A Heuristic Scheduling Algorithm with Variable-Cycle Approximate Functional Units in High-Level Synthesis

Koyu Ohata[1] Hiroki Nishikawa[2] Xiangbo Kong[1] Hiroyuki Tomiyama[1]

[1] *Graduate School of Science and Engineering, Ritsumeikan University, Shiga, Japan*
[2] *Graduate School of Information Science and Technology, Osaka University, Osaka, Japan*

Abstract
Computational approximation is a promising paradigm to exploit hardware capabilities or mitigate computational demands, taking advantage of inherent tolerance to errors in applications. For several decades, a variety of works for approximate computing have been extensively studied. In particular, accuracy-controllable approximate functional units have been developed to be configurable to control the accuracy at runtime. To satisfy the requirement of performance, hardware cost and accuracy, the techniques for design space exploration are important since the excessive use of approximate circuits results in loss of accuracy, while that of accurate circuits can hardly meet time constraints. This paper proposes a list scheduling algorithm in the high-level synthesis of accuracy-controllable approximate functional units with variable latency. The experiments demonstrate that our proposed scheduling method can efficiently explore the trade-offs between performance, hardware cost, and accuracy in a practical time, compared to the state-of-the-art methods.

Keywords
High-level synthesis, approximate computing, scheduling, approximate multiplication, list scheduling

1. Introduction

Approximate computing is attracting attention to rapidly designing high-performance, low-cost and low-power circuits for error-tolerant applications such as image processing and machine learning. Design techniques for approximate arithmetic circuits have been studied at various design levels, from the transistor to the architecture level [1-2]. The quality requirements of an error-tolerant application using approximate computing may vary significantly at runtime. There have appeared several works [3-5] on approximate functional units that can dynamically change accuracy at runtime. A carry-maskable adder [3] was proposed to enable dynamically selecting the length of the carry propagation to satisfy the quality requirements. Also, an accuracy-controllable multiplier [4] was proposed where the last stage is generated by a carry-maskable adder. The approximate multiplier has an additional input, and it is utilized to control the error at the output. If the error is large, the latency and power of the circuit will be small. On the other hand, if the error is reduced, the latency and power of the circuit will be large. Thus, the trade-off between error, power, and latency can be optimized at runtime by using a functional unit with variable accuracy. Further on, the work in [5] proposed an approximate multiplier that can dynamically change its accuracy by using a digital signal processor (DSP) with an embedded field-programmable gate array (FPGA). These studies show that the more inaccurate the computation, the faster it becomes.

There is a large amount of literature on high-level synthesis (HLS) of approximate computing circuits such as [6-8]. Unfortunately, most of the works previously mentioned have paid little attention to allocation, scheduling, and binding algorithms, which are the crucial techniques in HLS. Regarding HLS techniques aware of accuracy-controllable approximate functional units, there is the work [9]. The

The 4th International Symposium on Advanced Technologies and Applications in the Internet of Things (ATAIT 2022), August 24-26, 2022, Ibaraki, Japan
EMAIL: koyu.ohata@tomiyama-lab.org (K. Ohata); nishikawa.hiroki@ist.osaka-u.ac.jp (H. Nishikawa); kong@fc.ritsumei.ac.jp (X. Kong); ht@fc.ritsumei.ac.jp (H. Tomiyama)

work in [9] focused on accuracy-controllable functional units that can take variable cycles and proposed a scheduling scheme that takes into account the difference in delays between accurate and approximate computations in order to efficiently use accuracy-controllable approximate functional units. Their experiments show that it realized the tradeoffs between the errors, the resource, and the time constraints. However, the scheduling technique was based on integer linear programming (ILP) and the runtime for scheduling becomes exponentially longer as increasing the problem sizes. In order to solve the issue above, this paper presents a heuristic scheduling algorithm in HLS. Our proposal is based on a resource-constrained list scheduling algorithm and aims to minimize the output error by switching approximate functional units that cause large errors in the output values to accurately with satisfying the resource and time constraints.

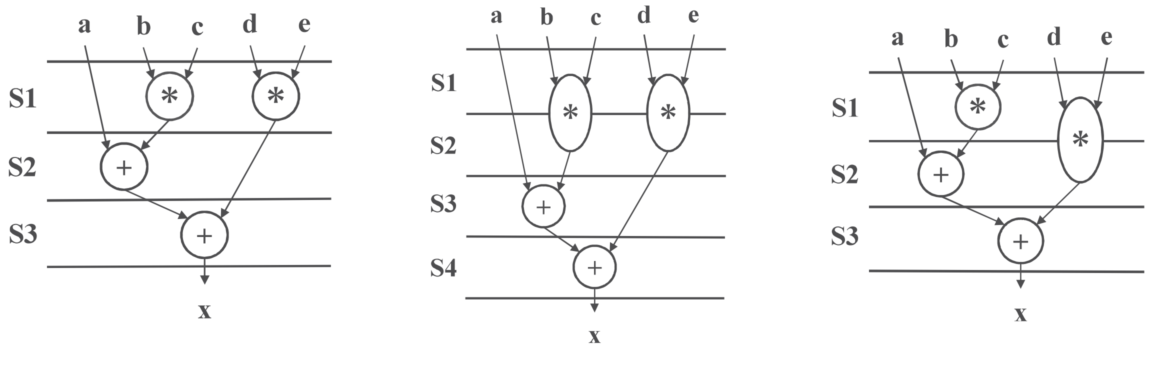
The remainder of the paper is organized as follows: Section II shows scheduling with variable-cycle functional units in HLS, and we propose a heuristic scheduling algorithm to solve this scheduling problem in polynomial time. Section III evaluates our proposed algorithm by comparing it to existing work. Finally, we conclude this paper in Section IV.

2. Variable-Cycle Scheduling in High-Level Synthesis

2.1. Variable-Cycle Scheduling

This section presents algorithm [9] for scheduling variable-cycle approximate functional units. In the following section, for simplicity, we have an assumption that there are two types of functional units in the scheduling problem: accuracy-controllable multiplier and arithmetic logic unit (ALU). An accuracy-controllable multiplier, which has been developed in [5], is capable of dual operation modes. One is called accurate mode that the multiplication takes two cycles and does not produce the error. The other is called approximate mode that takes a single cycle and allows the error. A function other than multiplication is assumed to use an ALU that takes a cycle. Variable-cycle scheduling determines not only a schedule of operations in control states but also the accuracy mode for each multiplication simultaneously, with the goal of error minimization at the output under a given resource and time constraint. It should be noted that our proposed algorithm can be easily extended to consider another specific functional unit, not ALU, and to enable its functional unit to take several execution modes as well as the accuracy-controllable multiplier. The number of cycles for each function is not limited to the assumption above. Although this work is targeted at multipliers [5] that can dynamically change their computational accuracy, it can extend to other approximate adders [10, 11] and multipliers [12, 13].

An example of variable cycle scheduling is shown in Figure 1. We are given a data flow graph (DFG) as a directed acyclic graph. Each node represents an operation and each edge represents precedence dependency between the operations. The DFG in this example consists of two accuracy-controllable multipliers and two ALUs. The notation labeled as S means a control state. For instance, $S1$ is the first state and $S2$ is the second state. Figure 1 (a) shows that the multiplications are approximated (determined to take one cycle), and it results in the shortest cycles (three cycles) but the



(a) Approximate multiplications (b) Accurate multiplications (c) Variable-cycle multiplications

Figure 1: An example of variable-cycle scheduling

largest error due to the approximation at the same time. In contrast, the multiplications in Figure 1 (b) are determined to execute in the accurate mode so that the operations can achieve the smallest error at the expense of the performance (four cycles). Figure 1 (c) shows variable-cycle scheduling. Unlike Figure 1 (b), the multiplication in the middle is approximated, and the generated circuit achieves shorter cycles than that of Figure 1 (b). In addition, multicycling the rightmost multiplication realizes a circuit with an accuracy higher than that of Figure 1 (a).

2.2. Proposed List Scheduling Algorithm

We propose a heuristic algorithm for the variable-cycle scheduling problem and show a pseudo code in Algorithm 1. Our scheduling algorithm is based on a typical resource-constrained list scheduling. Given a DFG $\mathrm{G(V, E)}$, our algorithm determines a schedule of operations in control states and the accuracy mode for each multiplication simultaneously. Let V denote a set of operations and E denote precedence dependency between the operations. Recall that the operations are classified into two types: the multiplication with accurate and approximate modes and the ALU-based operations. Let M denote a set of multiplications and A denote a set of ALU-based operations. In the set M, a subset called Apx is denoted, and Apx indicates a set of approximate multiplications. Therefore, the relationships are always held as follows; V=M∪A and Apx∈M. The number of cycles for each operation is given in advance. The multiplication takes either T_{acc} cycles for accurate mode and T_{apx} cycles for approximate mode and the ALU-based operations take T_{ALU} cycles. In addition, we denote $Const_{mul}$ and $Const_{ALU}$ as a resource constraint and denote $Const_{time}$ as a time constraint.

The first process in the algorithm conducts as-late-as-possible (ALAP) scheduling for the DFG where all the multiplications assume to be approximate mode and obtains the schedule t_alap_i for each operation i (Lines 2-4). Then, for each operation in the schedule, the produced error e_i is derived from the accurate and approximate outputs with considering the

Algorithm 1. Proposed list scheduling algorithm

```
1  ListScheduling (G(V,E)) begin
2    for $i \in$ V do
3      $t\_alap_i \leftarrow ALAP\_schedule$
4    endfor
5    for $i \in$ V do
6      $e_i \leftarrow$ Obtain $acc_i$, $err_i$ from G(V,E) and any input
7    endfor
8    for n in 1..|M|+1 do
9      for $i \in$ V do
10       if $i \in A \cup i \in Apx$ do
11         $p_i = t\_alap_i$
12       else $p_i = t\_alap_i - (T_{acc} - T_{apx})$ endif
13     endfor
14     $t = 0, \pi = \{\emptyset\}, \tau = \{\emptyset\}$
15     while $\pi \neq V$ do
16       $N_{mul} = Const_{mul}, N_{ALU} = Const_{ALU},\ t{+}{+}$
17       for $i \in$ V do
18         for $i \in \tau$ do
19           if $i \in$ M do
20             $tr_i = tr_i - 1, N_{mul} = N_{mul} - 1$
21             if $tr_i = 0$ do
22               $\pi = \pi \cup i, \tau = \tau \cap \neg i, tf_i = t$
23               $\sigma \leftarrow$ successor nodes of op $i$ in V
24             endif
25           endif
26           if $i \in$ A do
27             $tr_i = tr_i - 1, N_{ALU} = N_{ALU} - 1$
28             if $tr_i = 0$ do
29               $\pi = \pi \cup i, \tau = \tau \cap \neg i, tf_i = t$
30               $\sigma \leftarrow$ successor nodes of op $i$ in V
31             endif
32           endif
33         endfor
34         if $N_{mul} \geq 1 \cap \{i | i \in (\sigma \cap M) \cap \min(p_i)\}$ do
35           $ts_i = t, N_{mul} = N_{mul} - 1, \sigma = \sigma \cap \neg i$
36           if $i \in Apx \cap T_{apx} = 1$ do
37             $\pi = \pi \cup i, tf_i = t$
38             $\sigma \leftarrow$ successor nodes of op $i$ in V
39           elif $i \in Apx \cap T_{apx} > 1$ do
40             $\tau = \tau \cup i, tr_i = T_{apx} - 1$
41           else $\tau = \tau \cup i, tr_i = T_{acc} - 1$ endif
42         endif
43         if $N_{ALU} \geq 1 \cap \{i | i \in (\sigma \cap A) \cap \min(p_i)\}$ do
44           $ts_i = t, N_{ALU} = N_{ALU} - 1, \sigma = \sigma \cap \neg i$
45           if $i \in A \cap T_{ALU} = 1$ do
46             $\pi = \pi \cup i, tf_i = t$
47             $\sigma \leftarrow$ successor nodes of op $i$ in V
48           else $\tau = \tau \cup i, tr_i = T_{ALU} - 1$, endif
49         endif
50       endfor
51     endwhile
52     if $\forall i,\ tf_i \leq Const_{time}$ do
53       $best\_ts_i = ts_i, best\_tf_i = tf_i, best\_Apx = Apx$
54       $\{M' = M' \cap \neg i\} \cap \{i \mid i \in M' \cap \max(e_i)\}$
55     else $\{Apx = Apx \cup i\} \cap \{i \mid i \in M' \cap \max(e_i)\}$
56       $\{M' = M' \cap \neg i\} \cap \{i \mid i \in M' \cap \max(e_i)\}$
57     endif
58     $\{Apx = Apx \cap \neg i\} \cap \{i | i \in M' \cap \max(e_i)\}$
59   endfor
60 end
```

error propagation from predecessor operations due to approximation (Lines 5-7). In this step, it is not possible to use error models such as [14, 15] for post-design analysis. Therefore, we adopt the error propagation model presented in [16] and [17]. Consider a typical multiplication $p = a \times b$ and let the errors from the inputs a and b denote ε_a and ε_b respectively. Let the errors of the multiplication and its product be denoted as $\varepsilon_\times$ and ε_p. The error is derived as Formula (1):

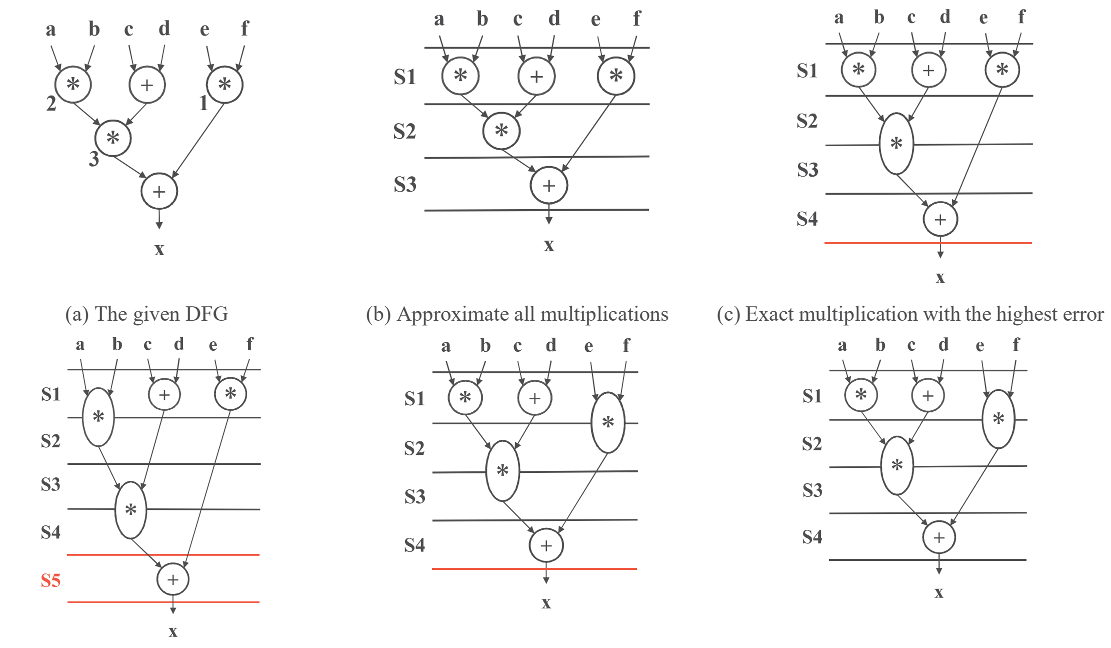
$$\varepsilon_p = a \times \varepsilon_b + b \times \varepsilon_a + \varepsilon_\times \tag{1}$$

Accuracy is evaluated by the magnitude of the error. In other words, the smaller the error, the higher the accuracy of the circuit.

At the same time, the priority p_i for each operation, i is created, based on t_alap_i (Lines 9-13). Next, the algorithm relaxes the schedule by multicycling the multiplication, where the multiplication with the highest priority is selected, to reduce the error at the output. The algorithm iteratively conducts its process while the time constraint is satisfied (Lines 15-50). Here, we introduce three sets of states as σ, τ, and π. Let σ denote a set of operations that are ready to execute, τ denote a set of operations during execution, and π denote a set of operations that are already finished. Initially, τ and π are set to empty. Let N_{mul} and N_{ALU} denote the number of available multipliers and ALUs, respectively. The resources N_{mul} and N_{ALU} are released after the execution of multiplication and an operation by ALU, respectively. tr_i represents the remaining cycles for each operation i. ts_i and tf_i represent the control states of start and finish or each operation i, respectively. If an operation i takes a single cycle, ts_i and tf_i indicate the same state. In Lines 18-34, functional units are ready to be executed and come to a set of ready operations σ under the resource constraint. If an operation in τ is finished, the operation is removed from τ and the resource is released. Then the operation becomes a member of π. In order of priority, the operations are started the execution (Lines 34-50). If the time constraint is missed, the schedule with multicycling the selected multiplication is infeasible and the multiplication is returned to approximate mode. Otherwise, the schedule is feasible and determined to be executed in accurate mode (Lines 52-57). The processes are repeatedly conducted until all the multiplication has tried to switch accurate mode from approximate mode. In the end, we obtain a feasible schedule such that the error is reduced under resource and time constraints. The computational complexity of our algorithm is $O(N^2 \log N)$ for the number of multiplications N at a worst case.

2.3. Proposed Algorithm Example

Our proposed method aims to minimize the error by accurately performing only those multiplications that cause large errors in the output. For the operation from step 8 to step 59 of Algorithm 1, an example problem in DFG with three multiplications and two additions is shown in Figure 2. We assume one cycle for approximate multiplications, two cycles for exact multiplications, and one cycle for additions. The resource constraint is list scheduling as two accuracy-controllable approximate multipliers and one adder. The time constraint is four cycles. Figure 2 (a) is the DFG given, and the number at the bottom left of the multiplication is assumed to be the magnitude of the influence of the error on the output value obtained in step 6 of Algorithm 1. First, as shown in Figure 2 (b), all the multiplications are approximate multiplications, and resource constraint-based list scheduling is performed. Next, the list scheduling is performed again with the multiplications with the largest errors made exactly. At this time, since all the operations can be performed within the time constraint of four cycles, the multiplication that was made exact in Figure 2 (c) is kept exact for the next scheduling. Next, as shown in Figure 2 (d), the multiplication with the second-largest error is made exact and list scheduling is performed. However, in Figure 2 (d) the time constraint of four cycles has been exceeded, so the upper left multiplication is returned to approximation. Next, the multiplication with the smallest error is made exact and list scheduling is performed. The result of Figure 2 (e) satisfies the time constraint, so we keep the multiplication as exact. Finally, since all the multiplications have been done correctly, the result of Figure 2 (e), which will have the smallest error so far, is output as the final solution as shown in Figure 2 (f). In this example, we have shown 1-2 cycles for simplicity, but our proposed method can be applied to operations with more than 3 cycles.



(a) The given DFG (b) Approximate all multiplications (c) Exact multiplication with the highest error

(d) Exact multiplication with the second highest error (e) Exact multiplication with the smallest error (f) Final output that satisfies resource and time constraints

Figure 2: An example of proposed list scheduling algorithm

3. Experiments

3.1. Setup

In the experiments, we have compared our heuristic algorithm to an existing ILP-based technique [9], which uses CPLEX 12.10 as a solver on a PC with an AMD Ryzen 7 PRO 4750G CPU and 64GB of main memory. We utilize MediaBench [18] for the benchmark programs. In the experiments, we are given resource and time constraints exhaustively to observe trade-offs. Note that the number of accuracy-controllable multipliers is restricted as a resource constraint but that of ALUs is ignored since sharing of ALUs largely incurs an overhead. The runtime for evaluation is limited to up to an hour in wall-clock time. If an optimal solution is not found within an hour, the best solutions at that time is employed to compare.

3.2. Results

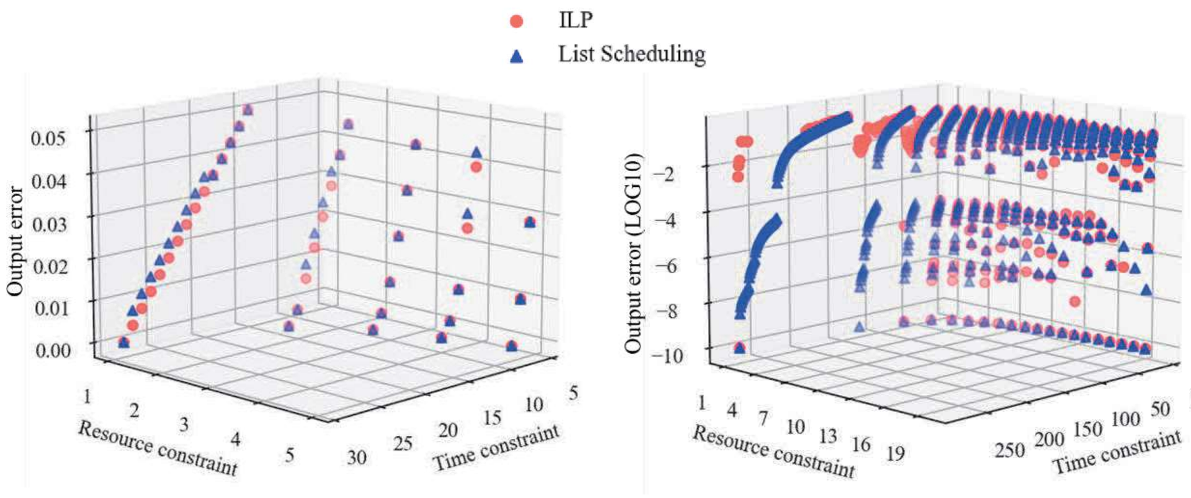
The experimental results are shown in Table 1 and Figure 3. Table 1 shows the comparison between our proposed algorithm and the ILP-based technique [9]. In the table, Nodes represent the total number of operations for each benchmark. Mult means the number of multiplications among them. Designs indicate the number of design problems for a variety of resource and time constraints. Wins, Losses, and Draws are the number of designs that our proposed algorithm that outperforms the ILP-based technique in accuracy even for just a bit, the ILP-based technique outperforms our proposed algorithm, and they find the same solutions, respectively. Figure 3 shows, on the other hand, the relationship between the output error, resource constraint, and time constraint in the cases of Motion Vectors Detector and Matrix Inversion. The abscissa axis represents the resource constraint, the ordinate axis represents the time constraint, and the applicate axis represents the error at the output. Note that the output error is expressed as the relative value of the error contained in the output to the accurate computation result, and the applicate axis is scaled in logarithm in Figure 3 (b).

The results in Table 1 indicate that our proposed algorithm finds almost the same errors as the ILP-based technique in the cases where the number of operations is less than a hundred. In addition, the cases have enabled the ILP-based technique to find optimal solutions; therefore, our algorithm explicitly obtains an optimal solution in many cases. In Motion Vectors Detector, our algorithm produces larger errors than the ILP-based technique in 15 designs. According to the results in Figure 3 (a), however, the output errors look slightly larger but almost the same since the output errors by each technique are negligibly small.

Over a hundred of the nodes, the ILP-based technique can hardly find an optimal solution within the runtime. In especially Matrix Inversion, Figure 3 (b) implies the ILP-based technique fails to find good solutions in many designs. In contrast, we demonstrate the effectiveness of our proposed algorithm since it can obtain flexible solutions under a variety of resource and time constraints. Table 2 shows the results of runtime evaluation between the ILP-based technique and our proposed algorithm. Max, Min, and Mean indicate the maximum, minimum, and average runtime of scheduling among the various constraints, respectively. In the cases where the number of operations is less than a hundred, the runtime to find optimal solutions is almost the same. However, in a large benchmark with hundreds of operations, our proposed algorithm is successfully finding a feasible solution in a shorter time than the ILP-based technique.

Table 1. COMPARISON BETWEEN OUR PROPOSED ALGORITHM AND ILP [9]

Benchmarks [18]	Nodes	(Mult)	Designs	Wins	Losses	Draws	ILP [9] exceeds an hour
HAL	11	(6)	14	0	0	14	0
FIR filter	21	(11)	19	0	0	19	0
Auto Regression Filter	28	(16)	36	0	0	36	0
Motion Vectors Decoder	32	(14)	37	0	15	22	0
Elliptic Wave Filter	34	(8)	16	0	3	13	0
Cosine	42	(14)	52	0	1	51	0
Feedback Points	53	(17)	43	0	10	33	1
Matrix Multiplication	109	(40)	129	1	20	108	4
Smooth Triangle	197	(69)	257	35	63	159	55
Matrix Inversion	333	(140)	516	235	104	177	257



(a) Motion Vectors Decoder (b) Matrix Inversion

Figure 3: The output error in Motion Vectors Decoder and Matrix Inversion with exhaustive constraints of resource and time

Table 2. RUNTIME EVALUATION BETWEEN ILP [9] AND OUR PROPOSED ALGORITHM (SEC.)

Benchmarks [18]	ILP [9]			Our algorithm		
	Max	Min	Mean	Max	Min	Mean
HAL	0.130	0.010	0.083	0.006	0.005	0.006
FIR filter	0.860	0.080	0.214	0.020	0.013	0.017
Auto Regression Filter	8.730	0.080	0.899	0.049	0.023	0.037
Motion Vectors Decoder	35.380	0.080	1.269	0.050	0.024	0.036
Elliptic Wave Filter	0.220	0.050	0.127	0.033	0.029	0.031
Cosine	2387	0.050	46.490	0.087	0.039	0.058
Feedback Points	>3600	0.080	85.617	0.138	0.066	0.100
Matrix Multiplication	>3600	0.200	149.748	2.044	0.527	1.092
Smooth Triangle	>3600	0.300	1155	15.804	2.865	7.204
Matrix Inversion	>3600	1.020	2116	170.288	16.217	70.913

4. Conclusions

We proposed a heuristic scheduling algorithm with variable-cycle approximate functional units in HLS. The proposed algorithm is polynomial, and we have demonstrated that it finds better solutions than existing techniques in a short time. Therefore, our algorithm is expected to be adopted in deep learning-based and image processing applications.

In the future, we plan to extend our algorithm to combine with other optimization techniques such as chaining, pipelining, and bit-width reduction. In addition, we are going to conduct a case study by the implementation of our algorithm into general HLS tools.

Acknowledgments

This work is supported partly by KAKENHI 20H00590, 20H04160, and 21K19776.

References

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys*, vol. 48, no. 4, 2016.

[2] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design & Test*, vol. 33, no. 1, 2016.

[3] T. Yang, T. Ukezono, and T. Sato, "A low-power configurable adder for approximate applications," *International Symposium on Quality Electronic Design*, 2018.

[4] T. Yang, T. Ukezono, and T. Sato, "A low-power high-speed accuracy-controllable approximate multiplier design," *Asia and South Pacific Design Automation Conference*, 2018.

[5] M. Sano, H. Nishikawa, X. Kong, H. Tomiyama, and T. Ukezoko, "Design of a 32-bit accuracy-controllable approximate multiplier for FPGAs," *International SoC Design Conference*, 2021.

[6] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," *Design, Automation & Test in Europe Conference & Exhibition*, 2014.

[7] S. Lee, L. K. John, and A. Gerstlauer, "High-level synthesis of approximate hardware under joint precision and voltage scaling," *Design, Automation & Test in Europe Conference & Exhibition*, 2017.

[8] J. C. Godínez, J. M. Vargas, M. Shafique, and J. Henkel, "AxHLS: Design space exploration and high-level synthesis of approximate accelerators using approximate functional units and analytical models," *International Conference On Computer Aided Design*, 2020

[9] K. Ohata, K. Shirane, H. Nishikawa, X. Kong, and H. Tomiyama, "Scheduling with variable-cycle approximate functional units in high-level synthesis," *International SoC Design Conference*, 2021.

[10] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, "On reconfiguration-oriented approximate adder design and its application," *International Conference on Computer-Aided Design*, 2013.

[11] V. Camus, J. Schlachter, and C. Enz, "A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision," *Design Automation Conference*, 2016.

[12] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," *Design, Automation & Test in Europe Conference & Exhibition*, 2014.

[13] C. H. Lin, and I. C. Lin, "High accuracy approximate multiplier with error correction," *International Conference on Computer Design*, 2013.

[14] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "MACACO: Modeling and analysis of circuits for approximate computing," *International Conference on Computer Aided Design*, 2011.

[15] W. T. J. Chan, A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Statistical analysis and modeling for error composition in approximate computation circuits", *International Conference on Computer Design*, 2013.

[16] J. C. Godínez, S. Esser, M. Shafique, S. Pagani, and J. Henkel, "Compiler-driven error analysis for designing approximate accelerators," *Design, Automation & Test in Europe Conference & Exhibition*, 2018.

[17] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint precision optimization and high-level synthesis for approximate computing," *Design Automation Conference*, 2015.

[18] C. Lee, M. Potkonjak, and W. H. M. Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," *International Symposium on Microarchitecture*, 1997.