Multiprocessor Implementation of Real-Time DSP Algorithms

Duen-Jeng Wang and Yu Hen Hu, *Senior Member, IEEE*

***Abstract*—In this paper, we consider multiprocessor implementation of real-time recursive digital signal processing algorithms. The objective is to devise a periodic schedule, and a fully static task assignment scheme to meet the desired throughput rate while minimizing the number of processors. Toward this goal, we propose a notion called *cutoff time*. We prove that the minimum-processor schedule can be found within a finite time interval bounded by the cutoff time. As such the complexity of the scheduling algorithm and the allocation algorithm can be significantly reduced. Next, taking advantage of the cutoff time, we derive efficient heuristic algorithms which promise better performance and less computation complexity compared to other existing algorithms. Extensive benchmark examples are tested which yield most encouraging results.**

I. Introduction

IN REAL-TIME digital signal processing applications, the required throughput rate often exceeds that of a single processing unit, and hence requires the use of an application specific multiprocessor system. The purpose is to exploit the parallelism inherent in the DSP algorithm so as to expedite the computation. For example, in high definition television systems, 2 million 24 b pixels are to be processed in 1/30 s. Other applications which requires ultra high throughput rate include digital library, virtual reality, as well as video compression.

The problem of deriving a minimum-processor implementation for *recursive DSP algorithms* with real-time constraint is considered in this paper. We assume each processor is capable of sequentially executing the assigned tasks according to a specified schedule. We further assume that when a processor needs to access data computed in another processor, dedicated data bus connecting these two processors will be available. Hence during the subsequent discussion, we will assume that the delay due to interprocessor communication is negligible compared to the computing time of each task. The argument that dedicated data bus may be used is reasonable for situations such as high level synthesis of multimodule DSP processing elements where on-chip parallel data buses can be afforded. The assumption of negligible interprocessor communication delay will be inadequate when the DSP algorithm is to be realized on a general-purpose message passing based distributed memory parallel processors.

Real-time constraint is defined as the *data sampling period*, the interval between the arrivals of two consecutive data samples, can not be smaller than the *initiation interval*, the interval between two consecutive initiations of the DSP algorithm. One popular design style to implement real-time DSP algorithms called *periodic schedule and fully static task assignment* (PSFS) [1] is adopted. With a PSFS implementation, we need only to schedule and assign the operations in a single iteration of the DSP algorithm. Operations in other iterations will have the same relative schedule measured against the starting time of each iteration, and exactly the same processor assignment.

To fulfill the goal, two questions are asked. First, how do we know the given data sampling period is large enough to have a PSFS implementation for the DSP algorithm? In a recursive DSP algorithm, one or more output variables at the present iteration depend on their values computed in previous iterations. Due to the cyclic dependent relationship, there is a lower bound on the initiation interval for a given recursive DSP algorithm [2]. *Look-ahead transformation* [3] can be used to reduce the low bound on initiation interval. However, look-ahead transformation alone can not guarantee the existence of a PSFS implementation [4].

Second, how to find a minimum-processor PSFS implementation when the existence of a PSFS for a given data sampling period is guaranteed. In order to find the minimum-processor PSFS implementation, all the possible PSFS implementations have to be searched. However, due to no constraint on how long one iteration of the DSP algorithm has to finish, there are possibly infinite number of possible PSFS implementations.

In this paper, we will present *Architectural synthesis for Real-Time DSP* (ART). It consists of unfolding algorithm transformation and a novel scheduling and allocation method. Earlier, we have proposed a criterion called *generalized perfect rate graph* (GPRG) [5]. We proved that a PSFS implementation which meets the real-time constraint exists if and only if the recursive DSP algorithm is a GPRG. Any recursive DSP algorithm can be transformed to a GPRG by unfolding at least *minimum collapsing factor* times [5].

Below, we will concentrate on how ART derives a minimum-processor PSFS implementation for a given GPRG. To reduce the number of possible PSFS implementations searched, we will first propose a method to compute the *cutoff time* for the given GPRG and a prespecified data throughput

Manuscript received June 9, 1993; revised May 12, 1994 and December 15, 1994.

The authors are with Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706 USA.

IEEE Log Number 9413458.

rate. We will show that the optimal PSFS implementation can be found among all the possible PSFS implementations within the interval of cutoff time as determined. Having derived the cutoff time, we proceed to develop a novel two-phase algorithm in this paper: First we derive a minimum-overlapped periodic schedule using an artificial intelligence approach called planning [6]. Then, based on this schedule, we assign each task to a processor to derive a fully static assignment scheme while minimizing the total number of processing elements. This is achieved using a graph coloring algorithm [7]. Compared with previously reported algorithms, our method is faster, and produces better results when tested with an extensive set of benchmark examples.

We now briefly survey several relevant works: Nonoverlapped PSFS realization used by [8]–[11] takes advantage from the well investigated conventional multiprocessor scheduling problems [12], where the tasks to be scheduled get executed only once. Many scheduling heuristics, such as critical path scheduling, are available. Algorithm transformation methods such as retiming [11], [13], software pipelining [9], loop winding [8], and loop folding [10] are used to minimize the initiation interval under the resource constraint, namely, a fixed number of processors. In general, retiming alone can not guarantee the existence of a PSFS implementation for a given initiation interval [4]. Incorporating node decomposition, in [13] a method to derive a nonoverlapped PSFS implementation for any given initiation interval is reported. However, for fine grain flow graphs where each node is not decomposable, this scheme is not applicable.

In [14]–[16], a cyclo-static realization for multiprocessor implementation of DSP algorithms is adopted. A *principal lattice vector* is used to denote the displacement in the space-time coordinate of successive iterations. However, an efficient way to determine the optimal principal lattice vector does not yet exist. Furthermore, the cyclo-static scheduling algorithm for a given principal lattice vector, as proposed in [16], has an exponentially growing worst case computational complexity.

Renfors and Neuvo [2] derived a PSFS implementation by exploring the maximal spanning tree from the signal flow graph (SFG). No attempt was made to minimize the number of processors, however. In [4], Parhi reported that a PSFS implementation for any initiation interval can be guaranteed by unfolding the DSP algorithm to a so-called *perfect-rate algorithm*. They further developed an architectural synthesis tool, called *MARS* [17], which comprised a heuristic scheduling algorithm and unfolding, retiming algorithm transformation. As shown in [5], the definition of perfect-rate algorithm may require excessive number of unfolding. Their scheduling algorithm has nonpolynomial-time worst case complexity. Recently, a range-chart-guided scheduling algorithm has been proposed [18], which deals with both performance constrained and resource constrained scheduling problems. The scheduling range for each node used in the range-chart algorithm is heuristically determined which may exclude the optimal schedule or induce extra scheduling overhead.

We note that there are other works [19]–[22] concerning the implementation of linear recurrence systems using a multiprocessor system have been reported. However, these work considered only finite length data stream which are readily available at the beginning of the execution. There are no real time throughput rate constraint to match. As such, none of these methods will be applicable to solve the problem discussed in this paper.

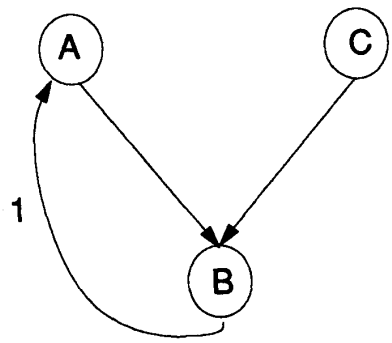


Fig. 1. An example ICDG.

The remaining sections of this paper are organized as follows. In Section II, the concept of GPRG is reviewed. In Section III, an algorithm to determine a cutoff time will be presented. The scheduling and task assignment algorithm will be described in Section IV. In Section V, the results of some benchmark examples are reported.

II. Generalized Perfect Rate Graph

A recurrent DSP algorithm can be formulated as an infinite *DO* loop [1]. The loop body, which corresponds to the operations to process incoming data samples, can be represented graphically by an *iterative computational dependence graph, (ICDG)*, $G = (N, E)$, where N is a set of nodes corresponding to operations of the algorithm, and E is a set of arcs representing data dependencies. If an operation i of the xth iteration depends on the results from operation j of the yth iteration, the dependence arc is labeled with a *dependence distance* $\delta_{i,j} = x - y$. In Fig. 1, we have $\delta_{B,A} = 1$ which signifies the dependence of $X[i]$ (node A) on $Y[i-1]$ (node B). The label of dependence distance is omitted when $\delta_{i,j} = 0$. These directed dependence arcs may form one or more cycles in the ICDG. For each cycle C in the ICDG, we denote *cycle computing time* $\mathcal{T}_C = \sum_{i \in C} \tau_i$ where τ_i is the computation delay in node i. Also, we denote $\Delta_C = \sum_{(i,j) \in C} \delta_{i,j}$ to be the total dependence distance in cycle C. For a computable iterative algorithm, we must have $\Delta_C > 0$ for all the cycles in its ICDG. Due to cyclic dependent relationship, there is a theoretical lower bound on the initiation interval. It is defined as

$$I_{\min}(G) = \max_{\forall C \in G} \frac{\mathcal{T}_C}{\Delta_C}. \tag{1}$$

It has been shown in [4] that for any ICDG there may not exist a PSFS implementation for any given $d \geq I_{\min}$. To solve the problem, in [5] we proved the following theorem:

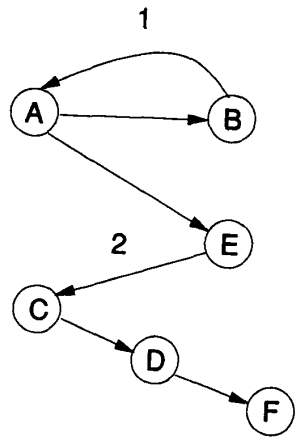


Fig. 2. A GPRG example where the delay of each node is $\tau_A = 2$, $\tau_B = 2$, $\tau_C = 2$, $\tau_D = 2$, $\tau_E = 3$, and $\tau_F = 3$.

Theorem 1: Given a cyclic ICDG $G = (N, E)$ and a desired initiation interval $d \geq I_{\min}(G)$, a PSFS implementation of G exists if and only if

$$\max_{\forall\ n \in N} \tau_n \leq d. \tag{2}$$

An ICDG which satisfies 2 is called *Generalized Perfect-Rate Graph* (GPRG) with respect to d.

Every ICDG can be transformed into a GPRG by unfolding. In [5] we showed that an ICDG can be converted into its corresponding GPRG by unfolding

$$\alpha_{\text{gprg}} = \left\lceil \frac{\max_{i \in N} \tau_i}{d} \right\rceil \tag{3}$$

times, where α_{gprg} is called the minimum collapsing factor (for GPRG).

III. Cutoff Time for Optimal PSFS Implementation

For the brevity of discussion, let us call the time duration in which operations in one iteration are scheduled an *iteration span.* In an overlapped periodic schedule, the iteration span of successive iterations may overlap. Potentially, the iteration span can be very large provided the precedence constraints are satisfied. This implies the scheduling algorithm has to search a very long time interval for an optimal solution.

In this section, we will derive a theoretical upper bound for the iteration span and prove that an optimal solution must lie within this interval. We call this upper bound the *cutoff time* (T_{cutoff}). The derivation of the cutoff time is based on the notion of equivalent periodic schedules.

A. Equivalent Periodic Schedules

Refer to the ICDG in Fig. 2, we may devise two different overlap schedules with the same period ($d = 6$) as depicted in Fig. 3(a), (b). Although these two schedules are different, they share the same *processor request profile* which is shown

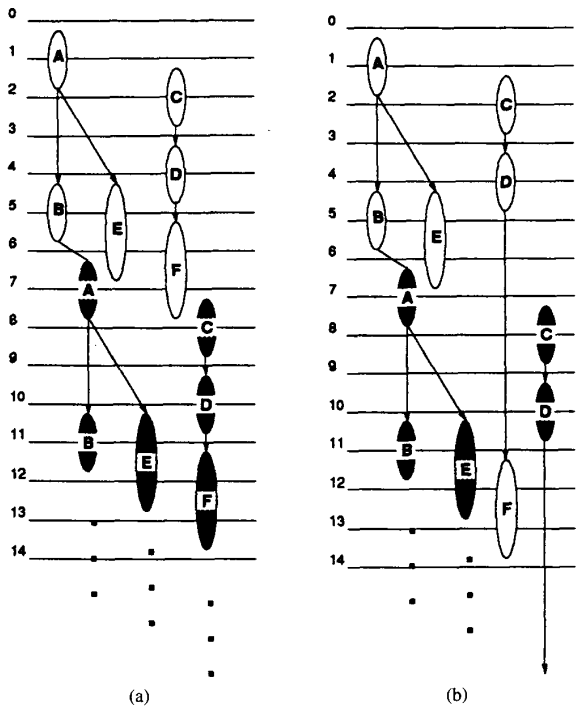


Fig. 3. Two possible schedules for Fig. 2 with initiation interval equal to 6.

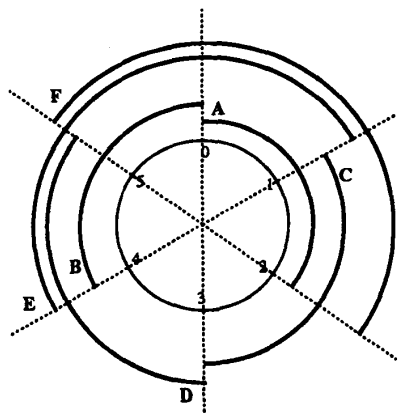


Fig. 4. Processor-request profile for the schedules in Fig. 3.

in Fig. 4. In this chart, the entire circle is equally divided into d sections, each representing a time slot. An arc spanning across one ore more sections indicates that it is scheduled to the corresponding time slots.

Definition 1: Two valid d-periodic schedules $\mathcal{F}_u$ and $\mathcal{F}_v$ for $G = (N, E)$ are *equivalent*, if and only if $(\mathcal{F}_u(n_i)$ modulo $d) = (\mathcal{F}_v(n_i)$ modulo $d)$ for all $n_i \in N$.

For every periodic schedule, $\mathcal{F}_u$, there are infinite many equivalent schedules. We define the *finish time* of a periodic schedule $\mathcal{F}_u$ as $T_u = \max_{\forall\ n_i} \mathcal{F}_u(n_i) + \tau_{n_i}$. The *primary schedule*, denoted by $\mathcal{F}_u^{\text{pri}}$, of $\mathcal{F}_u$ is defined as the one with smallest finish time. Thus, the cutoff time can be defined as follows.

Definition 2: (Cutoff Time) Denote the finish time of a primary schedule $\mathcal{F}_u^{\text{pri}}$ as T_u^{pri}. The cutoff time of a given ICDG, T_{cutoff} is defined as: $T_{\text{cutoff}} = \max_{\forall\ \mathcal{F}_u} T_u^{\text{pri}}$.

Lemma 1: Denote T_*^{pri} the finish time of the optimal periodic primary schedule, then $T_*^{\text{pri}} \leq T_{\text{cutoff}}$.

Lemma 1 follows directly from Definition 2. The question now is how to find T_{cutoff} or a tight upper bound of it. For this purpose, we will first discuss two special cases, namely acyclic ICDG's and strongly connected ICDG's. Then we will present the results for the general case.

B. Cutoff Time for Acyclic ICDG's

For any primary schedule $\mathcal{F}_u^{\text{pri}}$, any node should be scheduled within d clock ticks after the completion of its last finished predecessor to ensure a valid primary schedule. That is,

$$\max_{\forall\ (i,j)\in E} \mathcal{F}^{\text{pri}}(i) + \tau_i - d \times \delta_{i,j} \leq \mathcal{F}^{\text{pri}}(j) \leq \max_{\forall\ (i,j)\in E} \mathcal{F}^{\text{pri}}(i) + \tau_i - d \times \delta_{i,j} + d - 1.$$

Based on this observation, the computation of $T_{\text{cutoff}}(d)$ can be formulated by the algorithm illustrated below.

```
Acyclic(G_a, d):
    π(n_i) = (d − 1) for all n_i ∈ N;
    repeat
        modified = 0;
        for all (n_i, n_j) ∈ E
            l(n_i, n_j) = τ_{n_i} + (d − 1) − d × δ_{n_i,n_j};
            if (π(n_j) − π(n_i) < l(n_i, n_j))
                π(n_j) = π(n_i) + l(n_i, n_j);
                modified = 1;
        endfor
    until (modified == 0)
```

Below, we show that T_{cutoff} determined by *Acyclic*() satisfies the condition given in Definition 2. To facilitate the proof, we first introduce two special types of nodes in an ICDG, *source nodes* and *terminal nodes*. Source node is defined as a node has no incoming edge with a dependence distance equal to zero. We denote N_s as the set of source nodes in an ICDG. In the opposite, terminal node is a node with no outgoing edge with dependence distance equal to zero. N_t is the set of terminal nodes in an ICDG. An isolated node of an ICDG can be either a source node or a terminal node. Now we present the Theorem:

Theorem 2: For any valid periodic schedule $\mathcal{F}_u$ of G_a, its primary schedule $\mathcal{F}_u^{\text{pri}}$ has $T_u^{\text{pri}} \leq T_{\text{cutoff}}(d)$, where $T_{\text{cutoff}}(d)$ is derived from Acyclic().

Proof: Without loss of generality, the $\mathcal{F}_u$ of any node $n_i \in N$ can be represented as $x_{n_i} + z_{n_i} \times d$, where x_{n_i} and z_{n_i} are nonnegative integer and $0 \leq x_{n_i} \leq (d-1)$. Any equivalent schedule of $\mathcal{F}_u$, denoted as $\mathcal{F}'_u$, can thus be written as $\mathcal{F}'_u(n_i) = x_{n_i} + z'_{n_i} \times d$ for all $n_i \in N$. Due to precedence constraints, for any edge $(n_i, n_j) \in E$, z'_{n_i} and z'_{n_j} must satisfy the following inequality

$$x_{n_j} + z'_{n_j} \times d - x_{n_i} - z'_{n_i} \times d \geq \tau_{n_i} - d \times \delta_{n_i,n_j}. \quad (4)$$

The primary schedule of $\mathcal{F}_u$ will be an equivalent schedule that has the smallest z'_{n_j} that satisfies the inequality above. That is,

$$z'_{n_j} = \text{Max}\left[0, \max_{\forall (n_i,n_j)\in E}\left(z'_{n_i} + \left\lceil \frac{(x_{n_i} - x_{n_j}) + \tau_{n_i}}{d} \right\rceil - \delta_{n_i,n_j}\right)\right] \quad (5)$$

$z'_{n_s} = 0$ for all $n_s \in N_s$, since there is no predecessor for any source node n_s.

After constructing $\mathcal{F}_u^{\text{pri}}$, we will have to show $T_u^{\text{pri}} \leq T_{\text{cutoff}}$.

From 5, each z'_{n_i} of $\mathcal{F}_u^{\text{pri}}$ is determined by a *longest path* from any source node to n_i. Let z'_{n_t} of a terminal node n_t be determined by z'_{n_i} of n_i on the path from a source node n_s to n_t, $\{n_s, \ldots, n_k, n_{k+1}, \ldots, n_{t-1}, n_t\}$. Without loss of generality, we assume $z'_{n_k} = 0$ and $z'_{n_i} \neq 0$ for all $(k+1) \leq i \leq t$. Thus, we have

$$z'_{n_t} = \sum_{i=k}^{t-1}\left(\left\lceil \frac{x_{n_i} - x_{n_{i+1}} + \tau_{n_i}}{d} \right\rceil - \delta_{i,i+1}\right). \quad (6)$$

For any edge (n_i, n_{i+1}) in the path,

$$d \times \left\lceil \frac{x_{n_i} - x_{n_{i+1}} + \tau_{n_i}}{d} \right\rceil \leq x_{n_i} - x_{n_{i+1}} + \tau_{n_i} + d - 1. \quad (7)$$

From *Acyclic*(), we note that $\pi(n_i) \geq (d-1) \geq x_{n_i}$ for any node $n_i \in N$. Using 7 and 6, $\mathcal{F}_u^{\text{pri}}(n_t)$ is computed by

$$\begin{aligned}\mathcal{F}_u^{\text{pri}}(n_t) &= x_{n_t} + \sum_{i=k}^{t-1} d \times \left(\left\lceil \frac{x_{n_i} - x_{n_{i+1}} + \tau_{n_i}}{d} \right\rceil - \delta_{i,i+1}\right) \\ &\leq \pi(n_k) + \sum_{i=k}^{t-1}\left(d - 1 + \tau_{n_i} - d \times \delta_{n_i,n_{i+1}}\right) \\ &= \pi(n_k) + \sum_{i=k}^{t-1} l(n_i, n_{i+1}) \leq \pi(n_t). \end{aligned} \quad (8)$$

Therefore,

$$\begin{aligned} T_u^{\text{pri}} &= \max_{\forall\ n_t \in N_t}\left(\mathcal{F}_u^{\text{pri}}(n_t) + \tau_{n_t}\right) \\ &\leq \max_{\forall\ n_t \in N_t}\left(\pi(n_t) + \tau_{n_t}\right) = T_{\text{cutoff}}(d). \end{aligned}$$

□

C. Cutoff Time for Strongly Connected ICDG's

In a strongly connected ICDG, each node has at least one directed path to every other nodes. In this section, we will show that the iteration span of any valid periodic schedule of a strongly connected ICDG with a given initiation interval is bounded. A method will be given to derive an upper bound for the corresponding iteration span.

We first define a *distance graph* associated with a given ICDG.

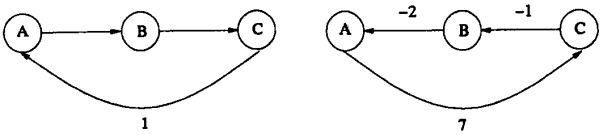


Fig. 5. (a) A strongly connected ICDG, (b) distance graph of (a).

Definition 3: (Distance Graph) A distance graph $G_D = (N, E)$ is a directed graph. There is a distance, $dist(n_i, n_j)$, associated with each directed edge $(n_i, n_j) \in E$.

For example, the distance graph of the ICDG in Fig. 5(a) is shown in Fig. 5(b).

To motivate our approach, let us examine the ICDG in Fig. 5(a). The node delays for A, B, and C are 2, 1, and 3 time units, respectively. Suppose that the given initiation interval $d = 10 > I_{\min} = 7$. Due to the interiteration dependency of A on C in the prior iteration, a valid schedule $\mathcal{F}$ has to satisfy the following constraint

$$\mathcal{F}(C) + \tau_C \leq \mathcal{F}(A) + d \times \delta_{C,A}$$

or,

$$\mathcal{F}(C) \leq \mathcal{F}(A) + (-3 + 10 \times 1).$$

Similarly, we have the following inequalities for the other two edges

$$\mathcal{F}(A) \leq \mathcal{F}(B) + (-2 + 0) \quad \text{for edge } (A, B)$$
$$\mathcal{F}(B) \leq \mathcal{F}(C) + (-1 + 0) \quad \text{for edge } (B, C).$$

From the inequalities above, we can use distance graph as shown in Fig. 5(b) to represent the relationship. The distance graph is constructed by adding edge (n_i, n_j) if there is an edge (n_j, n_i) in the corresponding ICDG and $\text{dist}(n_i, n_j) = d \times \delta_{n_j,n_i} - \tau_{n_i}$. To determining an upper bound of iteration span is equivalent to finding the shortest path from the *earliest starting node* to *latest finishing node*. That is the maximum of the shortest path between all pairs of source nodes and terminal nodes. The shortest path from source node A to terminal node C is the edge (A, C). Thus the time span of any schedule for the ICDG in Fig. 5(a) with $d = 10$ is bounded by (*shortest distance from A to* $C + \tau_C$) $= 7 + 3 = 10$. To determine the upper bound systematically, we have following algorithm

Strong(G_D, d):
  $\kappa = 0$;
  for all $n_s \in N_s$ and $n_t \in N_t$
    distance = shortest_distance(n_s, n_t);
    if ($\kappa <$ distance)
      $\kappa =$ distance;
  endfor

Many algorithms to find shortest path between two nodes are available [23]. This leads to the following Theorem:

Theorem 3: Let $G_s = (N, E)$ be a strongly connected ICDG. For any valid periodic schedule $\mathcal{F}_u$ with $d \geq I_{\min}$, there exists a constant κ, such that

$$\kappa \geq \max |\mathcal{F}_u(n_i) + \tau_{n_i} - \mathcal{F}_u(n_j)| \quad \forall\, n_i, n_j \in N.$$

Proof: To prove this Theorem, we only need to show there is a shortest path between any pair of source node and terminal node in the distance graph. That is the shortest path algorithm will terminate.

In a strongly connected ICDG, there is at least one path from a terminal node (n_t) to any source node (n_s). Thus a path from n_s to n_t exists in the distance graph. Furthermore, since $d \geq I_{\min}$, the sum of the distance of edges around any cycle in the distance graph must be nonnegative. Therefore, a finite-distance shortest path from any n_s to n_t in the distance graph exists. And κ can be defined as the maximum among all the shortest distance from any n_s to $n_t + \tau_{n_t}$. □

From Theorem 3, we may consider a strongly connected ICDG as a *super node* with delay equal to κ in order to estimate an upper bound of the iteration span of all the primary schedules. This observation is justified by the following Corollary.

Corollary 1: Given a strongly connected ICDG $G_s = (N, E)$ and an initiation interval $d \geq I_{\min}$, the cutoff time is $\kappa + d - 1$, where κ is computed by Strong().

Proof: We will show that any valid schedule $\mathcal{F}_u$ of G_s have a primary schedule $\mathcal{F}_u^{\text{pri}}$ whose $T_u^{\text{pri}} \leq (\kappa + d - 1)$.

Without loss of generality, $\min_{\forall n_i \in N} \mathcal{F}_u(n_i) = m \times d + r$, where m and r are integers and $0 \leq r \leq (d - 1)$. We define $\mathcal{F}_u^{\text{pri}}(n_i) = \mathcal{F}_u(n_i) - m \times d$. From Theorem 3, we have

$$\max_{\forall n_i \in N} (\mathcal{F}_u^{\text{pri}}(n_i) + \tau_{n_i}) - \min_{\forall n_i \in N} \mathcal{F}_u^{\text{pri}}(n_i) \leq \kappa.$$

Since $\min_{\forall n_i \in N} \mathcal{F}_u^{\text{pri}}(n_i) = r$ and

$$\begin{aligned} T_u^{\text{pri}} &= \max_{\forall n_i \in N} \left(\mathcal{F}_u^{\text{pri}}(n_i) + \tau_{n_i}\right) \\ &= \kappa + r \\ &\leq \kappa + d - 1. \end{aligned}$$

Thus, the Corollary is proved. □

D. Cutoff Time for General ICDG's

Since the iteration span of a strongly connected ICDG with a given d is bounded by κ, a strongly connected ICDG can be treated as a *super node* with delay equal to κ when determining the cutoff time. Adopting this idea, we thus generalize our previous results to find T_{cutoff} for any ICDG's. The scenario to determine a cutoff time for any given ICDG, $G = (N, E)$ is as follows.

1) Identify all the strongly connected subgraphs, G_{s_i}, from G and compute κ_{s_i} for each strongly connected subgraph.
2) Reduce G to a reduced graph G_r by replacing each G_{s_i} by a *super node*, s_i, with delay set to κ_{s_i}.
3) Since G_r is acyclic, Theorem 2 can be applied to compute a T_{cutoff} for G_r.

To summarize, we have the following theorem.

Theorem 4: Given an initiation interval d, any valid periodic schedule of G has an equivalent periodic schedule within the interval $[0, T_{\text{cutoff}}(d)]$, where T_{cutoff} is derived from G_r, the reduced G, using Theorem 2.

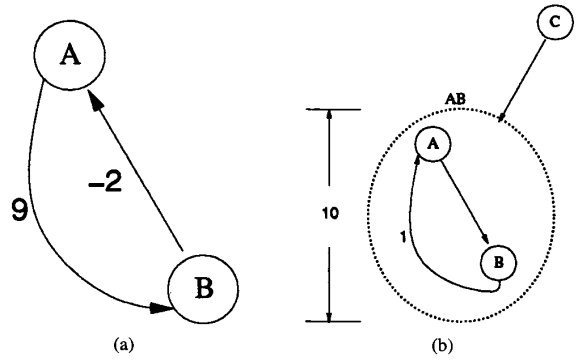


Fig. 6. (a) The distance graph of the strongly connected subgraph of Fig. 1; (b) a reduced ICDG of Fig. 1.

Proof: The proof follows directly from Theorem 2 and Theorem 3. □

To illustrate how to compute T_{cutoff}, let us consider the following example.

Example 1: Consider the ICDG in Fig. 1 as an example. Let the delay of A, B and C be 2, 1, and 2, respectively and desired data sampling period $d = 10$. Since $10 > (\tau_A = \tau_C) = 2$, the ICDG is a GPRG. We first compute κ for the strongly connected subgraph. From the distance graph used to determine κ shown in Fig. 6(a), $\kappa(10) = 9 + \tau_B = 10$. Then, the reduced ICDG, shown in Fig. 6(b), is constructed by clustering a strongly connected component labeled with AB. This leads to $T_{\text{cutoff}}(10) = 30$ after applying the algorithm *Acyclic*(). □

IV. Scheduling and Allocation Algorithm

Once the cutoff time T_{cutoff} is computed, we are ready to determine a schedule and processor assignment for each node in the given ICDG. In this paper, we propose a two-phase heuristic algorithm to solve this scheduling and assignment problem in two consecutive phases: First, each task is scheduled to start at a time slot and extended through a duration equal to the node computing time. The objective is to minimize the maximum number of tasks scheduled on the same time slot subject to the constraint that the desired initiation interval is maintained. Second, we assigns each node to processors such that the number of processors used is minimized under the constraint that only one task can be executed on a processor at a time.

A. Periodic Schedule

A common feature of combinatorial optimization problems is that the solution to the overall problem can be decomposed into the solution of a set of subproblems. There are often several candidate solutions to each subproblem. The order of which these subproblems are being solved, and the candidate solution chosen for solving each subproblem will dictate the quality of the solution. In the periodic scheduling problem, the schedule of each node in the ICDG is a subproblem. The time interval $[0, T_{\text{cutoff}}]$ contains the set of potential solutions to each subproblem.

The *most critical principle* applied here is to determine the order of which these subproblems are being solved. In particular, this principle calls for the prioritization of the subproblems according to a problem-specific heuristic cost function called *criticalness*. In this research, we use a criterion called *feasible scheduling interval* as a measure to determine the criticalness of each node in the ICDG. A node is more critical if it has fewer available feasible schedules. The criticalness of each node is dynamically updated for the set of un-scheduled nodes as more critical nodes are being scheduled.

The *least impact principle* is designed to help choosing an appropriate solution from a set of feasible solutions to solve a particular subproblem. By committing the current subproblem to any of its feasible solution will unavoidably consume precious resources, and thereby reducing the feasible solution space of remaining unsolved subproblems. The least impact principle favors the choice of a solution which will impose the least amount of impacts to the feasible solution space of those unsolved subproblems. To apply this principle, in this paper, we measure the impact of scheduling a particular node at a time slot using a criterion called *scheduling cost*. A low cost schedule is preferred to a high cost schedule as the former leaves more flexibility to schedule remaining un-scheduled nodes.

1) Feasible Scheduling Interval and the Most Critical Node: Due to the precedence constraints, a node in an ICDG can only be scheduled within the interval between the *as soon as possible schedule* (*ASAP*) and the *as late as possible schedule* (*ALAP*). Because of the cyclic nature of the ICDG's, the algorithm used in conventional high level synthesis research [24] to determine ASAP and ALAP schedules for acyclic DFG's (data flow graphs) can not be applied here.

The ASAP schedule assigns the earliest possible starting time S_i to a node i in G. Due to the precedence constraints, i can not be started until all its predecessors complete. That is

$$S_j = \max_{\forall (i,j) \in E} (S_i + \tau_i - d \times \delta_{i,j}). \tag{9}$$

While the ALAP schedule of node i denoted as L_i is defined as the schedule of i closest to T_{cutoff}. Since the completion time of i can not be later than the starting time of all its successors, we have

$$L_i = \min_{\forall\ (i,j) \in E} (L_j - \tau_i + d \times \delta_{i,j}). \tag{10}$$

Both ASAP and ALAP schedules can be derived by Bellman-Ford's algorithm [25] with at most $O(|N||E|)$ operations, where $|N|$ and $|E|$ represent the number of nodes and edges in G, respectively.

In our algorithm, we schedule each node sequentially. According to the most critical principle, a node should be scheduled first if it is the current most *critical* node. The criticalness for a node i, denoted by $C_{\text{cri}}(i)$, is measured by the ratio of the length of the feasible scheduling interval and delay of that node. That is

$$C_{\text{cri}}(i) = \frac{\tau_i}{L_i - S_i + 1} \quad \forall\, i \in N. \tag{11}$$

TABLE I
THE ASAP AND ALAP SCHEDULES FOR EXAMPLE 2

Node	delay	ASAP	ALAP	C_{cri}
A	2	0	27	$\frac{1}{14}$
B	1	2	29	$\frac{1}{28}$
C	2	0	27	$\frac{1}{14}$

TABLE II
THE ASAP AND ALAP SCHEDULES FOR EXAMPLE 3

Node	delay	ASAP	ALAP	C_{cri}
n1	2	0	5	$\frac{1}{3}$
n2	3	1	6	$\frac{1}{2}$
n3	1	4	7	$\frac{1}{4}$

A node with a larger value of $C_{\text{cri}}(i)$ implies that it has fewer alternative feasible schedules. Hence it will be given a higher priority to be scheduled early. If there are more than one nodes with the same criticalness, we arbitrarily pick one to schedule first.

Example 2: Consider the ICDG in Fig. 1 again. From Example 1, we have computed the cutoff time $T_{\text{cutoff}} = 30$. The ASAP schedule and ALAP schedule are shown in Table I. From 11, the criticalness of each node is computed and tabulated in Table I. A and C are the most critical nodes.

2) Least Impact Schedule: Once the most critical node is selected, it will be scheduled using the least impact principle. First, note that the time span of the schedule of node n_i is $\text{sp}(n_i) = [\mathcal{F}(n_i), \mathcal{F}(n_i) + \tau_{n_i})$ where $\mathcal{F}(n_i)$ is the scheduled starting time of node n_i. A time slot t is occupied by node n_i, if $[t, t+1) \subset \text{sp}(n_i)$. Since $S_{n_i} \leq \mathcal{F}(n_i) \leq L_{n_i}$, we have $\text{sp}(n_i) \subset [S_{n_i}, L_{n_i} + \tau_{n_i})$. Let $M = \min(L_{n_i} - S_{n_i} + 1, \tau_{n_i})$, we define $p_{n_i}(m)$, the demand probability of node n_i to time slot m, as follows:

$$p_{n_i}(m) = \begin{cases} \frac{m - S_{n_i} + 1}{L_{n_i} - S_{n_i} + 1} & S_{n_i} \leq m < (S_{n_i} + M - 1); \\ \frac{L_{n_i} + \tau_{n_i} - m}{L_{n_i} - S_{n_i} + 1} & (L_{n_i} + \tau_{n_i} - M) < m \leq (L_{n_i} + \tau_{n_i} - 1); \\ \frac{M}{L_{n_i} - S_{n_i} + 1} & (S_{n_i} + M - 1) \leq m \leq (L_{n_i} + \tau_{n_i} - M); \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

If $p_{n_i}(m) = 1$, it implies that no matter how the node n_i is scheduled, it will occupy time slot $[m, m+1)$. Since we are deriving a periodic schedule of period d, if a node n_i occupies time slot m, it will also occupy time slots $m + k \times d$, $k = 1, 2, \ldots$. Hence the probability of node n_i occupying an interval $[m, m+1)$ within the window $[0, d)$ is

$$pw_{n_i}(m) = \sum_{j=0}^{\left\lceil \frac{L_{n_i} + \tau_{n_i} - m}{d} \right\rceil} p_{n_i}(j \times d + m). \tag{13}$$

The total demand to a time slot $[m, m+1)$ within the window $[0, d)$ hence can be evaluated as $\sum_{n_i \in N} pw_{n_i}(m)$, where N is the set of all the nodes in the ICDG. A time slot with high demand, thus becomes a scarce resource and, according to the least impact principle, should be avoided when scheduling a given node. To measure the impact of scheduling a node in a particular time duration, we define a scheduling cost of node n_i, denoted by $C_{\text{sch}}(n_i, t)$, as the sum of the total demands by *other nodes* to each of the time slots within $[t, t+\tau_{n_i})$. That is,

$$C_{\text{sch}}(n_i, t) = \sum_{n_j \in N \text{ and } n_j \neq n_i} \sum_{k \in [t \text{ modulo } d, (t+\tau_{n_i}) \text{ modulo } d)} pw_{n_j}(k).$$

The least impact schedule is chosen to be the one which minimizes $C_{\text{sch}}(n_i, t)$ over the range $[S_{n_i}, L_{n_i}]$.

Once a node n_i is scheduled at t, we will update the demand probability by setting $pw_{n_i}(m) = 1$ for each $m \in [t \text{ modulo } d, (t+\tau_{n_i}) \text{ modulo } d)$[1] to reflect the fact that these time slots have been formally occupied by node n_i.

Example 3: In Table II, the ASAP and ALAP schedules for a three-node ICDG with $d = 8$ are shown. According to most critical principle, $n2$ will be scheduled first. In order to determine the least impact schedule for $n2$, we need to compute $C_{\text{sch}}(n2, t)$ for all $t \in [1, 6]$. The operations in 12 and 13 to compute $pw_{n1}(m)$ and $pw_{n3}(m)$ for all $m \in [0, d)$ are summarized in Fig. 7, where each arc represents a possible schedule. By counting the number of overlapped arcs in Fig. 7 for each time interval, pw can be determined. Since the ASAP and ALAP for $n2$ are 1 and 6, respectively, six possible schedules and the corresponding C_{sch}s' are tabulated below. The time span (6, 9) has the smallest C_{sch}. Therefore, the least impact schedule is to start $n2$ at $t = 6$.

B. Processor Allocation

Once a periodic schedule is obtained, we need to assign tasks to processors such that the number of processors used in the implementation is minimized. The problem of finding a minimum-processor allocation for periodic schedule is equivalent to the circular-arc coloring problem [26].

Two tasks are *incompatible* if and only if their corresponding circular arcs of processor-request profile overlap. Two incompatible tasks cannot be assigned to the same processor. This relationship can be represented by a undirected graph, called *incompatibility graph*. The nodes represent nodes in ICDG and two nodes are connected if and only if their corresponding tasks are incompatible. In this paper, an efficient heuristic algorithm which was designed for the digital system test scheduling problem [7] is employed. We will only illustrate the algorithm by an example. Readers please refer to [7] for more details of the algorithm.

Let us consider the periodic schedule in Fig. 3(a) as an example. Its incompatibility graph is shown in Fig. 8. First,

[1] If t modulo $d \geq (t + \tau_{n_i})$ modulo d, the interval is equivalent to the union of two intervals, $[t \text{ modulo } d, d) \bigcup [0, (t + \tau_{n_i}) \text{ modulo } d)$. Note that for GPRG, the length of the span of any node can not be greater than d.

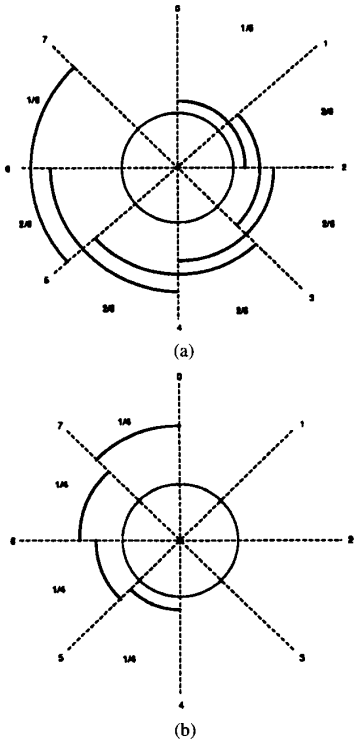


Fig. 7. Computing pw for (a) $n1$; and (b) $n3$.

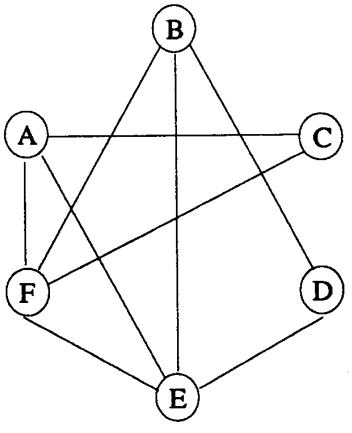


Fig. 8. The incompatibility graph of Fig. 3.

we determine a coloring sequence according to the degree (number of connected nodes) of each node in ICG. Each time a node with largest degree is chosen. Then delete the edges connected to that node and choose the another node until all the nodes are selected. If there are more than one node having the same degree, we arbitrarily pick one. The results of the example are tabulated below:

The coloring order of these nodes, in descending order, is E, F, A, B, D, and C. Next, we start to color the nodes according to the order. The color of node is the lowest number that is not used by any of its neighbor nodes. The results are listed below:

In this example, our coloring algorithm (processor assignment) uses only 3 colors (processors), which is optimal.

TABLE III
OUR RESULT VERSUS MARS'S [27]

ART				MARS [27]	
Unfold	PEs	T_{cutoff}	T_{cpu}	Unfold	PEs
2	3	77	0.03s	3	4

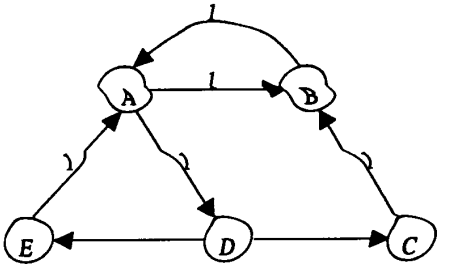


Fig. 9. An ICDG example ([4], Fig. 19).

C. Worst Case Complexity

The periodic scheduling algorithm described in Section IV-A1) and Section IV-A2) will schedule one or more nodes per iteration and terminate after at most $|N|$ iterations where $|N|$ is the number of nodes to be scheduled. Within each iteration, the *ASAP* and *ALAP* schedules are derived using at most $O(|N||E|)$ operations. The criticalness for all the nodes can be evaluated with $O(|N|)$ operations. Computing the demand probability and finding the least impact schedule for the critical node will take at most $O(|N|d+d)$ operations, where d is the desired initiation interval. Hence, the worst case complexity of the proposed scheduling algorithm is $O(|N|^2(d+|E|)+|N|d)$. The worst-case complexity of the fully static processor assignment algorithm is $O(|N|^2)$, when all the nodes have to be assigned to different processors.

Thus, our scheduling and assignment algorithm will take at most $O(|N|^2(d+|E|+1)+|N|d)$ operations. The complexity of range-chart-guided scheduling algorithm reported in [18] is proportional to $|N|^3$. The cyclo-static scheme [16] may even lead to a nonpolynomial time scheduling complexity.

V. EXPERIMENTAL RESULTS

The periodic scheduling and allocation algorithms have been implemented and tested on several benchmark examples reported in the literature [16], [18], [27]. The CPU time ($T_{\rm cpu}$) reported in our results is measured on a SUN SPARC Station 10/30 with 128 Mbyte main memory.

Example 4: In this example, we compare our method with MARS, an architectural synthesis tool [27] using an ICDG depicted in Fig. 9. By applying our scheduling and allocation algorithm to the twice-unfolded ICDG, we can achieve the minimum-processor PSFS implementation. In Table III, we compare our result with that derived from MARS. We note

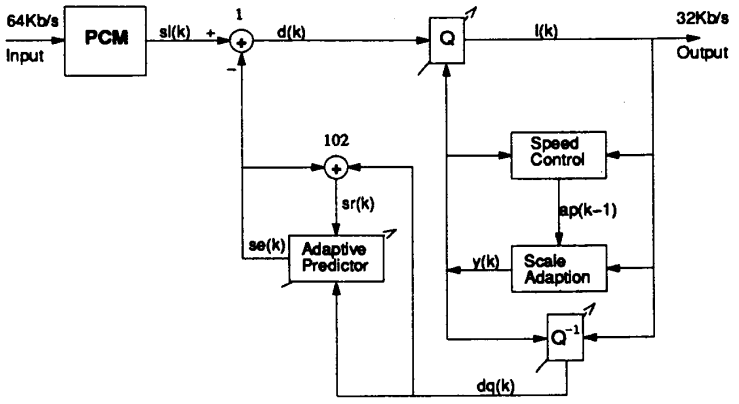


Fig. 10. ADPCM encoder block diagram.

TABLE IV
OUR RESULTS VERSUS THOSE USING RANGE-CHART SCHEDULING ALGORITHM [18]

Example	d	Delay +,*	T_{cutoff}	ART PEs	ART T_{cpu}	[18] PEs	[18] T_{cpu}
Second-order section	3	1,2	15	4	0.01s	4	0.09s
4^{th}-order Jaumann filter	16	1,5	64	3	0.03s	3	0.17s
4^{th}-order All-pole filter	14	1,5	48	3	0.01s	3	0.13s
5^{th}-order elliptic filter	16	1,2	31	4	0.1s	4	0.33s

that not only ours needs fewer unfolding, but also use fewer processors.

Example 5: In the example, we compare our results with those using the range-chart guided scheduling algorithm as reported in [18]. All these examples satisfy the condition of a GPRG, hence no unfolding is needed. The results are tabulated in Table IV. The number of processors needed for each of the example is the same for both methods.

Example 6: The results reported in [16], which are obtained using cyclo-static scheduling method, are compared. According to [16], the delay of addition operation and multiplication operation are set to 1 and 2 clock ticks, respectively. No unfolding is needed for all benchmarks, since they all satisfy the condition of GPRG's. The results are summarized in Table V Note that in the first two benchmarks, our implementations need fewer processors than those reported in [16].

Example 7: In many DSP applications such as linear equalization and echo cancellation, higher order filters are desired. In this example, we test our algorithm on higher order digital filters. We assume the addition and multiplication take 1 and 2 clock ticks. The results are summarized in Table VI. With reasonable computing time, our algorithm produces very satisfactory results.

Example 8: In this example, the design of an encoder for *adaptive differential pulse code modulation* (ADPCM) [28] according to CCITT Rec. 32 kbps G.721 is shown. Referring to Fig. 10, the 8-bit PCM input signal, $s(k)$, is converted from μ-law (A-law) to a linear PCM signal, s_l, according to the

TABLE V
OUR RESULTS VERSUS THOSE USING CYCLO-STATIC SCHEDULING SCHEME [16]

Example	d	T_{cutoff}	ART PEs	ART T_{cpu}	[16] PEs
2nd orthogonal filter	8	50	4	0.02s	5
6th orthogonal filter	9	65	14	1.05s	16
2nd Gray-Markel filter	7	61	3	0.01s	3
6th Gray-Markel filter	7	76	8	0.21s	8
2nd normalized filter	6	61	5	0.04s	5
6th normalized filter	6	92	14	0.49s	14
2nd LMS filter	7	23	2	0.01s	2
6th LMS filter	9	29	6	0.05s	6
4th Jaumann wave filter	8	21	3	0.03s	3

TABLE VI
EXAMPLES OF HIGHER ORDER FILTERS

Example	# nodes	# edges	d	PEs	T_{cpu}	Util %
40th orthogonal filter	473	590	9	93	665.63s	94.3
48th Gray-Markel filter	289	424	7	62	161.45s	100
42th normalized filter	337	461	6	96	235.24s	95.1
32th LMS filter	129	247	11	24	9.01s	73.5

procedure given by CCITT Rec. G711. An estimated value of this signal, $s_e(k)$, obtained from the adaptive predictor, is subtracted from $s_l(k)$ to produce the difference signal $d(k)$. The difference signal is encoded with a 16-level quantizer, where parallel search is used, to yield the 4-bit ADPCM value $I(k)$ that is transmitted to a distant decoder. The inverse quantizer Q^{-1} generates the quantized difference signal $d_q(k)$ from $I(k)$ using the scale factor $y(k)$. An adaptive predictor consisting of 2-pole and 6-zero produces the signal estimate thus closing the feedback loop. The adaptation algorithm for updating the coefficients of the linear predictor is shown in Table VII summarizes the numbers of different types of operations for the encoder.

In Fig. 11, we demonstrate a PSFS implementation derived from our proposed algorithm with data sampling period equal

TABLE VII
OCCURRENCES OF OPERATIONS FOR APDCM ENCODING ALGORITHM. PLEASE NOTE THE COUNTS MAY VARY DEPENDING ON THE WAY OF IMPLEMENTATION

Operation	#	Delay
Floating Point Multiplication	8	7
Multiplication	1	2
Addition/Subtraction	38	1
Shift (>>)	27	1
Misc. logic	30	1

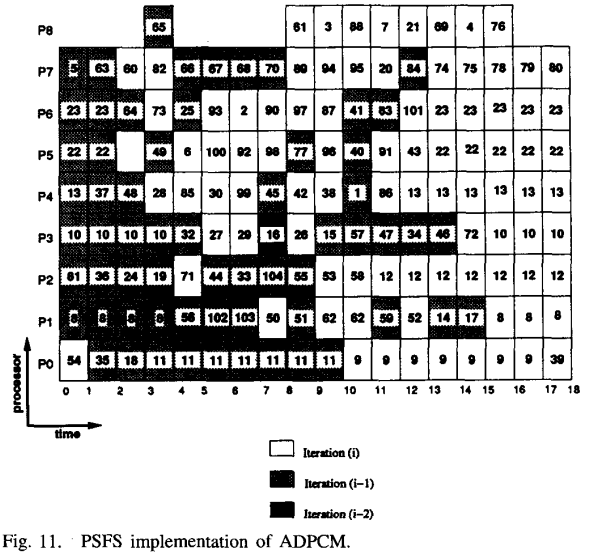


Fig. 11. PSFS implementation of ADPCM.

to 18. Compared with a processor lower bound computed by $\lceil \frac{\sum_{\forall n_i \in N} \tau_{n_i}}{d} \rceil = 9$, our implementation is optimal.

VI. CONCLUSION

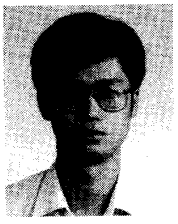
In this paper, we proposed a methodology to synthesize synchronous multiprocessor systems for real-time DSP algorithms. The style of periodic schedule and fully-static processor assignment is adopted. After transforming the ICDG to a previously proposed GPRG, our scheme consists of two steps. We first compute a cutoff time for periodic schedule, by which all the operations of an iteration have to be finished. A systematic way of determining a cutoff time is proposed. We showed that cutoff time derived from our scheme guarantees the existence of the minimum-processor implementation within the interval of cutoff time. Then, a novel heuristic method to schedule and map a GPRG onto a multiprocessor system is then proposed. By experimenting an extensive set of benchmarks, our algorithm can achieve the most processor efficient design in very low complexity. At the same time we can still preserve the real-time constraint.

Although our assumption for interprocessor communication delay is valid if all the processors can be put on a chip, this might not be true for more complicated DSP algorithms. In [29], the authors have studied the impact of the communication delay to the method proposed in this paper. A modified algorithm which takes interprocessor communication delay into account has been proposed in [30].

REFERENCES

[1] K. K. Parhi, "Algorithm transformation techniques for concurrent processors," *Proc. IEEE*, vol. 77, pp. 1879–1895, Dec. 1989.
[2] M. Renfors and Y. Neuvo, "The maximum sampling rate of digital filters under hardware speed contraints," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 196–202, Mar. 1981.
[3] D. J. Kuck, *Structures of Computers and Computations*. New York: Wiley, 1978.
[4] K. K. Parhi and D. G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," *IEEE Trans. Comput.*, vol. 40, pp. 178–195, Feb. 1991.
[5] D.-J. Wang and Y. H. Hu, "Fully static multiprocessor array realizability criteria for real-time recurrent DSP applications," *IEEE Trans. Signal Processing*, vol. 42, pp. 1288–1292, May 1994.
[6] D. Y. Y. Yun and N. P. Keng, "A planning/scheduling methodology for the constrained resource problem," in *Proc. Int. Joint Conf. Artificial Intell.*, 1989, pp. 998–100.
[7] G. L. Craig, C. R. Kime, and K. K. Saluja, "Test scheduling and control for VLSI build-in self-test," *IEEE Trans. Comput.*, vol. 37, pp. 1099–1109, Sept. 1988.
[8] E. F. Girczyc, "Loop winding—A data flow approach to functional pipelining," in *Proc. Int. Symp. Circuit Syst.*, May 1987, pp. 382–385.
[9] M. Lam, "Software pipelining: An effective scheduling technique for VLIW machines," in *Proc. SIGPLAN Compiler Construct.'88*, 1988.
[10] S. Note, F. Catthoor, G. Goossens, and H. J. De Man, "Combined hardware selection and pipelining in high-performance data-path design," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 413–423, Apr. 1992.
[11] L.-F. Chao and E. H.-M. Sha, "Unfolding and retiming data-flow DSP programs for RISC multiprocessor scheduling," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, vol. V, 1992, pp. 565–568.
[12] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," in *Ann. Discrete Math.*, vol. 5, pp. 287–326, 1979.
[13] P. Hoang and J. Rabaey, "Hierarchical scheduling of DSP programs onto multiprocessors for maximum throughput," in *Proc. Int. Conf. Appli. Spec. Array Processing*, Berkeley, CA, Aug. 1992, pp. 21–36.
[14] T. P. Barnwell, III, and C. J. M. Hodges, "Optimal implementation of signal flow graphs on synchronous multiprocessors," in *Proc. Int. Conf. Parallel Process.*, Aug. 1982, pp. 90–95.
[15] D. A. Schwartz and T. P. Barnwell III, "Cyclo-static multiprocessor scheduling for the optimal implementation of shift invariant flow graphs," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 1985, pp. 1384–1387.
[16] P. R. Gelabert and T. P. Barnwell III, "Optimal automatic periodic multiprocessor scheduler for fully specified flow graphs," *IEEE Trans. Signal Processing*, vol. 41, pp. 858–888, Feb. 1993.
[17] C.-Y. Wang and K. K. Parhi, "Dedicated DSP architecture synthesis using the MARS design system," in *Proc. Int. Conf. Acoust., Speech, Signal Processing*, 1991, pp. 1253–1256.
[18] S. M. Heemstra de Groot, S. H. Gerez, and O. E. Herrmann, "Range-chart-guided iterative data-flow graph scheduling," in Special Issue on Fundamental Theory and Applications, *IEEE Trans. Circuits Syst. ——I*, vol. 39, pp. 351–364, May 1992.
[19] R. H. Karp, R. E. Miller, and S. Winograd, "The organization of computations for uniform recurrence equations," *J. Assoc. Comput. Machin.*, vol. 14, pp. 563–590, Jul. 1967.
[20] P. M. Kogge, "Parallel solution of recurrence problems," *IBM J. Res. Develop.*, pp. 138–148, Mar. 1974.
[21] S.-C. Chen and D. J. Kuck, "Time and parallel processor bounds for linear recurrence systems," *IEEE Trans. Comput.*, vol. 24, pp. 701–717, July 1975.
[22] A. Nicolau and H. Wang, "Optimal schedules for parallel prefix computation with bounded resources," in *ACM SIGPLAN Symp. Princ. & Pract. Parallel Programm.*, Williamsburg, VA, Apr. 1991, pp. 1–10.
[23] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart, and Winston, 1976.
[24] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proc. IEEE*, vol. 78, pp. 301–318, Feb. 1990.
[25] L. R. Ford, Jr., and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton University Press, 1962.
[26] A. Tucker, "Coloring a family of circular arcs," *SIAM J. Appl. Math.*, vol. 29, no. 3, pp. 493–502, Nov. 1975.

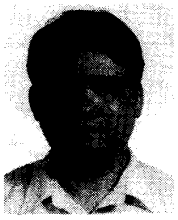
[27] L. E. Lucke, A. P. Brown, and K. K. Parhi, "Unfolding and retiming for high-level DSP synthesis," in *Proc. Int. Symp. Circuit, Syst.*, Singapore, 1991.
[28] T. Nishitani, I. Kuroda, M. Satoh, T. Katoh, and Y. Aoki, "A CCITT standard 32 kbits/s ADPCM LSI codec," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, pp. 219–225, Feb. 1987.
[29] Y. H. Hu and D.-J. Wang, "Static scheduling for real-time recurrence DSP algorithms with fixed interprocessor communication delays," in *Proc. Int. Symp. Circuit Syst.*, San Diego, CA, May 1992, pp. 180–183.
[30] ——, "Clustering approach for mapping recursive DSP algorithms to multiprocessor with fixed ipc delays," in *Proc. IEEE Int. Workshop VLSI Signal Processing*, Napa, CA, Oct. 1992, pp. 365–374.



Duen-Jeng Wang received the B.S.E.E. degree from National Taiwan University, Taipei, Taiwan, R.O.C., in 1984, and the M.S.E.E. degree from National Cheng Kung University, Tainan in 1986. He is currently a doctoral candidate in Electrical and Computer Engineering at University of Wisconsin, Madison.

From 1986 to 1988, he served in the Chinese Army as an Aide-de-Camp to Chief of Penghu Defense Command. He then joined Electronics Research & Service Organization (ERSO), Industrial Technology Research Institute (ITRI), Hsinchu in 1988, where he designed mini-supercomputers. His research interests include VLSI signal processing, high-level synthesis, parallel processing and architectures, and compilation for parallelism.

Mr. Wang is a member of IEEE Computer Society and the Association for Computing Machinery.



Yu Hen Hu (M'83–SM'87) received the B.S.E.E. degree from National Taiwan University, Taipei, Taiwan, R.O.C., in 1976. He received the M.S.E.E. and Ph.D. degrees in electrical engineering from University of Southern California, Los Angeles in 1980 and 1982, respectively.

From 1983 to 1987, he was an Assistant Professor of the Electrical Engineering Department of Southern Methodist University, Dallas, TX. He joined the Department of Electrical and Computer Engineering, University of Wisconsin, Madison in 1987 as an Assistant Professor from 1987 to 1989, and is now an Associate Professor. His research interests include VLSI signal processing, artificial neural networks, spectrum estimation, fast algorithms and parallel computer architectures, and computer-aided design tools for VLSI using artificial intelligence. He has published more than 100 journal and conference papers in these areas.

Dr. Hu is a member of SIAM. He is a former Associate Editor (1988–1990) for the IEEE TRANSACTIONS ON ACOUSTICS, SPEECH, AND SIGNAL PROCESSING in the areas of system identification and fast algorithms. He is a founding member of the Neural Network Signal Processing Technical Committee and VLSI Signal Processing Technical Committee of the Signal Processing Society.