Proc. Natl. Sci. Counc. ROC(A)
Vol. 22, No. 3, 1998. pp. 362-371

Conversion, Iteration Bound and X-Window Implementation for Multi-Rate Data Flow Graphs

DANIEL YUH CHAO

Department of Management Information Systems
National Cheng Chi University
Taipei, Taiwan, R.O.C.

(Received May 15, 1997; Accepted November 10, 1997)

ABSTRACT

Techniques for finding iteration bounds (IB; i.e., minimum iteration period) of single rate data flow graphs (SRDFG; i.e., sampling input data at a single rate) have been well documented in the literature. Lee *et al.* pointed out that SRDFG is too restricted and proposed heuristics for scheduling multi-rate DFGs (MRDFG; i.e., sampling input data at multiple rates). Parhi suggested converting an MRDFG into an equivalent SRDFG to find the IB and presented an explicit procedure to convert an MRDFG into its equivalent SRDFG. He reduced the time required for computing the iteration bound of the equivalent SRDFG by eliminating node and edge redundancies. Based on this result, we show that the lower bound of the iteration bound can be achieved for certain special cases. In addition, the algorithm for SRDFG can be applied to identify critical loops (CL), scheduling ranges, initial scheduling to avoid transients and static scheduling under steady state. There is no need for heuristics.

Key Words: concurrent processing, Data Flow Graph (DFG), General Petri Net (GPN), Weighted T-Graph (WTG), P-semiflow, T-semiflow, liveness, boundedness, single-rate DFG, multi-rate DFG, iteration bound, loop bound, loop-combination, system performance, critical loop and scheduling

I. Introduction

Real time applications such as image processing constantly execute a set of tasks. With the cost of Very Large Scale Integrated Circuits (VLSI) going down, multiprocessing is increasingly used to enhance execution speed by executing tasks concurrently. In a multiprocessor environment, the data flow graph (DFG) (Ackerman, 1982) is frequently used to model program specifications and to express the available concurrency. DFG is a directed graph where directed arcs model the precedence constraints between nodes. Each node accepts input data from its predecessors, executes some tasks for some deterministic period of time and outputs data to the next node. Given a DFG, we need to optimize the scheduling of processors assigned to nodes to achieve the shortest iteration period (IP) which results in a maximum throughput. Traditionally, one can construct the acyclic precedence graph (APG) from a DFG such that the root nodes of the APG carry enough initial data to start execution. We find the critical path of this APG by finding the longest path which determines the IP. The execution of nodes propagates towards the leaves of the APG.

By distributing initial data to different arcs, a different APG results, which may turn out to be a better scheduling with a shorter IP. This data redistribution process is referred to as *retiming* in the literature (Parhi, 1989). The aforementioned scheduling exploits the concurrency within one IP constrained by the intra-iteration precedency. One may improve speedup by further exploiting the concurrency among the multiple iterations constrained by the inter-iteration precedency.

To view inter-iteration precedency more clearly, one may unfold the DFG a number of times as suggested by Parhi (1989). An APG can then be used to construct an Admissible Scheduling (AS) with a shorter IP. One cannot improve the speedup factor or shorten the iteration period further by continuously supplying more processors. Such a shortest iteration period is termed the iteration bound. Chao and Sha (1992) have performed research to find the optimal combination of retiming and loop unfolding such that the resulting IP equals the iteration bound (IB).

Another technique is based on the observation that an single rate DFG (SRDFG) is equivalent to a marked graph which is a Petri net (PN), where each place has single input- and single output-transitions, and all arcs have unit weights. Ramamoorthy and Ho (1980) showed

that the IB is achievable. Therefore, there is no need for retiming and unfolding–a result unknown to digital signal processing (DSP) professionals. PN theory has been well developed, and its applications to DSP must be explored. The IB is referred to as the *minimum cycle time* for the corresponding PN (Ramamoorthy and Ho, 1980). Chao and Wang (1992, 1993a, 1993b, 1994) have developed techniques to find the IB, critical loops (CL), next-critical loops, subcritical loops, scheduling ranges, initial scheduling to avoid initial transients and steady state scheduling. This technique is fully static (no dynamic scheduling is necessary) and rate-optimal (maximum throughput), that is, IP=IB. No retiming nor unfolding is required.

The IB for an SRDFG can be found (Parhi, 1989) by finding the maximum of the loop bounds (LBs) of all the loops in a DFG. The LB of loop L_k is defined as $\frac{T^k}{D^k}$, where T^k and D^k stand for the sum of the execution times and the number of register elements or delays along L. Loops with the largest LBs are called critical loops. Each register element holds one piece of data.

This cannot be applied directly to multiple rate DFG (MRDFG) according to Chao and Wang (1993a) (Fig. 1) which is similar to an SRDFG except that an arc may carry multiple weights at its two ends. The weight at its source (destination) end indicates that when the source node (destination) executes, it injects (needs) the same number of data samples as the weight onto (from) the arc. For instance, the arc from node A to B in Fig. 1 carries weights 1 and 2, respectively, implying that when node A executes, it injects one data sample onto the arc, that and in order for node B to execute, the arc must have at least two data samples, and that after node B executes, it consumes two data samples from the arc. Thus, the execution of a node may require more than one input data from each input arc and produce more than one output data to be sent to each output arc.

As indicated by Lee *et al.* (1987), the SRDFG model is too restricted since it constrains the number of samples produced or consumed on an arc to unity. They also proposed heuristics for static scheduling of synchronous data flows (SDF), which is a large grain model similar to MRDFG. However, they did not discuss how to find IB and CL or their application for scheduling. The model of MRDFG is useful for describing some communications and signal processing algorithms involving interpolation and decimation operations (Lee *et al.*, 1987; Parhi, 1989). Unlike SRDFG, designers of MRDFG are constantly confronted with the problem of avoiding deadlocks and loss of data due to insufficient buffer space. The PN theory can resolve this

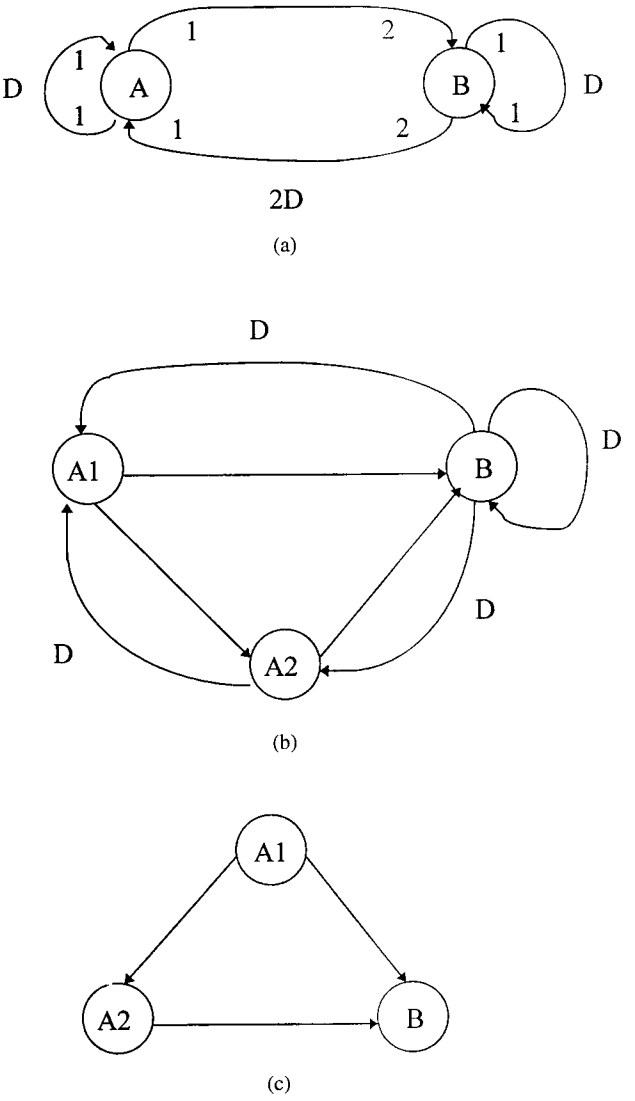


Fig. 1. (a) An example of a multiple-rate DFG. (b) The equivalent single-rate DFG. (c) The acylic precedence graph.

problem. Similar to SRDFG, we show in Section II that an MRDFG is equivalent to a Weighted T-Graph (WTG) (Teruel *et al.*, 1992), properties of which have been studied by Lien (1976) and Teruel *et al.* (1992). A WTG is a special class of General PN (GPN, where arcs carry multiple weights) and a natural generalization of a marked graph with arcs carrying multiple weights. The WTG is useful for modeling bulk arrivals and services (Lee *et al.*, 1987; Parhi, 1989; Teruel *et al.*, 1992) and automatic manufacturing systems (Chao and Wang, 1994; Chao *et al.*, 1994). We will not discuss the logical properties of WTGs in this paper; rather, we will concentrate on the performance of MRDFG, rarely discussed in the literature. To the best of the authors' knowledge, this work is the first show-

ing the above equivalence and suggesting the application of WTG theory to MRDFG. Most performance results of PNs (Magott, 1985; Morioka and Yamada, 1991; Parhi, 1989; Ramamoorthy and Ho, 1980) are for ordinary PN, rather than for GPN. Murata (1989) gave the lower bound of cycle times of general PNs similar to that provided by Campos *et al.* (1991) and Hillion (1988), which, however, dealt with unique consistent firing vectors but with random transition times. We will show that this lower bound is achievable when an MRDFG can be converted into an SRDFG.

The example in Fig. 2 illustrates the difficulty of deriving the IB for an MRDFG. In order for loop L_1 to execute once, loop L_2 has to execute twice. The IB is

$$I = \begin{cases} T^1 + T^2 & \text{for } T^2 \le T^1, \text{ and} \\ 2T^2 & \text{for } T^2 \ge T^1. \end{cases} \quad (1)$$

Thus, one can see that IB can be a linear combination of LBs of more than one loop. In this case, none of the loops are a CL, and we say that there is a *loop-combination*. This complicates the procedure for calculating the IB. By putting one more token in place p_2 of loop L_2, the two tokens flow together. Now, when loop L_1 executes once, loop L_2 also is executed once in the sense that the two tokens return once together to place p_2. The iteration bound is

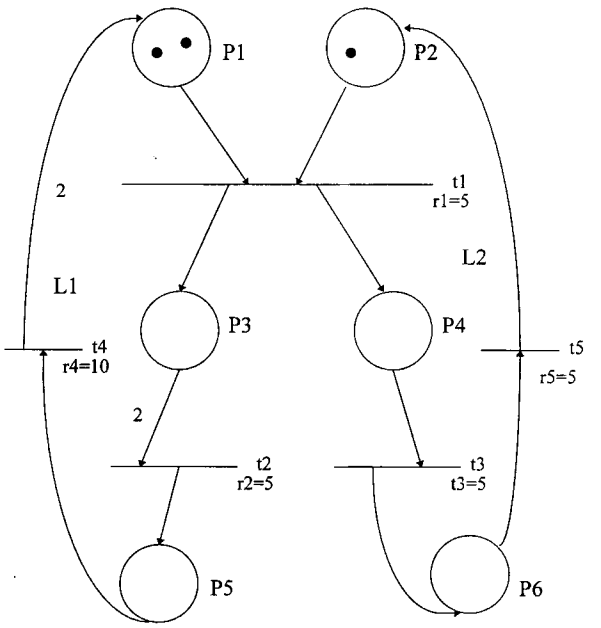


Fig. 2. An example of a multiple-arc Petri net whose cycle time is a linear sum of loop bounds.

$$I = \begin{cases} T^1 & \text{for } T^1 \ge T^2, \text{ and} \\ T^2 & \text{for } T^1 \le T^2. \end{cases} \quad (2)$$

Clearly, the CL does not involve loop-combinations. Also, I is shortened. Thus, loop-combination is marking related. It is also structure related: in Fig. 2, if all the edges carry a weight of one, there will be no loop-combination.

Parhi (1989) suggested converting MRDFG into the equivalent SRDFG to find the exact IB without presenting an explicit procedure. He did not, however, point out when such a conversion cannot be performed. This paper improves Parhi's result as follows: we (1) propose an algorithm to convert an MRDFG into its equivalent SRDFG; (2) identify the feasibility condition for the above conversion; and (3) under the above feasibility condition, develop a formula and the associated algorithm for calculating the IB, thus avoiding conversion, which takes more time and memory.

When an MRDFG behaves like an SRDFG, we can extend the technique for SRDFG to MRDFG. The IB for MRDFG with no loop-combination is similar to that for SRDFG except that the sum of tokens must be weighted; i.e., $I=\max\{\frac{T^k}{D_e^k};\ k=1, 2, \ldots, q\}$, where the equivalent D, $D_e^k = \sum_{h=1}^{K} Z_h$ and the weighted token $Z_h=\frac{M_h}{a_o(p_h)R_{h+1}}$. $a_i(p_k)$ and $a_o(p_k)$ are the number of input and output arcs, respectively, of place p_k, R_{h+1} is the $(h+1)$-th component of the minimal positive T-semiflow defined in Section II, and q is the total number of loops. Thus, the lower bound mentioned earlier by Murata (1989), Campos *et al.* (1991) and Hillion (1988) is achievable.

One special case of SRDFG equivalence would be that all Z_h are integral values (Section IV). The algorithm for SRDFG can be reused with slight modification to identify IB, critical loops, scheduling ranges, initial scheduling to avoid transients and a static scheduling under steady state for MRDFGs. As a result, the scope of the difficult problem of MRDFG performance and scheduling now reduces to that with non-SRDFG-equivalence.

We assume that the reader is familiar with various terminology of PN and DFG. Please refer to the paper by Murata (1989) for PN and Ackerman (1982) for DFG, respectively. This paper is organized as follows. Section II shows that an MRDFG can be converted to a WTG. The algorithm for converting an MRDFG into an SRDFG is presented in Section III. Section IV derives the explicit formula of the minimum cycle time for WTG or MRDFG. Section V presents the technique

for the determination of IB and CL. An example is shown in Section VI, followed by the X-Window implementation in Section VII for finding IB, CL, subcritical loops, and scheduling. Finally, conclusions are drawn in Section VIII.

II. Multi-Rate DFG

For the MRDFG in Fig. 1(a), the arc from A to B indicates that node A produces output to be consumed by node B. The numbers 1 and 2 at the head and tail of this arrow, respectively, show that one execution of node A produces one output sample, and that one execution of node B consumes two input samples from node A. It can be seen that nodes A and B are invoked twice and once, respectively, in one iteration of the DFG. The MRDFG models the following program (Parhi, 1989):

Initial Conditions: $aa(0)$, $bb(0)$, $ba(-1)$, and $ba(0)$
for each $\{n$=1 to $\infty\}\{$
for each $\{i$=1 to $2\}\{$
$aa(2n+i-2)=f_{aa}[aa(2n+i-3),\ ba(2n+i-4)]$
$ab(2n+i-2)=f_{ab}[aa(2n+i-3),\ ba(2n+i-4)]\}$.
$ba(2n-1)=f_{ba}[ab(2n-1),\ ab(2n),\ bb(n-1)]$
$ba(2n)=f_{ba}[ab(2n-1),\ ab(2n),\ bb(n-1)]$
$bb(n)=f_{bb}[ab(2n-1),\ ab(2n),\ bb(n-1)]\}$,

where $xy(n)$ indicates the data output from node x to node y at the n-th iteration, and $f_{xy}[]$ stands for the function to calculate $xy(n)$. In the n-th iteration of the MRDFG, the first invocation of node A computes $aa(2n-1)$ and $ab(2n-1)$, and the second invocation of node A computes $aa(2n)$ and $ab(2n)$. At the n-th iteration, node B computes $bb(n)$, $ba(2n-1)$ and $ba(2n)$.

The equivalent SRDFG in Fig. 1(b) shows that nodes A_1 and A_2 execute the odd and even invocations of A, respectively. Figure 1(c) illustrates the corresponding APG. It takes $(2r_a+r_b)$ units of time to execute one iteration of this MRDFG using a two-processor system, where r_a and r_b represent the computation times of nodes A and B, respectively.

The above initial condition is indicated by a "D" on the arc from node A to itself ($aa(0)$), a "D" on the arc from node B to itself ($bb(0)$), and a "2D" on the arc from node B to node A (two initial data $ba(-1)$ and $ba(0)$). It seems that we can use "D" to represent a data. The initial distribution of "D" on the DFG is called the initial marking M_0. $M(p)$ denotes the number of "D"s on arc p.

Based on these initial data, only node A can execute. Node A executes, by consuming $aa(0)$ and $ba(-1)$, to produce data $aa(1)$ on arc AA and data $ab(1)$ on arc AB. Note that both $aa(1)$ and $ab(1)$ are new data (rather than the initial data); they are produced in the first iteration. Thus, one can also consider "D" as both an initial data and a *delay element* (such as a register) such that the execution of a node using this initial data produces new data belonging to the next iteration. Node B still cannot execute because it requires two data on arc AB while node A can execute the second time by consuming $aa(1)$ and $ba(0)$ to produce $ab(2)$. Now node B can execute since arc AB has two data, $ab(1)$ and $ab(2)$ (indicated by "2D" on arc AB), arc BB has data $bb(0)$, and it produces new data $ba(1)$ and $ba(2)$ on arc BA. At this moment, the data distribution of the DFG returns to its initial state (marking) except that these data carry new iteration numbers, which in turn initiate new iterations. Thus, by the execution rule described in Parhi (1989), a node can execute as soon as each of its input arcs has enough data (equal to the number at the arrow end of the arc), and after it executes, it produces data equal to the number near the start end of each of its output arcs. These node execution semantics are similar to that of transition firing in a GPN (Lien, 1976), where a transition can fire if and only if all its input places hold enough tokens (equal to the weight of the corresponding input arcs to the transition).

Thus, one can consider nodes as transitions, arcs as places, "xD" on an arc as x tokens in the corresponding place, and node executions as transition firings. Thus, the number of "Ds" on all the arcs corresponding to the initial data can be viewed as the initial marking of the equivalent PN. Thus, any MRDFG is equivalent to a PN with multiple weights, i.e., a GPN. Note each the place of the equivalent PN has only one input and one output transition since each arc has only two ends. This kind of GPN is called a *Marked Graph* (MG) if all the arc weights of the PN are one. Otherwise, it is called a Weighted T-Graph (WTG) according to Teruel *et al.* (1992); the corresponding system is called the Weighted T-System. Further, we assume that the DFG and its corresponding PN are strongly connected; that is, there exists a directed path from any node to any other node. In the remainder of this paper, we use WTG and MRDFG interchangeably since they are equivalent. The following two definitions are useful in regard to weighted sum of tokens.

Definition (T-semiflow): $\boldsymbol{X}$ is called a *T-semiflow* if $\boldsymbol{X}$ is nonnegative and $\boldsymbol{A}^T\boldsymbol{X}=0$, where $\boldsymbol{A}^T$ is the transpose of the incidence matrix $\boldsymbol{A}$.

Definition (P-semiflow): $\boldsymbol{Y}$ is called a *P-semiflow* if $\boldsymbol{Y}$ is nonnegative and $\boldsymbol{A}\boldsymbol{Y}=0$.

Let $\boldsymbol{X}$ be a firing vector; then $\boldsymbol{M}=\boldsymbol{M}_0+\boldsymbol{A}^T\boldsymbol{X}$. If $\boldsymbol{X}$

is a T-semiflow, then $M=M_0$; i.e., it returns to M_0 and is *consistent*. Let R be the minimal positive T-semiflow such that starting with the initial marking, R_i is the minimum number of firings of t_i required to return to M_0, $\forall t_i$ in the PN. For a P-semiflow Y, $Y^TM=Y^TM_0+Y^TA^TX=Y^TM_0$. Thus, Y^TM stays constant independent of reachable M and is conservative.

III. Conversion of MRDFG to SRDFG

In this section, we will present an algorithm to convert an MRDFG to an SRDFG and identify the feasibility condition for such conversion. The explicit formula for the IB will be presented in the next section. In the example of Fig. 1, each node n_k is duplicated R_k times, and each arc from n_h to n_j is duplicated a_iR_h times with its delays evenly distributed among the duplicated arcs, where $a_i(a_o)$ denotes the multiplicity at the starting (ending) side of an arc. Thus, we propose the following:

1. Conversion Algorithm

(1) Find the minimal positive T-semiflow R.
(2) Duplicate each node n_k in the MRDFG R_k times, denoting as n_{ku}, u=1, 2, ..., R_k.
(3) Duplicate each arc (from n_h to n_j with ρ delays) in the MRDFG a_iR_h times. Each such arc connects from a n_{hu} to a n_{ju} in the following fashion:
   (i) There are totally a_i (a_o) arcs from (to) each n_h (n_j),
   (ii) a_i=1 and a_o=1 for each above arc, and
   (iii) each above arc carries a delay of $\frac{\rho}{a_iR_h}=\frac{\rho}{a_oR_j}$.

In the sequel, the SRDFG obtained using the above algorithm is defined as the equivalent SRDFG. Note that the number of delays on a duplicated arc may not be an integer in the above procedure; thus, the normal procedure for finding the IB for an SRDFG may not be applicable. Figure 3 shows such an example, but if we retime it first; the resulting SRDFG has integer delays. We will consider the special case where $\frac{\rho}{a_oR_j}$ is an integer.

The above equivalent SRDFG has, in general, many more nodes and edges than does the MRDFG, but many edges and nodes are redundant in determining the IB. Eliminating these redundancies reduces much the time for computing the IB. Using the results in Ito and Parhi (1994), each set of the above duplicated edges (nodes) can be degenerated (i.e., reduced) to a single edge (node) since they have identical delays. The resulting SRDFG has the same set of nodes and edges as does the original MRDFG with the number

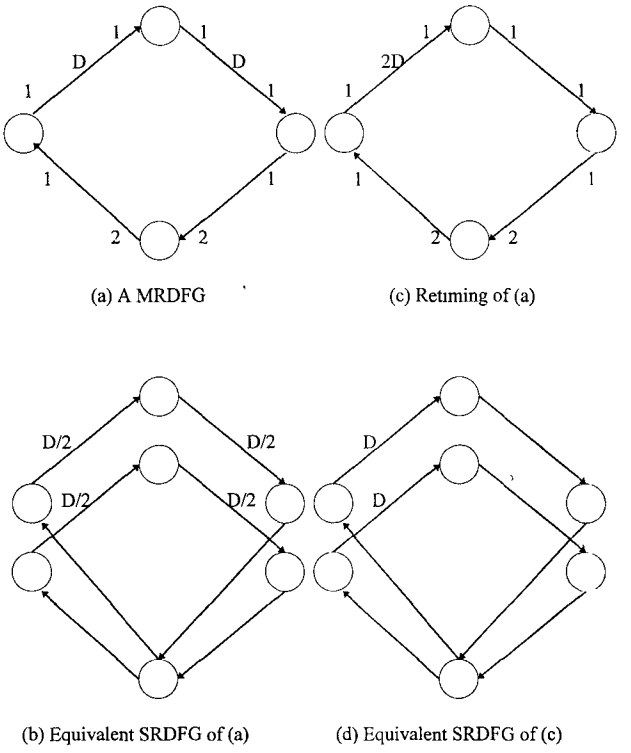


Fig. 3. An example of fractional delays.

of the delay elements replaced by $\frac{\rho}{a_oR_j}$. Thus, the MRDFG behaves like the corresponding SRDFG, and we difine such behavior equivalence as **SPDFG-equivalent**. It implies no loop-combination, but the converse is not true. The PN in Fig. 2 is a negative example unless we add one token to P2.

2. Determination of R

The above conversion algorithm requires knowledge of R, which will also be useful for finding the IB for MRDFG. We pick a transition t_1 randomly and assign x_1=1. For any other transition t_k, we pick an arbitrary directed path t_1–p_1–t_2–p_2–...–t_{k-1}–p_{k-1}–t_k; then, because A^TX=0, we have

$$x_k = x_1 \prod_{j=1}^{k-1} \frac{a_i(p_j)}{a_o(p_j)}. \qquad (3)$$

Since some x_k may not be an integer, we multiply all x_k by the least integer to make all new x_k integers. The resulting X vector is R. Applying this procedure to the PN in Fig. 2 and picking x_2=1, we have x_1=1/2 and x_3=1/2. Multiplying all x's by 2, we have R^T= [2 1 1], where the superscript "T" indicates the transpose.

When every $\frac{\rho}{a_i R_h}$ is an integer, we can simplify the IB calculation of the equivalent SRDFG as shown in the following section. Such an SRDFG is defined as a reducible SRDFG, and the corresponding MRDFG is said to have SRDFG-equivalence.

IV. Performance of MRDFG under SRDFG-Equivalence

Intuitively, when there is no loop-combination, an MRDFG behaves like an SRDFG because the number of delays of any loop is sufficient to fire the nodes in the loop in succession and to return all the delays to their initial arcs in one iteration. Thus, the techniques for calculating the IB and CL of SRDFG can be extended to MRDFG. To formally prove this, the concept of loop-combination is formally defined as follows:

Definition (Loop-Combination): A WTG is said to have *loop-combination* if there exists a transition t_i in a loop L such that the firing sequence $\sigma=\sigma_A\sigma_1\sigma_L\sigma_2\sigma_B$, where σ_A and σ_B are arbitrary firing sequences, and σ_1 and σ_2 contain t_i's only,

$$|\sigma_1|+|\sigma_2|=F_i^L, \tag{4}$$

and σ_L is a firing sequence containing all the transitions in another loop L, where F_i^L is defined as the R_i of t_i in L when L is isolated from the rest of the WTG. □

Recall that transitions in a WTG execute periodically. The corresponding period is defined as the *system period*. R_i is the number of firings of t_i in one *system period*. Similarly, if we consider each L in isolation (i.e., deleting all nodes and arcs not in the loop), F_i^L is the number of firings of t_i in one period. Such a period is specific to the loop and is defined as the *loop period*.

Equation (4) implies that no loop can complete one *loop period* without other loops completing their *loop periods*–a phenomena of loop-combination. It is rather hard to prove that the IB of an MRDFG can be calculated in a similar way to that for SRDFG from the above definition. Hence, a more constrained situation where every Z_h is an integer in the MRDFG is considered in the sequel. In such a case, every node n_i in the MRDFG, when enabled, can be fired an integral R_i times, and there will be no loop-combination. A more formal proof is provided in the sequel using the concept of Degenerating Action (DA).

Degenerating Action on the arc from n_i^u to n_j^v with γ delay elements in an equivalent SRDFG consists of the following subactions:

(1) Delete all n_i^u ($u\neq 1$) and n_j^v ($v\neq 1$).
(2) Move all incoming arcs to n_i^u ($u\neq 1$) to end at n_i^1.
(3) Move all outgoing arcs to n_v^v ($v\neq 1$) to start from n_v^1.

Lemma 1: The DA does not alter the IB of an equivalent SRDFG with every $\frac{\rho}{a_i R_h}$ being an integer.

Proof: Consider any loop L^a containing n_i^u ($u\neq 1$) and n_j^v ($v\neq 1$): n_α–n_β–n_γ–...–n_i^u–n_j^v–...–n_α. The DA results in loop L^b: n_α–n_β–n_γ–....–n_i^1–n_j^1–...–n_α. Since all n_i^u (n_j^v) have the same execution times, $T^a=T^b$. Also, all arcs $n_i^u\rightarrow n_j^v$ carry the same number of delay elements (by the assumption), $D^a=D^b$. Thus, the two loops have the same LBs. For any other loop in the equivalent SRDFG not containing any n_i^u and n_i^v, its LB is not affected by the DA.

Since IB is the maximum of all LBs, the DA does not alter the IB of the equivalent SRDFG. □

Based on the above lemma, we have the following theorem:

Theorem 1: If $\forall$ arc A_h, Z_h is an integer, then the IB of a strongly connected live and bounded MRDFG is given by

$$I=\max\{\frac{T^k}{D_e^k};\ k=1,\ 2,\ \ldots,\ q\}. \tag{4}$$

where q is the number of loops.

Proof: While a proof similar to the one in Ramamoorthy and Ho (1980) can be given, a simpler proof is sketched as follows. Repeatedly applying DA to the equivalent SRDFG leads to an SRDFG with sets of nodes and arcs identical to those of the original MRDFG and with Z_h delay elements on each arc A_h. Such an SRDFG has the same set of loops and the corresponding T^as as does the MRDFG. The D for each loop, however, must be replaced by the corresponding D_e. Thus, the IB of MRDFG equals that in Eq. (4). □

Note that the above formula for I is almost identical to that for SRDFG except for the subscript "e"; therefore, $\frac{T^l}{D_e^l}$ is the LB for L_l. Thus, we can apply the techniques for SRDFG to MRDFG without conversion by treating the MRDFG as an SRDFG with all arc multiplicities set to unity. For the example in Fig. 2,

$Z_e^1=1$ and $Z_e^2=1/2$. If we put one more token to p_2, then $Z_e^2=1$ and the loop bounds for L_1 and L_2 are 20 and 15, respectively. Hence, L_1 is the critical loop.

1. Comparison with the Lower Bound by Linear Programming

Note the above I is achievable as it is in the equivalent SRDFG. The above I was identified by Campos *et al.* (1991) and Murata (1989) as the lower bound of the minimum cycle time. Campos *et al.* converted the lower bound problem into a linear programming problem (LPP) (see also Morioka and Yamada (1991)), which, however, takes more time complexity than does the matrix approach of Chao and Wang (1992, 1993b) and Chao *et al.* (1993c). In addition, Campos *et al.* (1991) did not discuss how to extend the LPP to find critical loops, subcritical loops, scheduling ranges, steady state scheduling, and initial scheduling to avoid transients. Our matrix-based approach (Chao, 1995), nevertheless, can do all these tasks.

Lemma 2: For every p_k ($1\le k\le K$) of a loop L_l, t_1–p_1–t_2–p_2–...–t_{K-1}–p_{K-1}–t_K–p_K–t_1, there exists a constant c such that $Y_k=\frac{cZ_k}{M_k}$, where Y_k is the k-th component (for p_k) of the P-semiflow $\boldsymbol{Y}$ vector, and $Y_j=0$ if p_j is not in L_l.

Proof: $(AY)_k=\frac{ca_i(p_k)}{a_o(p_k)R_{k+1}}-\frac{ca_o(p_{k-1})}{a_o(p_{k-1})R_k}=c\frac{R_{k+1}}{R_k}\frac{1}{R_{k+1}}-\frac{c}{R_k}=0$. Hence, by the definition of P-semiflow, $Y_k(=\frac{cZ_k}{M_k})$ ($1\le k\le K$) is one component of a P-semiflow. □

Thus, $D_e^l=\sum_{k=1}^{K}Z_k=\frac{1}{c}\sum_{k=1}^{K}M_kY_k$ and $\frac{T^l}{D_e^l}=\frac{cT^l}{YM_0}=\frac{Y^T(A^-)^TGR}{Y^TM_0}$. Thus, the maximum LB corresponds to $\max_l\{\frac{Y^T(A^-)^TGR}{Y^TM_0};\ l=1, 2, ..., q\}$, where A^- is a $|P|\times|T|$ matrix with each entry, $a_{ij}^-=a_o(p_i)$ equals the weight between p_i and its output transition t_j, and G is a diagonal matrix with the i-th diagonal entry being the execution time for t_i. Thus, the lower bound by Campos *et al.* and Murata equals I in Theorem 1. The achievability of this lower bound, however, was not discussed in Campos *et al.* (1991), Morioka and Yamada (1991), and Murata (1989). Under loop-combination, the IB is a linear combination of LB of some loops; hence the lower bound, which is the maximum of all LB, may not be achievable. Further, Campos *et al.* did not present the complexity of their *LP* technique. The complexity of *LPP* is $O(n^5)$, according to Tardos (1986). This is still not as good as the $O(n^3\log n)$ in this work (see next section).

Note that because Y is a P-semiflow, $\sum_{k=1}^{K}M_kY_k$; hence, the sum of the weighted delays D_e^l is a constant for any M reachable from M_0 (Murata, 1989). Thus, even though Theorem 1 assumes that every loop contains only one marked place, it also holds for any M_0 from which existing a firing sequence σ to reach an M where every loop contains only one marked place.

Another contribution of this work is identification of the equivalence between MRDFG and SRDFG. Thus, most techniques for SRDFG are also applicable to MRDFG. The difficulty of MRDFG performance determination is now reduced to that of determining fractional values of D_e^l, thus reducing the scope of the problem.

V. Determination of Iteration Bound and Critical Loop

We follow the matrix-based approach (Chao and Wang, 1992, 1993b; Chao, 1994c) for SRDFG to find the IB and CL for MRDFG with no loop-combination. First, we build the initial matrix $Q^{(1)}$ with entry value $q_{ij}^{(1)}=RZ_k-r_i$, where R is a guess value for the minimum cycle time, and place p_k is in between transitions t_i and t_j. Based on the initial matrix, a series of intermediate matrices $Q^{(m)}$, m=2, 3, ..., can be found according to Floyd algorithm, taking time $O(n^3)$, where n is the total number of transitions (Floyd, 1962). Precisely, $q_{ij}^{(m)}=\min_{k\in\{1,2,...,n\}}[q_{ij}^{(m-1)}, (q_{ik}^{(m-1)}+q_{kj}^{(m-1)})]$. This is similar to matrix multiplication of $Q^{(m-1)}$ by itself if we view the operations "min" and "+" as "Σ" and "×" (multiplication), respectively. Let "*" indicate this kind of matrix multiplication; then, we have $Q^{(m)}=Q^{(m-1)}*Q^{(m-1)}$. There exists a limiting value of m such that the above operation fails to produce q_{ij} smaller than $q_{ij}^{(f)}$. The m-th intermediate matrix, with entries $q_{ij}^{(m)}$, represents the shortest distances between any two transitions, among all paths. These paths are composed of the intermediate transitions, which do not pass through more than m transitions. In particular, $q_{ii}^{(f)}$ indicates the shortest distance among all the loops through transition t_i. There are three cases associated with the diagonal entries as follows:

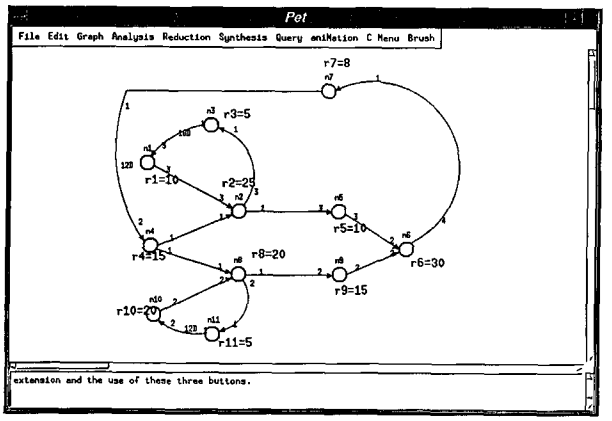
Theorem 2 (Ramamoorthy and Ho (1980)): In the final matrix $Q^{(f)}$, if the diagonal entries are (1) all positive, $R>I$, (2) some negative, $R<I$, and (3) one or more zeros and the rest positive, then $R=I$.

Thus, we can make a guess at I, apply Theorem

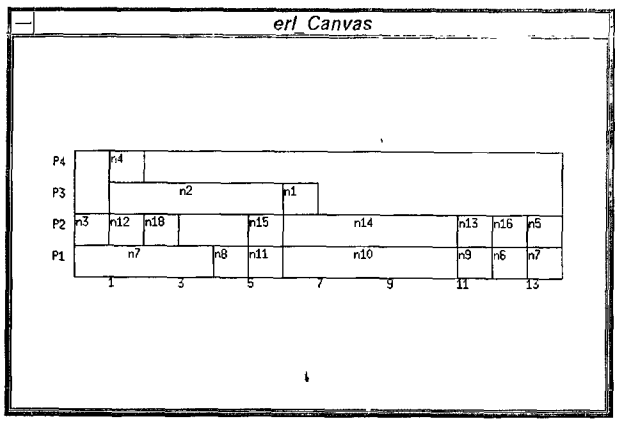
2 to determine which case applies and adjust the guess accordingly; e.g., if case (1) holds, we decrease the guess. We can choose zero as the lower bound of the guess since the iteration bound is always positive and the sum of the execution times of all the transitions as the upper bound. Using binary search, we can obtain I within tolerable errors. Nodes on critical loops and I can be identified. The total time complexity is $O(n^3 \log n)$ since there are $O(\log n)$ binary searches.

VI. Example

A parallel program is specified by the data flow graph in Fig. 4(a), and its execution requires an 11-node computer system. Each node is capable of bulk consumption and generation of data. For example, each execution of node n_1 requires that three data samples be available and produces another three. Initially, there are 18, 12, and 12 data samples on arcs (n_3, n_1), (n_7, n_4), and (n_{11}, n_{10}), respectively. All the other arcs have none. We assume that each execution of n_5 produces useful information for users.



(a)



(b)

Fig. 4. (a) IB, critical loops, subcritical loops, scheduling of a parallel program. (b) The level diagram for the DFG in Fig. 4(a).

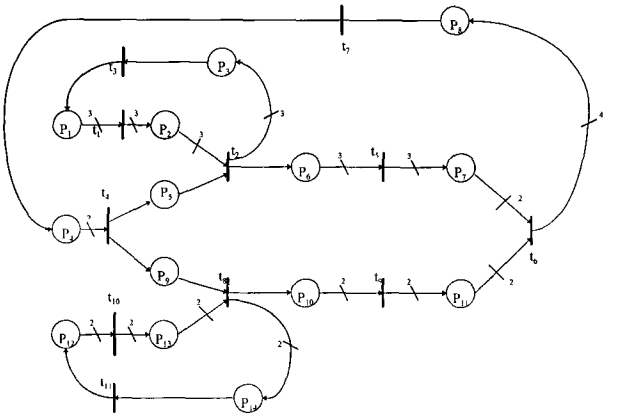


Fig. 5. The WTG model of the DFG in Fig. 4.

The corresponding WTG can be obtained by mapping each node in the data flow graph into a transition and each arc into a place, one input arc and one output arc. For example, n_i is mapped into t_i for $1<i<11$ in Fig. 4(a). Arc (n_3, n_1) is mapped into a place p_1, a single input arc (t_3, p_1), and arc (p_1, t_1) with multiplicity 3. 14 arcs in Fig. 4(a) lead to 14 places in Fig. 5. The initial markings of the places are determined by the initial data samples in the data flow graph. Thus, p_1, p_4, and p_{13} initially have 18, 12, and 12 tokens, and others have none. Delay times of transitions are execution times at nodes as shown in Fig. 4(a).

Based on Eq. (3), we solve R^T=[6, 6,18, 6, 2, 3,12, 6, 3, 6, 12]. With the given initial marking, the system cycle time is 88 units. Since R_5=2, the system node n_5 delivers twice the amount of useful information, or on average, for every 88 units of time, it does so once every 44 time units. To improve the speed, we can add more data samples for place p_4. Suppose that the number of tokens is doubled, i.e., 24. Then, the weighted token numbers for loops L_2 and L_3 are doubled, and the system cycle time becomes 45 units, which represents a significant improvement in efficiency compared with the original 88 units.

VII. X-Window Implementation

This section presents the software structure (Fig. 6, also shown in Chao and Wang (1993b)) of the X-Window implementation of the above algorithms. We reused the current software for SFDRG and included a new procedure invt_petri() for the calculation of R to find weighted delays. It can draw and simulate multi rate DFGs. Please refer to Chao and Wang

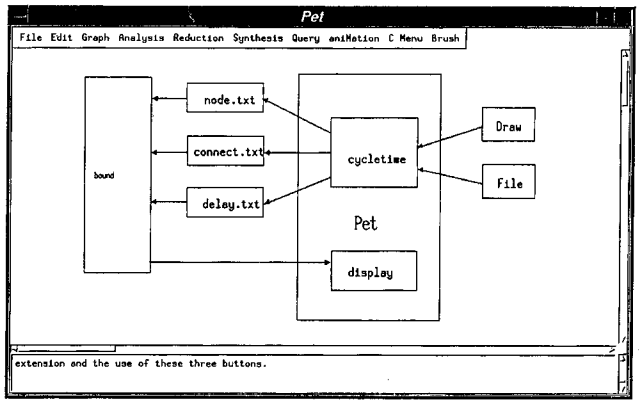


Fig. 6. Software structure used to calculate performance.

(1993b) for the X-Window implementation of SRDFG.

Unlike the tool which we presented in Chao and Wang (1993b), there is no need to construct the equivalent PN model of the DFG. The user can construct the SRDFG or MRDFG using the "DRAW" button to draw nodes and arcs, or by using the "FILE" button to input a DFG file. After the DFG is drawn or displayed on the screen, the graphical interconnections among nodes are automatically converted into an internal PN with a place for each node, a transition for each arc, and their input and output relationship.

Figure 6 shows the software structure used to calculate the cycle time. By clicking the "Cycletime" button of the "Query" menu, the procedure "cycletime" writes the characteristics of the input net into three files: connect.txt, node.txt, and delay.txt. File node.txt contains the execution time for each node. File connect.txt contains the information about connections among nodes. File delay.txt stores the weighted marking (token) information of places. It contains a two-dimensional matrix, and the number in the i-th row and j-th column indicates the weighted number of tokens on the arc between nodes i and j.

The program "bound" first reads in all the data mentioned above and calculates and writes the performance results to file "perf.dat". The main program "Pet" then reads this file and displays its contents in the bottom widget (window) called text_w (Fig. 4(a)). Unlike the program in Chao and Wang (1993b), it also displays the ranges of each side on its two sides and thickens all the edges on all critical loops.

Thus, it can be seen in Fig. 4(a) that there are two critical loops: $(n_4n_2n_5n_6n_7n_4)$ and $(n_4n_8n_9n_6n_7n_4)$. They have zero slackness while n_1, n_3 and n_{10}, n_{11} have slackness of 48 and 43, respectively. The level diagram is shown in Fig. 4(b). Due to the large size of the processor assignment chart, we do not show it here, but it is easy to analyze the total number of processors required. On all critical loops, the node with largest R_i=12, is n_7. Hence, we need at least 12 processors. Thanks to the large slackness of nodes not on critical loops, we can schedule each of the R_i executions at different time epochs within each system period in order to reduce the number of processors required. Note that only node n_3 holds R_i=18 greater than 12, so we need to stagger its start execution times within one X system period. From the level diagram, we can shift the start execution times from 0 to 15 for the last 6 executions of 18. The first 12 executions keep the start execution times at 0. Since n_1 is the node next to n_3 in the same loop, the last two executions of 6 must be staggered to 20 from 0. As a result, the total number of processors required is 12. It is easy to see that the utilization of some processors is low. Thus, for MRDFG, the benefits of high utilization of processors and rate-optimal may not be achieved at the same time. One may sacrifice the goal of an optimal rate (i.e., the iteration period is greater than the IB) to reduce the number of processors required.

VIII. Conclusion

As indicated by Lee and Messerschmitt (1987), SRDFG is restricted, and they presented heuristics to schedule the MRDFG. Parhi solved the problem of IB determination of MRDFGs by converting them into SRDFGs. This approach, however, suffers from several drawbacks. First, it works only under no loop-combination. Second, it increases the memory requirement by duplicating nodes and arcs, which further prolongs the time needed for the IB calculation of the equivalent SRDFG. Third, its use is suggested only without the procedure for conversion. We remove all these drawbacks by proposing an algorithm for conversion, by identifying the condition for no loop-combination, and by developing an explicit formula for IB similar to that of SRDFG. Intuitively, when there is no loop-combination, an MRDFG behaves like an SRDFG because the number of delays of any loop is sufficient to fire the nodes in the loop in succession and to return all the delays to their initial arcs in one iteration. Thus, the CAD tool that we have implemented to calculate the IB, CL, scheduling ranges, initial schedules to avoid transients of SRDFG and steady state schedules of SRDFG can be reused and extended to MRDFG. Further, an improvement on the work of Lee and Messerschmitt (1987) is that no heuristics are needed. The scope of the MRDFG performance problem is now reduced to the problem of determining the performance of the MRDFG with fractional values of D_e^K.

References

Ackerman, W. B. (1982) Data flow languages. *IEEE Computer*, **15**(2), 15-25.

Campos, J., G. Chiola, J. M. Colom, and M. Silva (1991) Ergodicity and throughput bounds of Petri nets with unique consistent firing vectors. *IEEE Trans. on Software Engineering*, **17**(2), 117-125.

Chao, D. Y. (1995) Application of final matrix to data flow graph scheduling using multiprocessors. *MIS Review*, **5**, 65-80.

Chao, L. F. and E. H. Sha (1992) Retiming and unfolding data-flow graphs. *Proc. of the 1992 International Conf. on Parallel Processing*, volume II, pp. 33-40.

Chao, D.Y. and D. T. Wang (1992) Theory, parallelization and scheduling of critical loop, subcritical loops, and next-critical loops. *Proc. Int'l Comp Symp* , pp. 285-292. Taichung, Taiwan, R.O.C.

Chao, D. Y. and D. T. Wang (1993a) Minimum marking for no loop-combination of general Petri nets. *Proc. MASCOTS'93*, pp. 265-270. San Diego, CA, U.S.A.

Chao, D. Y. and D T. Wang (1993b) Iteration bounds of single-rate data flow graphs for concurrent processing. *IEEE Trans. on Circuits and Systems*, **40**(CAS-I), 629-634.

Chao, D. Y. and D. T. Wang (1994) A synthesis technique of general Petri nets. *J. Systems Integration*, **4**(1), 67-102.

Chao, D. Y., M C Zhou, and D. T. Wang (1993) Multiple-weighted marked graphs. *Proceeding IFAC 1993 World Congress*, pp. 259-264, Sydney, Australia.

Chao, D. Y., M. C. Zhou, and D. T. Wang (1994) Extending knitting technique to Petri net synthesis of automated manufacturing systems. *Computer Journal*, **37**(1),1-10. British Computer Society, Oxford University Press. London, U.K.

Floyd, R. W. (1962) Algorithm 97: shortest path. *Comm. ACM*, **5**(6), 344-345.

Hillion, H. P. (1988) Timed Petri nets and application to multi-stage production systems. *Proc. of the 9th European Workshop on Applications and Theory of Petri Nets*, pp. 164-182. Venice, Italy.

Ito, K. and K. K. Parhi (1994) Determining the iteration bounds of single-rate and multirate data-flow graphs. *Proc. of the 1994 IEEE-Asia Pacific Conf. on Circuits and Systems*, pp. 163-168. Taipei, Taiwan, R.O.C.

Lee, E. A. and D. G. Messerschmitt (1987) Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, **36**(1), 24-35

Lien, Y. L. (1976) Termination properties of generalized Petri nets. *SIAM Journal of Computers*, **5**(2), 251-265.

Magott, J. (1985) Performance evaluation of systems of cyclic sequential processes with mutual exclusion using Petri nets. *Information Processing Letters*, **21**, 229-232.

Morioka, S. and T. Yamada (1991) Performance evaluation of marked graphs by linear programming. *Int'l J. Systems Sci* , **22**,1541-1552.

Murata, T. (1989) Petri Nets: properties, analysis and applications. *IEEE Proc.*, **77**(4), 541-580.

Parhi, K. K. (1989) Algorithm transformation techniques for concurrent processors. *IEEE Proc.*, 1879-1895.

Ramamoorthy, C. V. and G. S. Ho (1980) Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Trans. Software Engng.*, **6**(5), 440-449.

Tardos, E (1986) A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, **34**, 250-256

Teruel, E., P. Chrzastowski-Watchel, J. M. Colom, and M. Silva (1992) *On Weighted T-systems. Application and Theory of Petri Nets, Lecture Notes in Computer Science*, pp. 348-367. Springer-Verlag, Berlin, Germany.

多比率資料流程圖之轉換，週期限及X視窗實作

趙玉

國立政治大學資訊管理學系

摘　要

尋找「單比率資料流程圖」之週期限的技術早已存在於文獻中。李等指出單比率資料流程圖太多限制，而提出經驗法則去作「多比率資料流程圖」之排程。培西建議由「多比率資料流程圖」轉換成對等之「單比率資料流程圖」以尋找週期限，且提出一個轉換的程序。爲了成功轉換，他經由消除多餘的節點及線而縮短了尋找對等之「單比率資料流程圖」之週期限所需要的時間。基於此一結果，我們指出特殊案例的週期限的下界可由而實現。此外，一個單比率資料流程圖演算法可以應用於「辨別臨界迴圈」，排程範圍，初始排程去避免過度現象及位於穩定狀態中靜態的排程，而無需要去使用經驗法則。