = 2$, therefore,

$$X_1[j] = \begin{cases} \min_{1\le k\le 2}\{c_k(1) \text{ if } j - t_k(1) \ge 0\} = \min\ \{\ 2 \text{ if } (j-1) \ge 0,\ \ 1 \text{ if } (j-2) \ge 0\}, \\ \text{No feasible solution} \quad \text{Otherwise} \end{cases}$$

Inputting j from 1 to 7, the value of $X_1[j]$ can be calculated based on this equation. For example, when $j = 1$, $X_1[1] = \min\ \{\ 2 \text{ if } (1-1) \ge 0,\ \ 1 \text{ if } (1-2) \ge 0\} = 2$. Then, the cost and its corresponding FU type, are recorded in $X_1[1]$ and the dashed-line box below $X_1[1]$, respectively, as shown in Figure 8(c). The calculation for node v_2 in step 2 is similar to that for node v_1 since node v_2 has no child nodes either.

In step 3, node v_3 is processed. It has two child nodes, v_1 and v_2. First, we calculate the execution time of the longest path of the subtree rooted by v_3 by assigning each node with the minimum execution time type in the subtree. $T_{min}(v_3) = 1$ is then obtained by removing v_3 from the longest path. Next, we compute $X_{3'}[j]$. From Equation 3, $X_{3'}[j] = X_1[j] + X_2[j]$ since v_1 and v_2 are all child nodes of v_3. After finishing calculation of $X_{3'}[j]$ and $T_{min(v_3)}$, we can get the equation to calculate $X_3[j]$ as follows:

$$X_3[j] = \begin{cases} \min_{1\le k\le 2}\{\ X_{3'}[j - t_k(3)] + c_k(3) \text{ if } (j - t_k(3)) \ge T_{min}(v_3)\ \} = \\ \qquad \min\{\ X_{3'}[j-1] + 3 \text{ if } j - 1 \ge 1,\ X_{3'}[j-3] + 1 \text{ if } j - 3 \ge 1\}, \\ \text{No feasible solution} \quad \text{Otherwise} \end{cases}$$

Inputting j from 1 to 7, we can obtain the value of $X_3[j]$. For example, given $j = 1$, no condition is satisfied as $1 - 1 < 1$ and $1 - 3 < 1$, so no feasible solution; given $j = 2$, $X_3[2] = \min\{X_{3'}[1] + 3\} = 6 + 3 = 9$. Similarly, in step 4, we can compute $X_4[j]$ for node v_4. After all nodes have been processed, the assignment for the DFG can be obtained by tracing how to reach $X_4[7]$ using similar method in Section 5.1. For this example, the optimal assignment is $P_2 \rightarrow v_1$, $P_2 \rightarrow v_2$, $P_1 \rightarrow v_3$, and $P_2 \rightarrow v_4$ as shown in Figure 8(c).

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 three heuristics: DFG_Assign_Once, DFG_Assign_Repeat, and DFG_Assign_CP, to solve the general *heterogeneous assignment problem*. The DFG_Assign_Once algorithm and the DFG_Assign_Repeat algorithm are based on the Tree_Assign algorithm, and the DFG_Assign_CP algorithm directly works on DFGs. In the following, we first show algorithm DFG_Assign_Once and DFG_Assign_Repeat, and then algorithm DFG_Assign_CP is given.

5.3.1 The DFG Algorithms Based on Trees

Algorithm DFG_Assign_Once and DFG_Assign_Repeat are designed 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.

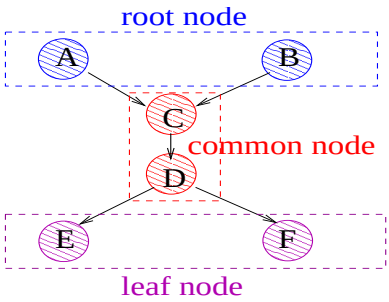


Figure 9: A DFG and its root, leaf and common nodes.

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.

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.

Input: a DFG G.

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

Algorithm:

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.

Algorithm DFG_Expand duplicates the common nodes with multiple parent nodes in a DFG from the bottom up based on a 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 unique

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

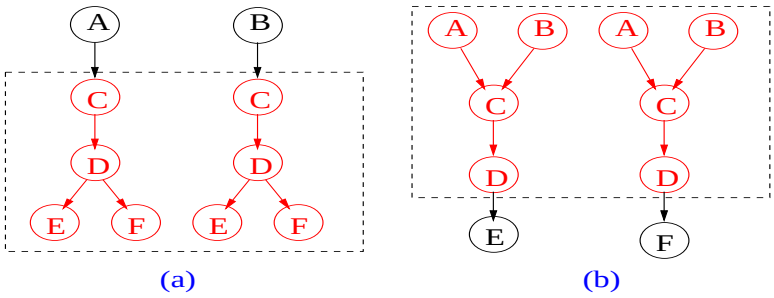


Figure 11: Two critical path trees for the DFG in Figure 9.

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.

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

Output: An assignment for the DFG.

Algorithm:

1. Let G^T be the transpose of G. Let G_{TR} and G_{TR}^T be the *critical path trees* obtained by running *Algorithm DFG_Expand* on G and G^T respectively. If $|V(G_{TR})| < |V(G_{TR}^T)|$, then $G_{TREE} = G_{TR}$; otherwise $G_{TREE} = G_{TR}^T$, 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.

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 assign-

ments 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.

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.

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

Output: An assignment for the DFG.

Algorithm:

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.

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. 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.

5.3.2 The DFG_Assign_CP Algorithm

Algorithm DFG_Assign_Once and DFG_Assign_Repeat are based on algorithm Tree_Assign. Since Algorithm Tree_Assign can obtain optimal solution for trees, DFG_Assign_Once and DFG_Assign_Repeat can achieve near-optimal solution. However, their computation time is related to the size of expanded trees so it takes longer time when the expanded tree is larger. Therefore, in this section, we propose another heuristic algorithm, DFG_Assign_CP, to solve this problem. Algorithm DFG_Assign_CP directly works on DFGs, where CP means Critical Path because we only pick one node from a critical path and change its type in each iteration in this algorithm. And the basic idea is to first assign each node with the best cost type and then iteratively change the type of the node in the critical path such that the timing constraint can be satisfied with the minimal cost increased. The algorithm is shown in Figure 14.

In Algorithm DFG_Assign_CP, we first assign the best cost type to each node and mark the type as assigned. We then find a critical path (CP) that has the maximum execution time among all possible paths based on the current assigned types for DFG G. Next, if the execution time of the critical path is greater than TC, the given timing constraint, we reduce it by changing the type of a node that is selected from the critical path. The node is selected from the critical path, since only the nodes in the critical path can influence the longest execution time of a DFG. For all other nodes that are not in the critical path, we want to keep their current types since they have been assigned the best types initially.

We select the node in the critical path base on a ratio. This ratio is calculated for each node and its each unmarked type as follows:

Input: M different types of FUs, DFG G=$\langle V, E, d\rangle$, and timing constraint TC.

Output: An assignment for the DFG.

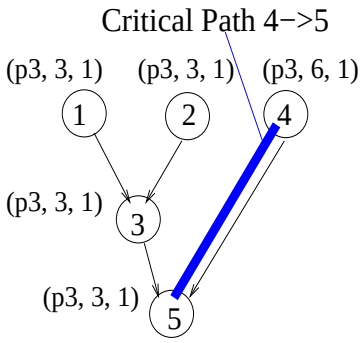
Algorithm:

1. For each node $v_i \in V$, assign type p to v such that v_i has the minimal cost among its all possible types; mark type t of node v_i.
2. Find CP, a critical path of G.
3. If (Time(CP) $>$ TC) then {
   (a) Find_Flag $\leftarrow$ NO; Min_Ratio$\leftarrow+\infty$;
   (b) For each node v_i in critical path CP and each unmarked type p of node v_i {
      i. Let $CTime$ and $CCost$ be the time and cost of v_i with current assigned type, respectively;
      ii. Let $PTime$ and $PCost$ be the time and cost of v_i with type p, respectively;
      iii. ReducedTime $\leftarrow$ $CTime$-$PTime$; IncreasedCost $\leftarrow$ $PCost$-$CCost$;
      iv. If ReducedTime $>$ 0, then Ratio $\leftarrow$ IncreasedCost/ReducedTime; else Ratio $\leftarrow+\infty$;
      v. If Min_Ratio $>$ Ratio, then set Min_Ratio $\leftarrow$ Ratio; Min_Node $\leftarrow$ i; Min_Type $\leftarrow$ p; Find_Flag $\leftarrow$ YES;
   }
   (c) If (Find_Flag == YES) {
      i. Change the type of node Min_Node to be Min_Type;
      ii. Mark type Min_Type of node Min_Node;
      iii. Goto Step 2.
   }else{ Output "No Solution"; exit.}
   }
4. Output the assignment of G.

Figure 14: Algorithm DFG_Assign_CP.

$$Ratio \leftarrow IncreasedCost/ReducedTime$$

Given node u_i in the critical path and its unmarked type p, ReducedTime and IncreasedCost are the corresponding reduced time and increased cost if u_i is changed from its current assigned type to type p. This ratio is used to represent the average increased cost per reduced time unit. Since we want to reduce the execution time with the minimal cost increased, we pick the node with the minimal ratio among all nodes with all possible unmarked types in the critical path. We keep the record of the node and its type that has the minimal ratio during processing. After the node and the type have been found, we change the node to this type and mark this type as assigned. The procedure is repeated until the timing constraint is satisfied or we can not reduce the execution time of the critical path any more.



(a)

The minimal ratio

Nodes	Current Type	Unmarked Types	Ratios for unmarked types
4	P3	P1, P2	P1: (7−1)/(6−1)=6/5 P2: (2−1)/(6−2)=1/4
5	P3	P1, P2	P1: (8−1)/(3−1)=7/2 P2: (5−1)/(3−2)=4

(b)

Figure 15: The ratio calculation by the DFG_Assign_CP algorithm for the given DFG in Figure 1. (a) All nodes are first assigned P_3 (the best cost type). (b) The ratios for each node in the critical path and its each unmarked type .

An example is given in Figure 15 that shows how to obtain the minimal ratio among all nodes with all unmarked types in the critical path in the first iteration using our algorithm. The inputs are the DFG and FU types shown in Figure 1 and the timing constraint is 5 time units. In Figure 15-(a), all nodes are first assigned to type P_3 that gives them the minimal costs. The 3-tuple (p,t,c) above each node in Figure 15-(a) denotes the current assigned type p, the execution time t, and the cost c. Figure 15-(b) shows the calculation of the ratios for all nodes in the critical path. We can see that node 4 has the minimal ratio 1/4 with type P_2, so the type of node 4 will be changed to P_2.

Given a DFG G, let $|E|$ and $|V|$ be the number of edges and the number of nodes, respectively. It takes $O(|E|)$ to find a critical path for a given DFG using the clock period computation algorithm in [32].

In our algorithm, in each iteration, it takes at most $O(|V| * M)$ to calculate ratios, select a node from the critical path and change its type, where M is the number of FU types. And the algorithm iterates at most $|V| * M$ times, since each type of a node is only assigned one time. Therefore, the time complexity of our algorithm is $O(|V| * M * (|E| + |V| * M))$. Considering M is a constant, our algorithm takes $O(|V| * |E| + |V|^2)$.

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 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.

Input: A DFG with FU assignments and timing constraint L.

Output: Lower bound for each FU type

Algorithm:

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,
$$LB_{ASAP}[j] \leftarrow \max\{\frac{N_{ASAP}[1][j]}{1}, \frac{N_{ASAP}[1][j] + N_{ASAP}[2][j]}{2}, \cdots, \frac{\sum_{1 \le k \le L} N_{ASAP}[k][j]}{L}\}$$
5. For each FU type j,
$$LB_{ALAP}[j] \leftarrow \max\{\frac{N_{ALAP}[L][j]}{1}, \frac{N_{ALAP}[L][j] + N_{ALAP}[L-1][j]}{2}, \cdots, \frac{\sum_{1 \le k \le L} N_{ALAP}[L-k+1][j}{L}\}$$
6. For each FU type j, its lower bound: $LB[j] \leftarrow \max\{LB_{ASAP}[j], LB_{ALAP}[j]\}$.

Figure 16: Algorithm Lower_Bound_RC.

Algorithm Lower_Bound_RC is shown in Figure 16. In the algorithm, it counts the total number of each FU type in each control step in the ASAP (As Soon As Possible) and ALAP (As Late As Possible) 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.

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

Output: A schedule and a configuration.

Algorithm:

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 $S \leftarrow S + 1$;

   $\}$ **While (** $S \leq L$ **);**

Figure 17: Algorithm Min_RC_Scheduling.

Using the lower bound of each FU as an initial configuration, *Algorithm Min_RC_Scheduling* (Figure 17) 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.

Algorithm Lower_Bound_RC and *Min_RC_Scheduling* both take $O(|V|+|E|)$ to get results, where $|V|$ is the number of nodes and $|E|$ is the number of edges for a given DFG.

7 Experiments

Bench.	Tree or DFG	# of Nodes	# of Duplicated Nodes	# of Nodes in G_Tree
4-lat-iir	Tree	26	-	-
8-lat-iir	Tree	42	-	-
voltera filter	Tree	27	-	-
Diff. Eq. Solver	DFG	11	3	15
RLS-laguerre filter	DFG	19	3	32
elliptic filter	DFG	34	19	189

Table 1: The basic information for the benchmarks.

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 are DFGs. The basic information about the benchmarks is shown in Table 1, 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 [31] and the ILP model in [9]. 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 shown in Table 2 and Table 3 are finished in less than one second and the longest running time is 0.11 seconds which occurs when Algorithm DFG_Assign_Repeat is experimented on Elliptic Filter with timing constraint 100 time units.

The experimental results for 4-stage lattice filter, 8-stage lattice filter, and voltera filter, are shown in Table 2. In the table, Column "TC" presents the given timing constraint. The system costs obtained from different algorithms: *the greedy algorithm*, *Algorithm DFG_Assign_CP*, *Algorithm Tree_Assign*, *the ILP model*, *Algorithm DFG_Assign_Once*, and *Algorithm DFG_Assign_Repeat*, are presented in corresponding Column "cost" under different fields, "Greedy", "ILP", "DFG_Assign_CP", and "Tree_Assign, DFG_Assign_Once, DFG_Assign_Repeat, ILP". Algorithm Tree_Assign, the ILP model, DFG_Assign_Once, and DFG_Assign_Repeat, obtain optimal solutions since the graphs of these filters are trees, so their re-

Benchmark	TC	Greedy ([31])	DFG_Assign_CP		Tree_Assign, DFG_Assign_Once, DFG_Assign_Repeat, ILP ([9])		
		cost	cost	%(CP)	cost	%(Tree)	Configuration
	31	286	**193**	**32.5**	**191**	**33.2**	$2\,P_1\,2\,P_2\,5\,P_3$
	35	210	**162**	**22.9**	**159**	**24.3**	$1\,P_1\,4\,P_2\,5\,P_3$
	40	201	**151**	**24.9**	**149**	**25.9**	$2\,P_1\,2\,P_2\,5\,P_3$
4-stage	45	192	**145**	**24.5**	**140**	**27.1**	$1\,P_1\,2\,P_2\,5\,P_3$
Lattice	50	189	**133**	**29.6**	**133**	**29.6**	$1\,P_1\,2\,P_2\,5\,P_3$
Filter	55	184	**128**	**30.4**	**126**	**31.5**	$1\,P_1\,2\,P_2\,4\,P_3$
	60	184	**120**	**34.8**	**120**	**34.8**	$1\,P_1\,2\,P_2\,4\,P_3$
	65	176	**116**	**34.1**	**115**	**34.7**	$1\,P_1\,2\,P_2\,4\,P_3$
	70	170	**111**	**34.7**	**111**	**34.7**	$1\,P_1\,2\,P_2\,4\,P_3$
	75	168	**111**	**33.9**	**107**	**36.3**	$1\,P_1\,1\,P_2\,4\,P_3$
	55	490	**325**	**33.7**	**325**	**33.7**	$2\,P_1\,1\,P_2\,6\,P_3$
	60	369	**296**	**19.8**	**291**	**21.1**	$1\,P_1\,2\,P_2\,5\,P_3$
	65	350	**296**	**15.4**	**273**	**22.0**	$2\,P_1\,1\,P_2\,5\,P_3$
8-stage	70	340	**254**	**25.3**	**252**	**25.9**	$1\,P_1\,2\,P_2\,5\,P_3$
Lattice	75	329	**241**	**26.7**	**237**	**28.0**	$1\,P_1\,1\,P_2\,5\,P_3$
Filter	80	323	**234**	**27.6**	**226**	**30.0**	$1\,P_1\,1\,P_2\,5\,P_3$
	85	313	**224**	**28.4**	**215**	**31.3**	$1\,P_1\,1\,P_2\,4\,P_3$
	90	313	**218**	**30.4**	**207**	**33.9**	$1\,P_1\,1\,P_2\,4\,P_3$
	95	301	**204**	**32.2**	**199**	**33.9**	$1\,P_1\,1\,P_2\,4\,P_3$
	100	295	**200**	**32.2**	**193**	**34.6**	$1\,P_1\,1\,P_2\,4\,P_3$
	37	324	**245**	**24.4**	**243**	**25.0**	$2\,P_1\,2\,P_2\,3\,P_3$
	40	324	**219**	**32.4**	**214**	**34.0**	$2\,P_1\,2\,P_2\,4\,P_3$
	45	233	**211**	**9.4**	**191**	**18.0**	$1\,P_1\,3\,P_2\,3\,P_3$
Voltera	50	219	**177**	**19.2**	**175**	**20.1**	$2\,P_1\,2\,P_2\,4\,P_3$
Filter	55	211	**166**	**21.3**	**155**	**26.5**	$1\,P_1\,3\,P_2\,3\,P_3$
	60	209	**148**	**29.2**	**145**	**30.6**	$1\,P_1\,3\,P_2\,3\,P_3$
	65	209	**142**	**32.1**	**136**	**34.9**	$1\,P_1\,2\,P_2\,4\,P_3$
	70	197	**142**	**27.9**	**128**	**35.0**	$1\,P_1\,2\,P_2\,3\,P_3$
	75	191	**128**	**33.0**	**121**	**36.6**	$1\,P_1\,1\,P_2\,4\,P_3$
	80	177	**119**	**32.8**	**115**	**35.0**	$1\,P_1\,1\,P_2\,4\,P_3$

Table 2: Comparison of the system costs for 4-stage lattice filter, 8-stage lattice filter, and voltera filter, with different time constraints.

Benchmark	TC	Greedy ([31])	ILP ([9])	DFG_Assign_Once		DFG_Assign_Repeat		DFG_Assign_CP		
		cost	cost	cost	%(Once)	cost	%(Repeat)	cost	%(CP)	Configuration
	21	132	**111**	**111**	**15.9**	**111**	**15.9**	**111**	**15.9**	2 P_1 1 P_2 2 P_3
	25	128	**91**	**101**	**21.1**	**97**	**24.2**	**107**	**16.4**	1 P_1 1 P_2 2 P_3
	30	91	**63**	**81**	**11.0**	**81**	**11.0**	**76**	**16.5**	1 P_1 1 P_2 2 P_3
	35	74	**54**	**60**	**18.9**	**60**	**18.9**	**60**	**18.9**	1 P_1 2 P_2 2 P_3
Diff.	40	74	**45**	**45**	**39.2**	**45**	**39.2**	**45**	**39.2**	0 P_1 2 P_2 2 P_3
Eq.	45	74	**40**	**40**	**45.9**	**40**	**45.9**	**41**	**44.6**	0 P_1 1 P_2 2 P_3
Solver	50	74	**39**	**39**	**47.3**	**39**	**47.3**	**41**	**44.6**	0 P_1 1 P_2 2 P_3
	55	74	**38**	**38**	**48.6**	**38**	**48.6**	**41**	**44.6**	0 P_1 1 P_2 2 P_3
	60	56	**36**	**39**	**30.4**	**36**	**35.7**	**41**	**26.8**	0 P_1 1 P_2 2 P_3
	65	56	**35**	**35**	**37.5**	**35**	**37.5**	**35**	**37.5**	0 P_1 1 P_2 2 P_3
	23	204	**155**	**157**	**23.0**	**155**	**24.0**	**156**	**23.5**	1 P_1 3 P_2 3 P_3
	25	188	**139**	**141**	**25.0**	**139**	**26.1**	**146**	**22.3**	1 P_1 3 P_2 4 P_3
	30	156	**117**	**117**	**25.0**	**117**	**25.0**	**120**	**23.1**	1 P_1 2 P_2 4 P_3
RLS-	35	137	**104**	**122**	**10.9**	**104**	**24.1**	**104**	**24.1**	1 P_1 1 P_2 4 P_3
laguerre	40	112	**98**	**101**	**9.8**	**98**	**12.5**	**100**	**10.7**	0 P_1 1 P_2 4 P_3
Lattice	45	98	**92**	**93**	**5.1**	**92**	**6.1**	**92**	**6.1**	0 P_1 1 P_2 4 P_3
Filter	50	98	**90**	**93**	**5.1**	**90**	**8.2**	**90**	**8.2**	0 P_1 1 P_2 4 P_3
	55	96	**86**	**90**	**6.3**	**90**	**6.3**	**86**	**10.4**	0 P_1 1 P_2 4 P_3
	60	96	**85**	**85**	**11.5**	**85**	**11.5**	**86**	**10.4**	0 P_1 1 P_2 3 P_3
	65	88	**83**	**83**	**5.7**	**83**	**5.7**	**86**	**2.3**	0 P_1 1 P_2 3 P_3
	57	399	**318**	**378**	**5.3**	**318**	**20.3**	**320**	**19.8**	2 P_1 2 P_2 2 P_3
	60	395	**279**	**350**	**11.4**	**288**	**27.1**	**297**	**24.8**	2 P_1 2 P_2 2 P_3
	65	371	**247**	**309**	**16.7**	**247**	**33.4**	**249**	**32.9**	2 P_1 2 P_2 3 P_3
Elliptic	70	328	**228**	**278**	**15.2**	**229**	**30.2**	**231**	**29.6**	2 P_1 2 P_2 3 P_3
Filter	75	296	**211**	**249**	**15.9**	**224**	**24.3**	**215**	**27.4**	2 P_1 3 P_2 2 P_3
	80	293	**199**	**236**	**19.5**	**208**	**29.0**	**206**	**29.7**	2 P_1 3 P_2 2 P_3
	85	262	**185**	**223**	**14.9**	**192**	**26.7**	**197**	**24.8**	1 P_1 2 P_2 2 P_3
	90	236	**172**	**192**	**18.6**	**179**	**24.2**	**181**	**23.3**	1 P_1 3 P_2 2 P_3
	95	228	**157**	**184**	**19.3**	**169**	**25.9**	**169**	**25.9**	1 P_1 2 P_2 3 P_3
	100	209	**155**	**174**	**16.7**	**157**	**24.9**	**160**	**23.4**	1 P_1 2 P_2 3 P_3

Table 3: Comparison of the system costs for Differential Equation Solver, RLS-laguerre Lattice Filter, and Elliptic filter, with different time constraints.

sults are put under the same column (Column "cost" under Field "Tree_Assign, DFG_Assign_Once, DFG_Assign_Repeat, ILP"). 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 that x_1 FUs with type P_1, x_2 FUs with type P_2, and x_3 FUs with type P_3 are used. *"Greedy"* is the greedy algorithm implemented based on the idea in [31], and *"ILP"* is the ILP model from [9]. Column "%(CP)" under "DFG_Assign_CP" and Column "%(Tree)" under "Tree_Assign, DFG_Assign_Once, DFG_Assign_Repeat, ILP" show the percentage of reduction on system cost, respectively, comparing with the results obtained by *the greedy algorithm*. The average percentage reduction for DFG_Assign_CP is 27.9% and that for Tree_Assign is 30.1%.

The experimental results for differential equation solver, RLS-laguerre lattice filter and elliptic filter, are shown in Table 3. In the table, each column "cost" under different fields presents the system costs obtained from different algorithms: *the greedy algorithm* (Field "Greedy"), *the ILP model* (Field "ILP"), *DFG_Assign_Once* (Field "DFG_Assign_Once"), *DFG_Assign_Repeat* (Field "DFG_Assign_Repeat"), and *DFG_Assign_CP* (Field "DFG_Assign_CP"). The configurations corresponding to the assignment obtained by *Algorithm DFG_Assign_CP* are shown in Column "Configuration" under "DFG_Assign_CP". Column "%(Once)" under "DFG_Assign_Once", column "%(Repeat)" under "DFG_Assign_Repeat" and column "%(CP)" under "DFG_Assign_CP" present the percentage of reduction on system cost, respectively, compared to *the greedy algorithm*. The average percentage reduction for DFG_Assign_Once is 19.9%, that for DFG_Assign_Repeat is 24.7%, and that for DFG_Assign_CP is 23.6%. The experimental results also show that all of these three heuristic algorithms achieve near-optimal results for various benchmarks compared with the optimal results obtained by the ILP model.

Although the ILP model from [9] can obtain optimal solution for heterogeneous assignment problem, it is NP-hard problem to solve the ILP model. Therefore, the ILP model may take very long time to get results even when a given DFG is not very big. We unfold 4-stage Lattice Filter and perform experiments to compare time and system cost for the ILP model, DFG_Assign_Once, DFG_Assign_Repeat, and DFG_Assign_CP. The timing constraint is 200 time units. The experimental results are shown in Table 4, in which 4-stage Lattice Filter is unfolded from 10 to 17 times. From the experimental results, we can see the ILP algorithm takes much bigger time to get results compared with our three algorithms. When the unfolding factor is 17, it can not get the result even after 5 days. Among our three algorithms,

DFG_Assign_CP performs best with minimum cost and less time.

4-stage Lattice IIR Filter with unfolding factors from 10 to 17 (Timing Constraint is 200 time units)									
Unfolding	Node	ILP ([9])		DFG_Assign_Once		DFG_Assign_Repeat		DFG_Assign_CP	
Factor	Num	time (s)	cost	time (s)	cost	time (s)	cost	time (s)	cost
10	260	**5.92**	**910**	**0.59**	**910**	**3.8**	**910**	**0.01**	**910**
11	286	**11.05**	**1003**	**0.89**	**1003**	**5.82**	**1003**	**0.02**	**1003**
12	312	**22.07**	**1097**	**1.37**	**1099**	**8.47**	**1098**	**0.02**	**1098**
13	338	**71.07**	**1193**	**2.13**	**1196**	**13.56**	**1194**	**0.03**	**1194**
14	364	**270.42**	**1291**	**3.06**	**1302**	**22.35**	**1299**	**0.04**	**1292**
15	390	**1570.26**	**1391**	**4.55**	**1411**	**34.32**	**1407**	**0.05**	**1392**
16	416	**46869.11**	**1494**	**6.96**	**1530**	**54.99**	**1526**	**0.08**	**1501**
17	442	**-**	**-**	**9.49**	**1656**	**92.21**	**1645**	**0.11**	**1609**

Table 4: Comparison of the system cost and time for the ILP model [9] and our three DFG algorithms.

Through the experimental results from Table 2-4, we found that our algorithms have better performance compared with *the greedy algorithm*. While DFGs become too big for the ILP model to solve, our algorithms can still efficiently give results. When a given DFG is a tree, *algorithm Tree_Assign* is recommended to be used, which generates optimal results. To solve the general problem, *DFG_Assign_CP* is recommended to be used, which gives near-optimal results with less time compared with *DFG_Assign_Once* and *DFG_Assign_Repeat*.

8 Conclusion

We have proposed a two-phase approach for real-time digital signal processing applications to perform high-level synthesis of special purpose architectures using heterogeneous functional units. 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 (may relate to power, reliability, etc.) 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, 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, three other algorithms, *Algorithm DFG_Assign_Once*, *Algorithm DFG_Assign_Repeat*, and *Algorithm DFG_Assign_CP*, are proposed. *Algorithm DFG_Assign_CP* is the best especially when the input graph is large.

References

[1] P. Gl 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*, vol. 8, pp. 661–679, June 1989.

[2] H. De Man, F. Catthoor, G. Goossens, J. Vanhoof, S. Note J.L. Van Meerbergen, and J. A. Huisken, "Architecture-driven synthesis techniques for vlsi implementation of dsp algorithms," *Proceedings of the IEEE*, vol. 78, no. 2, pp. 319–335, 1990.

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

[4] K.Parhi and D.G.Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via. optimum unfolding," *IEEE Trans. on Computers*, vol. 40, pp. 178–195, Feb. 1991.

[5] C.-T. Hwang, J.-H. Lee, and Y.-C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, pp. 464–475, April 1991.

[6] C. H. Gebotys and M. Elmasry, "Global optimization approach for architectural synthesis," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, pp. 1266–1278, Sep. 1993.

[7] 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*, vol. 14, pp. 274–295, March 1995.

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

[9] 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.

[10] C.-Y. Wang and K. K. Parhi, "Resource constrained loop list scheduler for dsp algorithms," *Journal of VLSI Signal Processing*, vol. 11, pp. 75–96, Oct./Nov. 1995.

[11] Yun-Nan Chang, Ching-Yi Wang, and Keshab K. Parhi, "Loop-list scheduling for heterogeneous functional units," in *6th Great Lakes Symposium on VLSI*, March 1996, pp. 2–7.

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

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

[14] 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.

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

[16] O. Beaumont, A. Legrand, and Y. Robert, "Static scheduling strategies for heterogeneous systems," *Computing and Informatics*, vol. 21, pp. 413–430, 2002.

[17] O. Beaumont, A. Legrand, and Y. Robert, "The master-slave paradigm with heterogeneous processors," *IEEE Trans. Parallel Distributed Systems*, vol. 14, no. 9, pp. 897–908, 2003.

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

[19] O. Beaumont, A. Legrand, and Y. Robert, "Scheduling divisible workloads on heterogeneous platforms," *Parallel Computing*, vol. 29, pp. 1121–1152, 2003.

[20] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Scheduling strategies for mixed data and task parallelism on heterogeneous clusters," *Parallel Processing Letters*, vol. 13, no. 2, pp. 225–244, 2003.

[21] 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*, Dec. 1997, vol. 46, pp. 1338–1356.

[22] 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.

[23] A. Legrand, H. Renard, Y. Robert, and F. Vivien, "Mapping and load-balancing iterative computations on heterogeneous clusters with shared links," *IEEE Trans. Parallel Distributed Systems*, vol. 15, no. 6, pp. 546–558, 2004.

[24] 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*, Apr. 1990, vol. 1, pp. 184–194.

[25] 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*, June 1992, pp. 452–459.

[26] O. Beaumont, A. Legrand, L. Marchal, and Y. R obert, "Pipelining broadcasts on heterogeneous platforms," in *International Parallel and Distributed Processing Symposium IPDPS'2004*. 2004, IEEE Computer Society Press.

[27] 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*, vol. 13, pp. 308–323, March 2002.

[28] S. Srinivasan and N.. K. Jha, "Safety and reliability driven task allocation in distributed systems," *IEEE Trans. on Parallel and Distributed Systems*, vol. 10, pp. 238–251, March 1999.

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

[30] 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.

[31] 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*, vol. 12, pp. 1147–1156, Aug. 1993.

[32] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.