IPSJ Transactions on System LSI Design Methodology Vol.6 60–70 (Feb. 2013)

[DOI: 10.2197/ipsjtsldm.6.60]

Regular Paper

A Method to Reduce Energy Consumption of Conditional Operations with Execution Probabilities

Kazuhito Ito[1,a) Kazuhiko Kameda[1,†1]

Received: May 25, 2012, Revised: August 31, 2012,
Accepted: October 30, 2012, Released: February 15, 2013

Abstract: In conditional processing, operations are executed conditionally based on the result of condition operations. While the speculative execution of conditional operations achieves higher processing speed, unnecessary energy may be consumed by the speculatively executed operations. In this paper, reduction of the energy consumption of conditional processing is considered for time and resource constrained processing. An efficient method to calculate the probability of operation execution is presented. Based on the probabilities of execution, a scheduling exploration with the simulated annealing and a heuristic scheduling algorithm are proposed to minimize the energy consumption of the conditional processing by reducing unnecessary speculative operations. The experimental results show 5% to 10% energy can be reduced by the proposed methods for the same configuration of resources.

Keywords: conditional operation, speculative execution, scheduling, low energy

1. Introduction

In general processing algorithms, the execution of operations are controlled by the results of other operation(s) [1]. In **Fig. 1** (a), the operation *A*, for example, is a comparison between two data. If the comparison result is *TRUE*(T), the operation *B* is executed. Otherwise, i.e., the comparison result is *FALSE*(F), the operations *C*, *D*, and *E* are executed. The operation *A* in Fig. 1 (a) is called a *condition operation*, and the operations *B* to *E* are called *condition dependent operations*. The processing algorithm with condition operations and condition dependent operations is called *processing with conditional operations* or *conditional processing*.

There are data dependencies and conditional dependencies among operations. In Fig. 1 (a), the operation *D* uses the result of the operation *C*. This is a data dependency of *D* on *C* and it imposes the precedence constraint such that *C* must be executed before the execution of *D*. Figure 1 (b) shows a possible time chart of the execution of operations by assuming each operation takes one control step to execute. By the way, the condition dependency does not impose the precedence constraint. For example, *C* can start at the same time as *A* because no data dependency exists between *A* and *C* [2]. In that case, *C* is executed unconditionally about *A* and the result of *C* is discarded when *A* is completed to be T. Let it be said *A* is *resolved* as T. Such execution of condition dependent operation before the condition is resolved is called a *speculative execution* of operations [3]. By the speculative execution, the processing of Fig. 1 (a) can be completed within the smaller number of control steps as shown in Fig. 1 (c).

There have been much research on the high-level synthesis

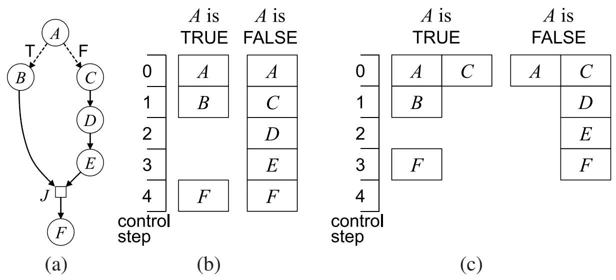

Fig. 1 An example of processing with conditional operations. (a) CDFG. (b) A time chart of the execution of operations without speculation. (c) Execution with speculation.

of conditional processing [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]. The basic scheduling algorithms are proposed in Refs. [1], [4], [5]. The speculative execution of operations is considered in Refs. [2], [3], [6], [7], [8], [9], [10], [11]. The objective in these work is to minimize the required number of functional units by conditional resource sharing, or to minimize the necessary control steps to execute the processing by resource sharing and/or speculative execution. In Ref. [12], the energy consumption of conditional processing is evaluated in the context of optimizing the average control steps to execute the processing probabilistically based on the conditional branches. However the speculative execution is not taken into account in Ref. [12]. Some commercial synthesis tools accept descriptions in high-level programming languages, such as C and C++, and support conditional operations [13], [14]. Though these tools perform many kinds of optimization, minimizing energy consumption with respect to the speculative execution is not mentioned.

There also have been much research on power- or energy-aware speculation in processors [15], [16], [17], [18], [19]. In Ref. [15], improvement of the speed performance is the target and hence the energy is increased in most cases. Reference [16] pro-

1 Saitama University, 255 Shimookubo, Saitama 338–8570, Japan
†1 Presently with Honda Motor Co., Ltd.
a) kazuhito@ees.saitama-u.ac.jp

poses a branch prediction method to reduced unnecessary power consumption. In Refs. [17] and [18], the prediction of branch and speculating thread is considered to reduce the energy consumption at the cost of latency degradation. In Ref. [19] a special data driven architecture is targeted. These techniques are difficult to apply to minimizing the energy consumption of operations with the constraints of time and resources.

The resource constrained list scheduling [20] is an efficient scheduling method and is extended to take conditional resource sharing into account [11]. In this scheduling, the operation k is executed as soon as (1) the preceding operations on which k is data dependent are executed and (2) a resource is available either unconditionally or conditionally sharable with other operation(s). This results in unnecessary speculative execution of operations and much energy will be consumed for operations whose results would be eventually discarded when conditions are resolved. Hence it is important to reduce unnecessary speculative executions to minimize the energy consumption.

In this paper, a high-level synthesis method is proposed to minimize the energy consumption of conditional processing with time and resource constraints. To precisely evaluate the energy consumption of conditional operations, a technique to efficiently calculate the probability of operation execution is proposed. Then two scheduling methods, one is by exploration and the other is by heuristics, are presented.

This paper is organized as follows. A motivating example is given in Section 2. A hardware model is presented in Section 3. Section 4 describes the technique to calculate the probabilities of execution of operations, and the scheduling methods are proposed in this section. The experimental results are given in Section 5. Section 6 concludes the paper.

2. Motivating Example

The conditional processing is given by the control data flow graph (CDFG) as shown in **Fig. 2**. The node represents an operation. A data dependency and a conditional dependency are represented by a solid arrow and a broken arrow, respectively. A character 'T' (TRUE) or 'F' (FALSE) near a broken arrow indicates which branch is taken depending on the condition. Where the branches join, a join node is placed. It is shown by a box in Fig. 2. A join node is used only to indicate the control flow and hence consumes no time and no energy. In addition, the join node J in Fig. 2 indicates that the condition among the results of C, D, and G is selected according to A and B.

The representative examples of condition operation are a comparison and an overflow detection. A comparison can be implemented by a subtraction. An overflow detection can be taken from the result of an addition. Therefore, the condition operations are considered in the same way as the operations in scheduling and evaluation of energy consumption.

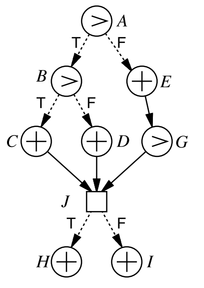

Fig. 2 An example of CDFG.

It is important to note that there are two kinds of probabilities in the conditional processing. One is the probability of branch (PB), which is the statistical probability of a condition resolved as either T or F. Let $PB_{m,\mathsf{T}}$ and $PB_{m,\mathsf{F}}$ denote the PB of condition operation m to be resolved as T and F, respectively. The PBs depend on the given processing algorithm and the input data. Thus, the PBs are profiled over the input data space in advance and are assumed to be known when the scheduling is considered. The other is the probability of execution (PE) of operations. Let $PE(n)$ denote the PE of operation n. While $PB_{m,\mathsf{T}}$ and $PB_{m,\mathsf{F}}$ are constant for the given processing, PEs vary depending on the schedule of operation execution.

Let $EC(n)$ denote the energy consumed by executing operation n. Then the energy consumption EC of the processing is calculated as

$$EC = \sum_{n \in N} PE(n)EC(n) \tag{1}$$

where N is the set of operations in the processing.

For simplicity, it is assumed that an operation takes one control step (CS) and $EC(n) = 1$ unit of energy (u.e.) for all $n \in N$. Without the speculative execution, 4 CSs are necessary as shown in **Fig. 3** (a). Assuming the PBs as Case I shown in **Table 1**, the PEs are $PE(A) = 1$, $PE(B) = 0.8$, and $PE(D) = 0.08$. This is because A is always executed, B is executed when A is resolved to branch to B with the probability of 0.8, and D is executed when both A and B are resolved to branch to D with the probability of $0.8 \cdot 0.1 = 0.08$. PEs of other operations can be derived similarly, and the total of consumed energy is 4.00 u.e. as shown in **Table 2**. The detail of calculating the PE is described in Section 4.2.

Schedules with 3 CSs are possible by the speculative execution. In the schedule shown in Fig. 3 (b), operation B is speculatively executed before the condition operation A is resolved. In

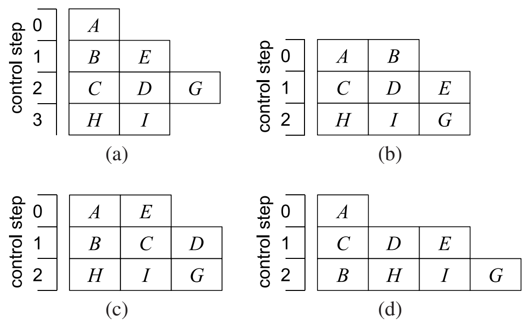

Fig. 3 Schedules of the CDFG shown in Fig. 2. (a) Without speculative execution. (b)(c)(d) With speculative execution.

Table 1 Probabilities of condition to branch.

	Case I	Case II
A	$PB_{A,\mathsf{T}} = 0.8$, $PB_{A,\mathsf{F}} = 0.2$	$PB_{A,\mathsf{T}} = 0.2$, $PB_{A,\mathsf{F}} = 0.8$
B	$PB_{B,\mathsf{T}} = 0.9$, $PB_{B,\mathsf{F}} = 0.1$	
C	$PB_{C,\mathsf{T}} = 0.6$, $PB_{C,\mathsf{F}} = 0.4$	
D	$PB_{D,\mathsf{T}} = 0.9$, $PB_{D,\mathsf{F}} = 0.1$	
G	$PB_{G,\mathsf{T}} = 0.7$, $PB_{H,\mathsf{F}} = 0.3$	

Table 2 Energy consumption of schedules in Fig. 3.

	Case I			Case II		
Schedule	(a)	(b)	(c)	(a)	(b)	(c)
A	1.000	1.000	1.000	1.000	1.000	1.000
B	0.800	1.000	0.800	0.200	1.000	0.200
C	0.720	0.720	0.800	0.180	0.180	0.200
D	0.080	0.080	0.080	0.020	0.020	0.020
E	0.200	0.200	1.000	0.800	0.800	1.000
G	0.200	0.200	0.200	0.800	0.800	0.800
H	0.644	0.704	0.652	0.686	0.926	0.688
I	0.356	0.496	0.428	0.314	0.874	0.332
Total	4.000	4.400	4.960	4.000	5.600	4.240

Fig. 3 (c), operation C is speculatively executed before the condition operation B is resolved. Operations H and I are also speculatively executed in these schedules. In Fig. 3 (b), $PE(B) = 1$ since B is always executed. In Fig. 3 (c), $PE(C) = 0.8$ since C must be executed when A is resolved as T with $PB_{A,\mathsf{T}} = 0.8$. The totals of consumed energy are 4.40 u.e. and 4.96 u.e. for Fig. 3 (b) and (c), respectively.

When PBs are given as Case II in Table 1, where the PB of A is changed, the totals of consumed energy for the schedules of Fig. 3 (b) and (c) are 5.60 u.e. and 4.24 u.e., respectively. Here the schedule of Fig. 3 (b) is preferable in Case I because the schedule achieves less energy consumption than the schedule of Fig. 3 (c). On the other hand in Case II, the schedule of Fig. 3 (c) is preferable. This example suggests that (1) the schedule to achieve the least energy consumption varies depending on the given PBs and (2) precisely calculating the PEs for the operation execution schedule is the key to minimize the energy consumption.

3. Hardware Model

The hardware model of the datapath is shown in **Fig. 4**. A functional unit FU_i ($1 \le i \le K$) receives data from either registers Reg_j ($1 \le j \le M$) or the external input(s). Multiplexors are used to select the source of the data. The FU has transparent latches (TLAT) at the input. A TLAT passes the input data to the output when enabled and keeps the output when disabled. Thus the FU works and consumes dynamic energy only when an operation is requested by enabling the TLAT. The FU does not have any output register and the operation result is stored in one of the registers Reg_j.

Figure 5 (a) shows a schedule of the CDFG shown in Fig. 1 (a). The operations C and D are executed speculatively before the condition A is resolved. The operation result of C is produced in CS 0, stored in a register at the end of CS 0 (at the beginning of CS 1), used by the operation D in CS 1, and the data becomes unnecessary at the end of CS 1. The *lifetime* of the data is represented with an arrow as shown in Fig. 5 (b). The speculative operation D is executed in CS 1. The condition A is also executed in CS 1 and if it resolves to TRUE, the data produced by D is unnecessary and needs not to be stored into a register. Because A is TRUE, the operation B is executed in CS 2 and the operation F is executed with the data produced by B. On the other hand, if A resolves to FALSE, the data produced by D is stored in a register, used by the operation E, and the operation F is executed with the data produced by E.

Consequently, the data produced by the operations specula-

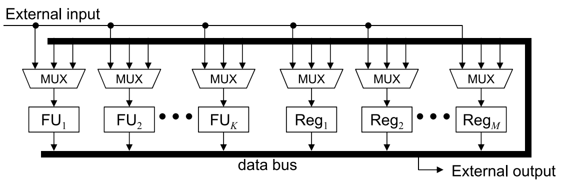

Fig. 4 A hardware model.

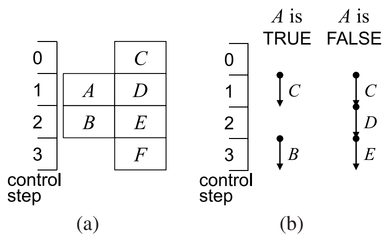

Fig. 5 Conditional operation execution and the lifetime of data.

tively executed before the condition (e.g., the operation C before A) are always stored into registers. If the condition resolves at the same time of speculative operations and the condition says the data are not necessary (e.g., the operation D when A is TRUE), the data are not stored into registers. This saves energy consumption of registers by avoiding unnecessarily altering the data in the registers.

4. Scheduling for Energy Minimization

4.1 Preliminary

The CDFG is given as (N, E), where N is the set of operations and E is the set of edges. The set E is divided into two subsets E_D and E_C. E_D is the set of the data dependency edges and E_C is the set of condition dependency edges. While a data dependency edge (n, m) imposes the precedence of n over m, a condition dependency edge does not impose any precedence. The set N is divided into two subsets N_T and N_D. N_T is the set of operations which have no incoming data dependency edge. N_D is the set of remaining operations, i.e., the operations with at least one incoming data dependency edge. The join node is included in N_D. Let $N_C \in N$ denote the set of condition operations. For example in the CDFG shown in Fig. 2, N_D consists of operation G and the join node J, and N_T consists of the remaining operations. N_C consists of A, B, C, D, and G.

4.2 Calculating Probability of Execution

Let the notation $x \Rightarrow \mathsf{T}$ ($x \Rightarrow \mathsf{F}$) denote that the condition x is resolved as TRUE (FALSE).

For the example CDFG shown in Fig. 2 and the statistical probabilities of branch given as Case I in Table 1, the condition B is executed at the statistical probability $PB_{A,\mathsf{T}} = 0.8$, and the operation C is executed at $PB_{B,\mathsf{T}} = 0.9$. Therefore, if both A and B are resolved before C, $PE(C)$ is $PB_{A,\mathsf{T}} \times PB_{B,\mathsf{T}} = 0.72$. If A is not yet resolved when C is executed, $PE(C) = 1 \times PB_{B,\mathsf{T}} = 0.9$. If B is not resolved, $PE(C) = PB_{A,\mathsf{T}} \times 1 = 0.8$. If both A and B are not resolved when C is executed, $PE(C) = 1$.

Consider calculating $PE(H)$ for the same CDFG. When a schedule shown in Fig. 3 (a) is given, all the conditions on which H depends are resolved before H. Then $PE(H)$ is calculated as the sum of the probabilities ($A \Rightarrow \mathsf{T}$, $B \Rightarrow \mathsf{T}$, $C \Rightarrow \mathsf{T}$), ($A \Rightarrow \mathsf{T}$,

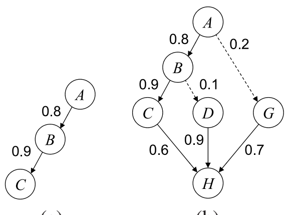

Fig. 6 CDPGs of operations in Fig. 2. (a) For the target operation C. (b) For the target operation H.

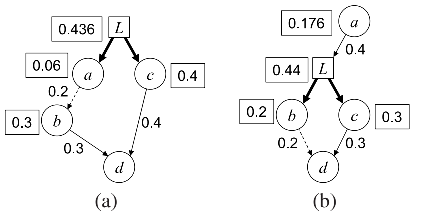

Fig. 7 CDPGs for target d. (a) For the condition $\{(a \Rightarrow \mathsf{F}, b \Rightarrow \mathsf{T})$ *or* $c \Rightarrow \mathsf{T}$. (b) For the condition $\{a \Rightarrow \mathsf{T}, (b \Rightarrow \mathsf{F}$ *or* $c \Rightarrow \mathsf{T})\}$.

$B \Rightarrow \mathsf{F}, D \Rightarrow \mathsf{T})$, and $(A \Rightarrow \mathsf{F}, G \Rightarrow \mathsf{T})$. Therefore,

$$\begin{aligned} PE(H) &= PB_{A,\mathsf{T}} \cdot PB_{B,\mathsf{T}} \cdot PB_{C,\mathsf{T}} + PB_{A,\mathsf{T}} \cdot PB_{B,\mathsf{F}} \cdot PB_{D,\mathsf{T}} \\ &\quad + PB_{A,\mathsf{F}} \cdot PB_{G,\mathsf{T}} \\ &= 0.8 \cdot 0.9 \cdot 0.6 + 0.8 \cdot 0.1 \cdot 0.9 + 0.2 \cdot 0.7 \\ &= 0.644. \end{aligned} \tag{2}$$

Figure 3 (d) shows another schedule of the CDFG where H is executed speculatively before the conditions B and G. If $A \Rightarrow \mathsf{T}$, the result of either C or D is selected by B and used as the condition to execute H. It implies that H must be executed to prepare to the case that C or D would be resolved as T. A straightforward method to calculate the $PE(H)$ is to enumerate the possible combinations of resolution of conditions. Hence $PE(H)$ for Fig. 3 (d) can be calculated as the sum of the probabilities $(A \Rightarrow \mathsf{T}, C \Rightarrow \mathsf{T}, D \Rightarrow \mathsf{T})$, $(A \Rightarrow \mathsf{T}, C \Rightarrow \mathsf{T}, D \Rightarrow \mathsf{F})$, $(A \Rightarrow \mathsf{T}, C \Rightarrow \mathsf{F}, D \Rightarrow \mathsf{T})$, and $(A \Rightarrow \mathsf{F})$, and therefore

$$\begin{aligned} PE(H) &= 0.8(0.6 \cdot 0.9 + 0.6 \cdot 0.1 + 0.4 \cdot 0.9) + 0.2 \\ &= 0.968. \end{aligned} \tag{3}$$

The drawback of this method is that the number of combinations of conditions grows exponentially to $|N_C|$ and thus requires many calculations.

A method to efficiently calculate PEs of operations for the given PBs and schedule is considered here. For each operation to calculate the PE, the *condition dependency graph* (CDPG) is constructed. A CDPG is a directed graph given as (V, E_R). V is the set of nodes representing the operation to calculate the PE (target), the condition operations on which the target depends conditionally, and the necessary OR nodes. The OR node is explained later. E_R is the set of edges which represent the conditional dependencies among the nodes in V. **Figure 6** (a) shows the CDPG for the target operation C of the example CDFG shown in Fig. 2. The target C depends on the condition operations A and B, and the condition to execute C is $A \Rightarrow \mathsf{T}$ *and* $B \Rightarrow \mathsf{T}$. The *and* relation is denoted as $\{A \Rightarrow \mathsf{T}, B \Rightarrow \mathsf{T}\}$. V includes A, B, and C, and the edges (A, B) and (B, C) are included in E_R. The *and* combination of conditions is represented with a series of edges connecting the conditions one after another as shown in Fig. 6 (a). The order of the conditions in the series of edges is arbitrary and does not affect the PEs. The edge (u, v) drawn as a solid (dashed) arrow means that v is executed when $u \Rightarrow \mathsf{T}$ $(u \Rightarrow \mathsf{F})$. Shown near the edge (u, v) is the PB of the condition u.

The condition to execute the target H in Fig. 2 is $\{A \Rightarrow \mathsf{T}, B \Rightarrow \mathsf{T}, C \Rightarrow \mathsf{T}\}$ *or* $\{A \Rightarrow \mathsf{T}, B \Rightarrow \mathsf{F}, D \Rightarrow \mathsf{T}\}$ *or* $\{A \Rightarrow \mathsf{F}, G \Rightarrow \mathsf{T}\}$. Here, '*and*' terms are combined by '*or*' relations. The CDPG is constructed as shown in Fig. 6 (b). The *or* relation is represented as multiple incoming edges to node(s).

In the case of H mentioned above, the conditions A and B are commonly included in the *and* terms and hence V includes only the target and the condition operations. If *and* terms do not include any common condition, then OR nodes are introduced. Suppose that the condition to execute a target d is $\{a \Rightarrow \mathsf{F}, b \Rightarrow \mathsf{T}\}$ *or* $c \Rightarrow \mathsf{T}$. In the CDPG, $\{a \Rightarrow \mathsf{F}, b \Rightarrow \mathsf{T}\}$ is denoted by the edge (a, b). Again, the order of a and b is arbitrary. As shown in **Fig. 7** (a), the *or* relation is denoted by an OR node L and the bold edges (L, a) and (L, c). If the condition to execute d is $\{a \Rightarrow \mathsf{T}, (b \Rightarrow \mathsf{F}$ *or* $c \Rightarrow \mathsf{T})\}$, the CDPG is constructed as shown in Fig. 7 (b). An OR node L is used to denote the *or* relation between b and c, and the edge (a, L) denotes the *and* relation.

There exists a node with no incoming edge in the CDPG created as mentioned above, and is called the *root* of the CDPG. The root is A in both CDPGs in Fig. 6. The roots are the OR node and a for CDPGs in Fig. 7 (a) and (b), respectively.

The $PE(d)$ of the target d for Fig. 7 (a) is calculated as follows. Here it is assumed that all the conditions are resolved before the execution of d. Starting from the target operation d, the CDPG is traversed in the opposite direction of arrows in the breadth first manner. Let $P_{x,y}$ denote the PE of the operation y with respect to the condition x. By definition, $P_{y,y} = 1$. $P_{b,d} = 0.3$ and it is shown in a box near the condition b in Fig. 7 (a). Similarly, $P_{c,d} = 0.4$. $P_{a,d}$ is calculated as $P_{a,d} = P_{a,b}P_{b,d} = 0.06$. Now $PE(d)$ is calculated as the sum of $P_{c,d} = 0.4$ (i.e., $c \Rightarrow \mathsf{T}$) and $(1 - P_{c,d})P_{a,d} = 0.036$ (i.e., $c \Rightarrow \mathsf{F}$, $a \Rightarrow \mathsf{F}$, and $b \Rightarrow \mathsf{T}$). Thus, $PE(d) = P_{L,d} = 0.436$.

As can be seen from this example, the calculation of PEs becomes complicated when the *or* relation is involved. It is noted that the execution of the operation d is not needed when both c and the combination of a and b do not request the execution of d. Here the notion of *probability of no execution* (PNE) is introduced. Let $Q_{x,y}$ denote the PNE of operation y with respect to the condition x. The relation $Q_{x,y} = 1 - P_{x,y}$ holds. For the case of Fig. 7 (a), $Q_{L,d}$ is calculated simply as $Q_{L,d} = Q_{L,c}Q_{L,a} = 0.6 \times 0.94 = 0.564$ and $PE(d) = P_{L,d} = 1 - Q_{L,d} = 0.436$. This is consistent with the result shown above.

Here a method to calculate the PNE is presented. The calculation of the PNE starts from the target operation y and ascends the CDPG to the root. **Figure 8** shows possible condition dependencies in calculating PNE. In Fig. 8 (a), y depends on only one side of x (T side in this case). If x is resolved before y, y is not executed at the probability of $Q_{m,y}$ when $x \Rightarrow \mathsf{T}$, and y is never executed when $x \Rightarrow \mathsf{F}$. If x is not resolved, $Q_{x,y} = Q_{m,y}$. Thus PNE $Q_{x,y}$ is calculated as

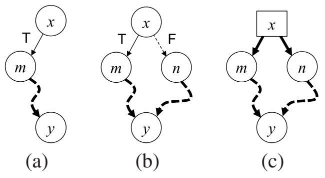

Fig. 8 Examples of condition dependencies. (a) Simple dependency. (b) Dependency branching and merging. (c) An OR node.

```
** Calculation of $PE(y)$ **
Input: CDPG $(V, E_R)$ with the root $r$
Output: $PE(y) = 1 - Q_{r,y}$

Unmark all the edges $(x, u) \in E_R$
$Q_{y,y} \leftarrow 1$
**for each** incoming edge $(x, y) \in E_R$ **do**
  Mark $(x, y)$
  **call** CalcPE( $x$ )
**end do**

**procedure** CalcPE( $x$ )
**if** an unmarked $(x, u) \in E_R$ exists **return**
(here, all the edges outgoing from $x$ are marked)
**if** $x$ is an OR node **then**
  Calculate $Q_{x,y} = \prod_{(x,m)\in E_R} Q_{m,y}$
**else** ($x$ is a condition operation)
  **if** only $(x, m) \in E_R$ is outgoing from $x$ **then**
      Calculate $Q_{x,y}$ as Eq. (4)
  **else** (two edges $(x, m)$ and $(x, n)$ exist)
      Calculate $Q_{x,y}$ as Eq. (5)
  **end if**
**end if**
**for each** incoming edge $(w, x) \in E_R$ **do**
  Mark $(w, x)$
  **call** CalcPE( $w$ )
**end do**
**return**
```

Fig. 9 Algorithm for calculating the PE.

$$Q_{x,y} = \begin{cases} PB_{x,\mathsf{T}}Q_{m,y} + (1 - PB_{x,\mathsf{T}}) & \text{if } x \text{ is resolved} \\ Q_{m,y} & \text{if } x \text{ is not resolved} \end{cases} \tag{4}$$

In Fig. 8 (b), y depends on both sides of x. Then $Q_{x,y}$ is calculated as

$$Q_{x,y} = \begin{cases} PB_{x,\mathsf{T}}Q_{m,y} + PB_{x,\mathsf{F}}Q_{n,y} & \text{if } x \text{ is resolved} \\ Q_{m,y}Q_{n,y} & \text{if } x \text{ is not resolved} \end{cases} \tag{5}$$

In Eq. (5), $Q_{x,y} = Q_{m,y}Q_{n,y}$ if x is not resolved, because y needs not be executed only when both m to y and n to y suggest the execution of y is not needed. If x is an OR node (Fig. 8 (c)), $Q_{x,y}$ is calculated as the product of $Q_{m,y}$ for all the outgoing edges (x, m). The algorighm to calculate the PE is shown in **Fig. 9**.

When the PBs are given as Case I in Table 1, $PB_{B,\mathsf{T}} = P_{B,C} = 0.9$, $PB_{B,\mathsf{F}} = P_{B,D} = 0.1$, $PB_{A,\mathsf{T}} = P_{A,B} = 0.8$, and $PB_{A,\mathsf{F}} = P_{A,G} = 0.2$, $PE(H)$ for Fig. 3 (a) and Fig. 3 (d) are calculated as shown in **Fig. 10**. In Fig. 10 (b), $Q_{G,H} = 0$, that is $P_{G,H} = 1$, because H must always be executed if $A \Rightarrow \mathsf{F}$ to prepare to the case of $G \Rightarrow \mathsf{T}$. The resultant $PE(H)$s are respectively identical to the results of Eq. (2) and Eq. (3).

The computational complexity of the method to calculate PE for an operation is $O(|N_C| + |E_C|)$ where E_C is the number of conditional dependencies in the CDFG. Since the conditions usually branch in two way (T or F), $|E_C| = 2|N_C|$. Therefore the computational complexity of calculating PEs for all the operations is $O(|N||N_C|)$. Consequently the above method is much efficient than enumerating the possible combination of conditions.

$$\begin{aligned}
Q_{C,H} &= 1 - 0.6 = 0.4 \\
Q_{D,H} &= 1 - 0.9 = 0.1 \\
Q_{B,H} &= P_{B,C} \cdot Q_{C,H} + P_{B,D} \cdot Q_{D,H} \\
&= 0.9 \cdot 0.4 + 0.1 \cdot 0.1 = 0.37 \\
Q_{G,H} &= 1 - 0.7 = 0.3 \\
Q_{A,H} &= P_{A,B} \cdot Q_{B,H} + P_{A,G} \cdot Q_{C,H} \\
&= 0.8 \cdot 0.37 + 0.2 \cdot 0.3 \\
&= 0.356 \\
PE(H) &= P_{A,H} = 1 - Q_{A,H} = 0.644
\end{aligned}$$

(a)

$$\begin{aligned}
Q_{C,H} &= 1 - 0.6 = 0.4 \\
Q_{D,H} &= 1 - 0.9 = 0.1 \\
Q_{B,H} &= Q_{C,H} \cdot Q_{D,H} \\
&= 0.4 \cdot 0.1 = 0.04 \\
Q_{G,H} &= 0 \\
Q_{A,H} &= P_{A,B} \cdot Q_{B,H} + P_{A,G} \cdot Q_{G,H} \\
&= 0.8 \cdot 0.04 + 0.2 \cdot 0 \\
&= 0.032 \\
PE(H) &= P_{A,H} = 1 - Q_{A,H} = 0.968
\end{aligned}$$

(b)

Fig. 10 Calculation of $PE(H)$. (a) For the schedule in Fig. 3 (a) where all the conditions are resolved. (b) For the schedule in Fig. 3 (d) where B and G are not resolved before H.

4.3 Exploration of Schedule to Reduce Unnecessary Speculative Execution

In the schedule obtained by the list scheduling, operations are executed as soon as possible (ASAP) if resources are available either conditionally or unconditionally [11]. Even when there is a time margin for an operation to wait for the depending condition being resolved, the operation is scheduled at the earliest control step where the speculative execution is possible. The better schedule which reduces such unnecessary speculative execution would be different from the ASAP schedule. A method to explore the schedule space is proposed to find a good schedule with the optimized cost function [21]. In that method, some time durations, called *strut*, are added to data dependency edges of CDFG to delay the execution of operations, and then the start control step (SCS) of each operation is determined by ASAP scheduling. The different combination of struts results in the different schedule of the CDFG. Hence the optimized combination of struts is explored which leads to a schedule with the optimized cost function.

The scheduling algorithm is shown in **Fig. 11** where the simulated annealing (SA) is used for the exploration. The initial schedule is the same as the result of the resouce constrained list scheduling. FT is the set of functional unit types, such as adder and multiplier. At most RC_f functional units can be used for the type $f \in FT$. Since an operation in N_T has no incoming data dependency edge, the operation can start at any control step as long as no precedence relation is violated among succeeding operations. The allowed maximum control step MCS is larger than or equal to the smallest control step obtained by the list scheduling with the given resource constraints. Therefore the scheduling problem is feasible, that is, at least one schedule exists which satisfies both the time and resource constraints.

The changes (a), (b), and (c) to generate a new scheduling candidate in step 4 are the same as ones described in Ref. [21]. These changes manipulate the values of struts. The change (d) is newly introduced for the conditional processing where the SCS of an operation in N_T is moved to an earlier or later control step.

Given a schedule satisfying all the precednce relations, the new schedule derived by one of these changes also satisfies the precedence. While the precedence is maintained after the changes, more resources than the limit might be needed to execute operations as scheduled since the ASAP scheduling does not take the resource constraint into account. Only accepting the change satisfying the resource constraint is one of the possible solutions. However it would reduce the chance for the exploration to escape from the local optima. Let $R_f(t)$ denote the number of required

```
Scheduling exploration for energy minimization of conditional processing
Input:   CDFG=$(N, E)$
         Probabilities of condition to branch
         Energy consumption of operations in $N$
         Resource constraint $RC_f$ for $f \in FT$
         Maximum control step $MCS$
Output:  Schedule with minimized energy consumption
(1) Set the strut $S_{ij} = 0$ for every data dependency edge $(i, j) \in E_D$. Set the
    SCS to the resouce constraind list scheduling result for every operation
    $j \in N_T$.
(2) Determine SCS for every operation in $N_D$ by ASAP scheduling method.
    This is the first scheduling candidate Sched and evaluate the initial ob-
    jective cost of Sched, $C(Sched)$.
(3) Set the temperature $T = T_{start}$.
(4) Generate a new scheduling candidate by performing one of the follow-
    ing changes to derive the new strut values $S'_{ij}$ of edges $(i, j) \in E_D$ or
    the new SCS of an operation in $N_T$ so that no precedence relation is
    violated.
    (a) increase or decrease a strut
    (b) move struts over single operation
    (c) rewind struts for a sub-tree of CDFG
    (d) move SCS of an operation in $N_T$
(5) Determine the SCS for every operation in $N_D$ by using the values of
    struts $S'_{ij}$ and ASAP scheduling. This is the new scheduling candidate
    Sched'.
(6) Evaluate the objective cost $C(Sched')$.
(7) Accept the new schedule candidate Sched' if $C(Sched') < C(Sched)$ or
    at the probability
        $prob = \exp\left(-\frac{C(Sched') - C(Sched)}{T}\right).$
(8) Repeat steps 4 to 7 for NI times where NI is a predetermined num-
    ber.
(9) Decrease the temperature $T$ by multiplying a constant $\alpha < 1$. If
    $T > T_{end}$, go to step 4. Otherwise end.
```

Fig. 11 The scheduling exploration algorithm.

functional units of the type f at control step t. $R_f(t)$ is calculated by considering conditional resource sharing with the method described in Ref. [11]. The resource penalty U_R is computed as

$$U_R = \sum_{f \in FT} \sum_{t=0}^{MCS-1} d(R_f(t) - RC_f) \qquad (6)$$

where the function $d(k) = k$ if $k \geq 0$ or $d(k) = 0$ otherwise. The cost $C(Sched)$ is defined as

$$C(Sched) = \beta_0 EC + \beta_1 U_R \qquad (7)$$

where EC is the energy consumption (Eq. (1)) and β_0 and β_1 are weighting factors. Minimizing $C(Sched)$ with a sufficiently large β_1 allows the exploration to temporarily exceed the resource limits and finally leads to the schedule where the energy consumption is minimized within the resource constraint.

4.4 A Heuristic Scheduling Algorithm

For a CDFG shown in **Fig. 12** with the resource constraint of one adder and one subtractor, the list schedule is obtained as shown in **Fig. 13** (a). The operations A3, A4, A8, S3, S6, and S8 are speculatively executed. PEs of operations for Fig. 13 (a) are shown in **Table 3** with all the PBs assumed to be 0.5. Generally, as an operation is assigned to later control step, more conditions are resolved and speculative executions are decreased. Thus the energy consumption by unnecessary speculative execution would be saved.

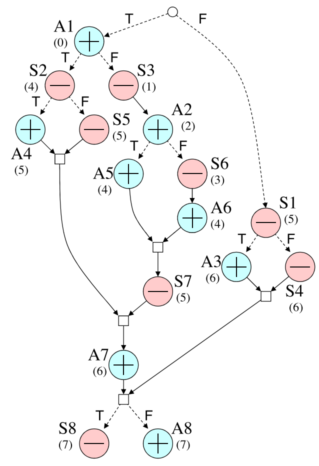

Fig. 12 The CDFG ‘MAHA’ [1].

control step					
0	A1	A3	S1	S3	
1	A2	A4	S2	S4	S6
2	A5	A6	S5		
3	A8		S7		
4	A7		S8		

(a)

control step							
0	A1				S3		
1	A2				S1	S2	S6
2	A3	A4	A5	A6	S4		
3	A8				S5	S7	
4	A7				S8		

(b)

Fig. 13 Schedules of MAHA for one adder and one subtractor. (a) Result of the list scheduling. (b) Result of the proposed method.

Table 3 Probabilities of execution in schedules.

Fig. 13 (a)				Fig. 13 (b)			
A1	0.500	S1	0.500	A1	0.500	S1	0.500
A2	0.250	S2	0.250	A2	0.250	S2	0.250
A3	0.500	S3	0.500	A3	**0.250**	S3	0.500
A4	0.250	S4	0.250	A4	**0.125**	S4	0.250
A5	0.125	S5	0.125	A5	0.125	S5	0.125
A6	0.125	S6	0.250	A6	0.125	S6	0.250
A7	0.500	S7	0.250	A7	0.500	S7	0.250
A8	0.750	S8	0.750	A8	0.750	S8	0.750

The proposed heuristic scheduling algorithm for low energy consumption is shown in **Fig. 14**. It refines the conventional resource constrained list schedule by rescheduling operations to later control steps so that the energy consumption is reduced. The priorities of operations are calculated as the result of as late as possible schedule with the assumption that the conditional dependencies as well as data dependency impose precedence constraints among operations. The priorities obtained for the CDFG shown in Fig. 12 are shown in parenthesis in the same figure. The larger value means the higher priority. In the algorithm, the set of operations with the same priority p, $S(p)$, is identified in the descending order of the priority. An operation $k \in S(p)$ is picked up one by one, and the SCS of k is searched from the current to the latest possible SCSs which most reduces the energy consumption without exceeding the resource limits. The satisfactory of the resource constraint is checked by calculating the required number of resources by considering conditional resource sharing [11]. The priority defined above ensures that, when operation k is picked up and rescheduling of k is performed, the CSs of operations succeeding k, either data dependently or conditional dependently, have been determined. This is necessary to precisely evaluate $\Delta EC(k, t)$, which is the energy reduction by rescheduling k to CS t. Among all the operations in $S(p)$, the operation ξ with the largest ΔEC at some CS t_ξ is selected, ξ is rescheduled to t_ξ, and ξ is removed from $S(p)$. If $S(p)$ is not empty, then repeat the selection of ξ again. If $S(p)$ becomes empty, then proceed to the next priority.

The allowed maximum control step MCS is larger than or equal to the smallest control step obtained by the list scheduling as mentioned in the previous section.

For example in the case of the CDFG shown in Fig. 12, the first $S(p)$ consists of S8 and A8. The SCS of S8 is not changed because S8 is already assigned to the latest possible CS. The SCS of A8 is also not changed because the resource constraint would be violated if A8 is moved to CS = 4. The second $S(p)$ consists of A3, S4, and A7. A3 can be moved to CS = 1, 2, or 4 (the resource constraint is not satisfied if A3 is moved to CS = 3), and the energy reductions are computed as $\Delta EC(\text{A3}, 1) = 1.539$, $\Delta EC(\text{A3}, 2) = 1.539$, and $\Delta EC(\text{A3}, 4) = -0.001$ (the energy consumption of operations are assumed as described in Section 5). S4 can be moved to CS = 2 or 3, and $\Delta EC(\text{S4}, 2) = 0$, $\Delta EC(\text{S4}, 3) = -0.769$. A7 is already at the last CS. Hence A3 is selected since $\Delta EC(\text{A3}, 2)$ is the largest and the CS 2 is the latest for A3. Then S4 is rescheduled to the CS 2. At the end of the algorithm, the schedule shown in Fig. 13 (b) is obtained. The speculative execution of A3 and A4 are eliminated. The PEs of

```
A time and resource constrained scheduling algorithm of conditional operations for energy minimization
Input:   CDFG=$(N, E)$
         Probabilities of condition to branch
         Energy consumption of operations in $N$
         Resource constraint $RC_f$ for $f \in FT$
         Maximum control step $MCS$
Output:  Schedule with minimized energy consumption
( 1 ) Calculate priorities of operations. Let $p$ be the highest priority.
( 2 ) Execute the conventional resource constrained list scheduling with
      speculative execution.
( 3 ) Identify the set $S(p)$ of operations with the priority $p$.
( 4 ) for $k \in S(p)$
          $t0 \leftarrow$ the current SCS of $k$
          $UB \leftarrow$ the latest possible SCS of $k$
( 5 )     for $t0 < t \leq UB$ without exceeding resource constraints
( 6 )         Calculate $\Delta EC(k, t)$ by assuming that the SCS of $k$ is $t$.
( 7 )     end for
( 8 )     Identify $t_k$ such that (1) $\Delta EC(k, t_k)$ is largest, and if more than one
          value of $t$ gives the largest $\Delta EC(k, t)$, then (2) $t_k$ is the latest.
( 9 ) end for
( 10 ) Identify $\xi \in S(p)$ with the largest $\Delta EC(\xi, t_\xi)$ and reschedule $\xi$ at the
       corresponding control step $t_\xi$ if $\Delta EC(\xi, t_\xi) \geq 0$. If $\Delta EC(\xi, t_\xi) < 0$ (the
       energy is increased), $\xi$ remains at $t0$.
( 11 ) Remove $\xi$ from $S(p)$. If $S(p)$ is not empty, go to Step 4. If $S(p)$ is
       empty, decrease $p$. If $p \geq 0$, go to Step 3. End otherwise.
```

Fig. 14 The heuristic scheduling algorithm.

operations in the schedule are shown in Table 3.

The computational complexity of calculating the priorities is $O(|N||E|)$. The cardinality $|S(p)|$ is at most N. For each operation $k \in S(p)$ in Step 4, the satisfaction of the resource constraint is checked and $\Delta EC(k, t)$ is computed for at most MCS control steps. The computational complexity of checking the satisfaction of the resource constraints is $O(RS)$ where RS is the number of resources. The computation of $\Delta EC(k, t)$ requires the complexity of $O(|N||N_C|)$. This is repeated for $|S(p)|$ times. The overall computational complexity of HA is $O(|N||E| + |N|^2 MCS(RS + |N||N_C|))$.

5. Experimental Results

The proposed method was implemented using C++ programming language and was run on a 2.66 GHz PC. The example CDFGs are MAHA (8 additions, 8 subtractions) [1] shown in Fig. 12, MAHA2 (16 additions, 16 subtractions) obtained by duplicating MAHA twice [2], MAHA5 (40 additions, 40 subtractions), MAHA10 (80 additions, 80 subtractions), MAHA25 (200 additions, 200 subtractions), and FIG17 (94 additions, 63 subtractions (including 15 comparisons), and 71 multiplications) [5] shown in **Fig. 15**, and QRS (7 additions, 19 subtractions (including 12 comparisons), and 17 multiplications) from HLSynth95 [22]. In the

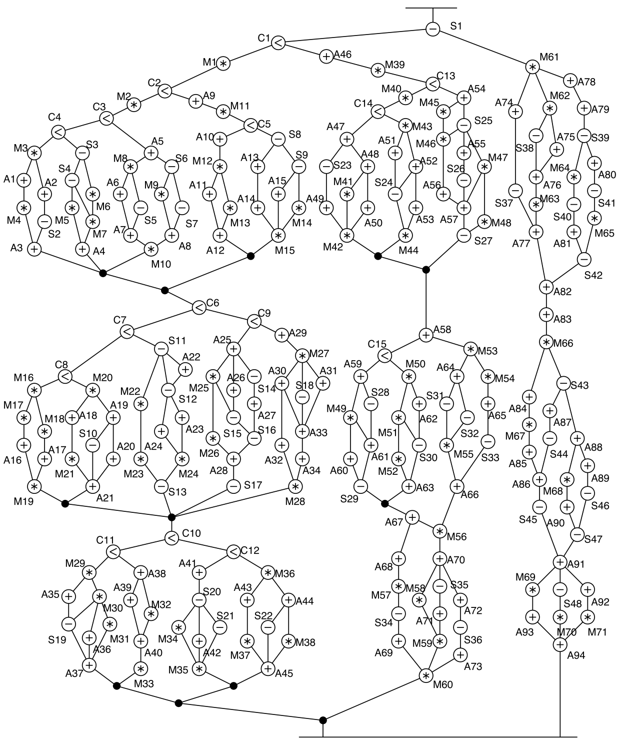

Fig. 15 CDFG 'FIG17'.

Table 4 Characteristics of functional units.

FU	CS	Energy [pJ]
Adder	1	3.3769
Subtractor	1	3.3887
Multiplier	2	47.955
Register	—	2.5932
2-to-1 Multiplexor	—	0.1854
Memory (Read)	2	155.60
Memory (Write)	1	206.71

original FIG17, there exist one 3-way branch condition and one 4-way branch condition. These are replaced respectively with the combination of 2 conditions and 3 conditions. MAHAue is the same CDFG as MAHA, but the PBs are set randomly. Other than MAHAue, the PBs are 0.5 for TRUE and 0.5 for FALSE. FIG1709 and FIG1701 are the same as FIG17 with the exception that the PB of the first condition C1 is 0.9 (0.1) for TRUE in FIG1709 (FIG1701).

Functional units of 16-bit integer are designed for a 0.18 μm CMOS process. The characteristics of the FUs are shown in **Table 4**. The execution of an addition or a subtraction takes one control step and the execution of a multiplication takes two control steps. The multiplier is pipelined into two stages. The aver-

Table 5 The parameters for SA.

CDFG	T_{start}	T_{end}	α	*NI*
MAHA25	1000	10	0.95	10000
others	1000	10	0.95	100000

Table 7 Scheduling results.

	RC				LS		HA		
CDFG	A	S	M	CS	*EC*	R	*EC*	R	CPU
FIG17	1	1	1	58	1771	8	1677 (94.7%)	8	6.34
				63	—		1644	6	6.89
	1	1	2	45	1851	7	1680 (90.7%)	8	6.17
				50	—		1650	8	6.56
	2	1	2	37	1903	11	1710 (89.9%)	7	5.95
				44	—		1661	8	6.41
FIG1709	1	1	1	58	1656	8	1523 (92.0%)	8	5.70
				63	—		1518	6	6.23
	1	1	2	45	1771	7	1547 (87.3%)	8	6.20
				50	—		1524	8	5.88
	2	1	2	37	1813	11	1599 (88.2%)	7	5.27
				44	—		1536	8	6.27
FIG1701	1	1	1	58	1885	8	1830 (97.1%)	8	6.39
				63	—		1769	6	6.27
	1	1	2	45	1931	7	1813 (93.9%)	8	6.19
				50	—		1776	8	5.79
	2	1	2	37	1992	11	1822 (91.5%)	7	5.99
				44	—		1787	8	6.42

Table 6 Scheduling results.

	RC				LS		SE			HA		
CDFG	A	S	M	CS	*EC*	R	*EC*	R	CPU	*EC*	R	CPU
MAHA	1	1	0	5	38.05	3	**35.74** (93.9%)	3	88	**35.74** (93.9%)	3	0.20
				6	—		**33.03**	3	93	**33.03**	3	0.21
				7	—		**31.86**	3	94	32.63	3	0.22
				8	—		**31.46**	3	95	**31.46**	3	0.23
	2	2	0	4	49.06	5	41.48 (84.5%)	4	91	**39.94** (81.4%)	4	0.22
				5	—		**37.60**	4	93	**37.60**	4	0.22
MAHAue	1	1	0	5	43.53	3	**41.04** (94.3%)	3	86	**41.04** (94.3%)	3	0.20
				6	—		**37.71**	3	91	**37.71**	3	0.22
				7	—		**37.20**	3	92	**37.20**	3	0.23
				8	—		**36.83**	3	93	**36.83**	3	0.24
	2	2	0	4	52.76	5	49.55 (93.9%)	4	92	**45.85** (86.9%)	4	0.22
				5	—		42.90	4	91	**42.89**	4	0.23
MAHA2	1	1	0	10	76.10	3	**71.49** (93.9%)	3	239	**71.49** (93.9%)	3	0.50
				12	—		**66.07**	3	248	68.38	3	0.55
				14	—		**63.72**	3	254	64.50	3	0.61
	2	2	0	7	98.56	5	84.99 (86.2%)	4	253	**81.05** (82.2%)	4	0.59
				9	—		**73.78**	4	266	75.99	4	0.73
MAHA5	1	1	0	25	190.3	3	**178.7** (93.9%)	3	1060	**178.7** (93.9%)	3	1.94
				27	—		177.0	3	1061	**175.6**	3	3.16
				29	—		174.0	3	1117	**171.7**	3	3.36
	2	2	0	16	247.1	5	**202.8** (82.1%)	4	1109	213.1 (86.2%)	4	2.88
				19	—		**192.7**	4	1176	208.7	4	4.37
MAHA10	1	1	0	50	380.5	3	**357.4** (93.9%)	3	4491	**357.4** (93.9%)	3	7.52
				52	—		374.3	3	4441	**354.3**	3	20.7
				55	—		378.5	3	4438	**350.0**	3	21.7
	2	2	0	31	494.6	5	489.5 (99.0%)	4	4569	**433.1** (87.6%)	4	15.4
				35	—		470.2	4	4758	**427.2**	4	27.0
MAHA25	1	1	0	125	951.3	3	901.3 (94.7%)	3	6296	**893.6** (93.9%)	3	72.0
				128	—		901.1	3	6299	**889.3**	3	415
				140	—		972.1	3	6316	**872.2**	3	493
	2	2	0	76	1237	5	1181 (95.5%)	4	6366	**1093** (88.4%)	4	335
				85	—		1279	4	6407	**1078**	4	596
				100	—		1270	4	6433	**1049**	4	747
QRS	1	1	1	32	682.3	11	661.0 (96.9%)	10	283	**654.9** (96.0%)	13	1.47
	1	1	2	17	651.5	8	**634.1** (97.3%)	13	262	**634.1** (97.3%)	11	0.96
	1	1	3	14	779.7	11	**641.2** (82.2%)	10	260	711.7 (91.3%)	11	0.80
				18	—		653.5	9	311	**650.4**	10	1.00
	1	2	3	12	669.5	8	**629.3** (94.0%)	8	317	637.1 (95.2%)	11	1.05
	2	1	3	14	791.6	11	**651.6** (82.8%)	9	258	722.1 (91.2%)	10	0.83
				15	—		**656.0**	9	314	701.1	10	0.88
				16	—		657.9	9	309	**654.9**	10	0.95

age energy consumptions are evaluated by circuit simulations. In the experiments, all the operations are assumed to consume these average energy. The energy consumption of a particular operation would be different from the average. In that case, it is possible that the energy consumptions of the operations are obtained by simulations of the processing in advance, and those energy consumptions of operations are used in minimizing the total energy consumptions.

The weighting factors $\beta_0 = 300$ and $\beta_1 = 1000$ in Eq. (7) are used where the value *EC* is expressed in the unit of pJ. In addition, the SA parameters shown in **Table 5** are used in the schedule exploration (SE). $T_{start} = 1000$ is selected so that the schedule exceeding the resource limits can be accepted at some probability in the early stage of the exploration.

Table 6 and **Table 7** summarize the results. Table 6 shows, from left to right, the CDFG, the resource constraints (RC) for adder RC_A (A), subtractor RC_S (S), for multiplier RC_M (M), the target number of control steps (CS), the energy consumption (*EC*) and the required number of registers (R) obtained with the conventional list scheduling with conditional resource sharing (LS) [11], [20], the *EC*, R, and CPU time of SE, and the *EC*, R, and CPU time of the heuristic algorithm (HA). The energy is shown in pJ and the CPU time in second for SE and in millisecond for HA. By analyzing the obtained schedule, the required number of registers is determined. It affects the configuration of the multiplexors and hence the energy consumption by the multiplexors. Since the LS assigns operations to the ASAP control step, the number of control steps and hence the energy consumption are uniquely determined with respect to the given CDFG and the resource constraint. On the other hand in the proposed methods, arbitrary control steps can be specified where operations are scheduled so as to minimize *EC*. For the smallest CS cases, the percentage of the energy against the LS is shown in parenthesis in Tables 6 and 7.

For MAHA and MAHAue, most results of SE and HA are identical. The energy reduction of 5% to 18% against LS is achieved. For other MAHAs and QRS, both SE and HA derives better schedules than LS. The result of the HA being worse than the SE means that the HA remains in suboptimal schedules. As the target CS increases, the results of SE become worse than the HA. This implies that the SE requires more CPU time in the exploration to find a good schedule. In the case of FIG17, FIG1709, and FIG1701, the SE could not derive the better result than the LS with the SA parameters shown in Table 5. Hence the results for the SE are omitted in Table 7. The CPU time was about 1,700 seconds for each run of the SE. The HA can derive the schedules with less energy consumption than the schedules obtained by LS

Table 8 Scheduling results with memory.

	RC				LS		HA		
CDFG	A	S	M	CS	*EC*	R	*EC*	R	CPU
MAHA	1	1	0	6	110.7	3	96.45 (87.2%)	3	0.23
				9	—		69.88	3	0.27
	2	2	0	5	112.0	3	96.21 (85.9%)	3	0.25
				6	—		93.88	3	0.25
MAHAue	1	1	0	6	122.2	3	102.7 (84.0%)	3	0.22
				9	—		74.74	3	0.27
	2	2	0	5	123.8	3	106.5 (86.1%)	3	0.25
				6	—		103.6	3	0.25
MAHA2	1	1	0	12	230.0	4	189.6 (82.5%)	4	0.58
				16	—		165.2	3	0.69
	2	2	0	10	230.0	5	194.7 (84.7%)	4	0.84
				12	—		182.9	4	0.86
MAHA5	1	1	0	30	610.7	6	469.2 (76.8%)	4	2.34
				34	—		444.8	4	4.61
	2	2	0	25	582.7	7	490.3 (84.2%)	4	6.03
				28	—		464.3	4	6.84
MAHA10	1	1	0	60	1238	6	935.2 (75.6%)	4	9.80
				65	—		911.1	4	34.1
	2	2	0	50	1256	7	982.9 (78.3%)	4	44.7
				54	—		956.1	4	49.5
MAHA25	1	1	0	150	3119	6	2333 (74.8%)	4	93.6
				170	—		2211	4	711
	2	2	0	125	3386	14	2461 (72.7%)	4	845
				135	—		2395	4	958
FIG17	1	1	1	60	2463	10	2287 (92.9%)	10	7.20
				65	—		2251	11	7.53
	1	1	2	46	2561	8	2352 (91.8%)	11	5.91
				52	—		2351	10	6.42
	2	1	2	39	2601	13	2400 (92.3%)	9	6.14
				46	—		2380	9	6.52
FIG1709	1	1	1	60	2305	10	2096 (90.9%)	10	8.47
				65	—		2092	11	8.83
	1	1	2	46	2468	8	2215 (89.8%)	11	7.20
				52	—		2180	10	7.66
	2	1	2	39	2506	13	2264 (90.3%)	9	6.31
				46	—		2239	9	6.52
FIG1701	1	1	1	60	2621	10	2479 (94.6%)	10	8.49
				65	—		2409	11	8.81
	1	1	2	46	2654	8	2489 (93.8%)	11	7.25
				52	—		2523	10	7.69
	2	1	2	39	2695	13	2535 (94.1%)	9	7.41
				46	—		2522	9	7.80
QRS	1	1	1	33	836.1	8	836.1 (100%)	8	1.84
	1	1	2	17	930.4	10	852.7 (91.6%)	12	1.23
	1	1	3	15	1021	13	931.0 (91.2%)	12	1.03
				19	—		844.4	11	1.37
	1	2	3	14	996.1	11	886.9 (89.0%)	12	1.34
	2	1	3	14	1018	13	937.7 (92.1%)	11	0.98
				15	—		916.8	11	1.11
				17	—		850.0	13	1.28

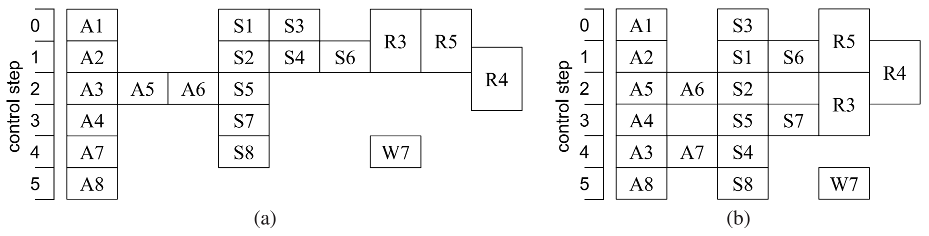

Fig. 16 The results of MAHA with memory for $RC_A = 1$, $RC_S = 1$, and 6 CSs. (a) LS. (b) HA.

for these larger CDFGs and longer CSs.

Consequently, the proposed SE can find a schedule with the minimized energy consumption for the case of shorter CSs and the proposed HA can find a schedule which is comparable to the one obtained by SE in short CPU time for larger CDFGs and longer CSs.

In some cases, more registers are required in the results of HA than LS. This is because the lifetime of data become long by putting operations to the later CS. These additional registers might be the source of increased static energy consumption in ultra deep submicron technologies.

The case with a work memory was also experimented. A 16-bit 256-word SRAM was assumed with the characteristics shown in Table 4. The read operation requires 2 CSs. An address is given to the SRAM in a CS and the corresponding SRAM content is output 2 CSs later. Before the content is output, another address for the next read operation can be given to the SRAM. In other words, the read operations are executed in a pipelined manner. The write operation requires 1 CS. In MAHA, it is assumed that the operations A3, A4, and A5 require an SRAM data (the read operations are denoted as R3, R4, and R5, respectively) and the result of S7 is written to the SRAM (denoted as W7). MAHA2, MAHA5, and so on are obtained by duplicating the MAHA twice, 5 times, and so on. Similarly in FIG17, FIG1709, and FIG1701, A5, A14, A24, . . ., A84, A93 [*1], and M3, M13, . . ., M63 require an SRAM data and the results of A10, A20, . . ., A90 are written to the SRAM. In QRS, 6 read operations and 3 write operations are added. The results are shown in **Table 8**. The meaning of the columns is the same as Table 7. The obtained schedules of MAHA for $RC_A = 1$, $RC_S = 1$, and 6 CSs are shown in **Fig. 16**. The speculative memory read operation R3 in Fig. 16 (a) is resolved in Fig. 16 (b). These results show that the proposed method reduces the energy consumption also when the memory operations are involved.

6. Conclusions

In this paper, the scheduling method for conditional processing to minimize energy consumption is proposed where the probability of execution and the energy consumption of operations are precisely evaluated. While high processing speed is achieved by speculative execution of operations, unnecessary speculative execution and therefore the energy consumption are reduced by the method.

The development of more efficient schedule exploration technique for conditional processing, sophisticating the heuristic algorithm, incorporating the constraint on the number of registers, and consideration for operation chaining remain as future work.

Acknowledgments This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Cadence Design Systems, Inc. and Synopsys, Inc.

[*1] A4 and A94 are avoided since these receive necessary input data immediately from the preceding operations. The addition A94 (A33, A37, A45, and so on as well) accepts three data strangely. This situation is inherited from the original CDFG in Ref. [5].

References

[1] Parker, A.C., Pizarro, J.T. and Mlinar, M.: MAHA: A Program for Datapath Synthesis, *Proc. Design Auto. Conf.*, pp.461–466 (1986).

[2] Wakabayashi, K. and Tanaka, H.: Global Scheduling Independent of Control Dependencies Based on Condition Vectors, *Proc. Design Auto. Conf.*, pp.112–115 (1992).

[3] Radivojevic, I. and Brewer, F.: A New Symbolic Technique for Control-Dependent Scheduling, *IEEE Trans. Computer Aided Design, Int. Circuit. Syst.*, Vol.15, No.1, pp.45–57 (1996).

[4] Wakabayashi, K. and Yoshimura, T.: A Resource Sharing Control Synthesis Method for Conditional Branches, *Proc. Int. Conf. Computer-Aided Design*, pp.62–65 (1989).

[5] Kim, T., Yonezawa, N., Liu, J.W.S. and Liu, C.L.: A Scheduling Algorithm for Conditional Resource Sharing — A Hierarchical Reduction Approach, *IEEE Trans. Computer Aided Design, Int. Circuit. Syst.*, Vol.13, No.4, pp.425–437 (1994).

[6] Yamada, A., Yamazaki, T., Ishiura, N., Shirakawa, I. and Kambe, T.: Datapath Scheduling for Behavioral Description with Conditional Branches, *IEICE Trans. Fundamentals*, Vol.E77-A, No.12, pp.1999–2009 (1994).

[7] Ishiwata, H., Togawa, N., Yanagisawa, M. and Ohtsuki, T.: A Time-Constrained Scheduling Algorithm for CDFG with Conditional Branches, *IEICE Tech. Report*, Vol.VLD95-133, pp.31–36 (1996).

[8] Ito, K. and Kawasaki, T.: An Overlapped Scheduling Method for an Iterative Processing Algorithm with Conditional Operations, *IEICE Trans. Fundamentals*, Vol.E81-A, No.3, pp.429–438 (1998).

[9] Li, J. and Gupta, R.K.: An Algorithm To Determine Mutually Exclusive Operations In Behavioral Descriptions, *Proc. DATE*, pp.457–463 (1998).

[10] Kountouris, A.A. and Wolinski, C.: Combining Speculative Execution and Conditional Resource Sharing to Efficiently Schedule Conditional Behaviors, *Proc. ASP-DAC*, pp.343–346 (1999).

[11] Wakabayashi, K.: Unified Representation for Speculative Scheduling: Generalized Condition Vector, *IEICE Trans. Fundamentals*, Vol.E89-A, No.12, pp.3408–3415 (2006).

[12] Ghiasi, S., Huang, P.-K. and Jafari, R.: Probabilistic Delay Budget Assignment for Synthesis of Soft Real-Time Applications, *IEEE Trans. Very Large Integration (VLSI) Systems*, Vol.14, No.8, pp.843–853 (2006).

[13] Synphony C Compiler, Synopsys, Inc. (online), available from ⟨http://www.synopsys.com/⟩ (accessed 2012-08-10).

[14] Catapult C Synthesis, Mentor Graphics Corp. (online), available from ⟨http://www.mentor.com/catapult⟩ (accessed 2012-08-10).

[15] Petric, V. and Roth, A.: Energy-effectiveness of pre-execution and energy-aware p-thread selection, Technical report, Proc. 32nd Int'l Symp. on Computer Architecture (2003).

[16] Parikh, D., Skadron, K., Zhang, Y. and Stan, M.: Power-Aware Branch Prediction: Characterization and Design, *IEEE Trans. Comput*, Vol.53, pp.168–186 (2004).

[17] Aragon, J.L., Gonzalez, J. and Gonzalez, A.: Power-Aware Control Speculation through Selective Throttling, *Proc. 9th Int. Symp. High-Performance Comp. Arch.*, pp.103–112 (2003).

[18] Nagpal, R. and Bhowmik, A.: Criticality Driven Energy Aware Speculation for Speculative Multithreaded Processors, *Proc. High Performance Computing*, pp.19–28 (2005).

[19] Farooq, M.U., John, L.K. and Jacome, M.F.: Compiler Controlled Speculation for Power Aware ILP Extraction in Dataflow Architectures, *Proc. High Performance Embedded Architectures and Compilers*, pp.324–338 (2009).

[20] Micheli, G.D.: *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York (1994).

[21] Ito, K. and Seto, H.: Reducing Power Dissipation of Data Communications on LSI with Scheduling Exploration, *IPSJ Trans. System LSI Design Methodology*, Vol.2, pp.53–63 (2009).

[22] Panda, P.R. and Dutt, N.: HLSynth95, University of California, Irvine (online), available from ⟨http://www.ics.uci.edu/pub/hlsynth/HLSynth95/⟩ (accessed 2012-05-20).

Kazuhito Ito was born in 1964. He received his B.E., M.E. and Ph.D. degrees in Electrical Engineering from Tokyo Institute of Technology, Japan, in 1987, 1989, and 1992, respectively. He is an associate professor of the Graduate School of Science and Engineering, Saitama University, Saitama, Japan. His research interests include high-level synthesis in VLSI design, VLSI signal processing, and design automation of system LSIs. He is a member of IEICE, IPSJ, and IEEE.

Kazuhiko Kameda received his B.E. and M.E. degrees in Electrical Engineering from Saitama University, Japan, in 2009 and 2011, respectively. His research interests include high-level synthesis in VLSI design, and low power design of LSIs.

(Recommended by Associate Editor: *Shinsuke Kobayashi*)