Soft-Error Tolerant Datapath Synthesis Based on Speculative Resource Sharing in Triple Algorithm Redundancy

Junghoon Oh and Mineo Kaneko
School of Information Science
Japan Advanced Institute of Science and Technology
1-1 Asahidai, Nomi, Ishikawa, Japan 923-1292
{junghoon.oh, mkaneko}@jaist.ac.jp

Abstract— As semiconductor technologies have advanced, the reliability problem caused by soft-errors is becoming one of the serious issues in LSIs. In this paper, we propose a method to synthesize soft-error tolerant application-specific datapaths via high-level synthesis. The novel feature of our method is speculative resource sharing between the retry parts and the secondary parts for hardware/time overhead mitigation. A scheduling algorithm using a special priority function to maximize the speculative resource sharing is also an important feature of this study. We found that our method is more effective when a computation algorithm possesses higher parallelism and a smaller number of resources is available.

I. Introduction

A soft-error is a transient fault which is triggered by cosmic rays induced neutron and alpha rays from radioactive contaminants in IC packing materials. The effect of soft-error is temporary but it can affect several spatial points simultaneously[1][2]. As the device size decreases, the reliability degradation caused by soft-errors has become one of the greatest issues in LSIs. Approaches to deal with soft-errors are roughly classified into the following three groups: (1) approaches on the device level such as the selection of IC packing materials and the improvement of well structures; (2) approaches on the circuit level such as a flip-flop (FF) with additional circuits for error detection, error recovery and error avoidance[3][4]; (3) approaches on the system level which include concurrent error detection (CED) and triple modular redundancy (TMR)[5]–[7].

Orailoglu and Karri[5] introduced a system which detects and corrects transient faults using CED, checkpointing and rollback. Wu and Karri[6] proposed a high-level synthesis of duplicated computation algorithms for fault detection considering partitioned data dependence of the algorithms. In [7], high-level synthesis for multi-cycle transient fault tolerant datapaths based on TMR was proposed. However, their study considered only a single fault on a single functional unit. While single-bit upsets are principally a reliability concern in memory devices and systems, multi-bit upsets and multi-cell upsets have recently become a serious problem, as well[1]. The work[8] reports that TMR can be defeated by even 2-bit multi-cell upsets caused by a single soft-error.

In this paper, we propose a method to synthesize single soft-error tolerant application-specific datapaths via high-level synthesis. Triple algorithm redundancy is our starting point for designing soft-error tolerant datapaths. Concerning the tolerability against multiple-component errors caused by a single soft-error, in our design, the main parts and the secondary parts are used for error detection, and the retry parts are used for error recovery. Two most important and novel features of our method are (1) speculative resource sharing between the retry parts and the secondary parts for hardware/time overhead mitigation in compensation for the assumption of sufficiently low probability of soft-errors, and (2) an inherent priority function used in the scheduling algorithm to maximize the speculative resource sharing.

This paper is organized as follows: Section II explains preliminaries of our method. Section III shows requested features for single soft-error tolerance and speculative resource sharing. In Section IV, a scheduling and resource binding algorithm with the specific features shown in Section III is proposed. Section V shows some extensions for multi-cycle soft-error tolerance. Section VI shows experimental results, and Section VII concludes this paper.

II. Preliminaries

A. *Error Detection and Error Correction Scheme*

Our method is based on triple algorithm redundancy. Considering the tolerability against multiple-component errors caused by a single soft-error, and mitigating resource/time overhead, the first copies and the second copies are used for detecting errors, and the third copies are used as retries in our design. For example, the first copy $A^{(1)}$ and the second copy $A^{(2)}$ of a computation block A are executed first, and their output data are compared by the comparator 'q'. If no error is detected, output data from $A^{(1)}$ is sent to the succeeding blocks. Only if a soft-error affects $A^{(1)}$ or/and $A^{(2)}$ and, as a result, an error is detected, the output data from $A^{(1)}$ and $A^{(2)}$ are abandoned and the third copy $A^{(3)}$ is executed as a retry. The retry data, then, is sent to succeeding computation blocks.

B. *Triple Algorithm Redundancy*

We assume that a computation algorithm to be implemented is given by a pair (G, D_P), where G is a dependence graph and D_P is the set of primary outputs. Let O be the set of all operations in the original computation algorithm, let D be the set of all variables in the original algorithm and let E be the set of all dependencies between operations and variables. Then a given dependence graph G can be described as $G = (O \cup D, E)$. To synthesize a soft-error tolerant datapath, the given graph G is triplicated (three copies are denoted as $G^{(1)}$, $G^{(2)}$ and $G^{(3)}$), and then comparison-operations and dependencies related to the comparison-operations are inserted to form a resultant algorithm $(\tilde{G}, \tilde{D_P})$ which is mapped on hardware and time domains by high-level synthesis in the end.

C. *Cone Partitioning/Comparison-Operation Insertion*

We define "check variables" as variables to be compared for soft-error detection, and we will choose them selectively, not to all operation results. Dependence subgraphs separated by those check variables are named "cones". Now we let D_Q ($\subseteq D$) be a set of the check variables and let Q be a set of comparison-operations corresponding to the variables in D_Q.

Definition 1 A "cone" c_d ($\subseteq G$) denotes a subgraph which is induced by tracing back from a check variable $d \in D_Q$ to primary inputs or other check variables. Similarly, triplicate copies $c_d^{(1)}$ (called "main-cone"), $c_d^{(2)}$ (called "second-cone") and $c_d^{(3)}$ (called "retry-cone") of c_d are induced from the triplicate check variables $d^{(1)} \in G^{(1)}$, $d^{(2)} \in G^{(2)}$ and $d^{(3)} \in G^{(3)}$, respectively.

Definition 2 The set of main-cone $c_d^{(1)} \subseteq G^{(1)}$, second-cone $c_d^{(2)} \subseteq G^{(2)}$ and retry-cone $c_d^{(3)} \subseteq G^{(3)}$ which are induced from a check variable d is called "stage".

Definition 3 The triplicate copies $o^{(1)}(\in c_d^{(1)})$, $o^{(2)}(\in c_d^{(2)})$ and $o^{(3)}(\in c_d^{(3)})$ of an operation o ($\in c_d$) are elements of the same stage, and they are called "series operations".

III. Conditions for Single Soft-Error Tolerant Datapaths

A. *Fault/Error Model and Fault Tolerant Condition*

In this study, we assume that a single soft-error can affect several spatial points simultaneously, and hence it causes multiple component errors, including errors on registers, functional units and other components at the same time. The proposed soft-error tolerant design is based on cone-level error masking, and relies on error detection by comparing the results of main-cone $c_d^{(1)}$ and second-cone $c_d^{(2)}$, and error correction by executing the retry-cone $c_d^{(3)}$ for each $d \in D_Q$.

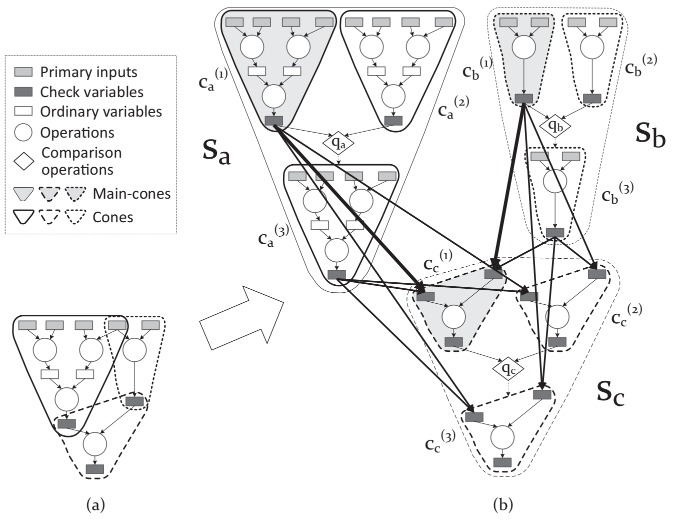

Fig. 1. A stage s_a consists of $c_a^{(1)}$, $c_a^{(2)}$ and $c_a^{(3)}$; (a) An example of an original computation algorithm (b) An example of a cone-partitioned triplicate algorithm

Retry-cones are executed to recover from erroneous results of the corresponding main-cones or second-cones caused by soft-errors. If the execution of a retry-cone overlaps the executions of the main-cone, the second-cone or the comparison-operation in the same stage, multiple component errors due to a single soft-error may affect two cones or more (retry-cone and either main- or second-cone) in the same stage simultaneously. For this reason, the execution order of operations in each stage should be constrained as follows.

Condition 1 [Execution order in each stage] After a main-cone and the corresponding second-cone are executed, then error detection with a comparison operation is performed to check the results of two cones. After that, the corresponding retry-cone is executed only if the results differ.

In the CED-retry mechanism, only if an error is detected, then corresponding retry-cone is executed and its result is used immediately for the succeeding operations without error detection of the retry result. The validity of this treatment relies on the following assumption.

Assumption 1 The probability of the recurrence of soft-errors in a short period is sufficiently low.

On the other hand, triplicate data in three standard registers which have outputs of three cones are not reliable anymore under the assumption of multiple component errors caused by a single soft-error. Therefore, in order to guarantee the correctness of the inputs to a retry-cone even if its main- and second-cone are affected by a single soft-error, we have decided to use specialized registers, such as BCDMR-ACFF[3], for input variables to each cone (they are primary inputs or check variables) and outputs of comparators.

B. *Speculative Resource Sharing*

In order to mitigate hardware resource/time overhead, we propose "speculative resource sharing" as follows. In our treatment, operations in a retry-cone are not executed as long as no error is detected. It means that resources bound to operations in a retry-cone are in idle state if two results of corresponding main-cone and second-cone are identical. If the probability of the recurrence of soft-errors in a short period is sufficiently low (Assumption 1), we can rebind the resources in idle state to other operations which have no dependency with the operations in the retry-cone. More specifically, operations in a retry-cone can share resources speculatively with operations of second-cones in different stages.

From Assumption 1, while a retry-cone is running (an error caused by a soft-error in the main-cone or/and the second-cone of the running retry-cone is detected), the correctness of other main cones which are executed after the erroneous main- and second-cones of the running retry-cone is guaranteed. Based on this observation, resources can be managed more efficiently by speculative resource sharing between error detection parts and error correction parts.

In Fig. 2, if operations in $c_m^{(3)}$ and $c_n^{(2)}$ share resources by speculative resource sharing ($m \neq n$ and there is no dependency between c_m and c_n), $c_n^{(1)}$ is unable to detect error when a soft-error affects some of $c_m^{(1)}$, $c_m^{(2)}$ and q_m, and $c_m^{(3)}$ is executed, since the execution of $c_n^{(2)}$ is abandoned by $c_m^{(3)}$. Hence, we need to carefully manage the execution of $c_n^{(1)}$ so that the soft-error which affects $c_m^{(1)}$, $c_m^{(2)}$ and/or q_m may not affect $c_n^{(1)}$. Considering such behavior, we introduce the second condition.

Condition 2 [Speculative resource sharing] Only if the execution of a main-cone $c_n^{(1)}$ is scheduled later than the execution of a comparison operation q_m, speculative resource sharing between operations in $c_m^{(3)}$ and operations in a second-cone $c_n^{(2)}$ corresponding to $c_n^{(1)}$ is possible. ($m \neq n$)

IV. Synthesis for Soft-Error Tolerant Datapath

In this section, we propose a scheduling and resource binding algorithm for single soft-error tolerant datapaths. The way how to select check variables for comparison and cone partitioning is an important factor for datapath optimization. However, in this paper, the set of check variables is assumed to be given as a part of an input description for high-level synthesis. Optimization of the selection of check variables is left as one of the important future works.

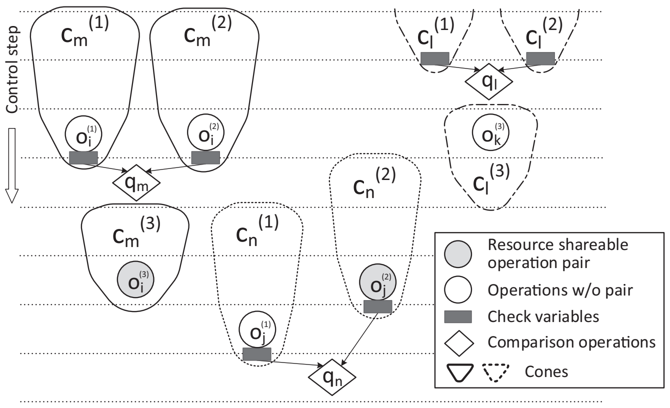

Fig. 2. Scheduled DFG; An example of speculative resource sharing between operations in second-cones and retry-cones. $o_i^{(3)}$ and $o_j^{(2)}$ can share a resource speculatively. However, $o_k^{(3)}$ and $o_i^{(2)}$ cannot.

Our algorithm is based on the list scheduling but has an inherent priority function and an inherent resource counting considering speculative resource sharing.

In the following, we let $\sigma : O^{(1)} \cup O^{(2)} \cup O^{(3)} \cup Q \to \mathbb{N}$ be an operation schedule (a control step assignment), where $O^{(\ell)}, \ell = 1, 2, 3,$ is the set of operations in $G^{(\ell)}$.

A. *Schedule Constraints*

1. Resource constraints for every resource type have to be satisfied at each control step cs. Let $A_{cs,r}$ be a set of operations which are scheduled at a control step cs with a resource type r. The actual number of resources of type r in use at step cs is described as follows: The speculative resource sharing conditions are described in the next section B.

$$\begin{aligned}&\text{Number of resources of type } r \text{ in use at step } cs\\ &= |A_{cs,r}| - \text{“Number of operations which share}\\ &\quad \text{resources speculatively in } A_{cs,r}\text{” } / 2\\ &\leq \text{Number of allocated resources of type } r \qquad (1)\end{aligned}$$

2. $c_m^{(3)}$, the retry-cone in a stage m, has to be executed after the comparison operation q_m that compares the results of $c_m^{(1)}$ and $c_m^{(2)}$ under Condition 1.

$$max(\bigcup_{\ell=1}^{2} \sigma(c_m^{(\ell)})) \ < \ \sigma(q_m) \ < \ min(\sigma(c_m^{(3)})), \quad (2)$$
$$\text{where } \sigma(c_m^{(\ell)}) = \{x | \sigma(o_i^{(\ell)}) = x, o_i^{(\ell)} \in c_m^{(\ell)}\}$$

3. If o_i is a predecessor of o_j,
   - For operations $o_i^{(\ell)}, o_j^{(\ell)} \in c_m^{(\ell)}$, the successor $o_j^{(\ell)}$ has to be scheduled after the execution of $o_i^{(\ell)}$. ($\ell = 1, 2, 3$)

$$\sigma(o_i^{(\ell)}) \ < \ \sigma(o_j^{(\ell)}) \qquad (3)$$

– For operations $o_i^{(\ell)} \in c_m^{(\ell)}$, $o_j^{(\ell)} \in c_n^{(\ell)}$ ($m \neq n$ and $\ell = 1, 2, 3$), the successor $o_j^{(\ell)}$ has to be scheduled after the execution of $o_i^{(3)}$ whose output is the result of $c_m^{(3)}$ $\left(\sigma(o_i^{(3)}) = max(\sigma(c_m^{(3)}))\right)$.

$$\sigma(o_i^{(3)}) \; < \; \sigma(o_j^{(\ell)}) \tag{4}$$

B. *Speculative Resource Sharing Conditions between Operations in Second-Cones and in Retry-Cones*

When there is no dependency between o_i and o_j $(i \neq j)$, $o_i^{(3)}$ and $o_j^{(2)}$ can be bound to the same resource under the following conditions (Fig. 2).

1. $o_i^{(3)}$ and $o_j^{(2)}$ belong to different stages.
$$o_i^{(3)} \in c_m^{(3)}, \; o_j^{(2)} \in c_n^{(2)} \quad (m \neq n) \tag{5}$$
2. $o_i^{(3)}$ and $o_j^{(2)}$ are scheduled at the same control step.
$$\sigma(o_i^{(3)}) \; = \; \sigma(o_j^{(2)}) \tag{6}$$
3. According to Condition 2,
$$\sigma(q_m) \; < \; min(\sigma(c_n^{(1)})) \tag{7}$$

When the above conditions (5)∼(7) are satisfied, $o_i^{(3)}$ and $o_j^{(2)}$ can share a same resource speculatively.

C. *Proposed Scheduling Algorithm*

Algorithm 1 shows the proposed algorithm which is to perform a list scheduling under given resource constraints. The following notations are employed to explain the proposed scheduling algorithm.

- $L_{cs,r}$: a list of ready operations of resource type r at control step cs
- $U_{cs,r}$: a list of executing operations of resource type r at control step cs
- $v_{r,k}$: the k-th operation of resource type r
- res_inuse : the number of functional units in use

Scheduling proceeds from control step 1 toward the last control step as the original list scheduling does. In each control step cs, first, operations of type r which are scheduled already and are executing at control step cs, are registered to $U_{cs,r}$. Ready operations which are not yet scheduled are registered to $L_{cs,r}$. After that, the function **count_occupied_FU** is called, which counts the number of presently occupied functional units (res_inuse) from $U_{cs,r}$, and then the function **sched_with_speculative_share** is called, which binds operations to control step cs considering the speculative resource sharing conditions (Section IV–B).

Algorithm 1 Modified list scheduling algorithm

```
Require: $N_{max} \leftarrow$ # of resource types
1: $cs \leftarrow 0$;
2: while (until all operations are scheduled) do
3:     $cs \leftarrow cs + 1$;
4:     for ($r \leftarrow 1$; $r \leq N_{max}$; $r$++) do
5:         $res\_inuse \leftarrow 0$;
6:         for ($k \leftarrow 1$; $k \leq$ # of operations (type $r$); $k$++) do
7:             if ($v_{r,k}$ is already scheduled) then
8:                 if (execution of $v_{r,k}$ is not finish yet) then
9:                     Register $v_{r,k}$ in $U_{cs,r}$;
10:                end if
11:            else if (executions of all immediate predecessors of $v_{r,k}$
                   are already finished) then
12:                Register $v_{r,k}$ in $L_{cs,r}$;
13:            end if
14:        end for
15:        $res\_inuse \leftarrow$ count_occupied_FU($U_{cs,r}$);
16:        sched_with_speculative_share($cs, r, res\_inuse, L_{cs,r}$);
17:    end for
18: end while
```

D. *Scheduling Priority*

First of all, we introduce priority and latency of a stage. Priority of a stage is defined as the smallest control step of ALAP schedule over all operations in the stage. Latency of a stage is defined as execution time from the start to the end of the stage based on ALAP schedule.

$$\text{Priority of } s_m = min(\bigcup_{\ell=1}^{2} \sigma_{ALAP}(c_m^{(\ell)})) \tag{8}$$

$$\begin{aligned}&\text{Latency of } s_m \\ &= max(\sigma_{ALAP}(c_m^{(3)})) - min(\bigcup_{\ell=1}^{2} \sigma_{ALAP}(c_m^{(\ell)})) + 1\end{aligned} \tag{9}$$

The priority of an operation is determined by applying the following factors in this order (a tie in the first factor is broken by the second factor, a tie in the second factor is broken by the third factor, and so on).

(1) Operations which are in a stage having higher priority have higher priority. As a result, a stage which started earlier can finish earlier. If a stage which started earlier remains without being chosen and only operations in other stages are scheduled, operations in the remained stage can be a bottleneck on the entire schedule.
(2) Operations which are in a stage having smaller latency have higher priority, since a stage which is expected to finish earlier can include speculatively resource sharable operation pairs easier.
(3) Operations which have smaller ALAP schedules are given higher priority.
(4) Operations which are placed on a critical path are given higher priority.
(5) Among series operations, operations in second-cones are given higher priority for speculative sharing.

Using these factors, priority is given to every operation.

E. *Selecting Operations*

Even if the priority of ready operations is fixed, selecting operations to be scheduled at the current control step is not straightforward, since we need to manage a complicated resource sharing between second-cones and retry-cones. In order to choose speculatively resource sharable pairs aggressively, we will use a bipartite graph H with X_T and Y_T as its partite sets.

We define "vertex weight" $w(z) \in \mathbb{Z}_+$ as priority of an operation z and "edge weight" $w(e = \{x, y\}) \in \mathbb{Z}_+$ as the sum of weights of two end vertices x and y, that is, $w(e = \{x, y\}) = w(x) + w(y)$.

- A bipartite graph $H = (X_T \cup Y_T, W)$ describes the following:
  - A partite set X_T is defined as $X_T = X \cup Y_C$, where $X = \{x \mid x \in L_{cs,r} \wedge x \in G^{(1)} \cup G^{(2)}\}$ and Y_C is a set of auxiliary vertices with $|Y_C| = |Y|$.
  - The other partite set Y_T is defined as $Y_T = Y \cup X_C$, where $Y = \{y \mid x \in L_{cs,r} \wedge y \in G^{(3)}\}$ and X_C is a set of auxiliary vertices with $|X_C| = |X|$.
  - The edge set W of H is defined as follows. $W = \{\{x, P_X(x)\} \mid x \in X\} \cup \{\{P_X(y), y\} \mid y \in Y\} \cup \{\{x, y\} \mid x \in X \cap G^{(2)}$ and $y \in Y$ can share a resource speculatively$\}$, where $P_X : X \to X_C$ and $P_Y : Y \to Y_C$ are arbitrary one-to-one mapping from X to X_C and from Y to Y_C, respectively.
- We consider that weights of all auxiliary vertices are 0, that is, $\forall z, w(z) = 0$, where $z \in X_C \vee z \in Y_C$.

Once H is constructed, selecting an edge $e = \{x, y\}$ from H means that two operations which are represented by the end vertices x and y of e share a resource speculatively. If an end vertex of e is an auxiliary vertex (a vertex in either X_C or Y_C), it means that the operation which is represented by the other end vertex occupies a resource without speculative sharing. As a result, an operation selection considering speculative resource sharing can be considered as a problem to find a size-constrained maximum weight matching from H.

Algorithm 2 shows a detailed description of function **schedule_with_speculative_share**. After a graph H is constructed, speculatively resource sharable operation pairs are chosen by a greedy selection procedure. More specifically, edges are selected in descending order of the edge weights within the number of available resources. Fig. 3 illustrates an example of operations selecting result by Algorithm 2.

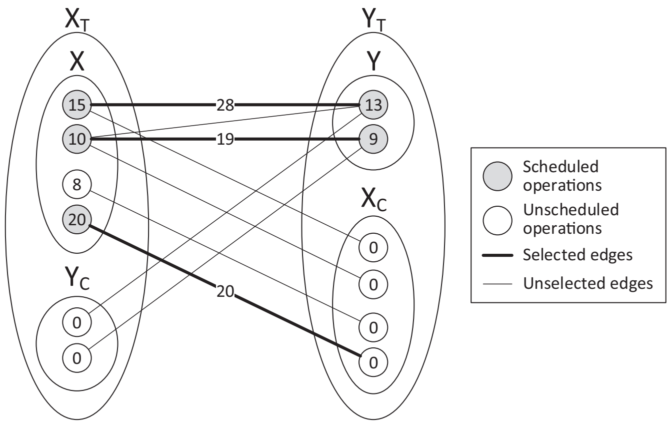

Fig. 3. An example of operations selecting result produced by greedy selection procedure; Operations which have edge weight 10 and 15 are elements of $G^{(2)}$. Other operations in X are elements of $G^{(1)}$. The number of available resources is 3.

Algorithm 2
Algorithm of function schedule_with_speculative_share()

```
Require: $cs$ ← Current control step
Require: $r$ ← Currently considered resource type
Require: $n_r$ ← Available number of resources of type $r$
Require: $L_{cs,r}$ ← List of ready operations of type $r$ at step $cs$
 1: Construct a bipartite graph $H$ based on $L_{cs,r}$;
 2: for ($i \leftarrow 0$; $i <$ Total number of $W$; $i$++) do
 3:     if ($n_r > 0$) then
 4:         Select an edge $e$ from $W$ which has the largest edge weight;
 5:         Schedule two end vertices of $e$ at the current step $cs$;
 6:         Remove the edge $e$ and its end vertices from $H$
 7:         $n_r \leftarrow n_r - 1$;
 8:     else
 9:         break;
10:     end if
11: end for
```

V. Multi-Cycle Soft-Error Tolerant Datapath

We can expand our schedule constraints and speculative resource sharing conditions so that they can manage multi-cycle soft-errors[7].

A. Schedule Constraints

When we consider k-cycle soft-error tolerant datapath synthesis, the difference in schedule constraints is Equation (2) in Section IV–A.

- $c_m^{(3)}$, the retry-cone in a stage m, has to be executed k steps or more after the completion of the comparison operation q_m which compares the results of $c_m^{(1)}$ and $c_m^{(2)}$.

$$max(\bigcup_{\ell=1}^{2} \sigma(c_m^{(\ell)})) + k - 1 \;<\; \sigma(q_m) \tag{10}$$

$$\sigma(q_m) + k - 1 \;<\; min(\sigma(c_m^{(3)})) \tag{11}$$

B. Speculative Resource Sharing Conditions between Operations in Second-Cones and in Retry-Cones

These conditions, excluding Equation (7), are the same as the conditions in Section IV–B,

$$\sigma(q_m) + k - 1 \;<\; min(\sigma(c_n^{(1)})) \tag{12}$$

VI. Experimental Results

We have implemented the proposed scheduling algorithm as a computer program, and applied it to 16-point fast Fourier transform (16FFT), 8-point inverse discrete cosine transform (8IDCT), 16-point FIR filter (16FIR), autoregressive filter (ARF), fifth-order elliptic wave digital filter (5EWDF), and inverse discrete cosine transform

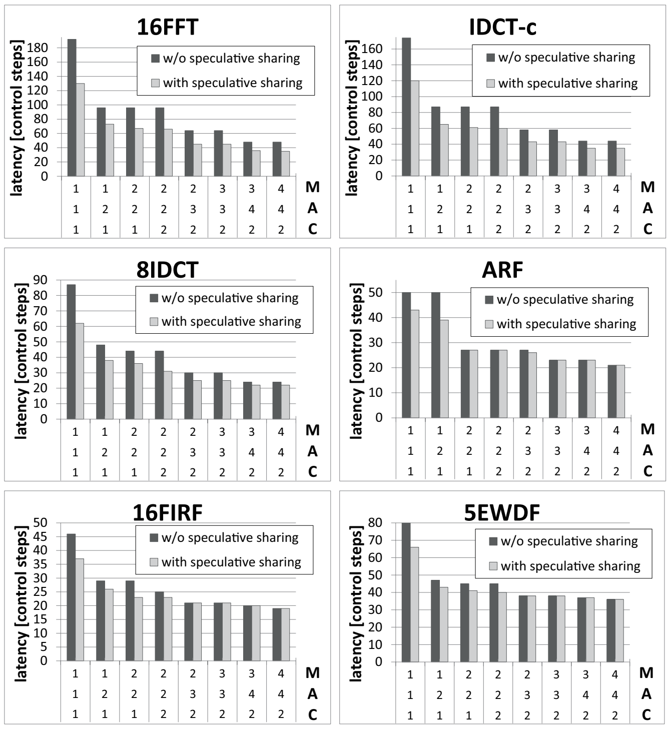

Fig. 4. Experimental results by various computational algorithms. Each column represents allocated resources (C:Comparator, A:ALU, M:Multiplier). Every column in each graph has two grouping bars. Left bars represent scheduling results without speculative resource sharing and right bars represent proposed scheduling results with speculative resource sharing.

with column-wise decomposition (IDCT-c). In order to evaluate our scheduling algorithm, we also implemented a conventional list scheduling algorithm without speculative resource sharing.

Fig. 4 and Table I show experimental results. Each graph in Fig. 4 illustrates the achieved latencies with different specifications of resources. Table I shows the maximum improvement rates in latency between conventional list scheduling results without speculative resource sharing and proposed list scheduling results with speculative resource sharing. We can find from those results that, for every application algorithm, the latency is improved more or less by the speculative resource sharing. Also we found that the proposed method is more effective when a computation algorithm possesses higher parallelism (16FFT, IDCT-c and 8IDCT) and the improvement in latency is larger when a smaller number of resources is allocated. The reason is that there are more possibilities to share resources speculatively between retry-cones and second-cones and, in consequence, many chances for speculative resource sharing can conceal the extension of latency.

VII. Conclusion

Concerning soft-error tolerant datapath synthesis based on triplication of an input computation algorithm via high-level synthesis, constraints for soft-error tolerance and a scheduling algorithm considering speculative resource sharing are proposed. Datapath circuits designed by our method tolerate multi-component and multi-cycle errors caused by single soft-errors. From the results of soft-error tolerant datapath synthesis experiments, we found that our speculative resource sharing achieves maximum 32.3% improvement in latency. Especially, our proposed method is more effective when an input computation algorithm possesses higher parallelism, and the number of allocated resource is relatively small. Optimization of cone partitioning and register binding are left as future works. Moreover, we needed to investigate that our method is applicable to more practical problems such as more large applications and CDFG with loop structures.

TABLE I
Experimental results; latency improvement rate

Computation algorithm	Total # of operations (triplicate algorithm)	Max. improvement rate [%]
16FFT	328	32.3
IDCT-c	234	31.0
8IDCT	160	29.5
ARF	96	22
16FIRF	80	20.7
5EWDF	120	17.5

References

[1] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in srams from a 250 nm to a 22 nm design rule," *Electron Devices, IEEE Transactions on*, vol. 57, no. 7, pp. 1527–1538, July 2010.

[2] H. Fuketa, R. Harada, M. Hashimoto, and T. Onoye, "Measurement and analysis of alpha-particle-induced soft errors and multiple-cell upsets in 10t subthreshold sram," *Device and Materials Reliability, IEEE Transactions on*, vol. 14, no. 1, pp. 463–470, March 2014.

[3] M. Masuda, K. Kubota, R. Yamamoto, J. Furuta, K. Kobayashi, and H. Onodera, "A 65 nm low-power adaptive-coupling redundant flip-flop," *Nuclear Science, IEEE Transactions on*, vol. 60, no. 4, pp. 2750–2755, Aug 2013.

[4] Y. Lin and M. Zwolinski, "Settoff: A fault tolerant flip-flop for building cost-efficient reliable systems," in *On-Line Testing Symposium (IOLTS), 2012 IEEE 18th International*, June 2012, pp. 7–12.

[5] A. Orailoglu and R. Karri, "Automatic synthesis of self-recovering vlsi systems," *Computers, IEEE Transactions on*, vol. 45, no. 2, pp. 131–142, Feb 1996.

[6] K. Wu and R. Karri, "Fault secure datapath synthesis using hybrid time and hardware redundancy," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 23, no. 10, pp. 1476–1485, Oct 2004.

[7] T. Inoue, H. Henmi, Y. Yoshikawa, and H. Ichihara, "High-level synthesis for multi-cycle transient fault tolerant datapaths," in *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, July 2011, pp. 13–18.

[8] H. Quinn, K. Morgan, P. Graham, J. Krone, M. Caffrey, and K. Lundgreen, "Domain crossing errors: Limitations on single device triple-modular redundancy circuits in xilinx fpgas," *Nuclear Science, IEEE Transactions on*, vol. 54, no. 6, pp. 2037–2043, Dec 2007.