PAPER *Special Section of Selected Papers from the 10th Karuizawa Workshop on Circuits and Systems*

An Overlapped Scheduling Method for an Iterative Processing Algorithm with Conditional Operations

Kazuhito ITO[†], ***Member and*** **Tatsuya KAWASAKI**[†], ***Nonmember***

SUMMARY One of the ways to execute a processing algorithm in high speed is parallel processing on multiple computing resources such as processors and functional units. To identify the minimum number of computing resources, the most important is the scheduling to determine when each operation in the processing algorithm is executed. Among feasible schedules satisfying all the data dependencies in the processing algorithm, an overlapped schedule can achieve the fastest execution speed for an iterative processing algorithm. In the case of processing algorithms with operations which are executed on some conditions, computing resources can be shared by those conditional operations. In this paper, we propose a scheduling method which derives an overlapped schedule where the required number of computing resources is minimized by considering the sharing by conditional operations.

key words: *scheduling, high-level synthesis, digital signal processing, conditional branch, overlapped schedule*

1. Introduction

One of the ways to execute a processing algorithm in high speed is parallel processing on multiple computing resources such as processors and functional units. The number of computing resources must be no less than the maximum number of operations in the processing algorithm to be executed concurrently. Therefore the required number of computing resources may vary depending on when each operation is executed. Scheduling in high-level synthesis is to determine when operations are executed for either of the following targets. One target is to minimize the required number of computing resources for a specified time limit to complete the processing algorithm. The other target is to minimizing the required time to complete the processing algorithm for a specified maximum number of computing resources. These are called *time-constrained scheduling* and *resource-constrained scheduling*, respectively. In this paper, we concentrate on time-constrained scheduling.

An iterative processing algorithm performs an identical processing repeatedly on each of an indefinitely continuing series of input data. An example of iterative processing algorithm is digital signal processing. Each repetition of the iterative processing algorithm is called an *iteration*. In a static execution of the iterative processing algorithm, if the input data arrive periodically, then an iteration must also be initiated periodically on each arrival of input data. However, it does not mean that all the operations in an iteration are to be completed within the period. We can postpone some operations to be executed in parallel to the operations of the next iteration. Hence several iterations can be executed in parallel so that iterations overlap with each other. Such a schedule is called a *overlapped schedule*. By overlapped scheduling, a shorter period can be achieved than the longest operations in the processing algorithm. Therefore higher speed processing can be realized by overlapped scheduling.

Operations in a processing algorithm may vary depending on the input data or intermediate results of the processing algorithm. In that case, some operations are executed only when a certain condition holds. The operation whose execution depends on a condition is called a *conditional operation*. If one conditional operation is executed when a condition holds and another conditional operations is executed when the condition does not hold, then these two operations can be scheduled to be executed at the same time on an identical computing resource since these operations are executed *mutually exclusively*. We should consider sharing a computing resource by mutually exclusive operations in scheduling a processing algorithm containing conditional operations.

For scheduling processing algorithms with conditional operations, some scheduling methods have been proposed [1]–[4]. Modeling scheduling as a binary integer nonlinear programming and using binary decision diagram (BDD) to solve it is proposed in [2]. Although this approach derives an optimal schedule, the problem size is limited by a BDD package used. A heuristic scheduling method for relatively large processing algorithm is proposed in [3]. This method hierarchically schedules blocks of operations and hence can efficiently schedule large scale processing algorithms. However, conditional operations are always executed mutually exclusively after the condition is resolved. Hence the lower bound of the iteration period is rather large and it does not suit for the target of overlapped scheduling. In [4], a heuristic scheduling algorithm is proposed where conditional operations may be executed before the condition

Manuscript received June 27, 1997.
Manuscript revised October 6, 1997.

[†]The authors are with the Department of Electrical and Electronic Systems, Saitama University, Urawa-shi, 338-0825 Japan.

is resolved if necessarily. All of these scheduling methods do not take into account the overlapped scheduling and therefore the minimum iteration period might not be achieved.

In this paper, we propose a scheduling method to derive an overlapped schedule for a given iterative processing algorithm with conditional operations. The rest of this paper is as follows. In Sect. 2, sharing computing resources by conditional operations is discussed. Data-flow graph to represent processing algorithm is introduced in Sect. 3. The overlapped schedule is revisited in Sect. 4. Precedence constraints which must be satisfied in any schedules are also discussed in this section. The range-chart-guided scheduling method proposed in [5] is briefly reviewed in Sect. 5. The proposed scheduling method is described and its computational complexity is analyzed in Sect. 6. Some experimental results are shown in Sect. 7.

2. Processor Sharing by Conditional Operations

Figure 1 (a) shows an example of a processing algorithm including conditional operations. In this processing algorithm, b is subtracted from a signal x if x is greater than a ($x > a$), or c is added to x otherwise ($x \leqq a$). Operation 1 compares the input signal x with a constant a. It determines if a condition holds and is called *a decision operation.* If the condition holds, i.e., the condition $x > a$ is true, then operation 2, that is a subtraction, is executed. On the other hand, if the condition does not hold, i.e., the condition $x > a$ is false, then operation 3, that is an addition, is executed. Let two operations each of which is executed on the condition exactly opposite to the other be said *mutually exclusive* to the other operation. For example, operation 3 is the mutually exclusive operation of operation 2. Since operations 2 and 3 are mutually exclusive to each other, it is possible to schedule operations 2 and 3 be executed at the same time on an identical processor as shown in Fig. 1 (b). After the operation 1, operation 2 is executed if the condition holds (Fig. 1 (c)), or operation 3 is executed if the condition does not hold (Fig. 1 (d)).

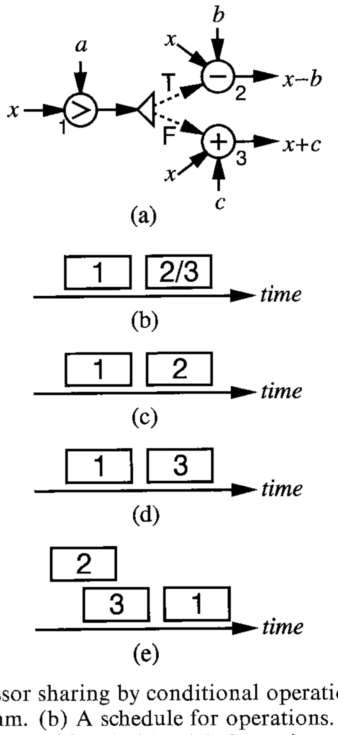


Fig. 1 Processor sharing by conditional operations. (a) A processing algorithm. (b) A schedule for operations. (c) Operations executed if the condition holds. (d) Operations executed if the condition does not hold. (e) Another schedule where conditional operations are executed before the decision operation.

In the rest of this paper, it is called *two operations share a processor* or *processor sharing* if these two operations are mutually exclusive and it is possible to schedule these operations to an identical time step on an identical processor. By processor sharing, we need only one processor for these conditional operations. The required number of processors to implement a processing algorithm can be reduced.

It must be noted that processor sharing is possible only if the conditional operations are executed after the condition is resolved. Hence, in order that the conditional operations share a processor, conditional operations must be scheduled after the completion of the decision operation which resolves the condition. For example, in the schedule shown in Fig. 1 (b), conditional operations 2 and 3 are scheduled after the decision operation 1. In this case, either operation 2 or operation 3 is executed and a processor is shared.

It also must be noted that if processor sharing is not necessary, the conditional operations can be executed before the decision operation. For example, Fig. 1 (e) is another schedule for a processing algorithm in Fig. 1 (a). In this schedule, both operation 2 and operation 3 are executed and one of the results is selected based on the condition resolved by the decision operation 1. This is possible since there are no data dependencies and therefore no precedence constraints from the decision operation to the conditional operations. The drawback is that we have to execute both conditional operations and may need more processors. However, a schedule with shorter iteration period could be derived. In addition, the number of processors could be as minimum as the schedule with processor sharing if processors are not fully utilized.

To minimize the required number of processors, processor sharing is tried in the proposed scheduling method. Moreover, executing conditional operations on both the true and false sides before a decision operation is used in our scheduling method if it is necessary to achieve the specified iteration period.

The conditions could be treated as ordinary computational results. Hence, similar to ordinary computational data, conditions are transfered from one processor to processors requiring the conditions, and/or stored in registers for controlling later operations in both the current iteration and later iterations. Of course, the data paths and registers should be optimized to 1 bit width, since a condition can be represented by only 1 bit.

3. Data-Flow Graph

A processing algorithm is represented by a data-flow graph (DFG) as shown in Fig. 2. The processing algorithm is periodically repeated on a series of input signals. In DFGs, data dependencies between operations are mostly considered and therefore input signals to the processing algorithm from outside and constants may often be omitted.

A DFG consists of nodes and edges. A circle node implies an operation and its functionality is shown as a symbol in the circle. A thin solid edge represents a data dependency from the tail node to the head node. This edge is called a *data dependency edge.* A data dependency edge is denoted as (i, j) where i is the tail node and j the head node. For example, a data dependency edge between nodes 1 and 2 in Fig. 2 is denoted as $(1, 2)$. A data dependency edge may have any number of delays on it. The delays on edges imply data dependencies over iteration cycles. If the number of delays is d on a data dependency edge (i, j), then the execution of operation j depends on the result of operation i in the d-th previous iteration cycle.

A triangle node implies a beginning of conditional branch and it is called a *branch node.* Each branch node has one incoming data dependency edge and two outgoing dashed edges. Dashed edges are *branch edges*

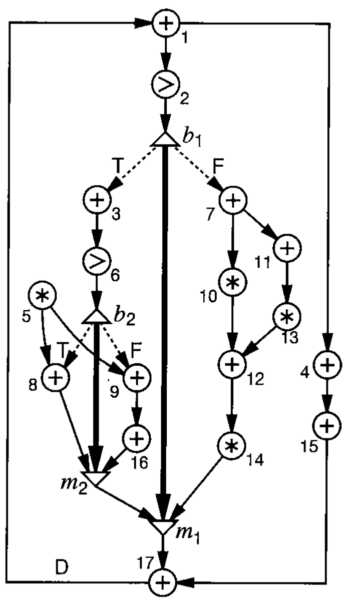


Fig. 2 An example data-flow graph.

and imply the operations adjoint to the branch edges are conditionally executed. The operation immediately preceding the branch node by the data dependency edge decides which branch edge is taken. This operation is called the *decision operation* of the branch. For example, in Fig. 2, operation 2 is the decision operation of branch b_1 and if the condition is true, the operations 3, 6, and so on are executed, and if the condition is false, the operations 7, 10, 11, and so on are executed. It must be noted that a branch edge does not imply data dependencies from the branch node to the conditional operations.

When the conditional part of the processing algorithm is completed, the conditional operation flows merge into a single operation flow. This is represented by a *merge node* which is symbolized as a reverse triangle in DFGs. The correspondence between a branch and a merger is represented by a *branch-merge edge,* which is shown as a thick solid edge in a DFG. For example in Fig. 2, two data flows which branch at the branch node b_2 merge when either conditional operation 8 or conditional operation 16 is completed. It is represented by the merge node m_2. A branch-merge edge is introduced only to clarify the correspondence between a branch node and a merge node. A branch-merge edge does not imply any data dependencies but it implicitly constrains precedence from a decision operation to operations after a merge node.

4. Overlapped Schedule and Precedence Constraints

4.1 Overlapped Schedule

A *directed path* of a data-flow graph is defined as a series of connected and non-repeated data dependency edges. The length of a directed path is the sum of execution times of operations on the path. Among all the directed paths which does not contain edges with delays, there exists one with the longest length. That directed path is called the *critical path.*

In the case of a non-overlapped schedule of a processing algorithm, all the operations in an iteration are scheduled to finish in the specified iteration period. Therefore, the iteration period cannot be shorter than the length of the critical path in a processing algorithm.

On the other hand, in an overlapped schedule, not all the operations in an iteration finish in the iteration period T and those operations not finish in the iteration are executed in parallel to the operations in the subsequent iterations. Therefore, unlike to the non-overlapped schedule, the lower bound of the iteration period is not limited by the critical path length.

The lower bound of the iteration period, or the iteration lower bound, inherent to a processing algorithm is limited not by the critical path but by the critical cycle[6]–[8]. In the case that the critical path length is longer than the iteration lower bound, any schedule in

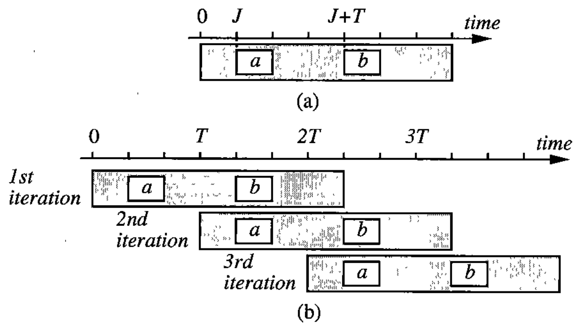


Fig. 3 Overlap of iterations. (a) A schedule longer than the T. (b) Repetition of the schedule with the iteration period T.

the non-overlapped manner cannot achieve the iteration lower bound. Only the overlapped schedule can always achieve the iteration lower bound. The technique to compute the iteration lower bound for a given processing algorithm can be found in [9]–[12].

Figure 3 (a) shows a schedule of some processing algorithm. The duration to execute every operation in the processing algorithm once is longer than the iteration period T. Operations a and b are scheduled to start at time steps J and $J+T$, respectively. The schedule is iteratively executed with the iteration period T. The first iteration starts at time step 0 and the second iteration starts at time step T. In general, n-th iteration starts at time step $(n-1)T$. Since the duration of an iteration is longer than T, iterations are executed so as to overlap with each other as shown in Fig. 3 (b).

In the n-th iteration, operation b is executed at time step $(n-1)T+J+T=nT+J$. In the next iteration, i.e., $(n+1)$-th iteration, operation a is executed at time step $nT+J$. Therefore, operation b in the current iteration and operation a in the next iteration are executed in parallel.

Consequently, operations scheduled to start at time steps $J, J+T, \ldots, J+nT, \ldots$ are executed concurrently in the overlapped schedule with the iteration period T. Let time steps be divided into *time classes.* Each time step, t, belongs to a time class denoted by $t - \lfloor \frac{t}{T} \rfloor T$[†], or t mod T. Hence, time steps $J, J+T, \ldots, J+nT, \ldots$ belong to the time class J. Operations in the same time class are executed concurrently in the overlapped schedule.

4.2 Precedence Constraint

A data dependency edge (i,j) from operation i to operation j implies that the execution of operation j requires the result of operation i. To achieve this data dependency, the start of operation j must be after the completion of operation i. Consequently, a data dependency edge (i,j) imposes a precedence constraint among executions of operations.

Let t_i denote the time step at which operation i starts in a schedule. If a data dependency edge (i,j) exists between operation nodes i and j, or a branch-merge edge (i,j) exists between a branch node i and a merge node j, the start time steps t_i and t_j must satisfy

$$t_j \geqq t_i + Q_i - D_{ij}T_0, \qquad (1)$$

where Q_i is the execution time of node i, D_{ij} the number of delays on the edge (i,j), and T_0 the specified iteration period. In the case of a branch node or a merge node, the execution time $Q_i = 0$.

The operation node immediately preceding a branch node is the decision operation which resolves the condition. Therefore, the start time step of the branch node can be as early as the completion time step of the decision operation. Let d_b denote the decision operation immediately preceding the branch node b. If a branch-merge edge (b,m) exists, then

$$t_b \geqq t_{d_b} + Q_{d_b}, \qquad (2)$$

$$t_m \geqq t_b - D_{bm}T_0. \qquad (3)$$

Therefore, the following precedence constraint must be satisfied;

$$t_m \geqq t_{d_b} + Q_{d_b} - D_{bm}T_0. \qquad (4)$$

For example, in Fig. 2, a branch-merge edge (b_1, m_1) exists and the operation node immediately preceding the branch node b_1 is operation 2. Therefore,

$$t_{m_1} \geqq t_2 + Q_2 \qquad (5)$$

must be satisfied.

As mentioned in Sect. 2, conditional operations k and l can share a processor only when their start time steps are not earlier than the completion of the decision operation. In our scheduling method, it is assumed that the conditional operations can share a processor even if the execution times are different. It is also assumed that the operation execution cannot be interrupted once the execution is initiated. By taking these assumptions into account, the precedence constraint

$$\min\{t_k, t_l\} \geqq t_{d_b} + Q_{d_b} \qquad (6)$$

must be satisfied as shown in Fig. 4 if conditional operations k and l are to share a processor.

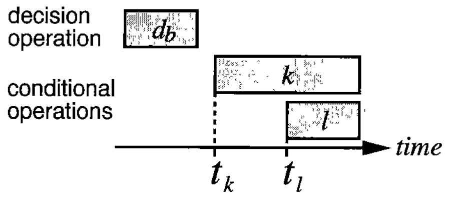


Fig. 4 Precedence between conditional operations and the corresponding decision operation.

† $\lfloor x \rfloor$ represents the largest integer less than or equal to x.

On the other hand, also as mentioned in Sect. 2, the precedence constraint (6) needs not be satisfied if conditional operations k and l do not share a processor.

5. Range-Chart-Guided Scheduling Method

Scheduling is to assign a start time step to each of the operations so as not to violate any precedence constraint. In order to execute operations as scheduled, we need as many processors as the maximum of required number of processors over all the time classes. Hence, in order to minimize the number of processors, we choose start time steps of operations so that the maximum number of concurrent operations is minimized.

The range-chart-guided scheduling method [5] is proposed as a heuristic scheduling method to minimize the number of processors for unconditional processing algorithms. Our proposed scheduling method is based on the range-chart-guided (referred as RCG in the remaining of this paper) scheduling method. In this section we briefly review RCG scheduling method.

The RCG scheduling method is summarized as follows.

1. Choose the reference operation and fix it to time step 0.
2. Compute scheduling ranges.
3. Select an operation.
4. Choose a start time step for the selected operation.
5. If all the operations are scheduled, then goto 6. Otherwise, goto 2.
6. Allocate operations to processors.

5.1 Scheduling Range

The scheduling range of an operation is the set of time steps at which the operation can start without violating the precedence constraints described by inequality (1). In order to minimize the number of processors required to execute a schedule, the time step should be selected within the scheduling range so that the number of concurrently executed operations is minimized. Therefore, the larger the scheduling ranges, the more possible to lead to the optimal schedule.

The lower bound of the scheduling range can be determined as *as soon as possible* (ASAP) schedule. This is because, in ASAP schedule, each operation starts as soon as its preceding operations finish and therefore the start time step is the earliest which satisfies precedence constraint. Similarly, the upper bound of the scheduling range can be determined as *as late as possible* (ALAP) schedule.

By choosing one operation in the DFG as the *reference operation*, the ASAP schedule can be obtained as the length of the longest path from the reference operation to other operations, where the weight of an edge (i, j) is $Q_i - D_{ij}T$ and the longest path length to the reference operation itself is defined as 0. On the other hand, the ALAP schedule of an operation can be obtained as the length of the shortest path from the same reference operation to other operations, where the weight of an edge (i, j) is $-Q_i + D_{ij}T$, the directions of the edges are reversed, and the shortest path length to the reference operation itself is defined as 0.

Each time an operation is scheduled, scheduling ranges for other operations must be recomputed.

5.2 Operation Selection

Among operations not yet scheduled, an operation is selected and a time step within its scheduling range is assigned to the operation. In other words, the time step at which the operation is executed is fixed. By fixing the time step for the operation, the scheduling ranges of other nodes are reduced in general. For example, if the time step is the lower bound of the scheduling range, then the scheduling ranges of the operations preceding the fixed operation would be reduced from the upper bound.

If an operation with the minimum scheduling range is selected and is fixed, the reduction of scheduling ranges of other operations would be minimum and the possibility of processor sharing could be kept as large as possible.

If more than one operations are the candidates to be selected, an operation is selected based on the following criteria in order of appearance: (1) the scheduling range is the smallest; (2) either the upper bound or the lower bound of the scheduling range is the *fixed bound.* The upper (lower) bound of the scheduling range of an operation is said *fixed* if all the operations immediately succeeding (preceding) the operation are already scheduled.

5.3 Start Time Assignment

For the selected operation, a time step is chosen within the operation's scheduling range and it is assigned as the start time step. Let P_J denote the number of concurrently executed operations in time class $J \in [0, 1, \ldots, T-1]$. In each time class J, we need as many processors as P_J. Therefore the required number of processors, $\hat{P}$, to execute a schedule is the maximum of P_J over all the time classes $J = 0, 1, \ldots, T-1$.

By assigning the selected operation to a start time step in time class J, the required number of processors is increased from P_j to $P'_j = P_j + 1$ for time class $j = J, J+1, \ldots, J+Q-1$ where Q is the execution time of the selected operation. Therefore, the time step is chosen for the selected operation where $\max P'_j$ for $j = J, J+1, \ldots, J+Q-1$ is minimized so that $\max P'_J$

does not exceed $\hat{P}$. If more than one time classes are with the minimum P_j', or there is no time step without increasing $\hat{P}$, then the time class closer to the fixed bound is chosen.

5.4 Processor Allocation

In start time assignment, start time steps of operations are determined to minimize the maximum of concurrently executed operations in time classes. To achieve the minimum number of processors to execute the schedule, it must be determined that on which particular processor each operation is executed. This process is called *processor allocation.*

First, give an index to each processors. If a number of P processors are used, the indices would be 0 to $P-1$. Then, from not yet allocated operations, choose one with the longest execution time and allocate it to the processor with the smallest index among processors which can execute the operation. This is repeated until all the operations are allocated.

6. Scheduling Method for Conditional Operations

In order to achieve processor sharing by conditional operations, a conditional operation must be scheduled in the same time step as the mutually exclusive operation. Therefore during the search for a start time step of a conditional operation within the scheduling range, each time step is checked if a mutually exclusive conditional operation is scheduled in the time step. If such a time step is found, then the conditional operation is scheduled in the time step so as to share a processor. In this way, processor sharing by conditional operations is efficiently included in RCG scheduling method.

6.1 Scheduling Range

The scheduling range of an operation is determined just the same as in the RCG scheduling method. As mentioned in 4.2 branch edges impose precedence constraints described by inequality (6) only when mutually exclusive operations have been scheduled to share a processor. Otherwise, branch edges are ignored in determining the scheduling ranges.

6.2 Scheduling Range for Operation Selection

In RCG scheduling, an operation with the smallest scheduling range is selected among operations not yet scheduled and the selected operation is scheduled to start at a time step within the scheduling range. However, the order of operations selected in this manner is not always suitable to fully utilize processor sharing.

Figure 5 shows scheduling ranges of a decision operation and a conditional operation. A grey rectangle is the execution time of operation and it can start at

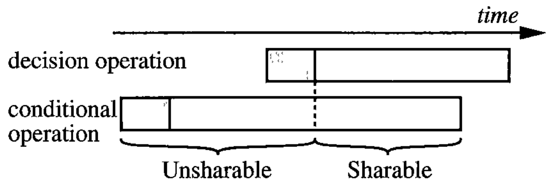


Fig. 5 Dividing the scheduling range of a conditional operation into sharable and unsharable scheduling ranges.

any time step in the scheduling range which is represented by a white rectangle. Let a scheduling range of a conditional operation be divided into *sharable scheduling range* and *unsharable scheduling range* as shown in Fig. 5. If the conditional operation is scheduled at a time step in the unsharable scheduling range, the conditional operation is executed before the decision operation. Hence processor sharing is impossible. On the other hand, if the conditional operation is scheduled in a time step in the sharable scheduling range, then a processor could be shared if the decision operation would be scheduled in an earlier time step than the conditional operation.

If a decision operation is selected before the conditional operations and is assigned to a late time step, then the sharable scheduling range becomes small and hence the possibility of processor sharing becomes small. Therefore, to maximize the possibility of processor sharing, it is preferable that conditional operations are first assigned to start time steps and then the decision operation is assigned to such a start time step that is earlier than the conditional operations.

In order that conditional operations are selected prior to decision operations in RCG scheduling, scheduling range for operation selection is introduced here. For a conditional operation with the sharable scheduling range, the scheduling range for operation selection is the same as the sharable scheduling range. For any other conditional operations without sharable scheduling ranges and unconditional operations, the scheduling range for operation selection is the same as the original scheduling range. For example, in Fig. 6 (a), operation a is a decision operation and operations b, c, d are the conditional operations. Figure 6 (b) shows scheduling ranges of the operations. It is assumed that the lower bound of the decision operation a is time step 2 and operation e has been scheduled at time step 5. The conditional operations c and d have sharable scheduling ranges as shown in Fig. 6 (c). Furthermore, by considering precedence constraints, scheduling ranges for operation selection is derived as shown in Fig. 6 (d).

Now conditional operations have smaller scheduling ranges for operation selection than the decision operation. Thus conditional operations are selected prior to the decision operation. Of course there can be the case that scheduling range of decision operation is

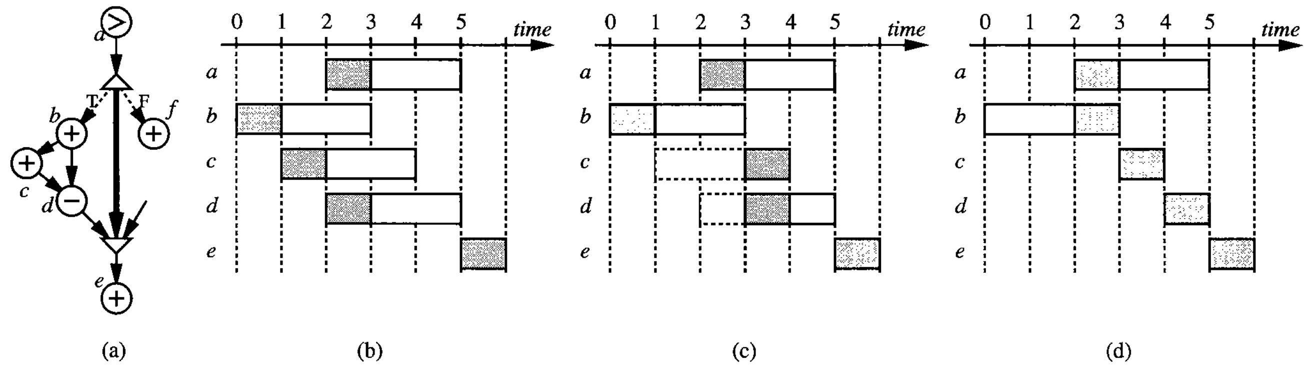


Fig. 6 Scheduling range for operation selection. (a) A sample DFG. (b) Ordinary scheduling range. (c) The lower bound of scheduling range for processor sharing. (d) The lower bound of scheduling range by precedence constraints.

smaller than the conditional operations. In that case the decision operation is selected prior to the conditional operations.

Scheduling ranges for operation selection are computed as follows. At first, compute the scheduling ranges and unmark all the operations. Then, do a depth first search, starting from already scheduled operation, on the DFG where the directions of edges are reversed.

If a not yet scheduled conditional operation c is unmarked, and has the sharable scheduling range, then mark c, span an imaginary edge from the corresponding branch node to c, and recompute the scheduling ranges. The imaginary edge implies only the precedence constraint from the branch node to the conditional operation. Resumed the depth first search from the beginning. When all the unscheduled conditional operations are marked, the recomputed scheduling ranges are the scheduling ranges for operation selection.

6.3 Operation Selection

In order to maximize processor sharing by conditional operations, we modify the criteria to select an operation which is not yet scheduled and to be assigned a start time step.

The primary criterion is that it is a conditional operation and the mutually exclusive operation has already been scheduled. If such a conditional operation is selected and could be scheduled in the same start time step as the already scheduled mutually exclusive operation, a processor is shared and the number of explicitly concurrent operations is not increased.

If two or more conditional operations satisfy the primary criterion, select an operation based on the following secondary criteria in order of appearance: (1) the scheduling range for operation selection is the smallest; (2) the upper bound of the scheduling range is the latest; (3) either the upper bound or the lower bound of the scheduling range is the fixed bound. If the selected operation is assigned to a late start time step, then the reduction of scheduling ranges of the preceding operations would be small. Hence the criterion (2) is introduced.

If there are no conditional operations which satisfy the primary criterion, then select an operation among conditional and unconditional operations according to the criteria (1), (2), and (3) described above.

6.4 Start Time Assignment

In the case that the selected operation is an unconditional operation, the start time is assigned just the same way as described in 5.3.

In the case that the selected operation is a conditional operation, it is checked whether the mutually exclusive operation is already scheduled in each time class within the scheduling range. If there is such a time class, then the number of processors in the time class would not be increased since a processor can be shared by the selected operation and the mutually exclusive operation. Hence the selected operation is assigned to the time class to share a processor. Otherwise, the number of processors in the time class must be increased by one just like the unconditional operations.

6.5 Reference Operation

In our proposed scheduling method, the selected operation is scheduled in the time step close to the fixed bound. If an operation immediately succeeding a merge node is chosen as the reference operation, then the last operation in the conditional flow has the fixed upper bound. Then the operation would be scheduled as late as possible and hence the scheduling ranges of other operations are not much reduced. Consequently, the operation immediately succeeding the merge node of the outer most conditional branch is chosen as the reference operation.

If there exist two or more such operations, then choose one with the latest upper bound as a reference operation and execute the scheduling method, or repeat the scheduling method by assuming each operation as a reference operation and choose the best schedule.

6.6 Algorithm Summary

Processor allocation for conditional operations is just the same as described in 5.4 by treating mutually exclusive operations as a single operation.

The scheduling method is summarized as follows.

1. Choose the reference operation and fix it to time step 0.
2. Compute scheduling ranges.
3. Compute scheduling ranges for operation selection.
4. Select an operation by the criteria in Sect. 6.3.
5. Choose a start time step for the selected operation.
6. If all the operations are scheduled, then goto 7. Otherwise, goto 2.
7. Allocate operations to processors.

6.7 Computational Complexity

The computational complexity of the proposed scheduling method is as follows. In the following, n denotes the number of operations, e the number of edges, b the number of conditions, p the number of computational resources.

At first, it is registered for each operation that if the operation depends on a condition, and if so, then on which side of the condition, the true side or the false side. This is performed by a depth first search and the computational complexity is $O(e)$.

1. *Computing scheduling ranges.* Scheduling ranges are computed as the longest path lengths. Hence the computational complexity is $O(ne)$.
2. *Computing scheduling ranges for operation selection.* The sharable scheduling range is identified by comparing the lower bound of a decision operation and the upper bound of the conditional operation. This is done in $O(nb)$ time about all the conditional operations. Then the longest path lengths are computed again for each time an imaginary edge is added. The total computational complexity is $O(nb+n^2e) = O(n^2e)$ (it is assumed that $b < ne$).
3. *Selecting operation to be scheduled.* For each of unscheduled operations (at most n), the possibility of processor sharing with already scheduled operations (at most n) is checked. It imposes the checking the mutually exclusiveness and the checking if the start time is included in the scheduling range. operation. The mutually exclusiveness can be checked in $O(b)$. Hence the computational complexity is $O(n^2b)$.
4. *Start time assignment.* For the selected operation, the mutually exclusiveness of already scheduled operation in every time class within the scheduling range is checked. The number of already scheduled operation is at most n and the mutually exclusiveness can be checked in $O(b)$. The number of time classes is the same as the iteration period T. Hence the computational complexity is $O(nbT)$.

The above steps from 1 to 4 are repeated until all the operations are scheduled.

The computational complexity of processor allocation is $O(nTp)$ [5]. Consequently the total computational complexity is $O(e + n(ne + n^2e + n^2b + nbT) + nTp) = O(n^3e+n^2bT+nTp)$ (it is assumed that $b < e$).

If the scheduling method is repeated by assuming each operation which immediately succeeds a merge node as a reference operation and choose the best schedule, the computational complexity would be $O(n^3be + n^2b^2T + nbTp)$.

7. Scheduling Results

The proposed scheduling method is implemented by using C programming language. Figure 7 shows the derived schedule for the DFG in Fig. 2 with the iteration period $T = 8$ units of time (u.t.). It is assumed that all types of operations are executed on a processor where a multiplication takes 2 u.t. and an addition and a comparison take 1 u.t. Two processors, $P1$ and $P2$, are necessary and are the minimum. Figure 7 (a)

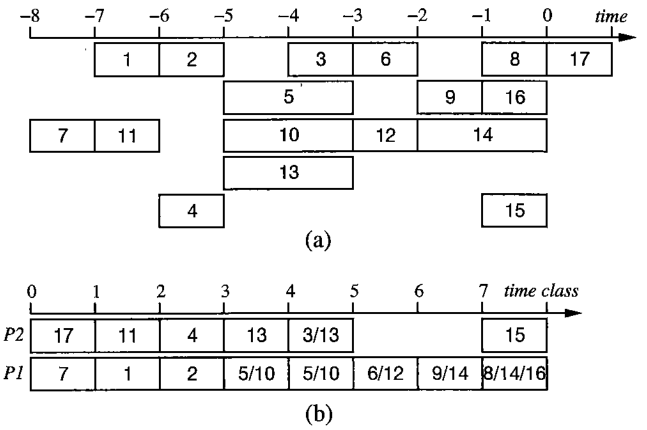


Fig. 7 A schedule for the DFG in Fig. 2. (a) The start time steps. (b) Processor allocation.

Table 1 Scheduling result of the DFG in Fig. 2.

Method	T	Result	CPU
		Processor	[sec]
Ours	**9**	**2**	0.332
	8	**2**	0.328
	7	**3**	0.299
	6	**3**	0.267
	5	**4**	0.309
	4	**5**	0.285

shows the start time steps of operations and Figure 7 (b) shows the processor allocation. Operation 17 is chosen as the reference operation and hence it is scheduled in time step 0. In this schedule, the mutually exclusive operations 5 and 10 are assigned to the identical time step -5 and share processor $P1$. Although the execution times of operations 3 and 13 are different, these operations are mutually exclusive and therefore share processor $P2$. Moreover, in order to realize the specified iteration period, mutually exclusive operations 7 and 11 are assigned to earlier time steps than the decision operation 2 and therefore both operations 7 and 11 are executed unconditionally. In time class 7, mutually exclusive operations 8 and 16 share processor $P1$. In addition, either operation 8 or operation 16 also shares the processor with the mutually exclusive operation 14. Hence more than two operation can share a processor.

The iteration lower bound of the DFG is 4 u.t. For the iteration periods smaller than 9 u.t., the proposed scheduling method derives a schedule with the minimum number of processors. The results are shown in Table 1.

When the iteration period is 6 u.t., we must execute operations 3, 5, 7, 10, 11, 12, 13 before the decision operation 2 to satisfy precedence constraints. Hence these operations are always executed regardless of the condition b_1. In addition, if b_1 is False, we must execute operations 1, 2, 4, 14, 15, and 17. The total execution time of these operations is 17 u.t. Therefore, to execute these operations with the iteration period of 6 u.t., we need at least $\lceil \frac{17}{6} \rceil$[†] $= 3$ processors. The proposed scheduling method achieves this lower bound (defined as *computational processor bound* in [5]) and therefore is optimal. Also for any other examples, it can be shown that the scheduling result achieves the processor bound.

The proposed scheduling method is compared with the existing scheduling methods in [2] and [4]. Tables 2 and 3 show the comparison for DFGs in [3] and [1], respectively. In this case, the functional units, such as adder, subtracter, and comparator, are distinguished. In these tables, BDD denotes the method in [2] and Match denotes the method in [4]. Tables show: the name of scheduling method; the specified iteration period T; the numbers of functional units; and the CPU time in seconds to execute the scheduling method on a 70 MHz Sparc workstation.

For the same iteration period, all three scheduling methods derive schedules with the minimum number of functional units. In addition, our proposed scheduling method can derive optimal schedules for the smaller iteration periods by considering overlapped scheduling.

Table 2 Scheduling result for the DFG in [3].

Method	T	Result			CPU
		Add	Sub	Cmp	[sec]
BDD	6	2	1	1	NA
Match	7	2	1	1	NA
	6	2	1	1	NA
Ours	**8**	**1**	**1**	**1**	0.668
	7	**2**	**1**	**1**	0.610
	6	**2**	**1**	**1**	0.563
	5	**2**	**1**	**1**	0.552
	4	**2**	**1**	**1**	0.490
	3	**3**	**2**	**1**	0.503
	2	**4**	**2**	**1**	0.497
	1	**8**	**4**	**2**	0.514

Table 3 Scheduling result for the DFG in [1].

Method	T	Result		CPU
		Add	Sub	[sec]
BDD	4	2	2	NA
Match	4	2	2	NA
Ours	**6**	**1**	**1**	0.371
	5	**2**	**2**	0.343
	4	**2**	**2**	0.336
	3	**2**	**2**	0.342
	2	**3**	**3**	0.331
	1	**6**	**6**	0.321

8. Conclusions

In this paper, a scheduling method for an iterative processing algorithm with conditional operations was proposed. The proposed scheduling method is the time-constrained scheduling and minimizes the required number of processors or functional units by means of resource sharing by conditional operations. By considering overlapped scheduling, the proposed scheduling method derives a schedule for the specified iteration period which can be as small as the iteration lower bound of the processing algorithm.

Scheduling results show that the proposed scheduling method can derive the same optimal schedules as the existing scheduling methods with respect to the number of computational resources. In addition, the proposed method can produce optimal schedules for small iteration periods which have not been handled by the other existing scheduling methods.

Acknowledgement

This work is supported partly by CAD21 research body at Tokyo Institute of Technology.

[†] $\lceil x \rceil$ represents the least integer greater than or equal to x.

References

[1] A.C. Parker, J.T. Pizarro, and M. Mlinar, "MAHA: A program for datapath synthesis," Proc. 23rd DAC, pp.461–446, 1986.

[2] A. Yamada, T. Yamazaki, N. Ishiura, T. Kambe, and I. Shirakawa, "A data-path scheduling method with conditional branches," IEICE 7th Karuizawa Workshop on Circuits & Syst., pp.49–54, 1993.

[3] T. Kim, N. Yonezawa, and J.W.S. Liu, "A scheduling algorithm for conditional resource sharing—A hierarchical reduction approach," IEEE Trans. Comput.-Aided Design, vol.13, pp.425–438, April 1994.

[4] H. Ishiwata, N. Togawa, M. Sato, and T. Ohtsuki, "A time-constrained scheduling algorithm for CDFG with conditional branches," IEICE Technical Report, VLD95-133, 1996.

[5] S.M. Heemstra de Groot, S.H. Gerez, and O.E. Herrmann, "Range-chart-guided iterative data-flow graph scheduling," IEEE Trans. Circuits Syst.-I: Fund. Theory & Appl., vol.39, pp.351–364, May 1992.

[6] M. Renfors and Y. Neuvo, "The maximum sampling rate of digital filters under hardware speed constraints," IEEE Trans. Circuits Syst., vol.CAS-28, pp.196–202, March 1981.

[7] D.A. Schwartz and T.P. Barnwell, III, "A graph theoretic technique for the generation of systolic implementations for shift invariant flow graphs," Proc. of the 1984 IEEE ICASSP, San Diego, CA, March 1984.

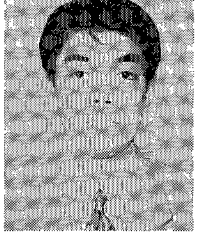
[8] K.K. Parhi and D.G. Messerschmitt, "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding," IEEE Trans. Computers, vol.40, pp.178–195, Feb. 1991.

[9] K. Ito and K.K. Parhi, "Determining the minimum iteration period of an algorithm," Journal of VLSI Signal Processing, vol.11, pp.229–244, Dec. 1995.

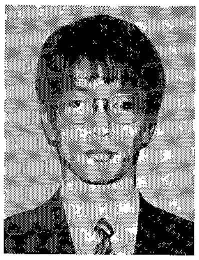
[10] D.Y. Chao and D.Y. Wang, "Iteration bounds of single-rate data flow graphs for concurrent processing," IEEE Trans. Circuits Syst.-I, vol.40, pp.629–634, Sept. 1993.

[11] R. Govindarajan and G.R. Gao, "A novel framework for multi-rate scheduling in DSP applications," Proc. 1993 Int. Conf. Application-Specific Array Processors, pp.77–88, IEEE Computer Society Press, 1993.

[12] S.H. Gerez, S.M. Heemstra de Groot, and O.E. Herrmann, "A polynomial-time algorithm for the computation of the iteration-period bound in recursive data-flow graphs," IEEE Trans. Circuits Syst.-I, vol.39, pp.49–52, Jan. 1992.



Kazuhito Ito received the BS, the MS, and the Ph.D. degrees in Electrical Engineering from Tokyo Institute of Technology, Tokyo (Japan), in 1987, 1989, and 1992, respectively. He was with the Tokyo Institute of Technology from 1992 and visited the University of Minnesota from April 1993 to July 1994. He is now an associate professor of the Department of Electrical and Electronic Systems, Saitama University, Urawa (Japan). His research interests include high-level synthesis in digital signal processing, VLSI signal processing, and asynchronous systems.



Tatsuya Kawasaki was born in Tokyo, Japan on July 28, 1973. He received the B.E. degree in Electrical and Electronic Engineering from Saitama University, Japan, in 1996. He is currently a graduate student at Saitama University. His research interests include high-level synthesis and scheduling for digital signal processing.