# 論文 / 著書情報
# Article / Book Information

| | |
|---|---|
| 論題(和文) | |
| Title(English) | Computational Ordering of Digital Networks under Pipeline Constraints and its Application to DSP Compilers |
| 著者(和文) | 杉野暢彦, 王尾誠司,, 西原明法 |
| Authors(English) | NOBUHIKO SUGINO, Seiji Ohbi, AKINORI NISHIHARA |
| 出典(和文) | 電子情報通信学会, Vol. E72, No. 12, pp. 1299-1306 |
| Citation(English) | Transactions of IEICE, Vol. E72, No. 12, pp. 1299-1306 |
| 発行日 / Pub. date | 1989, 12 |
| URL | http://search.ieice.org/ |
| 権利情報 / Copyright | 本著作物の著作権は電子情報通信学会に帰属します。<br>Copyright (c) 1989 Institute of Electronics, Information and Communication Engineers. |

PAPER <Special Issue on the 2nd Karuizawa Workshop on Circuits and Systems>

# Computational Ordering of Digital Networks under Pipeline Constraints and Its Application to DSP Compilers

Nobuhiko SUGINO†, *Member*, Seiji OHBI†, *Nonmember*
*and* Akinori NISHIHARA†, *Member*

**SUMMARY** To reduce users' load in writing programs for DSPs (Digital Signal Processors), especially those with several pipeline stages, an improved search algorithm is proposed which determines the computational order for a given digital network and generates codes automatically. To search an effective order, a new representation of the precedence form is proposed. A new technique to generate effective codes is also presented. Taking the DSP architecture into account, the method re-allocates the computational order to reduce blanks in the microcodes. These methods are applied to the compilers for $\mu$PD77230, $\mu$PD7720 and TMS32010/20, which give relatively effective codes.

## 1. Introduction

A digital signal processor (DSP) has been dramatically improved during the past decade and has been used for the wide applications, especially in communication and control fields. A digital filter plays a major role in digital signal processing algorithms used in such applications. A DSP can implement various digital filter network structures only by changing internal instruction codes. Systems are then developed by software rather than hardware. Then a DSP programmer is required to have enough knowledge of both the processor architecture and the processing algorithm to write an efficient (short) program.

Recently, so-called second generation DSPs have appeared. Usually these DSPs have short cycle time and several stages of pipeline to improve their performance. Pipelining causes much more difficulty in writing an efficient program ; since an instruction may output the result several cycles later than its execution, extreme care must be taken for the timing to access that data not to cause a pipeline hazard. Even for a skillful programmer, it is a difficult and a tiresome work to produce an efficient program for such a DSP.

On the other hand, from the viewpoints of circuit theory, many types of networks have been proposed ; e.g. wave digital filters[1] have very low-sensitivity and are free from parasitic oscillations. In return for such good characteristics, wave digital filters often have compli-cated structures, which causes much difficulty in DSP programming.

To reduce the cost and time for DSP programming, powerful software tools such as high-level language and its compiler would be very important. This theme has been researched in various way, and several compilers are now available from DSP manufacturers. In spite of these efforts, programmers' load is still heavy since the computational order for a given network must be assigned by themselves. Thus our aim is to develop such a system that generates optimum codes automatically from the structural description of given digital networks. It is known, however, that the optimum coding problem is NP-hard. Therefore efficient heuristic algorithm is necessary to be developed.

In Refs. (2)-(5), DIMPL (Digital network IM Plementation Language) and its compiler for $\mu$PD7720 (NEC) were proposed. The compiler itself determines the computational order by a heuristic method based on the depth-first-search and the network decomposition. That compiler consists of two parts ; the computational ordering part which is independent of target DSPs and the code generation part dependent on the target. By modifying the second part, a compiler for TMS32010/20 (TI) was also developed[4]. That ordering method, however, does not lead to efficient codes for recent DSPs having multiple stages of pipeline.

This paper shows new algorithms that can determine efficient ordering of a network for the pipelined DSPs. By the use of these algorithms to our compiler, users become free from cumbersome consideration of data movements related with the pipeline, yet efficient codes are obtained. The proposed methods also lead to satisfactory compilers for non-pipelined DSPs.

## 2. A New Computational Ordering

### 2.1 Precedence Form

A given network can be represented in a signal flow graph (SFG) notation, as shown in Fig. 1. The SFG can be divided into node/branch sets depending on the preced-

Manuscript received July 17, 1989.

† The authors are with International Cooperation Center for Science and Technology, Tokyo Institute of Technology, Tokyo, 152 Japan.

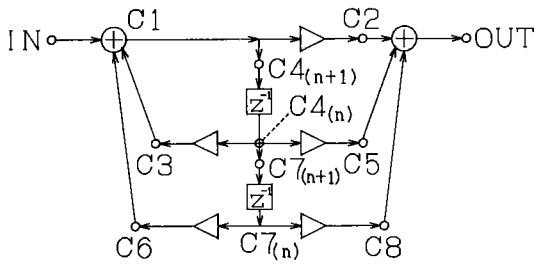
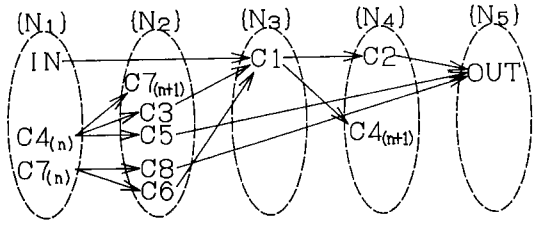
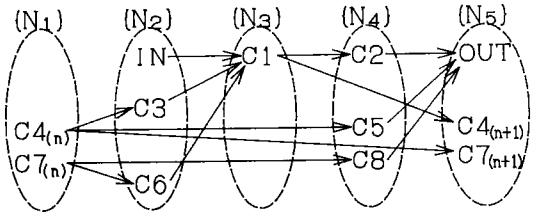



Fig. 1 An example digital network (SFG).




(a) The earliest representation (ERPF).




(b) The latest representation (LRPF).

Fig. 2 Precedence form for network in Fig. 1.

ence relation, to get a precedence form (PF) proposed by Crochiere and Oppenheim[6]. DSP programs or any computation of a network must not violate that precedence relation. From the viewpoint of writing a program, however, the PF of a given network is not sufficient, i.e. it does not lead to a unique program because it only gives a computational order between the sets (which is called as level here) and not the order between nodes/branches in the same set. Therefore, further order of nodes/branches should be decided. This is done by a search of nodes/branches.

The PF in Ref. (6) is obtained by iterative extraction of node sets which are computable from its preceding node sets. Thus each node set consists of all nodes which are computable at that level. Since a node is placed in the earliest possible level, this original PF is called here as the earliest representation (ERPF). The ERPF for the network shown in Fig. 1 is given in Fig. 2 (a).

To get low complexity and high efficiency of search, the number of candidates to be searched should be small, especially in the earlier stages. For this purpose, every node is moved to its latest possible level, which leads to the new "latest representation" of a precedence form (LRPF). Algorithm to get the LRPF are :

(1) Find the output and delay-inputs in a given network and assign them to the node set $\{N_1\}$.

(2) Find the nodes that are directly reachable to the elements of $\{N_k\}$ $(k=1, 2, \cdots)$, and assign those nodes to the node set $\{N_{k+1}\}$. If a node is included in both set $\{N_{k+1}\}$ and set $\{N_m\}$ $(m=1, 2, \cdots, k)$, that node is excluded from the node set $\{N_m\}$.

(3) Repeat step 2 until no node is assigned to the node set. The number of node sets is referred to as $n$. Then, the order of the node sets are reversed, i.e. node set $\{N_k\}$ $(k=1, 2, \cdots, n)$ becomes $\{N_{n-k}\}$.

If a node belongs to no set, that node is unnecessary in the computation of the network. If a delay-free-loop exists in the given network, this algorithm does not stop and repeats step 2 forever.

The example of LRPF is shown in Fig. 2 (b), where the configuration of nodes very much differs from that in Fig. 2 (a). Especially, nodes in earlier node sets, $\{N_1\}$ or $\{N_2\}$, such as nodes C5 and C8, show remarkable movements to the later node sets, and the earliest sets consists of fewer nodes.

Since the values of the nodes in set $\{N_1\}$ are given in advance, the computation starts from the nodes in $\{N_2\}$ for each sampling cycle. In the LRPF, the nodes in $\{N_1\}$ are the starting points of the longest or critical path(s) in the network. Computation along the longest or critical path(s) is often efficient in data handling, since data are accessed and/or manipulated successively using a few registers. Thus, if a search starts from $\{N_2\}$ of the LRPF and gives priority to the nodes in the critical path (s), the résultant order is expected to be efficient.

Note that the precedence relations are not violated in the proposed representation. The LRPF only gives the computational ordering between the node/branch sets as the ERPF does, and the further node/branch ordering is required.

## 2.2 Node Ordering

In an SFG, delay branches form directed cycles. By the extraction of all delay elements, the network reduces to a Directed Acyclic Graph (DAG). Since an optimum ordering of the nodes or branches in a DAG is an NP-hard problem, a heuristic algorithm is necessary. A method in Ref. (1) applies the 'Depth-First Search' to determine the order between the subgraphs, which have forms of binary trees and are derived by partitioning a given DAG. The optimal code generation algorithm is available for each subgraph, and an efficient final program results for non-pipelined DSPs.

Under the pipeline constraints, however, there are no optimal code generation algorithms and partitioning to subgraphs is no longer useful. Another heuristic method is required for pipelined DSPs to yield an efficient program.

A new ordering method is proposed here which determines the evaluation sequence of nodes in a net-

work with a search algorithm. It uses the number of pipeline stages of the target DSP to take the pipeline constraints into account. Since such pipeline stages as 'instruction fetch' have no effect on the computation and cause no pipeline hazard, these pipeline stages are not considered here. Thus, actual DSPs have a bit more pipeline stages than those treated here.

Initially, the nodes in a given network are labeled as the level of the LRPF (Fig. 3 (a)). The search uses three states of nodes as mentioned below.

Definition of the node state

not-yet-decidable : nodes whose order cannot yet be decided.
decidable : nodes whose order is ready to be decided.
decided : nodes whose order has already been decided.

For the two consecutive nodes, the following two terms are used in the algorithm to indicate their relation.

Definition of the node for two consecutive nodes

child (ren) : the preceding node(s)
parent (s) : the following node(s)

The whole nodes are 'not-yet-decidable' at first, except several 'decided' nodes. The 'decided' nodes are the outputs from the delay elements, which have already been evaluated at the end of the preceding clock cycle, and the input of the network, which is just taken at the beginning of the present cycle.

The proposed search algorithm is stated as follows ;

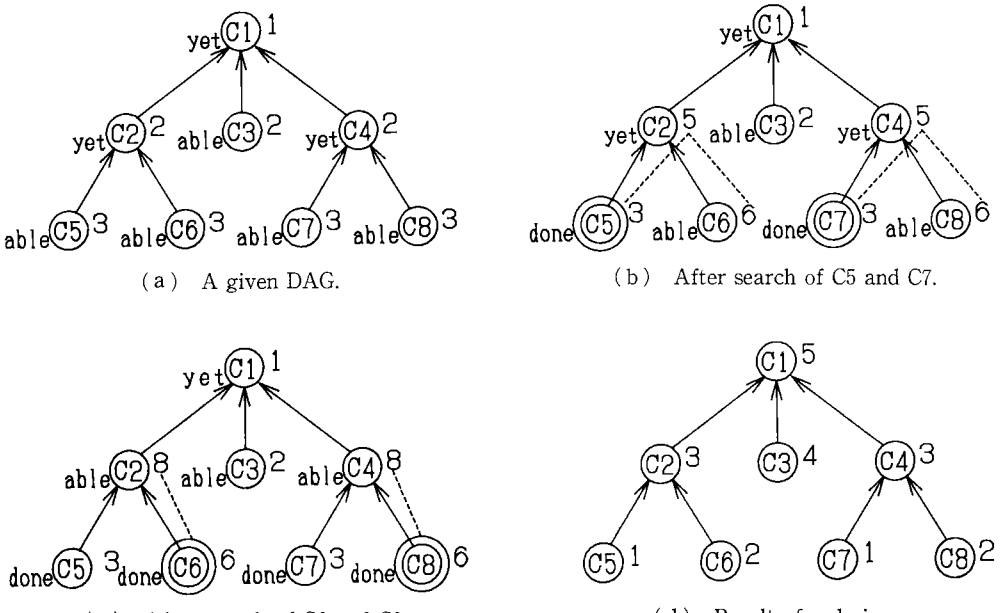
Algorithm : Node ordering for a given DAG

**procedure** search(a DAG of a given network)
**type** node status=(not-yet-decidable, decidable, decided) ;

**begin**
  **repeat**
    search 'not-yet-decidable' nodes whose children are 'decided' ; mark these nodes as 'decidable'
    select $N$ 'decidable' nodes having larger labels ;
    **for** each of $N$ nodes
      **begin**
        mark the node as 'decided' ;
        search a parent of that node with the largest label ;
        increase the label of the parent by a number M1 ;
        search the longest path preceding from the parent ;
        (comparing the label according to the ERPF)
        increase the labels of the nodes included in the path
        by a number M2 ;
      **end** ;
  **until** whole nodes are marked 'decided' ;
**end**.

In the algorithm, $N$ is the number of pipeline stages in a target processor, and M1 and M2 are numbers that can control the progress of the search. When M1 is larger than M2, the search is proceeded in a depth first way. On the contrary, the search is similar to the breadth first search, if M2 is larger. Here, these numbers are chosen to be the maximum number in the initial labels, to identify the nodes whose labels are increased.

The algorithm is now illustrated by a simple exam-




(a) A given DAG.
(b) After search of C5 and C7.
(c) After search of C6 and C8.
(d) Result of ordering.

Fig. 3 Example of the node ordering.

ple shown in Fig. 3 (a), where the numbers are labels assigned to the nodes. The state of each node is also shown in the figure, where 'done', 'able' and 'yet' mean the node is 'decided', 'decidable' and 'not-yet-decidable', respectively. Here, 2-stage pipeline is assumed.

First, {C3, C5, C6, C7, C8} are the decidable nodes in Fig. 3 (a). The nodes C5 and C7 are assumed to be decided in this search cycle, resulting in Fig. 3 (b). According to the algorithm, the labels of C2 and C4, the parents of C5 and C7, respectively, are increased by M1, that is 3 in this example. And also C6 and C8, children of C2 and C4 respectively, have their labels added by M2 or 3. The result is shown in Fig. 3 (b), where the dotted lines show the propagation of the process according to the child-parent relation.

Second, {C3, C6, C7} are the decidable nodes. For labels of C6 and C7 are larger than that of C3, C6 and C7 are searched, and the labels of their parents, C2 and C4, are increased (Fig. 3 (c)). In the same way, C2 and C4 are searched, and C3 follows. Finally, after the search reaches C1, the node order results as depicted in Fig. 3 (d).

The advantage of this method over the one used in Refs. (2), (3) is that $N$-stage pipeline can be taken into account. Moreover, the algorithm is simpler and takes less time to determine the order ; the complexity of the old method is $O(n^3)$, whereas $O(n^2)$ for the proposed method, where $n$ is the number of nodes/branches included in a network. The complexity is evaluated in Appendix 1.

## 3. A New Code Generation Method

The ordering algorithm uses the common information of various DSPs having different architectures and different instruction sets to cover as many DSPs as possible. So the code generation part must be flexible to adapt various DSPs. To generate efficient codes for those various DSPs, a code allocation method is proposed here. This method is similar to some horizontal microcode compaction methods[7], but differs in considering the machine cycle necessary for each instruction. Therefore, this method can be applied to DSPs having instructions with different cycles.

### 3.1 Decomposition of the Multi-Operation Node

For the code allocation, branches rather than nodes are convenient, since the branches of a given network just correspond to microinstructions such as add, subtract, multiply, and move. The order of branches can roughly be decided according to the order of nodes derived from the search algorithm.

In order for branches in a DAG to correspond to actual instructions, multi-input summation nodes are decomposed as shown in Fig. 4 (a). Here, every branch is attributed as an instruction such as 'add', 'sub', 'load',

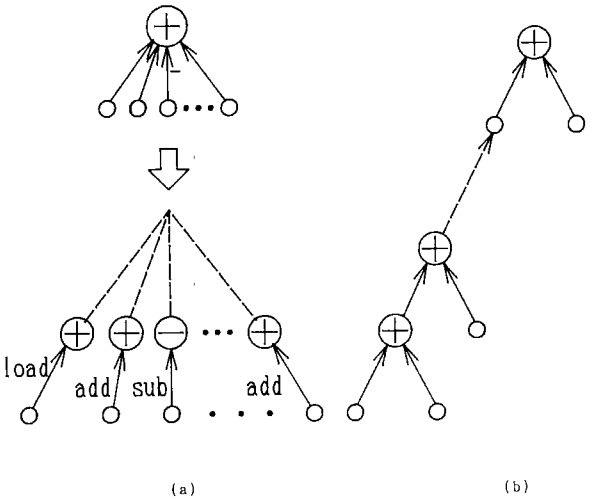



Fig. 4 Decomposition of multi-input node.
(a) Decomposition.
(b) A binary tree.

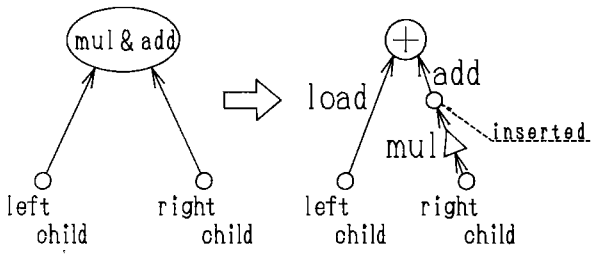



Fig. 5 Decomposition of multiply-add node.

etc. This decomposition is just similar to consider a binary tree in Fig. 4 (b), when the branches are ordered in the code allocation process.

Usually, DSPs are designed to execute multiply-add/sub operations effectively. In Ref. (3), every combination of multiply and add/sub operations. is treated as one operator, to yield an efficient program. Under the pipeline constraints, however, such a combination often causes the inefficient programs. Therefore, each multiply-add/sub node is decomposed into a summation node (which is decomposed further) and a branch with a multiplier as depicted in Fig. 5.

### 3.2 Code Allocation

After the decomposition, branches exactly correspond to microcodes and they are allocated for each instruction cycle according to the computational order of branches. To yield efficient codes, as many branches as possible are assigned in one cycle. Then the availability of data required by the branches is examined by identifying the contents of hardware resources such as a register, an arithmetic unit, a multiplier, a data-memory, etc. If the data is not yet available, the corresponding branch is allocated to the following cycle and another

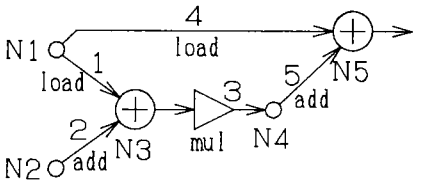



Fig. 6 A DAG for the code allocation.

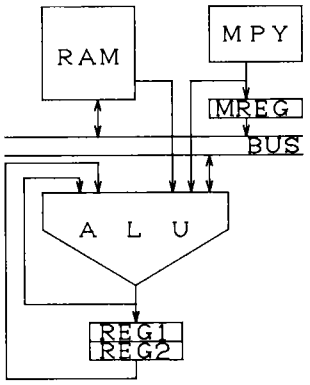



Fig. 7 Architecture of a DSP ($\mu$PD77230).

branch can be allocated to that cycle. This part is just like a detailed processor simulation.

The detail of the procedure in this method is explained using the DAG in Fig. 6. In this figure, each branch is accompanied by its instruction type and a number, which is the evaluation order just given by the preceding step. The procedure uses a content table of hardware resources as shown in Fig. 8 to examine the availability of the required data. In Fig. 8, the row shows the instruction cycle ; the numbers denoted to the rows as '+0', '+1', ··· correspond to the current cycle, the next cycle, ···, respectively. Thus this table shows when the data resulting from the current instruction become available, and the row size equal to the number of pipeline stages is sufficient. For the next instruction cycle, the contents in the table are popped up and renewed ; the data in the row '+0' disappears, and the data in '+1' scroll up to '+0', and so on.

Now, the target processor is supposed to have an architecture shown in Fig. 7 and to have instruction cycles summarized in Table 1. These parameters are the same as those of $\mu$PD77230.

In Fig. 8, the allocated codes are shown with the content table for each instruction cycle. At first, branch 1 can readily be executed, as the argument, N1, is on the memory. Based on the given parameter, data N1 is assigned to resource REG1 in the next (+1) cycle. Since no other branches can be allocated in this cycle, the content table is popped up for renewal, to proceed to the second cycle. Since data N1 is available in the register REG1 and data N2 is on the memory, branch 2 can be allocated for this cycle. As the parameter shows, data N3 is assigned to ALU in the next cycle and REG1 in the next-next cycle. In the third cycle, branch 4 is chosen

| #CYCLE | #BRANCH | CODE | | RESOURCE TABLE ALU | Reg1 | Reg2 | MPY | MReg |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | LD Reg1, N1 | +0 | - | - | - | - | - |
| | | | +1 | - | N1 | - | - | - |
| | | | +2 | - | N1 | - | - | - |
| 2 | 2 | ADD Reg1, N2 | +0 | - | N1 | - | - | - |
| | | ↓ | +1 | N3 | N1 | - | - | - |
| | | (N3) | +2 | - | N3 | - | - | - |
| 3 | 4 | LD Reg2, N1 | +0 | N3 | N1 | - | - | - |
| | | | +1 | - | N3 | N1 | - | - |
| | | | +2 | - | N3 | N1 | - | - |
| 4 | 3 | MOV MPY, Reg1 | +0 | - | N3 | N1 | - | - |
| | | ↓ | +1 | - | N3 | N1 | N4 | - |
| | | (N4) | +2 | - | N3 | N1 | N4 | N4 |
| 5 | 5 | ADD Reg2, MPY | +0 | - | N3 | N1 | N4 | - |
| | | ↓ | +1 | N5 | N3 | N1 | N4 | N4 |
| | | (N5) | +2 | - | N3 | N5 | N4 | N4 |

Fig. 8 Example of the code allocation.

Table 1 Parameter of an example target DSP.

| Instruction | Output | Cycles required |
|---|---|---|
| Add./Sub. | ALU | 1 cycle |
| Add./Sub. | REG | 2 cycles |
| multiply | MPY | 1 cycle |
| multiply | MREG | 2 cycles |
| load/store | REG, RAM, etc | 1 cycle |

instead of branch 3 requiring N3, which is not ready to be referred by the multiplier (MPY). In the same way, further allocation is also shown.

Another possibility for the code generation is the modification of the conventional method[2] ; NOPs are inserted not to cause pipeline hazard and the postpass compaction[8] is applied to improve the code efficiency. The proposed method, however, is much easier and more effective especially for many stages of pipeline.

## 4. Compiler

The whole procedure of the compiler is shown in Fig. 9. First, the compiler reads a network description written in DIMPL and represents it by nodes/branches for the internal use. The node/branch ordering follows the removal of meaningless nodes/branches, which is mentioned in Refs. (2), (3) and Appendix 2. Finally, instruction codes are generated by the code allocation method or other methods, such as code compaction method referred below. This part is closely related to the target DSPs, while the previous parts are dependent

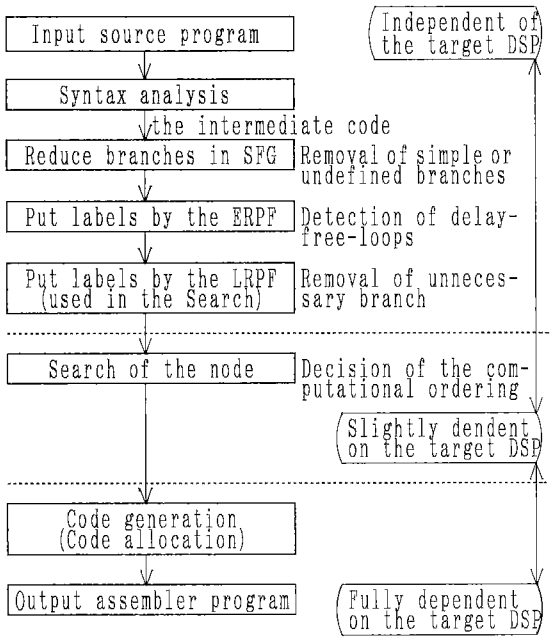



Fig. 9 Structure of the compiler.

on processors only for the number of pipeline stages. The compiler can be adapted to various DSPs by slight modification of the last part. Compilers for the non-pipelined DSPs can be built in the same structure.

The compiler itself is written in PASCAL (Turbo Pascal) on personal computers. The compiler is prepared for earlier DSPs ($\mu$PD7720 and TMS32010/20) and for so-called second-generation DSP, $\mu$PD77230 that has 2 stages of pipeline (additional 1 for the instruction fetch) and micro-programming style instructions.

For the $\mu$PD77230, Table 2 shows the comparison of the compilers in terms of the number of program execution steps for the various networks. The results of the previous version of our compiler[3] with the postpass code optimization algorithm[8] are also listed. It is obviously seen that the derived codes are up to 1.5 times as long as the ones assembled by an expert, and the proposed method can yield more efficient codes than the previous one.

For $\mu$PD7720 and TMS32020, so called first generation DSPs, Tables 3 and 4 show the length of generated programs, respectively. For the comparison, the optimal programs are also listed ; for TMS32020, the optimal programs are found among all the available programs[9], and for $\mu$PD7720, found among many programs carefully assembled by experts. The compiled codes are, for TMS32020 up to 1.1 times more than ones derived in Ref. (9), and for $\mu$PD7720, up to 1.5 times more than ones assembled by experts.

Table 2 Program steps for $\mu$PD77230.

| Network (Order) | Proposed algorithm | Previous compiler | Hand assembled (Optimal) |
|---|---|---|---|
| Wave (3) | 34 | 39 | 27 |
| Cascade (6) | 42 | 38 | 30 |
| Wave Lattice (7) | 46 | 49 | 38 |
| Lattice (5) | 32 | 38 | 24 |
| State-Space (3) | 25 | 43 | 22 |

Table 3 Program steps for $\mu$PD7720.

| Network (Order) | Proposed algorithm | Previous compiler | Hand assembled (Optimal) |
|---|---|---|---|
| Wave (3) | 46 | 44 | 25 |
| Cascade (6) | 51 | 51 | 48 |
| Wave Lattice (7) | 54 | 55 | 46 |
| Lattice (5) | 42 | 42 | 30 |
| State-Space (3) | 33 | 33 | 23 |

Table 4 Program steps for TMS32020.

| Network (Order) | Proposed algorithm | Optimal Code in [9] |
|---|---|---|
| FIR (4) | 13 | 13 |
| IIR (2) | 16 | 16 |
| Lattice (2) | 29 | 27 |
| GIC (2) | 28 | 28 |
| State-Space (2) | 23 | 22 |
| Wave (4) | 50 | 50 |

## 5. Conclusion

This paper proposed a new representation of precedence form, heuristic algorithms to decide the computational order, and a code allocation method to generate efficient codes. The proposed methods were used in the compilers for commercially available DSPs. Codes derived from the compilers are efficient enough. Such compilers are useful not only for beginners but also for experts to save the programming load. Applications of this system to adaptive filters and other DSP algorithms are studied now.

## Acknowledgement

The authors are grateful to Profs. T. Yanagisawa and N. Fujii of Tokyo Institute of Technology, for their valuable discussions, and thank H. Sugaya, Y. Kawakami, H. Ishizuka and M. Fukuda oaf NEC Corporation for their support of this work.

**References**

(1) A. Fettweis: "Wave digital filters: Theory and practice", Proc. IEEE, **74**, 2, pp. 270-327 (1986).
(2) H. Sakamoto, E. Watanabe and A. Nishihara: "Code optimization of digital networks for signal processors", Proc. of ISCAS, pp. 1403-1406 (June 1985).
(3) N. Sugino, A. Toshikiyo, E. Watanabe and A. Nishihara: "Computational ordering of digital networks and its application to a compiler system for digital signal processors", Trans. IEICE, **J71-A**, 2, pp. 327-335 (Feb. 1988).
(4) N. Sugino, S. Ohbi and A. Nishihara: "Improved computational ordering and its application to DSP compilers", IEICE Technical Report, **CAS88**-139, pp. 17-24 (March 1989).
(5) N. Sugino, S. Ohbi and A. Nishihara: "A computational ordering method for linear digital networks to be implemented on pipelined DSPs and its applications to compilers", Proc. of 2nd CAS Karuizawa Workshop, pp. 108-115 (May 1989).
(6) R. E. Crochiere and A. V. Oppenheim: "Analysis of linear digital networks", Proc. IEEE, 63, 4, pp. 581-595 (1975).
(7) M. Tokoro, E. Tamura and T. Takizawa: "Optimization of microprogram", IEEE Trans. Computer, **C-30**, 7, pp. 491-504 (1981).
(8) J. Hennessy and T. Gross: "Postpass code optimization of pipeline constraints", ACM Trans. on Programming Language and System, **5**, 3, pp. 422-448 (1983).
(9) O. Hivarinen, J. Honkanen, J.-P. Sairanen, I. Hartimo and O. Simura: "Signal processors code generation: an analytic study and practical consideration", Proc. of ISCAS, pp. 1053-1056 (June 1988).
(10) A. V. Aho, J. E. Hopcroft and J. D. Ullman: "The design and analysis of computer algorithms", Addison-Wesley (1974).

## Appendix 1: Complexity in the Algtorithms

Complexity, or time spent, in each algorithm is considered for the network of $n$ nodes as follows. The number of branches is assumed to be proportional to $n$ for simplicity. Most of the networks, e.g. cascades and wave digital filters, satisfy this assumption. Complexity in each algorithm in terms of the number of branches can be easily derived in the same way.

### A-1. 1 Precedence Form

The ERPF of the given network can be derived by extracting nodes iteratively as mentioned in Ref. (6). Since each node is scanned once, complexity is proved to be $O(n)$. This can also be shown by the fact that the algorithm can be developed by a simple extension of the depth-first-search[10].

Although the LRPF algorithm above is stated simply, it is not efficient in practice; some nodes are scanned more than once, and complexity is estimated between $O(n)$ and $O(n^2)$. Another algorithm can be derived, which depends on the depth-first-search and searches the network in the reverse way. According to this algorithm, each node is scanned once and complexity is proved to be $O(n)$[10].

### A-1. 2 Computational Ordering

[Decomposition of the digital network[2],[3]]
This method consists of two procedures:
the branching point detection
decomposition to the partitioned subgraphs
Both procedures have only to scan nodes of a given network once, so that the complexity in this method is $O(n)$.

[Computational ordering method in Ref. (2)]
The method decomposes the given network into subgraphs and orders the subgraphs by the depth-first-search. Complexity is $O(n)$, but generated codes are not efficient, so that the method in Ref. (3) is necessary.

[Computational ordering method in Ref. (3)] ]
In the method, the given circuit is decomposed into subgraphs first. Then, while the subgraphs are searched (costs $O(n)$), the method refers the following relation between nodes/subgraphs.
Relation between two nodes/subgraphs
Relation is defined as the minimum number of branches connecting the two nodes/subgraphs, i.e. to find the minimum path between two nodes/branches. Complexity is $O(n^2)$, where $n$ is the number of the nodes/subgraphs contained in the network.
Since all relations are examined every time subgraph is scanned, complexity of this method is estimated to be $O(n^3)$.

[Proposed computational ordering method]
This method can be divided into two parts;
Initial labeling by the LRPF
Complexity is $O(n)$ from A-1. 1.
Improved computational ordering
As mentioned before, the search algorithm selects nodes in the network once (costs $O(n)$). For each selection, the labels of the parent node and its children nodes are changed (costs $O(n)$). Thus, the complexity is estimated as $O(n^2)$.
In conclusion, complexity is proved to be $O(n^2)$.

## Appendix 2: Effect of the Node/Branch Removal

Digital networks sometimes include meaningless calculations, especially when the networks are modularly described using several identical subcircuits. For example, in wave digital filters, the reflection to the source (complementary output) such as 'UNREF' in Fig. A•1, is not necessarily to evaluate. The reflection from the matched load (which is always 0) such as 'ZEROINP' in Fig. A•1, also accompanies meaningless calculations such as $0+x$ and $0 * x$. Removal of these unnecessary nodes/branches reduces the complexity of

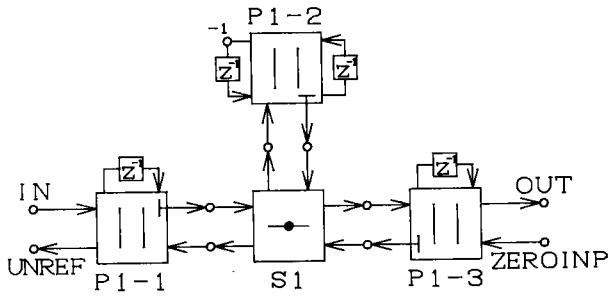
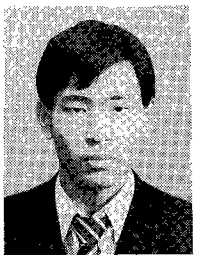



Fig. A·1 3rd order wave digital network.

the network, and leads to more efficient programs. In the proposed compiler, such nodes are reduced automatically rather than manually. This allows users to describe a network effectively. An abbreviated description for the network in Fig. A·1 is illustrated in below (See Ref. (3) for details about description).

```
FILTER WDF3RD (IN>OUT) ;
        ·····Declaration of nodes and coefficients
  SUBROUTINE P1 (A1, A2, A3>B1, B2, B3/M) ;
        ·····Internal description of the parallel adapter
  SUBROUTINE S2 (A1, A2, A3>B1, B2, B3/M1,
                 M2) ;
        ·····Internal description of the series adapter
BEGIN
  P1 (·····) ;
  S2 (·····) ;
  P1 (·····) ;
  P1 (·····) ;
        ·····Description of delays
END
```



Nobuhiko Sugino was born in Yokkaichi on November 19, 1964. He received B. E. and M. E. degrees in electronics from Tokyo Institute of Technology in 1986 and 1989, respectively. He is now a doctor course student in the same Institute. His main research interests are in the digital signal processing, its hardware and software.

Seiji Ohbi was born in Zentsuji, Kagawa on September 28, 1966. He received B. E. degree in electronics from Tokyo Institute of Technology in 1989. He is now a student in M. E. Course in the same Institute. His main research interests are in hardware and software of digital signal processors.

Akinori Nishihara was born in Fukuoka on February 26, 1951. He received B. E., M. E. and Dr. Eng. degrees in electronics from Tokyo Institute of Technology in 1973, 1975 and 1978, respectively. Since 1978 he has been with Tokyo Institute of Technology, where he is now Associate Professor of International Cooperation Center for Science and Technology. He is responsible not only for research cooperation with ASEAN countries but also for education and research in the field of circuits and systems. His main research interests are in filter design and applications of digital signal processors. He received Young Engineer Award in 1982. Dr. Nishihara is a member of IEEE.