MINIMUM INITIATION INTERVAL OF MULTI-MODULE RECURRENT SIGNAL PROCESSING ALGORITHM REALIZATION WITH FIXED COMMUNICATION DELAY

Hung-ying Tyan

Electrical Engineering Department,
Ohio State University,
Columbus, OH

Yu Hen Hu

Dept. of Electrical and Computer Engineering,
University of Wisconsin,
Madison, WI 53706-1691,
hu@engr.wisc.edu

ABSTRACT

A novel iterative algorithm is proposed to compute the theoretical minimum initiation interval of a given recurrent algorithm when there is a known, fixed inter-module communication delay. Specifically, for a twin-module implementation problem, a novel representation called necessary initiation interval is introduced to faciliate the development of an iterative algorithm which yields both the minimum initiation interval and the corresponding cut set of the cyclic iterative computational dependence graph (ICDG). The convergence of this iterative algorithm in finite iterations is also proved.

1. INTRODUCTION

A simple example of a recurrent algorithm is an infinite impulse response (IIR) digital filter:

$$y(n) = a * y(n-1) + b * u(n) \tag{1}$$

where $y(n)$ is the output and $u(n)$ is the input. The computation of $y(n)$ can not be commenced until the completion of the previous iteration where $y(n-1)$ is computed. In other words, the n^{th} iteration depends on the $(n-1)^{th}$ iteration. The time duration between the beginning of the execution of the n^{th} iteration and that of the $(n+1)^{th}$ iteration is called the initiation interval. Due to the inter-iteration dependence relation, the minimum initiation interval gives an upper bound of the throughput attainable on a multi-processor implementation of the recurrent algorithm assuming negligeble inter-processor communication delay. Hence, deriving the minimum initiation interval is a crucial first step for the multi-module implementation of a recurrent DSP algorithm.

In this paper, we consider a multi-module realization of recurrent algorithms with fixed inter-module communication delay. A module may refer to an IC chip or other physical packages such as multi-chip modules (MCM). Each module may contain one or more processing elements (PE). PEs within the same module have dedicated communication links with negligible delay. However, due to limited number of input/output links of each module, communications across module boundaries will incur a fixed communication delay which can not be ignored.

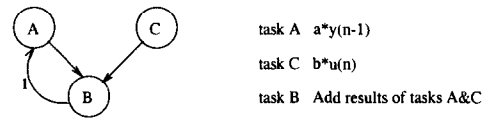

Figure 1: An ICDG example.

As shown in Fig. 1, a recurrent DSP algorithm can be graphically represented by a directed graph, called iterative computing dependence graph (ICDG), $G = (N, E)$. N is the set of nodes corresponding to the operations in the algorithm and is represented in the ICDG with associated computation time. E is the set of directed edges indicating data dependence between two operations. We define τ_c as the cycle computing time which equals to the sum of all the node computing time in a cycle C. We also define Δ_C (≥ 1) to be the total dependence distance of a cycle C, which is the sum of the delay elements on all edges of that cycle. The minimum initiation interval, denoted by $I_{min}(G)$, then can be computed as:

$$I_{min}(G) = \max_{\forall C \in G} \frac{\mathcal{T}_C}{\Delta_C} \tag{2}$$

In [3], a realizability criterion is proposed to determine if a multi-processor realization of a given recurrent algorithm is feasible with a specific initiation interval. In [1], a number of algorithm transformation techniques are given to derive the desired multi-processor realization. In [4], an efficient realization method is proposed. In these previous results, it is assumed that inter-processor communication delay is negligible.

2. TWO-MODULE IMPLEMENTATION PROBLEM

To take into account the inter-module communication delay, the ICDG must be modified to reflect how the tasks are partitioned into different modules. The set of edges which correspond to inter-module communication forms a cut set. A delay node will be inserted to each edge in this cut set so that the delay incurred on these edges is treated as if they were extra computing nodes while calculating the minimum initiation interval. In other words, the minimum initiation interval now is determined by both the original recurrent

algorithm (ICDG) and the way the ICDG is partitioned into different modules.

In this paper, we consider only the case of two-module partitioning (ie. M = 2). Work is underway to consider more general cases. The process of assigning computation nodes in the ICDG to two modules is equivalent to that of finding a cut set which bisects the ICDG into two subgraphs. The minimum initiation interval can be computed for this particular cut set. The objective of the two-module implementation problem then is to find the optimal cut set such that the corresponding minimum initiation interval is minimum among that of all possible cut sets. We note that an m-module implementation problem has been investigated preliminarily in [5].

Taking this cut set into account, we consider a generalized ICDG $G'(N, E, K)$ where K is the set of edges which forms the cut set. Let us denote G' to be the ICDG transformed from G by replacing each edge $e \in K$ with a branch of the same source and destination and a delay node which models the inter-module communication delay. Once this transformation is done, the resulting ICDG, G', can be analyzed as if there were no inter-module communication delay as long as tasks assigned to the same module stay within the same module. Therefore, based on Eq. (2), the minimum initiation interval corresponding to a particular cut set K can be found as:

$$\begin{aligned} I(K) &= \max_{\forall C \in G'} \frac{\tau_C}{\Delta_C}, \\ &= \max_{\forall C \in G} \frac{\tau_C + |e \in C \cap K| \cdot H}{\Delta_C} \end{aligned} \tag{3}$$

Where $|e \in C \cap K|$ is the number of edges in the intersection $C \cap K$, and, by definition, $|\emptyset| = 0$. With these notations, we are ready to formulate the two-module implementation problem:

Problem 1. Given a ICDG $G = (N, E)$ and a constant inter-module communication delay H. Denote $\mathcal{K}$ to be the set of all possible cut sets which can be imposed on G. Find an optimal cut set K^* such that

$$K^* = \arg \min_{\forall K \in \mathcal{K}} I(K)$$

Before proceed further, let us comment that since the minimum initiation interval is defined on directed cycles in the ICDG, any cut set which does not break any cycle will not affect the minimum initiation interval regardless the communication delay H. Hence in this paper, we shall focus on the partition of tightly coupled ICDG where every edge is part of a cycle. In the following three lemma, we will show that if G is not a tightly coupled ICDG, then there exists an optimal two-module partition of G such that the resulting minimum initiation interval will not be affected by the inter-module communication delay.

Due to space limitations, in the following development, proofs of lemma and theorems are omitted. They can be found in an accompanying full paper in preparation [2].

3. NECESSARY INITIATION INTERVAL

Let us denote $\mathcal{K}_e = \{K | e \in K, K \in \mathcal{K}\}$ to be the set of cut sets each of which contains a particular edge e. The necessary initiation interval of edge $e \in E$ is defined as :

$$I_e = \min_{K \in \mathcal{K}_e} I(K).$$

Thus, I_e is the smallest initiation interval of all cut sets which contain a specific edge e. Similarly, denote $\mathcal{K}_C = \{K | C \cap K \neq \emptyset, K \in \mathcal{K}\}$ to be the set of cut sets which intersect with a particular cycle C. The necessary initiation interval of cycle $C \in G$ is defined as :

$$I_C = \min_{K \in \mathcal{K}_C} I(K)$$

I_c is the smallest initiation interval of all cut sets which break a specific cycle C. From these definitions, it is easy to show that for $\forall K \in \mathcal{K}$,

$$I(K) \geq \max_{\forall e \in K} I_e; \quad I(K) \geq \max_{\forall C \cap K \neq \emptyset} I_C$$

With either I_e or I_C, the minimum initiation interval in problem 1 can be obtained by

$$I_{min}(H) = \min_{\forall e \in G} I_e \quad \text{or} \quad I_{min}(H) = \min_{\forall C \in G} I_C \tag{4}$$

The relations between I_e and I_C are summaried in the lemma below:

Lemma 1

$$I_e \geq \max_{e \in C} I_C \tag{5}$$

$$I_C = \min_{e \in C} I_e \tag{6}$$

The next lemma states that for each cycle C, there are at least a pair of edges which attend the minimum I_e:

Lemma 2 For $\forall C \in G$ and $\forall e \in C$, there exists another edge $e' \in C$ such that $I_{e'} \leq I_e$.

Consequently, let

$$e^* = \arg \min_{e \in C} I(e)$$

Then, there exists an $e' \in C$ and $e' \neq e^*$ such that $I_{e'} = I_{e^*}$. Or, equivalently,

$$I_C = \min \max_{e_1, e_2 \in C} \{I_{e_1}, I_{e_2}\} \tag{7}$$

3.1. Simple Cut Set

A cut set $K \in \mathcal{K}$ is a simple cut set if and only if for any cycle $C \in G$, $C \cap K \neq \emptyset$, $|K \cap C| = 2$. If K is a simple cut set, according to Eq. (3),

$$I(K) = \max_{\forall C \in G} \left\{ \max_{C \cap K \neq \emptyset} \frac{\tau_C + 2H}{\Delta_C}, I_{min}(0) \right\} \tag{8}$$

where

$$I_{min}(0) = \max_{\forall C \in G} \frac{\tau_C}{\Delta_C}$$

is the minimum initiation interval without any partitioning.

Lemma 3

$$I_e \geq \max\left\{\max_{\forall C, e\in C} \frac{\tau_C + 2H}{\Delta_C}, I_{min}(0)\right\}$$

$$I_C \geq \max\left\{\frac{\tau_C + 2H}{\Delta_C}, I_{min}(0)\right\}$$

Theorem 1 If K is a simple cut set, then

$$I(K) = \max_{\forall e\in K} I_e$$

3.2. Single-Node Cut Set

A single node cut set is a cut set which separates a specific node from the remaining nodes of the graph. More importantly, a single node cut set is a also simple cut set.

The following theorem states that to find optimal two-chip partition, we need to consider only simple cut sets.

Theorem 2 For any edge $e \in G$, there exists a simple cut set, say K_e, such that $I_e = I(K_e)$. Similarly, for any cycle $C \in G$, there exists a simple cut set, say K_C, such that $I_C = I(K_C)$.

3.3. The Iterative Algorithm

As described in previous sections, computing I_e becomes the main task to solve our problem, as well as to reduce the size of the multi-chip implementation problem. However, it is still not efficient to compute I_e by its definition. Instead, we propose an efficient iterative algorithm to compute I_e. The method is based on the recursive relation between I_e and I_C, which are described in eqs. (5) and (7).

If we can find a good set of initial values for either I_e's or I_C's, then we can use Eq. (5) and (7) as the iterative formulas to update I_e's and I_C's until they converge. Lemma 3 gives a good set of initial values for the iterative algorithm below.

Theorem 3 The Iterative Algorithm – Let

$$i_C(0) = \max\{\frac{\tau_C + 2H}{\Delta_C}, \max_{\forall C'\in G} \frac{\tau_{C'}}{\Delta_{C'}}\}, \quad \forall C \in G \quad (9)$$

$$i_e(0) = 0, \quad \forall e \in G \quad (10)$$

and the iterative formulas

$$i_e(t+1) = \max_{\forall C, e\in C} i_C(t), \quad \forall C \in G \quad (11)$$

$$i_C(t+1) = \min_{\forall e_1, e_2 \in C} \max\{i_{e_1}(t+1), i_{e_2}(t+1)\} \quad (12)$$

Then for some $1 \leq t_1 < \infty$, and $t > t_1$

$$i_e(t) = i_e(t_1) \quad \forall e \in G \quad (13)$$

$$i_C(t) = i_C(t_1) \quad \forall C \in G \quad (14)$$

The convergence criterion is either $i_C(t) = i_C(t-1)$, for $\forall C \in G$, or $i_e(t) = i_e(t-1)$, for $\forall e \in G$. Furthermore,

$$I_e = i_e(t_1), \quad \forall e \in G \quad (15)$$

$$I_C = i_C(t_1), \quad \forall C \in G \quad (16)$$

Outline of Proof: The proof is divided into four steps. Let $i_e(\infty)$ and $i_C(\infty)$ be the convergent values of $i_e(t)$ and $i_C(t)$, we will prove

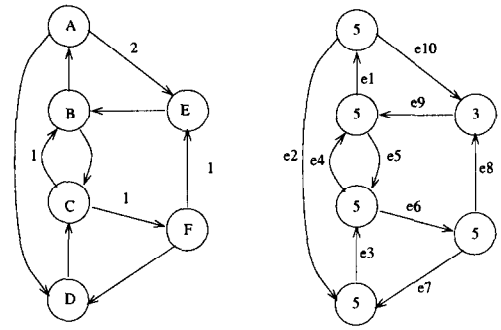

Figure 2: The ICDG in example 1.

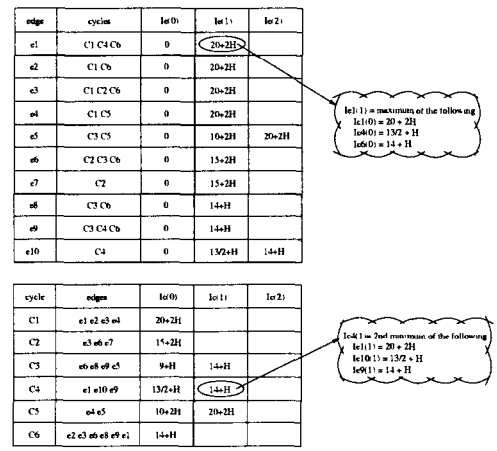

Figure 3: The list of computation for example 1.

1. $i_e(t)$ and $i_C(t)$ are non-decreasing in t.
2. The number of iterations for $i_e(t)$ and $i_C(t)$ to converge is finite (i.e., t_1 exists).
3. $I_e \geq i_e(t)$, $I_C \geq i_C(t)$ for all t. The result will be used in step four.
4. $I_e = i_e(\infty), \forall e \in G$, $I_C = i_C(\infty), \forall C \in G$.

The detailed proof can be found in [2].

The complexity of the iterative algorithm is O(the number of iterations $\times(\mid E \mid + \mid C \mid)$) of operations of finding extreme values in a partial set, where $\mid C \mid$ is the number of cycles.

4. TWO EXAMPLES

In this section, two examples are given to observe the convergence of the iterative algorithm.

Example 1 The graph on the left-hand side of Fig. 2 shows the ICDG demonstrated in this example. The graph on the right-hand side of the figure labels the edges in the ICDG. In Fig. 3, we list all the computation up to the convergence and indicate two cases of computations, $I_{e_1}(1)$ and $I_{C_4}(1)$, for better understanding. To focus on how the graph structure affects the changes of i_e and i_C values, we suppose that

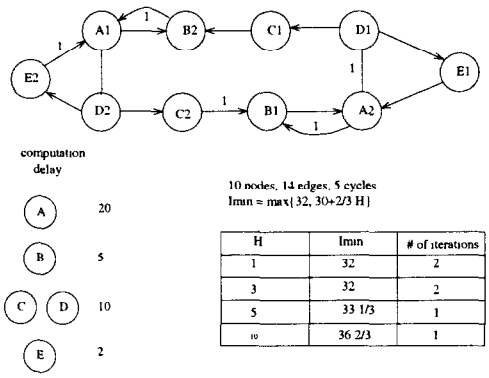

Figure 4: Example 2.

Example	# of nodes	# of edges	# of cycles	# of iter.
40th orthogonal filter	473	590	757	2
48th Gray-Markel filter	289	424	1176	3
32th LMS filter	129	247	48	2
42th normalized filter	337	461	871	2

Table 1: Experimental results.

H is large enough such that $\max_{\forall C \in G} \frac{\tau_C + 2H}{\Delta_C}$, the initiation interval in single-chip implementation, doesn't involve through the computation. The first column is the edge (cycle) list. The second column lists all the cycles (edges) contains (constructs) the edge (cycle). The remaining columns are the results of the iterations. In this example, it takes two iterations to converge.

Example 2 Figure 4 shows the ICDG of this example. In this example, we use several H values and observe the number of iterations needed to converge. The result shows the iterative algorithm is not sensitive to the change of H.

5. EXPERIMENTAL RESULTS

The iterative algorithm has been implemented for experimenting its convergence behavior and sensitivity to the change of H. Four higher order filters from [4] are tested in our experiments. We change H from 1 to 1000 non-uniformly for each benchmark and the result shows that the number of iterations to converge does not change with H in all the benchmarks tested. In additions, the iterative algorithm is shown very efficient in terms of the number of iterations to converge. For a large algorithm such as 48th Gray-Markel filter having 289 nodes, 424 edges and 1176 cycles, the number of iterations to converge is as low as three. The results are summarized in the table below:

6. DISCUSSION

We have proposed an efficient iterative algorithm to compute the necessary initiation interval of edges, I_e's. After I_e's are computed, the minimum initiation interval of two-chip assignment is obtained with ease. The minimum initiation interval of two-chip assignment serves a base of whether the DSP algorithm can be implemented in a multi-chip fashion. In additions, we can locate disqualified edges and prune them away before proceeding some m-chip assignment procedure. In such a way, the size of the resulting graph may be reduced significantly.

7. REFERENCES

[1] K. K. Parhi. Algorithm transformation techniques for concurrent processors. Proc. IEEE, 77:1879–1895, 1989.

[2] Hung-Yin Tyan and Yu Hen Hu. Minimum initiation interval of multi-module implementation of recurrent signal processing algorithm with fixed communication delay. IEEE Trans. on VLSI Systems, 1998 (submitted).

[3] Duen-Jeng Wang and Yu Hen Hu. Fully static multiprocessor array realizability criteria for real-time recurrent dsp applications. IEEE Trans. Signal Processing, 42:1288–1292, May 1994.

[4] Duen-Jeng Wang and Yu Hen Hu. Multiprocessor implementation of real-time dsp algorithms. IEEE Trans. on VLSI Systems, 3(3):393–403, September 1995.

[5] Duen-Jeng Wang and Yu Hen Hu. Synthesis of real-time recursive dsp algorithms using multiple chips. Proc. Great Lake Symposium on VLSI, pages 8–13, April 1996.