한국산업정보학회논문지 제6권 제2호 (2001.6)

A Multiprocessor Scheduling Methodology for DSP Applications†

Chun Pyo Hong* Jin Mo Yang*

요 약 본 논문은 디지털 필터와 같이 연산 단위가 적은 디지털신호처리 알고리즘을 효과적으로 구현할 수 있는 새로운 형태의 다중 프로세서 시스템 및 이를 위한 스케쥴링 알고리즘을 제안한다. 본 논문에서 제안한 다중 프로세서 시스템에서는 한 개 또는 그 이상의 공유 버스를 이용하여 프로세서 사이를 연결하였으며, 각 프로세서에서 명령어가 실행될 때 일정 크기의 시간 차이가 존재한다. 이 시스템은 프로세서 사이의 통신 문제를 효과적으로 해결할 수 있을 뿐만 아니라, 다중프로세서 시스템의 스케쥴링 시 프로세서간의 통신 시간을 반영할 수 있다는 장점이 있다. 또한 본 논문에서는 플로우 그래프로 표시된 디지털 필터를 새로운 형태의 다중프로세서 시스템에 최적으로 구현할 수 있는 스케쥴링 알고리즘을 개발하였다. 마지막으로 본 연구에서 개발된 스케쥴러를 이용하여 잘 알려진 디지털 필터에 대하여 시뮬레이션을 한 결과 대부분의 경우 이론적으로 얻을 수 있는 최소의 반복 주기를 만족시켜주는 스케쥴링 결과를 얻을 수 있음을 확인하였다.

Abstract This paper presents a new multiprocessor system and corresponding scheduling algorithm that can be applied for implementation of fine grain DSP algorithms such as digital filters. The newly proposed system uses one or more shared buses as the basic interconnection network between processors, and fixed amount of clock-skew is maintained between instruction execution of processors. This system not only can handle the interprocessor communications very efficiently but also can explicitly incorporate the interprocessor communication delay time into the multiprocessor scheduling model. This paper also presents a new scheduling strategy for implementing digital filters expressed in fully-specified flow graphs on the proposed system. The simulation result shows that well-known digital filters can be implemented on proposed multiprocessor in which the implementation satisfies the iteration period bound.

1. Introduction

For many Digital Signal Processing (DSP) applications, the need for real-time processing is required. Algorithms that integrate speech coding and decoding typically require processing rates of 1-30 million instructions per second (MIPS), and video algorithms may require rates of 0.1-10 billion instructions per second (GIPS) [3].

For many DSP applications, DSP processors (DSPs) have been widely used. However, while there continues to be improvements on the computing capability for DSPs, most DSPs can execute very few instructions during each sample period for video algorithms. One alternative for real time implementation of DSP systems is to use multiple processing elements. In this approach, as the number of processing elements increases, so does the difficulty of orchestrating their cooperative efforts increase.

A fundamental problem with all of the previous multiprocessor processing models [1][2][3][4][7][9] is that they did not adequately incorporate the interprocessor communication cost in the scheduling process. In particular, researchers usually assumed that each processor could exchange results with other

* School of Computer and Communication Engineering, Taegu University
† This research was supported by the Taegu University research grant 1996

processors whenever the data precedence constraints were satisfied. It is indeed possible to build parallel processing systems in which this is true. However, since such systems require that the interprocessor communications delay be included as part of the arithmetic operational delay, they usually must be implemented with a lower clock rate than for systems in which the I/O transfers are pipelined. In addition, such systems also require massive, on-demand communication which often makes them prohibitively expensive.

This paper proposes a new class multiprocessor system, called Clock-Skewed Parallel Processing (CSPP) system, and examines the problem of how to schedule such processors statically. Also this paper presents a new interprocessor communication scheduling strategy and an associated multiprocessor scheduler for the automatic generation of fine-grain DSP algorithms for Clock-Skewed Parallel Processing (CSPP) system. In such a CSPP system, the clock-skewing between instruction executions guarantees the conflicts free data transfer through the shared bus system, and no bus contention control mechanism are required. The hardware necessary to realize the required data bus network is therefore quite modest. In addition to the data busses themselves, each Processing Element (PE) must have a synchronous, programmable I/O control unit which controls the interaction with the bus network.

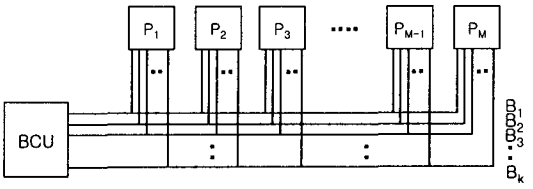
In the CSPP system, all the algorithm operations and all the data transfers are determined at compile time. Thus, at run time, global optimality is achieved using completely local control. All the data transfers occur synchronously, and are part of the predetermined static schedule. The schedule includes the source and destination PEs at each interprocessor communication instance, as well as explicit timing information such as when the source PE must put data into the shared bus systems and when the destination PE can take the data from shared bus system..

2. Algorithms and Architectures

The CSPP scheduler described here transforms an algorithm represented as a cyclic flow graph into a complete schedule for a synchronous CSPP system. The two primary inputs to the compiler are the computational algorithm and the characteristics of the target architecture.

In this research, all the computational algorithms are described as Fully Specified Flow Graphs (FSFGs) [5]. The FSFGs are class of shift-invariant flow graphs which can represent time-invariant linear filter structures, adaptive filter structures, and many other important DSP algorithms [8]. In addition, two bounds for optimal implementation has been defined in previous research [4][5]. An additional constraint in this research is that arithmetic operations are assumed to have the same durations, i.e., Ta(i) = Ta for all i, where Ta(i) is the arithmetic operation time delay of the ith node of a defining FSFG.

All the target architectures of interest belong to the class of synchronous MIMD machines. In addition, this research directly addresses multibus systems such as the one illustrated in Fig. 1 and systems for which a fixed time skew is maintained between PEs. In the target architecture of Fig. 1, each PE is considered to have local registers and local memory, and each is capable of performing all the arithmetic operations required for the algorithm as well as reading and writing to the local memory.



<Figure 1> Target architecture

3. Clock-Skewed Parallel Processing System

The Clock-Skewed Parallel Processing (CSPP) system is defined as a synchronous system in which each PE is connected to one or more shared busses. The primary difference that characterizes CSPP systems is the time skew between processors. This leads to the concept of clock class, which is a set of all PEs that share the same clock-skew.

The major traditional problem with shared bus systems is the necessity for a complex bus contention

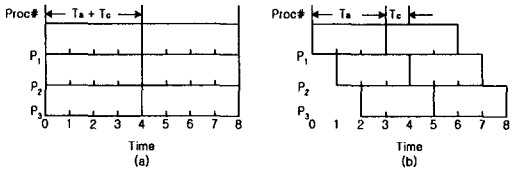
control mechanism. As the number of PEs connected to shared bus increases, the bus contention control mechanism generally becomes more complex. For systems in which the arithmetic operational delays are large compared to the bus transfer times, it is possible to achieve a simple and elegant solution to this problem. In system such as we propose, the busses are automatically time division multiplexed between PEs in different clock classes. In such a system, the Bus Control Unit (BCU) is very simple, and is not involved in controling the contention of shared bus. Instead the BCU is only responsible for writing results into the local memory of the correct PE(s) at right time.

In the target architecture of Fig. 1, if an operation scheduled on Pj requires an operation result of a node scheduled on Pi, the result has to be in the local memory of Pj before that instruction is executed. If Pi finishes an arithmetic operation, the result will be stored into the local memory of Pi itself or will be passed to Pj through the shared bus. The interprocessor communication time delay, Tc(ij), represents the minimum time delay between two processors Pi(or Pj) and Pj(or Pi) such that the operation result scheduled on the processor Pi(or Pj) is available to the processor Pj(or Pi). Thus, there are two types of communications :

i) communication is occurred between Pi and Pi, and it is assumed that Tc(ii) = 0; and

ii) communication is occurred between Pi and Pj, and it is assumed that Tc(ij) = Tc when $i \neq j$.

The simple example of Fig. 2 illustrates the basic concepts of a CSPP system. In this example, it is assumed that the CSPP system has a single shared bus, where Ta = 3, and Tc = 1. In the conventional synchronous parallel processing model of Fig. 2.(a), the clock-skew is zero and all the PEs are belong to the same clock class. In contrast, in the CSPP system of Fig. 2.(b), the operations at each PE are shifted. Thus, the three PEs belong to three different clock classes. In general, for a CSPP with N PEs, a clock-skew of Tc is maintained between instruction execution on Pi and P(i+1) mod N.



<Figure 2> Comparison of two parallel processing model. (a) Conventional synchronous parallel processing model. (b) Clock-skewed parallel processing model

One of two important characteristics of the CSPP system is that it vastly reduces the bus contention problem. Consider a single bus CSPP system in which each PE belongs to one of N different clock class. In such a system, each processor must communicate with either itself (same clock class) and/or with one or more other processors (other clock classes). If it communicates with itself, there is no problem since it can write into its own local memory. More importantly, so long as the bus transfer time is less than Tc, then it can also communicate with any other (set of) processor(s). Thus, all of the processors are functionally fully connected without contention.

The other important characteristic of the CSPP system is that it can explicitly incorporate the interprocessor communication time delay into the scheduling model. Consider again a single bus CSPP system above. For the conventional model, if a node operation is assigned to Pi at time ti, then the result will be available to Pi after the time ti+Ta, since Tc(ij) = 0 for i = j. But at that time, this result will not be available to Pj after the time ti+Ta+Tc, since Tc(ij) = Tc for $i \neq j$. If Pj requires the from Pi, there has to be at least the time space of Tc to explicitly incorporate the interprocessor communication delay into the scheduling model. The clock-skew of Tc (at least) is always maintained between Pi and Pj as long as $j \geq (i+j) \bmod N$, and this implicitly gives the time delay for interprocessor communications. Thus, the operation durations used by the compiler for the conventional system must be longer by Tc then those used by the CSPP compiler.

One limitation of the CSPP system is that the number of different clock class (N) which can be handled by each shared bus is limited by the ratio of Ta and Tc. If an arithmetic operation is executed on

Pi at time ti, Pi can begin another arithmetic operation at the time ti+Ta. Since the fixed amount of clock skew must be kept in the CSPP system, P(i+j) mod N can begin an arithmetic node operation at time ti + Tc, P(i+2) mod N can begin an arithmetic operation at time ti+2Tc, and so on. In this way, during the time interval between ti and ti + Ta, exactly N PEs can begin arithmetic operation with clock skewing of Tc. So N is bounded by Ta/Tc.

In this processing environments, no matter how many PEs are physically connected to the shared bus, the number of PEs which can be involved in parallel processing without bus contention is limited by N. One way to increase the number is to increase the number of shared busses. As the number of shared busses increases, the number of different clock class is still the same (N). But the number of PEs which can be involved in parallel processing increases. For example, in Fig. 1, M PEs are connected to each shared bus (where $M \geq N$). Since each shared bus can handle N different clock class, and the number of PEs belong to each clock class are K, maximum number of PEs that can be involved in parallel processing is equal to $M = N \times K$.

4. Interprocessor Communication in the CSPP System

All processors int a CSPP systam which share the same clock are said to be in the same clock class [3]. In a single bus CSPP system, there is no bus contention among PES because each PE belongs to a different clock class. In a multi-bus CSPP system, all processors which share a particular bus belong to different clock classes. Thus, all that is required is an appropriate fully synchronous system for implementing the bus communications required by the multiprocessor schedules generated by the compiler.

4.1 Interprocessor Communication Primitives

Let Ps denote the source processor which produces the operation result and let Pd denote the destination processor which receives the operation result from the source processor. The interprocessor communication is always achieved through the channel from Ps to Pd. Every time a PE finishes an instruction execution, it becomes a Ps. In the single bus CSPP system, since each PE belongs to a different clock class, there is only one Ps in every interprocessor communication cycle.

If Ps and Pd are identical, no communication is required using the bus system, and the APU must realize a data feedback loop. If Ps and Pd are different, then the data must be transferred through the shared bus. For the interprocessor communication through the shared bus, two primitive operations, data send and data receive, are defined (Fig. 3(a)).

i) Data Send Operation : The data send operation is defined as the transmission of a data element from the APU of Ps to the shared bus. This operation is executed by the I/O CU of Ps. In a data send operation, the I/O CU of Ps receives an operation result from the APU of Ps, and put it on the shared bus.

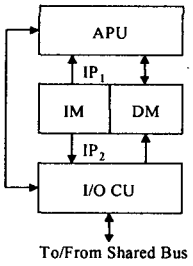
ii) Data Receive Operation : The data receive operation is defined as the reception of data element from the shared bus, and the storing of that data element in the DM of Pd. This operation is executed by the I/O CU of Pd. In a data receive operation, the I/O CU of Pd receives data from the shared bus and stores it into the DM of Pd.

All the interprocessor communications through the shared bus use these two primitive operations. Since the two primitive operations are executed by two different I/O CUS, two I/O CUS are always involved during the interprocessor communication. The schedule for these two primitive operations includes the information about the processor ID of Ps and Pd for each interprocessor communication cycle. In addition, the schedule also includes explicit timing information such as when the Ps must execute the data send operation, and when the Pd must execute the data receive operation.

4.2 Parallel Instruction of APU and I/O CU

In the CSPP system, each PE has four separate components. These are 1) Arithmetic Processing Unit

(APU), 2) Instruction Memory, 3) Data Memory (DM), and I/O Control Unit (I/O CU), and described in Fig. 3(a). Although the APU and I/O CU are part of the PE, they realize different aspects of the multiprocessor schedule. The APU realizes the arithmetic operations while the I/O CU realizes the communications operations. Since the entire system is synchronous and deterministic, they can share the same controller, in which they occupy different locations in a horizontal microcode word.



(a)

(b)

(c)

<Figure 3> Parallel instruction for APU and I/O CU. (a) Two instruction paths for a single instruction. (b) Instruction fields in a microcode control word. (c) Multiple elements for a complete schedule.

As described in Fig. 3, there are two instruction paths and N+4 instruction fields in the microcode control word. As described in Fig. 3(b), these include the operation code for the instruction being executed by the APU (OC), two APU operand addresses in the DM (OA1, OA2), one bus source DM address (SA), N-1 bus destination DM addresses (DA1 to DAN-1), and interprocessor communication instruction (I/O). For the system to operate at full speed, the DM must be a dual port memory which operates at the bus cycle time. The system would be capable of receiving N-1 data elements on a single cycle. Only occasionally would more than one datum be received in a single APU cycle.

In order to create all of the fields in the control word, the compiler must generate a complete schedule, including the processor assignment. Bacause each PE is assigned to a particular bus cycle, the processor communications reduces to a decision to put data on the shared bus, and the time slots in which to receive any required data. Thus, as shown in Fig. 3.(c), for each execution cycle for each PE, the compiler must generate the node number (operation type), a send flag, and two receive times. Note that, since each node of FSFG has at most two predecessor nodes [5], two receive operations are enough for this application.

For the interprocessor communication in the multi-bus CSPP system, all the PES are divided into Sub-Processing Groups(SGi). The SGi is defined as a set of processors in which each PE belongs to different clock class. If there are K sub-processing groups, there must be K shared busses (B0, B1, ..., Bk-1). In this case, the interprocessor communication between a PE which belongs to SGi and a PE which belongs to SGj is achieved through the shared bus Bi. Since each PE belongs to a different clock class in each sub-processing group, conflict free communication is always possible in the multi-bus CSPP system.

5. The CSPP Scheduler

Gives a FSFG, the scheduler problem is to assign each node operation of the defining FSFG to a PE such that the iteration period is minimized. The limitations on the iteration period are the data dependencies of the defining FSFG and the interprocessor communication time delay between PEs of the CSPP system. The scheduling is accomplished in three steps: 1) preprocessing; 2) depth-first search;

OC	OA_1	OA_2	SA	DA_1	···	DA_{N-1}	I/O

and 3) interprocessor communication scheduling.

Execution Cycle	Node Number	Send Flag	Receive Time 1	Receive Time 2

5.1 Preprocessing

In the first step, the scheduler abstractly transforms the CSPP system to the conventional parallel scheduling model with the inherent communication constraints just as in [5]. The clock-skew in the CSPP

system matches exactly with time shift between pseudo-processors in pseudo-processor representation of the pipelined processor. Hence the clock-skew can be modeled as a constraint in the scheduling using the approach described by the author in [5].

In the second step, the scheduler does an analysis of the flow graph to find performance bounds which the scheduler will attempt to achieve. For a given FSFG, two performance bounds are defined [5][9]: 1) Iteration period Bound (IPB); and 2) Processor Bound (PB).

5.2 Depth-First Search

The scheduler begins by trying to achieve the IPB by pruning away all partial schedules with can be proved to have a longer iteration period. If a schedule is found which achieves the IPB, then there is guaranteed to be no other schedule which is faster. If no such solution is found, then the search procedure constitutes a proof that no solution exists, and the process is repeated for the next largest iteration period. The scheduler enumerates all of the possible schedules that meet the precedence constraints of the defining FSFG and have iteration periods close to the IPB of the defining FSFG. However the search space will be extremely large if all the paths on the search tree are to be considered for every incremental depth. To reduce the search, three specific types of constraints are applied. The first constraint is simply the data precedence relationships from the FSFG. The second constraint is the communications constraint imposed as valid data links in time and space between PEs as described in abstract transformation. This constraint is a direct function of the CSPP system architecture. As the final constraint, the processor modulo constraint limits the maximum number of parallel operations at a given tim to the number of PEs [9]. If the solution has an iteration period of T, then there are T equivalence classes into which the operations of one iteration can fall.

5.3 Interprocessor Communication Scheduling

In the preprocessing and depth-first search, the primary problem is finding a schedule that satisfies the IPB of the defining flow graph. The schedule not only specifies which operations are performed at each time instance but also specifies which processor will perform the operation. As the final step of the CSPP compiler, the scheduler assigns the source processor and the destination processor(s). Basically, the algorithm consists of two parts: 1) Ps and Pd scheduling; 2) send and receive time scheduling. Fig. 4 describes the interprocessor communication scheduling algorithm.

```
Ps and Pd SCHEDULING
for all p ∈ P do
 begin
   for every instruction cycle ∈ v is assigned do
      begin
         Ps ← p;
         Pd ← all p ∈ successor(v) is assigned;
      end
 end
SEND FLAG and RECEIVE TIME SCHEDULING
for all Ps and Pd do
 begin
   if Ps at least one Pd (≠Ps) then
      SEND_FLAG ← TRUE;
   else SEND_FLAG ← FALSE;
 end

for all Pd that has at least one Ps(≠Pd) do
 begin
   if Pd has one Ps then RECEIVE_TIME_1 ← DecideRtime(Ps, Pd);
   else if Pd has two Ps (Ps1 and Ps2) then
     begin
       if Ps1 and Ps2 are executed at the same instruction cycle then
         begin
           RECEIVE_TIME_1 ← DecideRtime(Ps1, Pd);
           RECEIVE_TIME_2 ← DecideRtime(Ps2, Pd);
         end
       else RECEIVE_TIME_1 ← DecideRtime(Ps, Pd);
     end
 end
```

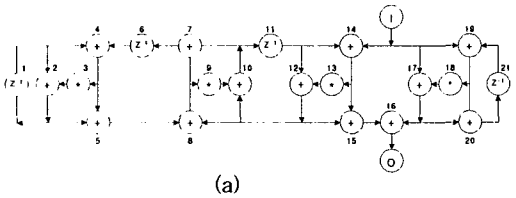
<Figure 4> Interprocessor communication scheduling algorithm.

5.4 An Example

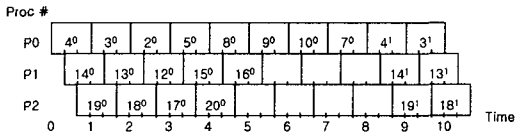
As an example, Fig. 5 illustrates an implementation of a wave filter on the CSPP system in which the system scheduling includes the operation scheduling and the interprocessor communication scheduling. In this example, it is assumed that each instruction cycle of the PE consists of three sub-cycles, and clock-skewing between PE is equal to one sub-cycle. In addition, each node of the filter, regardless of the operation type, is executed in one instruction cycle.

Fig. 5(b) is an operation schedule that has been reported in the previous paper by the author[5]. Based on the operation schedule, the interprocessor communication schedule is obtained. For example, in

Fig. 5(b), node 12 is scheduled at the 2nd instruction cycle of P1 and one of it's successor nodes, 8, is scheduled at the 4th instruction cycle of P0. In this case, P1 becomes a Ps and P0 becomes a Pd. Since the Ps and the Pd are different PEs, at the 2nd instruction cycle of P1, the send flag is set to TRUE. At the same time, since the 3rd instruction cycle is the earliest instruction cycle of P0 after the 2nd instruction on P1 has been executed, the receive operation on P0 is carried out on the 3rd instruction cycle. In addition, since P1 belongs to clock class 1 and P1 belongs



(a)



(b)

	P_0				P_1				P_2			
EC	NN	SF	RT1	RT2	NN	SF	RT1	RT2	NN	SF	RT1	RT2
0	4	F			14	F			19	F		
1	3	F			13	F			18	F		
2	2	F			12	T			17	T		
3	5	F	1		15	F	1		20	F		
4	8	F			16	F						
5	9	F										
6	10	T					2					
7	7	F										

(c)

<Figure 5> System scheduling for an implementation of a wave filter on 3 PEs CSPP system. (a) Fully-Specified Flow Graph of a 4th order Jaumann wave filter. (b) Operation schedule. (c) Complete schedule for each PE.

to clock class 0, the delay is determined by the formula of (3-(0-1+3)) = 1. As a result, the receive operation is carried out at the 1st sub-cycle of the 3rd instruction cycle of P0.

Fig. 5(c) illustrates a complete schedule of each PE in which all the information for interprocessor communication is explicitly included. Depending on the data of each schedule, the CSPP system is running in fully synchronous form without overhead. For example, in Fig. 5(c), at the 3rd instruction cycle of P0, two different operations are executed in parallel. The APU of P0 reads two data from the DM, executes the operation of node 5, and stores the operation result into the DM. In addition, the I/O CU of P0 receives data from the shared bus at the 1st sub-cycle, in which the data is sent at the 2nd instruction cycle of the P1. The data received by the I/O CU is stored into the DM of the P0.

<Table 1> Performance of CSPP scheduler for several digital filter structures. PB is the Processor Bound, IPBcspp is the Iteration Period Bound with the CSPP system, IPcspp is the achieved Iteration Period with the scheduler described in the CSPP system, IPBcyclo is the Iteration Period Bound with the parallel processing model of Fig. 2(a) and the Cyclo-Static scheduler [9], and IPcyclo is the achieved Iteration Period with the Cyclo-Static scheduler on parallel processing model of Fig. 2(a). Note that the unit of IPBcyclo, IPcyclo, IPBcspp, and IPcspp is not the instruction cycle but the sub-cycle.

Filter Structure	No. of nodes	*PB*	IPB_{cspp}	IP_{cspp}	IPB_{cyclo}	IP_{cyclo}
2nd oder IIR	10	4	8	8	10	10
4th oder IIR	20	8	16	16	18	18
lattice (3stages)	22	4	20	22	25	25
lattice (5stages)	36	7	35	37	40	40
lattice (10stages)	71	13	65	67	70	70
lattice (20stages)	141	25	125	127	130	130
wave filter	21	3	24	24	32	32
state-space	15	5	15	16	18	18

5.5 Performance of CSPP Scheduler

The scheduler has been implemented to run on the newly proposed multiprocessor system for a wide variety of digital filter types. Table 1 summarizes the performance of the CSPP scheduler. In addition to describing the performance of the CSPP scheduler itself, the simulation results show an important point with regard to the interprocessor communication. As described in Table 1, the IPcspp is less than the IPcyclo for all the filter structures. These results clearly show that, although the cyclo-static system always can achieve the IPB, since it fails to incorporate the interprocessor communication time delay into the scheduling model, the IPcyclo will always be larger than the IPcspp. As a result, even if the scheduler described in this paper fails to achieve the IPB in some cases, the scheduler along with the newly proposed parallel processing system (CSPP system) is very effective for a parallel implementation of recursive FSFGs.

6. Conclusion

This paper proposed a new class of multiprocessor system, called Clock-Skewed Parallel Processing system, and examined the problem of how to schedule such processors statically. Two important characteristics of CSPP system are: 1) they can handle the interprocessor communications very efficiently; and 2) they can explicitly incorporate the interprocessor communication delay into the parallel scheduling model.

The interprocessor communication strategy described in this chapter is a synchronous control mechanism in which each processor is connected to one or more shared bus. In such a CSPP system, the clock-skewing between instruction executions guarantees the conflict free data transfer through the shared bus system, and no bus contention control mechanisms are required. The hardware necessary to realize the required data bus network is therefore quite modest. In addition to the data busses themselves, each processor must have a synchronous, programmable I/O control unit which controls the interaction with the bus network.

In the CSPP system, all algorithm operations and all data transfers are determined at compile time. Thus, at run time, global optimality is achieved using local control. All the data transfers occur synchronously, and are part of the predetermined static schedule. The schedule includes the source and destination processors at each interprocessor communication instance, as well as explicit timing information such as when the source processor must put data into the shared bus system and when the destination processor can take the data from shared bus system.

The performance of CSPP scheduler was compared with the cyclo-static system. The results showed that the scheduler presented in this chapter is both powerful and practical because it can obtain optimal or near optimal solutions for most common digital filter structures.

REFERENCES

[1] T. P. Barnwell III and C. J. M. Hodges, "Optimal Implementation of Signal Flow Graphs on Synchronous Multiprocessors," 1982 International Conference on Parallel Processing, Belaire, Michigan, pp. 90-95, Aug. 1982.

[2] J.P. Brafman, J. Szczupak and S.K. Mitra, "An Approach to the Implementation of Digital Filters Using Microprocessors," IEEE Trans. on Acoustics, Speech, and Signal Processing, Vol. ASSP-26, No. 5, pp. 442-446, Oct. 1978.

[3] 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.

[4] S.J.A. McGrath, C.P. Hong, and T.P. Barnwell III, "A Scheduling Methodology for a Synchronous Cyclo-Static Multiprocessors" International Conference on Systolic Arrays, Killarney, Co. Kerry, Ireland, pp. 641-652, May, 1989.

[5] C.P. Hong and T.P. Barnwell III, "Implementation of Shift-Invariant Flow Graphs on Clock-Skewed Parallel Processing System," Proc. on International Symposium on Circuit and Systems, New Orleans, Louisiana, pp. 2658-2661, May 1990.

[6] C.P. Hong and J.J. Woo, "An Optimal

Implementation of Digital Filters on Multiple Pipelined Processor," Proc. on International Workshop on Intelligent Signal Processing and Communication, Seoul, Korea, pp. 345-350, Oct. 1994.

[7] S.Y. Kung, "On Supercomputing with Systolic/Wavefront Array Processors," Proceedings of the IEEE, Vol. 72, No. 7, pp. 867-884, July, 1984.

[8] A. V. Oppenheim and R. W. Schafer, Digital Signal Processing, Englewood Cliffs, NJ: Prentice-Hall, 1975.

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

홍 춘 표 (Chun-pyo Hong)

1978년 2월 경북대학교 전자공학과(학사)

1986년 12월 Georgia Institute of Technology (미국), 전기·컴퓨터공학과 (석사)

1991년 12월 Georgia Institute of Technology (미국), 전기·컴퓨터공학과 (공학박사)

1978년 3월~1985년 6월 국방과학연구소, 연구원

1990년 1월~1991년 12월 Atlanta Signal Processors, Inc. (미국), Consultant

1992年 9月~(현재) 대구대학교 정보통신공학부, 부교수

주관심 분야: DSP 하드웨어 및 소프트웨어, VLSI 신호처리, 컴퓨터 구조

양 진 모 (Jin-mo Yang)

1980년 2월 경북대학교 전자공학과(학사)

1989년 5월 버지니아공과대학 (미국), 전기공학과(석사)

1993년 12월 Georgia Institute of Technology (미국), 전기·컴퓨터공학과 (공학박사)

1979년 12월~1987년 2월 국방과학연구소, 연구원

1994년 9월~(현재) 대구대학교 정보통신공학부 조교수

2001년 6월~(현재) 한국전자통신연구원(ETRI) 초빙연구원

주관심 분야 : RF 능·수동소자 모델링, 통신용 RF 집적로 설계, VLSI 시스템 설계