한국산업정보학회논문지 제4권 제2호 (99.6)

Implementation of Digital Filters on Pipelined Processor with Multiple Accumulators and Internal Datapaths[†]

Chun-Pyo Hong*

요 약 본 논문은 순환이동불변 플로우 그래프로 표시된 디지털 필터를 여러 개의 누산기 및 내부 데이터패스를 가진 파이프라인 프로세서에 최적으로 구현할 수 있는 기법에 대하여 기술하였다. 이와 관련하여 본 논문에서는 상용의 DSP 프로세서를 이용하여 다중프로세서를 구성했을 때를 고려한 스케쥴링 기법을 개발하였으며, 연구 결과는 다음의 세 가지로 요약할 수 있다. 첫째, 상용 DSP 프로세서의 구조와 유사한 n 개의 누산기와 3 개의 내부 데이터패스를 가지는 파이프라인 프로세서의 모델을 제시하였다. 둘째, 주어진 구조를 가지는 시스템에 순환이동불변 플로우 그래프로 표시된 디지털 필터를 구현하고자 할 때 얻을 수 있는 최소 반복 주기 및 간단한 스케쥴링 모델을 구했으며, 제약조건을 부여한 깊이 탐색기법에 바탕을 둔 최적의 스케쥴링 기법을 개발하였다. 마지막으로 본 연구에서 개발된 스케쥴러를 이용하여 잘 알려진 디지털 필터에 대하여 성능 시험을 한 결과 대부분의 경우 이론적으로 얻을 수 있는 최소의 반복 주기를 만족시켜 주는 스케쥴링 결과를 얻을 수 있음을 확인하였다.

Abstract This paper presents a set of techniques to automatically find rate optimal or near rate optimal implementations of shift-invariant flow graphs on pipelined processor, in which pipeline processor has multiple accumulators and internal datapaths. In such case, the problem to be addressed is the scheduling of multiple instruction streams which control all of the pipeline stages. The goal of an automatic scheduler in this context is to rearrange the order of instructions such that they are executed with minimum iteration period between successive iteration of defining flow graphs. The scheduling algorithm described in this paper also focuses on the problem of removing the hazards due to inter-instruction dependencies.

1. Introduction

In many Digital Signal Processing (DSP) applications, such as real time processing of wide-band or multidimensional signals, processors with high speed computing capability are required. Recent improvements in VLSI technology have now lead to single chip programmable Digital Signal Processors of unprecedented speed and computing capability, and most of those processors are deeply pipelined[2][3][4].

In some cases, the use of pipelining can provide the required computing power, since the inherent nature of DSP to repeat the same computation to each frame of the input stream. However, if there is recursion in the problems, pipelining can degrade the performance as it may alter the computation intended[1]. This paper presents a set of techniques to automatically find an implementations of digital filters on pipelined processor, in which digital filters are represented by recursive Shift-Invariant Flow Graphs (SIFG)[6] and pipelined processor has multiple accumulators and internal datapaths.

For a linear pipelined processor, author has previously described and demonstrated a compiler[5]. In

* School of Computer and Information Engineering, Taegu University
† The author wish to acknowledge the financial support of Korea Research Foundation made in the program year of 1997

previous research, we assumed that the pipelined processor consist of two units: Instruction Unit (I-unit) and Execution Unit (E-unit). For DSP applications, different forms of pipelining can be structured depending on the applications. In most real systems, the structure of the E-unit is system dependent and usage dependent. This paper proposes some different forms of the E-unit for DSP applications, and investigates the effects of different forms of pipelining. The E-unit proposed in this research has multiple accumulators and internal datapaths.

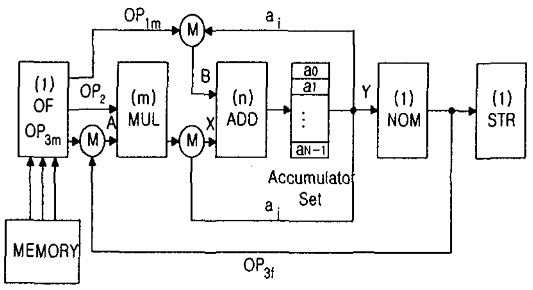
This paper also describes an instruction scheduling algorithm for the implementation of recursive SIFG on the new pipelined E-unit. Since the operand fetch from accumulator or normalizer decreases the latency between operations, the internal datapaths can decrease the achievable iteration period bound. As a result, the chance of implementation with shorter iteration period increases. However, from the view point of complexity, the internal datapaths and the multiple accumulators *make the instruction scheduling problem more complex.* The scheduler described in this paper also focuses on the problem of removing the hazards due to inter-instruction dependencies.

2. Pipelined Processor with Multiple Accumulators and Internal Datapaths

The architecture of basic functional unit is determined by the nature of the algorithms that characterize the task to be performed. By considering the characteristics of SIFGs, three kinds of operations, i.e., multiplication, addition, and multiplication/addition should be performed in an efficient way. <Figure 1> illustrates the structure of a pipelined E-unit which is optimized for these three arithmetic operations.

As described in <Figure 1>, the E-unit is consists of five modules, i.e., Operand Fetch (OF), Multiplier (MUL), Adder (ADD), Normalizer (NOM), and Store Result (STR). In this unit, it is assumed that the MUL module consists of m stages, the ADD module consists of n stages, and the remaining three modules consist of a single stage apiece. Although each module clearly can either more or less be partitioned, the partitioning must be considered together with the instruction scheduling.

<Figure 1> Structure of (m+n+3) stages Pipelined E-unit



$A=OP_{3m}$ or OP_{3f}
$B=OP_{1m}$ or a_i
$X=(OP_2 \times OP_{3m})$ or
$(OP_2 \times OP_{3f})$ or
a_j

$Y=(OP_2 \times OP_{3m}) + OP_{1m}$ or
$(OP_2 \times OP_{3m}) + a_i$ or
$(OP_2 \times OP_{3f}) + OP_{1m}$ or
$(OP_2 \times OP_{3f}) + a_i$ or
$a_j + a_i$ or
$a_j + OP_{1m}$

In the E-unit, regardless of the specified operations, three operands are fed from memory, accumulator, or NOM. Then, the multiply/add operation is performed on the operands in the MUL and the ADD modules, and the result is latched in the accumulator. In the NOM module, the accumulator value is rounded to provide the correct data size to match with the size of data memory. After normalization, the result of the specified operation is stored into memory or accumulator.

The E-unit has two inherent characteristics. First, since the basic function of E-unit is the multiply/add operation, there are some redundant steps for the other operations. The ADD module is a redundant step for the multiply operation, and the MUL module is a redundant step for the add operation. However, there is no loss of efficiency as long as the throughput of the ADD and the MUL is equals to the throughput of control logic and memory.

Second, three different arithmetic operations are performed in the same way using the same resources. This is achieved by simply providing a special constant (one or zero) for a specific operand. Since these operands are simple constants, in some case, these constants can be stored in a different memory unit. In addition, in the typical DSP applications such as FFT or digital filters, one of the two operands to the multiplication is a constant (or coefficients). Thus this constant value also can be stored in a different memory unit.

The internal datapath from accumulator to ADD provides a function for a fast multiply and accumulate operation. With the addition of this data path, even if the ADD is not modified, the ADD can function as two different modes. If the operand B for the ADD comes from the memory OP1m, then the ADD function as a normal add mode. In contrast, if the operand B for the ADD comes from the accumulator ai, then the ADD function as an accumulate mode.

The internal data path from NOM to MUL is a kind of short-circuiting. Instead of storing the normalized data into memory, a copy of data to be stored is directly fed to an operand for the MUL (OP3f). This data path can save the time for store result and operand fetch. As an example, AT&T's DSP32C[4] which uses deep pipelining employs this type of internal datapath.

In addition to the internal data path, the E-unit has multiple accumulators. By employing multiple accumulators, it provides more flexibility in choosing operands and storing results. With this pipelined E-unit, if several multiply/accumulate instructions are to be executed one after the other, then the instructions are pipelined such that one instruction completes in every pipeline clock cycle. Sometimes, both operands for the ADD can come from different accumulators. In such a case, the output of the ADD becomes ai + aj. In addition, in the E-unit of <Figure 1>, to guarantee conflict free accumulator allocation, the minimum (m+n+2) accumulators are required.

3. Instruction Scheduling on the Pipelined E-unit

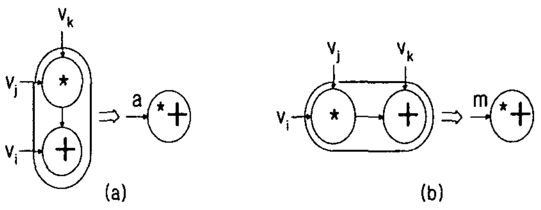
In the E-unit of <Figure 1>, although the additional control circuitry are required, it can achieve a faster solution. Even if the shorter computational latency between operations does not always guarantee the shorter iteration period, it is clear that the shorter computational latency give a greater probability of achieving a shorter iteration period.

3.1 Atomic Operations Set

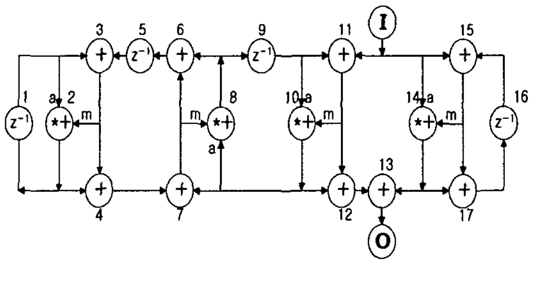
In the E-unit of <Figure 1>, the multiply/add operation can be executed in a single instruction. Since the atomic operation is defined as an operation that can be scheduled independently in the specified target architecture, the multiply/add operation is considered as an atomic operation. Thus, with the E-unit of <Figure 1>, the new atomic operations set is defined as: 1) multiply; 2) add; and 3) multiply/add.

Based on the new atomic operations set, the defining DSP algorithms can be specified as a SIFG which contains three types of atomic operation. If the defining algorithm is originally defined as a SIFG which does not include the multiply/add operation as an atomic operation, then a new SIFG can be constructed by combining the multiply and the add into the multiply/add. In such a case, the multiply and the add can be combined in two different ways. In addition, depending on the original structure,

<Figure 2> Two multiply/add nodes which have different implicit connection. (a) case 1: predecessor is connected to add node. (b) case 2: predecessor is connected to multiply node.



<Figure 3> Wave filter which has multiply/add node as an atomic operation.



the computational latency between two nodes which include multiply/add node is different. <Figure 2>

represents two different types of multiply/add node which is connected in different way implicitly. In <Figure 2>, depending on the connection with the predecessor vi, the computational latency between the node vi and the multiply/add node will be different. By applying this rule, the wave filter referenced in previous research[4] can be transformed to an equivalet SIFG described in Figure 3

3.2 Pipeline Iteration Period Bound

The iteration period bound is a function of the computational time delay of node in defining SIFG. With the piplined E-unit of <Figure 1>, the computational latency between two operations is different depending on the operation types. This is because the operands for the MUL can be fetched from the NOM or Memory, and because the operands for the ADD can be fetched from Accumulator (s) or Memory.<Figure 4>

<Figure 4> Minimum achievable computational latencies between two operations with pipelined E-unit in <Figure 1>

Data Path	Latency
$(+) \rightarrow (+)$	n
$(*) \rightarrow (*)$	m+n+1
$(+) \rightarrow (*)$	m+n+1
$(+) \xrightarrow{a} (*+)$	n
$(+) \xrightarrow{m} (*+)$	m+n+1
$(*) \xrightarrow{a} (*+)$	n
$(*) \xrightarrow{m} (*+)$	m+n+1
$(*+) \rightarrow (+)$	n
$(*+) \rightarrow (*)$	m+n+1
$(*+) \xrightarrow{a} (*+)$	n
$(*+) \xrightarrow{m} (*+)$	m+n+1

summarizes the minimum achievable computational latency between possible combinations of two operations.

As described in <Figure 4>, since the computational latency between two operations is different depending on the operation types, the pipeline iteration period bound is computed by equation (1).

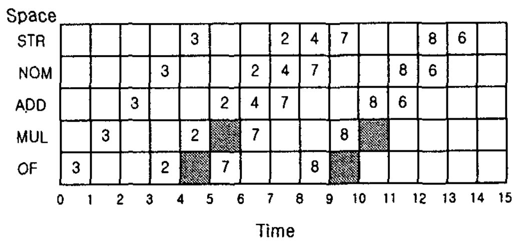
$$T_{0p} = \max_{p \in loops} \lceil \frac{L_p}{n_p} \rceil \qquad (1)$$

where $L_p = \sum_{i,j \in p} l_{ij}$ is the sum of computational latency between two arithmetic operation nodes vi and vj around the loop p, and np is the number of ideal delays in loop p. In equation (1), lij is obtained directly from <Figure 4>. If there is one or more ideal delays between vi and vj, then compute lij by removing ideal delay.

3.3 Static Scheduling Model

With the pipelined E-unit of <Figure 1>, the operands can be fetched from three different locations, i.e., accumulator, normalizer, and memory. If both operands are fetched from accumulators, then OF and MUL modules are not used. In such a case, the usage of each pipeline stage is non-uniform. Consider the implementation of the critical loop of wave filter in <Figure 3> on pipelined E-unit of <Figure 1>. <Figure 5> illustrates non-uniform usage of each pipeline stage. In this example, we assumed that m = 1 and n = 1.

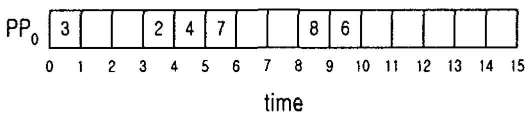
<Figure 5> Non-uniform usage of each pipeline stage



As illustrated in <Figure 5>, the usage of each pipeline stage is non-uniform. Thus each pipeline stages must be scheduled independently depending on the available operands. However independent scheduling of each pipeline stage requires enormous computation costs. In this research, instead of scheduling each pipeline stage independently, a simple scheduling model is applied.

In <Figure 5>, although some pipeline stages are not used at certain times, no other operations require the usage of the same pipeline stages at the same time. In other words, no other operations require the usage of the OF at time 4 and 9. Similarly no other operations require the usage of the MUL at time 5 and 10. In such a case, arbitrary modifications can be made for shaded areas in <Figure 5>. In such a case, if the first stage of E-unit is scheduled without conflicts, then remaining four stages can be automatically scheduled without any conflicts. More generally, for (m+n+3) stages of E-unit, instead of scheduling each pipeline stage independently, the problem is simplified to schedule only the first stage explicitly. <Figure 6> represents an equivalent representation of the schedule of <Figure 5> in the static scheduling model

<Figure 6> Equivalent representation of the schedule of Figure 5 in static scheduling model.



If the number of pipeline stages of the MUL and the ADD is fixed, then the pipeline iteration period is bounded in two different way. If T0p is greater than or equal to the total number of arithmetic operation nodes in defining SIFG, then this system is flow graph dependent. In contrast, if T0p is less than the total number of arithmetic operation nodes in defining SIFG, then this system is resource dependent. Consequently, if the number of pipeline stages of the MUL and the ADD is fixed, then the pipeline iteration period bound is determined by equation (2).

$$T_{0p} = \left\{ \begin{array}{ll} T_{0p} & \text{if } T_{0p} \geq |V| \\ |V| & otherwise \end{array} \right\} \qquad (2)$$

where V is the cardinality of arithmetic operation nodes in defining SIFG.

3.4 Determination of Static Schedule

The objective of instruction scheduling is to find an implementation which has minimum iteration period. The scheduling algorithm is an iterative procedure and is divided into two steps.

▶ Step 1: SIFG Analysis

As the first step, the scheduler does an initial analysis of the SIFG to find the bounds which the scheduler will attempt to achieve. Then the next step is to decide the loop scheduling order, that is the order of each loop to be integrated into the final operation schedule. The loop scheduling order is determined depending on the slack time of each loop that is specified in equation (3).

$$t_s(p) = n_p T_{0p} - L_p \qquad (3)$$

where ts(p) is the slack time of loop p, np is the number of ideal delays around loop p, T0p is the pipeline iteration period bound, and Lp is the sum of computational latency between two arithmetic operation nodes vi and vj around the loop p. Although Lp may be different depending on the implementation, the slack time is computed based on minimum achievable latency between operations.

▶ Step 2: Constrained Depth-First Search

As the second step, the scheduler finds an operation schedule for an implementation of the given SIFG onto the static scheduling model. The basic scheduling methodology is to construct a deterministic schedule by a constrained enumeration of all possible schedules that meet the precedence relation of the given SIFG. The constrained enumeration of possible schedules is achieved through the depth-first search operation.

During the depth-first search, the scheduler enumerates all of the possible schedules that meet the precedence constraints of the given SIFG and have iteration periods consistent with the pipeline iteration period bound of the given SIFG. However the search space will be extremely large if all the paths on the depth-first search tree are to be considered for every incremental depth. To reduce the search space, a set of constraints such as pipeline iteration period bound, data precedence constraint, pipeline latency constraint described in <Figure 4>, and equivalence class constraint[5] is applied for the depth-first search. <Figure 7> is an example of implementation of the wave filter.

4. Discussion

In this research, a new pipelined E-unit was proposed, and the effects of new pipelined E-unit to instruction scheduling were investigated. It was shown that the new pipelined E-unit is suitable for sum-of-products, which is one of key operation for DSP algorithms.

The instruction scheduling methodology for the new pipelined E-unit has the same features as the scheduling algorithm described in previous research[5]. One major difference is that some additional constraints are applied during the depth-first search. It was also shown that, due to the feedback paths and multiple accumulators, the new pipelined E-unit can achieve a shorter iteration period. However, it requires a higher computational cost since the depth-first search requires a larger search space.

<Figure 7> Implementation of wave filter in <Figure 3> on pipelined E-unit in <Figure 1>. (a) Loop scheduling order. (b) Four step operations for constructing the final schedule.

Loop	L_p	Slack Time	Scheduling Order
3-2-4-7-8-6-5	10	3	1
11-10-7-8-9	8	5	2
3-2-4-7-6-5	7	6	3
3-4-7-8-6-5	7	6	4
11-10-8-9	5	8	5
10-7-8-9	5	8	6
15-14-17-16	5	8	7
3-4-7-6-5	4	9	8
3-2-1	4	9	9
10-8-9	2	11	10
15-17-16	2	11	11
2-1	1	12	12

(a)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
step1	3			2	4	7			8	6					
step2	3	11		2	10	4	7			8	6				
step3	3	11	15	2	10	4	7	14	17	8	6				
step4	3	11	15	2	10	4	7	14	17	8	6	12	13		

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

(b)

References

[1] P. Dewilde, E. Deprettere, and R. Nouta, "Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms," VLSI and Modern Signal Processing, S.Y. Kung, H.J. Whitehouse, and T. Kailath, editors, Prentice-Hall Press, pp. 257-276, 1985.

[2] R. Ernst, "Long Pipelines in Single-Chip Digital Signal Processors - Concepts and Case Study," IEEE Trans. on Circuits and Systems, Vol. 38, No.1,

pp. 100-108, 1991.

[3] E.A. Lee, "Programmable DSP Architectures: Part I," IEEE ASSP Magazine, Oct. 1988.

[4] E.A. Lee, "Programmable DSP Architectures: Part II," IEEE ASSP Magazine, Jan. 1989.

[5] C.P. Hong, "An Optimal Implementation of Digital Filters on Multiple Pipelined Processors," Proc. on International Workshop on Intelligent Signal Processing and Communication, pp. 345-350, Seoul, Korea, Oct. 1994.

[6] D.A. Schwartz and T. P. Barnwell III, "Cyclo-Static Multiprocessor Scheduling for the Optimal Realization of Shift-Invariant Flow Graphs," Proc. of International Conference on Acoustics, Speech, and Signal Processing, pp. 1384-1387, 1985.