V1.2

A Globally Static Rate Optimal Scheduling for Recursive DSP Algorithms

Lih-Gwo Jeng and Liang-Gee Chen*
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan 10764, R. O. C.

Abstract

This paper presents a set of techniques for rate-optimal scheduling of recursive DSP algorithms. The underlying multiprocessor system is based on the RISC architecture. The retiming process evenly redistributes the delays and optimizes the scheduling time of each iteration. The unfolding of recursive data flow graph exploits intra-iteration and inter-iteration parallelism, and the schedule can be more compacted. It is shown that after evenly retiming and unfolding with the optimal unfolding factor, a fully-extended precedence graph is generated, which can be scheduled in rate-optimal by a simple contiguous scheduling. The optimal unfolding factor is merely equal to the delay counts of the critical loop.

1 Introduction

For hard real-time implementation, special techniques are needed. In particular, ASIC implementations of DSP applications require static scheduling, in which the DSP application algorithm is mapped onto the hardware at design time or at compile time, but certainly not at run time [1,2,3]. For example, systolic arrays can be synthesized from the dependence graph description of an algorithm [4].

In this paper, we introduce the notion of a *fully-extended precedence graph* which specified the operation parallelism and precedence relations both in one iteration of the algorithm and in contiguous iterations. The underlying synchronous, multiprocessor system is based on the RISC architecture , each instruction is complete within one clock cycle. The retiming process [5] is applied here to evenly redistribute the delay nodes (data buffers). The unfolding process involves multiple iterations of the algorithm to generate a compacted precedence graph. It is shown that construction of a *fully-extended precedence graph* is derived directly from the recursive data flow graph and can be executed on a rate-optimal schedule. The optimal unfolding factor is given by the loop delay counts of the critical loop of the recursive data flow graph. This will generates a reasonable problem size of the intermediate *fully-extended precedence graph* representation to specify the inter-iteration and intra-iteration operation parallelism and a more compacted rate-optimal schedule can be derived.

*This work is supported in part by National Science Council under Grant NSC79-0404-E002-34 and NSC78-0404-E002-47.

The scheduler described in this paper are incorporated into the DSP/DA system [6], which is an automatic and interactive design system targeted at high performance signal processing applications.

1.1 Graph Bound and Retiming

Considering a DFG involving loops or recursion or feedback cycles, which has an upper bound on the computation rate or a low bound on the task iteration period [1]. This iteration period bound, P_{min}, is given by '$P_{min} = \max_{l \in loops} \frac{T_l}{n_l}$' where the maximum is taken over all loops in the DFG, and T_l is the sum of the computation times associated with all the nodes in loop l, and n_l is the number of delay elements in loop l. Any loop l for which $T_l/n_l = P_{min}$ is defined to be a critical loop. In the context of this work, any realization that achieves an iteration period equal to the iteration period bound is considered to be rate-optimal, because no faster realization of the underlying graph can exist.

Traditional retiming technique is used to improve the clock rate of synchronous circuit by Leiserson, Rose and Saxe [5], and is applied here to improve the iteration period of the static multiprocessor schedule of the recursive DFG. The retiming technique attempts to evenly redistributes the delays [5], and also corresponds to a cutset transformation around the node in the systolic realization [4]. The retiming model of the system is based on the RISC architecture.

Section 2 describes the *fully-extend precedence graph* and the basic concepts underlying our optimization process of the scheduling algorithm. Section 3 presents the details of the optimization by retiming and unfolding. The rate-optimal scheduling are addressed in section 4 . Section 5 presents our conclusions.

2 Fully-Extended Precedence Graph

The precedence graph is an acyclic, directed graph originally used for the representation of sequential precedence of operations. Owing to its close relationship to the data flow graph of a recursive algorithm, the precedence graph representation is suitable for static scheduling of the recursive algorithm onto a multiprocessor system. Since a recursive DSP algorithm is naturally described by a data flow graph with directed loops, which will be used to set up the corre-

CH2977-7/91/0000-1005 $1.00 © 1991 IEEE

sponding *fully-expended precedence graph*.

Definition 1 *A fully-extended precedence graph is defined as the precedence graph whose critical path is equivalent to the computation path of the critical loop of the original recursive data flow graph.*

A *fully-extended precedence graph*, which put emphasis on the equivalence between the critical path and the critical loop of the original recursive DFG, is different from the precedence graph [4]. A *fully-extended precedence graph* may consist of successive iterations of the recursive data flow graph; or in another word, is a compacted combination of several successive precedence graph. Comparing the recursive data flow graph shown in figure 3 and the corresponding *fully-extended precedence graph* in figure 6. The following relations should be identified.

Relation 1 *The Intra-iteration Precedence Relation is the path without any delay operators on it in the recursive DFG. This relationship indicates the operation precedence order during one sampling period (iteration period).*

This kind of relation is denoted as $X_n \rightarrow Y_n$ or $X \rightarrow Y$, for simplicity, which means node X and node Y must be completed in the same iteration of the algorithm and node Y is fired only after node X has been completed. In the recursive data flow graph, the intra-iteration precedence relation can not be chained as a loop; otherwise, a dead look is caused. An example of a recursive data flow graph is shown in figure 1. Figure 2 shows the intra-iteration precedence relations of this recursive data flow graph.

Relation 2 *The Inter-iteration Precedence Relation is the path with the dummy delay operators. This relationship specified the recursion or feedbacks among different iterations.*

The term delay is used in the recursive sense of the signal processing, and corresponds to a sample offset between the input and the output. A unit delay element on the path from node X to node Y means that the nth sample consumed by node Y will be the $(n-1)$th sample produced by node X. Figure 5 shows the inter-iteration precedence relation of the retimed recursive data flow graph of figure 3. In figure 5, considering the precedence path $A \rightarrow d \rightarrow B$ of loop L_3, which is interpreted as $A_n \rightarrow B_{n+1}$. The 'd' node represents the delay. The node A_n is complete in the nth iteration and the node B_{n+1} is in the $(n+1)$th iteration.

Hence, in the recursive data flow graph to *fully-extended precedence graph* conversion, the delay nodes are used as dividers among successive iterations. The delay nodes will be cut and the delay parts will be interpreted as starting and terminating nodes of each iteration. Since the original recursive DFG must not contain any delay-free loop, the corresponding *fully-extended precedence graph* is always acyclic.

We have described the precedence constraints, the starting nodes and terminating nodes of the fully-extended precedence graph which corresponds to the elements in the original recursive data flow graph. Thus, the information of

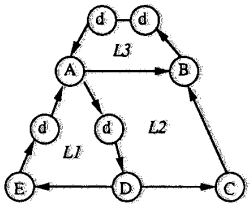

Fig. 1.A Recursive DFG with three loops. The node computation delay is 1 clock cycle.The cycled upper case letter in- dicates the node index, and the cycled 'd' indicates the delays (arc buffer).

L1 : EAD = 3/2 * (Iteration period bound)
L2 : ADCB = 4/3
L3 : AB = 2/2

Critical Loop : *L1* : D -> E
A

Noncritical Loop : *L2* : D -> C -> B (retiming)
A
L3 : A -> B

Fig.2 The iteration period bound 3/2 cycles, and the precedence relations of one iteration of the recursive data flow graph in figure 1.

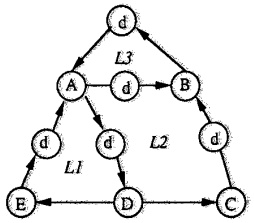

Fig.3 A retiming version of Fig.1.

Dn -> En An
Dn -> Cn Bn

Fig.4 The intra-iteration precedence realtion of recursive DFG of Fig.3.

An $\xrightarrow{d}$ D_{n+1}
An $\xrightarrow{d}$ B_{n+1}
Cn $\xrightarrow{d}$ B_{n+1}
Bn $\xrightarrow{d}$ A_{n+1}
En $\xrightarrow{d}$ B_{n+1}

Fig.5 The inter-iteration precedence realtion among the contiguous iterations of Fig.3.

the original recursive data flow graph can be totally preserved by the corresponding *fully-extended precedence graph*. Since the system bases on the RISC architecture and the high throughput requirement demands a rate-optimal design, the optimization method of retiming and unfolding can be applied on the original data flow graph in an efficient way to construct the fully-extended precedence graph.

3 Retiming and Unfolding

Considering the recursive data flow graph of figure 1 and the corresponding precedence graph in figure 2. It shows that the loop L_1 has the maximum iteration period 3/2 cycles which is greater than other loops. Other loops are con-

sidered to be the non-critical loops, since they have a smaller iteration period than that of L_1. But, in figure 2, the path $D \rightarrow C \rightarrow B$ of the non-critical loop L_2 is longer than the path $D \rightarrow E$ of the critical loop L_1. If we take the precedence graph of figure 2 to schedule, it is obviously that no possible static rate-optimal scheduling of $P_{min} = 3/2$ cycles exists because the static scheduling period is limited by the path $D \rightarrow C \rightarrow B$ as 3 cycles.

The retiming and unfolding process is applied here to make the critical loop of the recursive data flow graph equivalent to a critical path of the *fully-extended precedence graph*, and preserves the intra-iteration and inter-iteration concurrency. The process of retiming involves moving around the delays in the DFG such that the total number of delays in any loop remains unaltered, and the stead-state input-output behavior and the P_{min} of the system is preserved [5]. In the research [3], it has shown that only retiming of the critical loop does not guarantee a rate-optimal design. Thus, we have made a different approach and target our multiprocessor system on the RISC architecture. Every operation is complete within one clock cycle. The retiming process is applying on all loops. Since a non-critical loop always have a smaller iteration period and there are only discrete positions along the length of the paths, the delays can be evenly redistributed by retiming. In order to depict the object of the optimization retiming process in detail, a lemma is expressed as below:

However, we need a few simple definitions first. A path is a sequence of precedence related operations. All operations are assumed to be complete within a single instruction cycle. A segment of a path is a subpath from an output of a delay node to an input of a delay node.

Considering a loop L_l with n_l dummy delay operators, we then define a function T as below:

$$T(L_l, n_l) = T_l \quad , \; l \in loops$$

Where T_l is the sum of the computation time associated with all the operation nodes in loop L_l. The n_l represents the delay counts of the loop L_l, which also corresponds to the number of separated segments in a loop. Thus, a segment $S_l^i(0)$ of the loop L_l is defined by the function T as below:

$$T(L_l, 1) = S_l^i(0) \quad , \; l \in loops$$

Where the parameter '0' of the $S_l^i(0)$ represents no delay node in the segment, and i is the index of the segment.

Lemma 1 *Based on the RISC architecture, any non-critical loop can be retimed such that $T(L_c, n_c) \geq T(L_m, n_c)$, where L_c is the critical loop , n_c represents the number of delay nodes in the critical loop, and $L_m \in$ the set of all non-critical loops.*

Proof: Since the static schedule is implemented on a RISC machine, every operation node can be mapping on a certain hardware component which complete its operation within one clock cycle. Assume the segments $S_c^i(0)$ of the computation path of the critical loop is expressed as:

$$S_c^1(0),\ S_c^2(0), ...,\ S_c^{n_c}(0)$$

and, the segments $S_m^i(0)$ of the computation path of the non-critical loop is expressed as:

$$S_m^1(0),\ S_m^2(0), ...,\ S_m^{n_c}(0)$$

Since the computation path of the critical loop and non-critical loop both are discrete and the average segment length of the critical loop is the maximum of all loops, we can retime each segment $S(0)$ as follows:

$$S_c^1(0) \geq S_m^1(0),\ S_c^2(0) \geq S_m^2(0),\ .....,\ S_c^{n_c}(0) \geq S_m^{n_c}(0),$$

To sum up the two sides of the inequality, we can derived:

$$T(L_c, n_c) \geq T(L_m, n_c) \square$$

Comparing the recursive data flow graph of figure 1 and a retimed version shown in figure 3, the retiming process is locally applied on node B. The removal of one delay node form the only outgoing arc $B \rightarrow A$ and add one delay node to all incoming arcs. A more detail description of the segment length of the original data flow graph and that of the retimed version can be identified by comparing figure 2 and figure 4. Now, we consider how an admissible schedule is constructed for the recursive data flow graph.

Lemma 2 *A static schedule of all segments $S_l(0)$, $l \in$ loops, of the original DFG is an admissible schedule for the recursive data flow graph.*

Proof: The $S_l(0)$ begins from a starting node and ended on a terminating node, which does not alter any precedence constrains. Hence, an admissible static schedule of all segments $S_l(0)$ of the original data flow graph is an admissible schedule for the original recursive data flow graph. □

The static admissible schedule can be constructed by all $S_l(0)$ segments, we now consider the segments $S_l(n)$. This follows the Lemma 3.

Lemma 3 *The static schedule of all $S_l(n)$ segments is an admissible schedule of the original recursive data flow graph.*

Proof: Since the set of all $S_l(n)$s does not alter the intra-iteration and inter-iteration precedence constraints, and represents n iterations of the original data flow graph. Thus, the static schedule of all $S_l(n)$ segments is an admissible schedule of the original recursive data flow graph. □

Since a fully-extended precedence graph is desired and the optimization process of retiming and unfolding can be applied to the recursive data flow graph efficiently, the construction of a fully-extended precedence graph can be depicted as theorem 1.

Theorem 1 *The set of all $S_l(n_c)$ segments after appropriate retiming, such that $T(L_c, n_c) \geq T(L_m, n_c)$ is a fully-extended precedence graph.*

Proof: The set of all $S_l(n_c)$ segments which consists of the intra-iteration and inter-iteration precedence constraints. It also represents n_c iterations of the recursive data flow graph.

The computation time associated with all operation nodes of the critical loop is $S_c(n_c)$, which can be expressed as $T(L_c, n_c)$. Since $T(L_c, n_c) \geq T(L_m, n_c)$, for any non-critical loop m, the critical path of the precedence graph constructed by the set of all $S_l(n_c)$ segments is equivalent to the computation path of the critical loop. Thus, the set of all $S_l(n_c)$ segments after appropriate retiming, such that $T(L_c, n_c) \geq T(L_m, n_c)$ is a fully-extended precedence graph. □

4 Rate-Optimal Scheduling

A simple scheduling method called *contiguous scheduling* is used here, all operations are scheduled as soon as possible under no hardware limitation. The *fully-extended precedence graph* of figure 3 is shown in figure 6. Then, the rate-optimal scheduling of a fully-extended precedence graph is depicted in the following theorem:

Theorem 2 *A contiguous scheduling of the fully-extended precedence graph is a rate-optimal schedule of the original recursive data flow graph.*

Proof: Since the critical path of the fully-extended precedence graph is equivalent to the total computation path of the critical loop, the contiguous schedule of the fully-extended precedence graph is also a contiguous schedule of the critical loop without any intermediate idle cycles. Thus, the iteration period bound of the critical loop remains unchanged, and a rate-optimal schedule can be derived. □

Figure 7 shows the contiguous scheduling of the *fully-extended precedence graph* of figure 6. The square block represents the time instance of the processor. The sets of blocks with different block patterns represent different iterations of the recursive algorithm. The computation path of the critical loop is contiguous scheduled and becomes the critical path of the static schedule. The scheduling period is the same as the *iteration period bound* $= 3/2$ *cycles*. The schedule is globally static and rate-optimal. Since we take a fully-extended precedence graph with the problem instance of n_c iterations of the original recursive data flow graph, the result is running in a globally static and locally dynamic way, as in figure 7. Note that, our approach can achieve a rate-optimal solution but does not ensure whether a processor-optimal solution can be derived.

5 Conclusion

We have found a solution of the rate-optimal scheduling for the recursive signal processing algorithms. In the research of [3], unfolding the data flow graph with the factor of least common multiple of all loop delay counts also can produce a rate-optimal scheduling. The main disadvantage of [3] is that necessary memory and controller is proportional to the unfolding factor, and the unfolding factor is also great; e.g. $LCM(3,5,7) = 105$. By using the modern signal processing hardware components of RISC architecture, the recursive DFGs can be evenly retimed and unfold to an equivalent fully-extended precedence graph. The optimal unfolding factor is merely equal to the delay counts of the critical loop. And, a rate-optimal schedule can be generated from the fully-extended precedence graph by a simply contiguous scheduling.

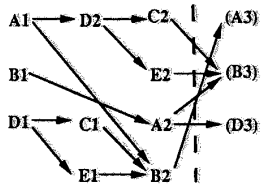

Fig.6 A fully-extended precedence graph (only need an unfolding factor = 2, not LCM(2,3) = 6).

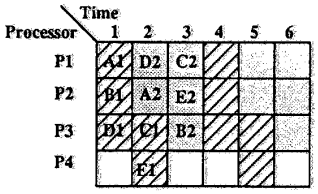

Fig.7 A rate-optimal scheduling where 1th iteration & 2nd iteration are locally dynamic scheduling and every two iterations are globally static scheduling on rate-optimal for iteration period bound 3/2 cycles.

References

[1] D. A. Schwartz and T. P. Barnwell III, "Cyclo-Static Multiprocessor Scheduling for the Optimal Realization of Shift-Invariant Flow Graphs," Proc. of ICASSP'85. pp. 1384-1387.

[2] S. H. Lee and T. P. Barnwell III ,"Optimal Multiprocessor Implementations From a Serial Algorithm Specification," Proc. of ICASSP'88. pp. 1694-1697.

[3] Keshab K. Parhi and David G. Messerschmitt, "Rate-Optimal Fully-Static Multiprocessor Scheduling of Data-Flow Signal Processing Programs," Proc. of ISCAS '89. pp. 1923-1928.

[4] Sun-Yuan Kung, "On Supercomputing with Systolic/Wavefront Array Processor", Proceedings of IEEE, Vol. 72, No. 7, July 1984.

[5] Leiserson C. E., Rose F. and Saxe J., "Optimizing Synchronous Circuitry by Retiming," Third Caltech Conference on VLSI, Pasadena, CA, March 1983, pp. 87-116.

[6] Liang-Gee Chen, Lih-Gwo Jeng, K. T. Chao, D. J. Lin and C. T. Chao, " CAD System for a Application-Specific DSP Processor Design," Proceedings of SASIMI '90, Japan. October 1990, pp 199-206.