Unfolding of Multirate Data-Flow Graph to Compute Iteration Bound

Daniel Yuh Chao

Department of Management and Information Science
National Cheng Chi University
Taipei, Taiwan, ROC
Ph.: (02)29387694; Fax: (02)29393754
yaw@mis.nccu.edu.tw

Abstract. Parhietal. find the iteration bound (IB) by considering the equivalent single-rate data-flow graph (SRDFG) N' of N, which is generally an exponential time task and the transformed SRDFG is much larger (grows exponentially) than the MRDFG. Ito et al. proposed a novel algorithm to remove node/edge redundancies taking extra time and memory, but losing schedule information of removed nodes. We propose to reduce the MRDFG in a loop-wise fashion (reduce the nodes/edges in a loop as a whole) with fewer nodes/edges. The scheduling of nodes in the MRDFG can be derived from that of the reduced SRDFG., where one invocation of a node n corresponds to a consecutive number of invocations of n in the MRDFG.

Keywords: Concurrent Processing, Data Flow Graph (DFG), iteration bound, Petri nets.

1 Introduction

External data to the system are sampled periodically at a certain period termed the iteration period which is also the time period of a DFG between two successive invocations of a node. System throughput is improved by shortening the sampling period with more processors. The improvement cannot be continued indefinitely, the lower bound to the iteration period is termed the iteration bound. The corresponding scheduling is called rate-optimal [1].

Signals in a multirate system modeled by a multirate data-flow graph (MRDFG) are sampled at different rates due to interpolation and decimation operations [2]. The invocation of a node (or actor) in an MRDFG, unlike SRDFG (Single-Rate Data-Flow Graph), may require more than one data from each input edge and may produce more than one output data to each output edge. Large grain MRDFGs have been widely used as a programming model for DSP applications.

Very few methods [3, 4] are available for MRDFG svheduling since scheduling for SRDFG always begins with a computed IB based on the minimum cost-to-time ratio cycle algorithm described in [5]. This algorithm, however, cannot be extended to multirate data-flow graph (MRDFG).

Parhi et al. find the IB by considering the equivalent single-rate data-flow graph (SRDFG) N' of N, which is generally an exponential time task and the transformed

A. Hua and S.-L. Chang (Eds.): ICA3PP 2009, LNCS 5574, pp. 460–465, 2009.

SRDFG is much larger (grows exponentially) than the MRDFG. Ito et al. [1] proposed a novel algorithm to remove node/edge redundancies taking extra time and memory, and in the worst case, no reduction may be available. Furthermore, it is unclear how to schedule removed nodes.

We propose to reduce the MRDFG in a loop-wise fashion (reduce the nodes/edges in a loop as a whole) with fewer nodes/edges. The scheduling of nodes in the MRDFG can be derived from that of the reduced SRDFG, where one invocation of a node n corresponds to a consecutive number of invocations of n in the MRDFG. Thus, unlike [1], there is no loss in scheduling information. Using the technique in [6] for SRDFG scheduling, we can compute the scheduling time τ of each node n in the reduced SRDFG, which corresponds to some consecutive number of invocations of n in the MRDFG. These invocations are scheduled at the same time τ.

We break the MRDFG into a number of loop-combination–free islands and duplicate each island a number of times. Each node n in the duplicated island represents some consecutive invocations of the same node n in the original MRDFG; thus each kth invocation of n can be scheduled avoiding the scheduling deficiency suffered in the approach by Ito et al. If two consecutive invocations of n is represented by the same n' in one island Πi but different n' in another island Πj, then we say the two islands are incompatible—fixed by expanding n' in Πi into two nodes with an intermediate place. Further reducing the computation time, we compute the final IB to be the maximal among iteration bounds of all islands with a time complexity faster by a factor of f^2 where f is the average number of islands.

The rest of the paper is organized as follows. Section 2 presents the preliminaries about data-flow graphs (DFG) followed by the proposed approach in Section 3. We propose the algorithm in Section 4. Section 5 concludes the paper.

2 Preliminaries

A data-flow graph (DFG) is denoted by $G=(N, E, W, r, d)$ where N is the set of nodes (also called actors), E is the set of directed edges, W is the set of weights associated with the edges, r is the set of execution times of the nodes, and M_0: $E \rightarrow Z^+$ is the net *initial marking* or *delay counts* assigned to each edge $e \in E$, $d=M_0(e)$ *delays* or *tokens*. G is an SRDFG (MRDFG) if none (some) of the weights are more than one. The set of input (output) nodes of an edge e is denoted by ${}^\bullet e$ ($e^\bullet$). Similarly, the set of input (output) edges of a node n is denoted by ${}^\bullet n$ ($n^\bullet$). This notation is also generalized to any set of edges or nodes, X, e.g. ${}^\bullet X = \cup_{x \in X} {}^\bullet x$. The ordered set $X = < x_1 \ldots x_n > \in (N \cup E)^*$ is a path, if and only if (iff) $x_{i+1} \in x_i^\bullet$; $i = 1, \ldots, n - 1$. Furthermore, a path X is characterized as a *circuit* or *loop* iff $x_1 \equiv x_n$. Associated with each edge (u, v), there are two weights I_{uv} (Input function) and O_{uv} (output function) at the beginning and ending part of the edge, respectively.

Given a marking M, a node v is *enabled*, if $\forall e \in {}^\bullet v$, $M(e) \geq O_{uv}$, $e=(u, v)$ and this is denoted by $M[n>$. $v \in N$ is said to be *disabled* by $e \in {}^\bullet v$ at M iff $M(p) < O_{uv}(e=(u, v))$. Firing an enabled node v results in a new marking M_1, which is obtained by removing O_{uv} tokens from each edge $e \in {}^\bullet v$, and placing $I_{vu'}$ tokens in each edge $e' \in v^\bullet$ $(e=(v, u'))$ moving the system state from M_0 to M_1. Repeating this process, it reaches M' by firing a sequence $\sigma = n_1 n_2 \ldots n_k$ of nodes. M' is said to be reachable from M_0; i.e.,

$M_0[\sigma>M'$. Input function I_{uv} is the number of tokens or data items consumed by v of edge (u, v) in each firing of v, output function O_{uv} is the number of tokens or data items produced by v of edge (u, v) in each firing or invocation of u, r is the set of invocation times of the nodes, and marking $M(e)$ [resp. $M_0(e)$] is the set of (resp. initial) delay counts or tokens on edge e. G is an SRDFG (MRDFG) if none (some) of I_{uv} and O_{uv} are more than one. For SRDFG, $T_k = \sum_{ni \in Lk} r_i$ denotes the sum of the invocation times r_i of nodes n_i in loop L_k. $N_k = \sum_{ei \in Lk} M_i$ denotes the total number of delay elements M_i of every edge e_i in loop L_k. *Loop Bound* of loop L_k, $LB_k = T_k/N_k$, is the total loop computation time, T_k, divided by the number of delay elements, N_k, inside loop L_k. *Cycle time or Iteration Bound* ($I = \max\{T_k/N_k. ; k\ 1, 2, \ldots, q\}$, q being the number of loops) of a system is the lower bound on the achievable iteration period. The loop with the (resp. next) largest loop bound is called the (resp. sub-) critical loop. R_u (denoted by k_u in [1]) denotes the number of invocations of node u in an iteration.

3 The Approach

When an MRDFG behaves like an SRDFG (i.e., can complete one system iteration by executing each loop once), we can extend the technique for SRDFG [6] to MRDFG [2, 3]. The IB for MRDFG with no loop-combination is similar to that for SRDFG except the sum of delays must be weighted; i.e., $I = \max\{T_k/D_e^{\,k}; k = 1,2, \ldots, q\}$, where the equivalent D, $D_e^{\,k} = \sum e_{h \in Lk}\ \mathfrak{I}_h$ with the weighted delay $\mathfrak{I}_h = \mathfrak{I}(e_h) = M_h/(a_o(e_h)R(n_o))$. Note that M_h is the delay at edge e_h in the loop L_k , $a_o(e_h) = O_{eh}$ denotes the number of data consumed from e_h by one invocation of the end node n_o of the edge, $R(n_o)$ is the the number of invocations of node n_o in an iteration, and q is the total number of loops.

For the example in Fig. 1, R^T=[2 2 1 1 2], $q=2$, $\mathfrak{I}^T$=[1 1/2 0 0 0 0] where $\mathfrak{I}_2 = 1/2$ is not an integer and hence it does not behave like an SRDFG (unless we add one delay to the edge e_2 to make $\mathfrak{I}_2 = 1$) since in order for n_2 to execute for the first time, each node (except n_1) in L_2 must have been executed once — a so-called loop-combination which occurs when there is a loop which must execute more than once in order to complete one system iteration. We propose to develop an algorithm to find iteration bounds of MRDFG with loop-combination.

We attack this problem first from simple cases such as the one in Fig 1. Note that in order for n_3 to execute once, n_1 must execute twice. The first invocation of n_1 leads to a delay blocked at the input edge of n_3. The delay produced by the second invocation of n_1, however, will not lead to delay blocking and the net behaves like an SRDFG. The time duration between the two invocations of n_1 is T_2, the sum of invocation times in L_2. Hence, after T_2, the net behaves like an SRDFG and after $T' = max(T_1, T_2)$, the net returns to its initial state. Thus, we have the iteration period $I = T_2 + max(T_1, T_2)$.

Note that the MRDFG can be reduced to an SRDFG (single rate DFG) in Fig. 2 with the same set of nodes, edges and node invocation times except for nodes (e.g. n_1, increased by T_2) involved with the loop combination. In general, if n_1 executes k times for n_2 to execute once, $I = (k-1)T_2 + max(T_1, T_2)$. In the reduced net, the invocation time of node $n_1$is increased by $(k-1)T_2$.

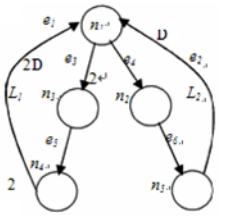

Fig. 1. Example of an MRDFG with loop combination

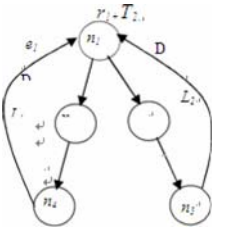

Fig. 2. Reduced SRDFG where all arcs are of unit weights and all r remain the same except r_1 is replaced by r_1+T_2

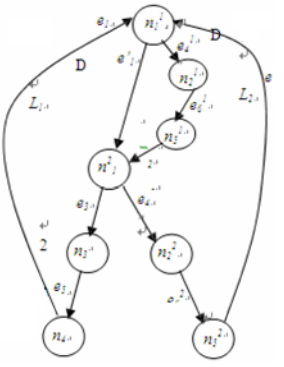

Fig. 3. SRDFG reduced from the MRDFG in Fig. 2

4 The Algorithm

We propose to compute the iteration bound (IB) of an MRDFG by converting the MRDFG into an equivalent SRDFG. It consists of the following steps: 1) decomposing it into a number of loop-combination free subnets or islands (lcf-islands), 2) find the duplication factor d_i for each lcf-island Π_i, 3) Setting all edge weights unity and expanding Π_i d_i times, 4) Merge all Π_i along common subislands, and 5) Compute the IB for the resulting SRDFG. Each step is explained after the following example.

There are 2 loops in Fig. 1 and each is an lcf-island. For L_1 to execute once, L_2 must execute twice implying $u=2$ for L_2. Thus, we unfold L_2 twice while making all edge weights unity and setting all nonzero initial delays to unity also. Now we expand (Step 6 in Algorithm I) n_1 in L_1 to n^1-e'_1- n_1^2 since n_1 is enlarged to n_1 and n'_1 in L_2. We now merge (Step 7 in Algorithm I) the new L_1 with the expanded L_2 along the common nodes n_1 and n'_1. The IB for the resulting net in Fig. 3 equals that of the MRDFG in Fig. 1.

We assume that Π_i is *single-enabled*; that is only one node in each loop is executable initially. However, the results also apply to the case of multiple-enabled Π_i using the retiming technique [2]. That is, we fire nodes in Π_i so that only one node is executable in Π_i.

Definition 1: Let R_i denote the R for island Π_i. Let $R(n)$ ($R_i(n)$) be the component of reproduction vector of node n for the DFG (island Π_i) and n be an initially enabled node in Π_i. The *total enabling factor* of Π_i, $\varsigma^n_i = M_0(e')/(a_o(e')R_i(n))$ where $n \in e'^{\bullet}$ is the ouput node of edge e'. $d_i=R(n)/R_i(n)$ is the *duplicate factor* of Π_i,

ς^t_i, when being an integer, is the number of island iterations (the system iteration for the isolated island) that can be performed by firing one round of all nodes in Π_i to return to the initial marking of Π_i. We say that Π_i is enabled ς^n_i times. d_i indicates the number of island iterations needed to complete one system iteration. We assume in Algorithm I that the fraction part alone would not make the island live; i.e.,

continuous executions of the island will reach a deadlock to simplify the presentation. Thus, $\varsigma^n{}_i$ will be assumed to be an integer in the algorithm.

If $\varsigma^n{}_i \geq d_i$, then Π_i is able to complete one system iteration by firing one round of all transitions in Π_i to return to M_0 and there is no need to unfold the island and the unfolding factor $q=1$. On the other hand, if $\varsigma^n{}_i < d_i$, the island should be unfolded $q=d_i=\lceil x \rceil$ (to be explained below) rather than d_i times where $x= d_i/\varsigma^n{}_i$ since one loop execution completes $\varsigma^n{}_i$ loop iterations and it needs $\lceil x \rceil$ loop executions to finish one system iteration. The procedure to perform these steps to find unfolding factor q is denoted by *unfolding_factor*().

Definition 2: Let ¶ be a subisland in island Π_i. $(¶)^k{}_i$ is the unfolding of ¶ for its kth execution in island Π_i. $(¶)_i^{a,(a+1),\ldots b}$ represents the a-th, (a+1)-th, …, b-th executions of ¶ in island Π_i indicating that $(¶)^a{}_i=(¶)^{a+1}{}_i=\ldots=(¶)^b{}_i$. A subisland ¶ in both Π_c and Π_d, is called to be incompatible if $\Pi_c \neq \Pi_d$, $\exists k>0$, such that $(¶^{k-1})_c=(¶^k)_c$, $(¶^{k-1})_d \neq (¶^k)_d$. The set of incompatible subislands in N is denoted by $¶_c$.

Island Π_i (each node n) is unfolded into Π_i^1, Π_i^2, …, and Π_i^q (n^1, n^2, …, and n^q). Except for the last Π_i^q, the execution of each Π_i^j, $j \in \{1,2,\ldots,q-1\}$ completes $\varsigma^n{}_i$ loop iterations. One execution of Π_i^q finishes $d_i - (q-1)\ \varsigma^n{}_i$ ($<\varsigma^n{}_i$) loop iterations.

When unfolding Π_i, we set all arc weights unity and reduce the initial marking so that each firing of node n^j (j-th unfolding of n) in N' represents the a-th, (a+1)-th, …, b-th firings of n in N. Hence we can also use $(n)^{a,(a+1),\ldots b}$ to represent n^j in N'. The procedure to do these steps is called *unfold* (q).

Note that ¶ may be a single node n. When $(n^{k-1})_c=(n^k)_c$ and $(n^{k-1})_d \neq (n^k)_d$, the $(k-1)$- and k-th firings of n in Π_c can be represented by a single node while that in Π_d cannot do so. As a result, in the final reduced SRDFG, $(n^{k-1})_c$ and $(n^k)_c$ in Π_c (e.g., Π_1) must be separated by an intermediate edge e'_1 so that node $(n^{k-1,k})_c$ in Π_c is now expanded to $(n^{k-1})_c \rightarrow e'_1 \rightarrow (n^k)_c$; i.e., edge e'_1 from node $(n^{k-1})_c$ to node $(n^k)_c$. The procedure to do such is called *expand*(Π_i).

Algorithm I for MRDFG conversion into SRDFG

Input: A MRDFG

Assumption: 1. Each island Π_i is *single-enabled*. 2. Fractional part of each $\varsigma^t{}_i$ can be ignored; i.e., only integral part of each $\varsigma^t{}_i$ is considered

Output: the equivalent SRDFG and the IB

1) Find R_i for each node based on the technique in [2].
2) Divide the net into separate loop-combination free islands (lcf-islands) (denoted by Π_i) based on the graph-transversal algorithms.
3) For each lcf-island Π_i, find its duplicate factor $u_i=R(n)/R_i(n)$ and unfolding factor using Procedure q=*unfolding_factor*().
4) Unfold each Π_i using Procedure *unfold*(q).
5) Expand the duplicated Π_i using the procedure *expand*(Π_i).
6) Merge all Π_i along common subislands.
7) Output the resulting MRDFG N'.
8) Compute and output the IB for N'.

5 Conclusion

Rather than duplicate and remove redundant nodes/edges individually, we reduce them in a loop-wise fashion (reduce the nodes/edges in a loop as a whole) which is more efficient and results in fewer nodes/edges. We further fold several duplicated loops of the same loop into a single one; thus reducing more nodes/edges which is not possible using Ito *el al.'s* approach. We break the MRDFG into a number of loop-combination–free islands and duplicate each island a number of times. Each node *n'* in the duplicated island represents some consecutive invocations of the same node *n* in the original MRDFG.

Compared with the approach by Ito *et al*., ours posts the following advantages:

1. In the best case, we are able to reduce an MRDFG to an SRDFG with the same number of nodes and edges, while it is not possible in [1].

2. We do not lose the scheduling information of some nodes while it is possible in [1] if some duplicates of them are removed in the reduced SRDFG.

3. In general, we are able to reduce an MRDFG to an SRDFG with a less number of nodes and edges for two reasons. First, Ito *et al.* duplicate and reduce nodes and edges individually. This is less efficient than our uniform approach where we consider a loop as a whole and duplicate each node and edge in a loop the same number of times without any reduction and it results in a less number of nodes and edges.

References

1. Ito, K., Parhi, K.K.: Determining the minimum iteration period of an algorithm. Journal of VLSI Signal Processing 11, 229–244 (1995)
2. Chao, D.Y.: Conversion, iteration bound and X-WINDOW implementation for multi-rate data flow graphs. Proceedings of the National Science Council, Part A: Physical Science and Engineering 22(3), 362–371 (1998)
3. Chao, D.Y.: Performance of multi-rate data flow graphs for concurrent processing. Journal of Information Science and Engineering 13(1), 85–123 (1997)
4. Chao, D.Y., Wang, D.T.: Iteration bunds of single-rate data flow graphs for concurrent processing. IEEE Trans. Circuits and Systems, CAS-1 40(9), 629–634 (1993)
5. Parhi, K.K.: Algorithm transformation techniques for concurrent processors. Proc. IEEE 77, 1879–1895 (1989)
6. Chao, D.Y.: A Fast Implementation For Recurrent DSP Scheduling Using Final Matrix. Journal of Information Science andEngineering 16(3), 391–421 (2000)