Snow Ablation Optimizer Accelerator Based on High Level Synthesis

Maoshuo He*[1], Renjing Hou*[2], Zirui Li [2], Kang Zhao †[2],
[1]Xidian University [2]Beijing University of Posts and Telecommunications
maoshuohe@stu.xidian.edu.cn, {hourj, lzr_official, zhaokang}@bupt.edu.cn

***Abstract*—How to reduce the execution time in the snow ablation optimizer (SAO) is a key problem. However, it is very hard because the space for achieving time optimization through code modification is limited. To resolve this issue, this paper utilizes the Vitis HLS tool to deploy the SAO onto the FPGA and optimize the SAO's execution time. Vitis HLS is a key tool in high-level synthesis (HLS). Until now, there have been few attempts to optimize the execution time of this type of algorithm through HLS. This paper proposes a pragma integration model combined with a modified linear feedback shift register (LFSR) algorithm to reduce execution time. The experimental results show that this approach reduces the latency to 44.50% while maintaining an acceptable convergence error, compared to previous optimizations.**

***Index Terms*—High-Level Synthesis, Snow Ablation Optimizer, FPGA, Accelerators, Linear Feedback Shift Register**

I. Introduction

In the field of very large-scale integration (VLSI), high-level synthesis (HLS) [1] has emerged as a crucial technology and is now extensively adopted by VLSI design companies. When contrasted with the traditional register transfer level (RTL) design approach, HLS exhibits remarkable technical advantages in VLSI development. Its core value lies in the elevation of the design abstraction level: by leveraging high-level languages such as C/C++/SystemC to depict algorithmic behavior, HLS redirects the design emphasis from the microcontrol of sequence and circuit structure to the efficient implementation of functional logic. This can significantly abbreviate the development cycle and reduce the threshold for hardware development.

At the optimization stage, the HLS tool deploys strategies including pipeline, loop unroll, and array partition to generate a pareto-optimal architecture with respect to throughput, latency, and hardware resource consumption. For example, as illustrated in Fig. 1, in this case, the HLS tool initiates the next operation before the current one is fully completed. Suppose the major cycle consists of three cycles and the initiation interval is set to 1. Once one cycle is finished, the next major cycle commences. This approach can effectively reduce latency. Through synthesis directives, other structures like arrays can also be manipulated. Arrays can be synthesized into register-based random access memory (RAM) or fully

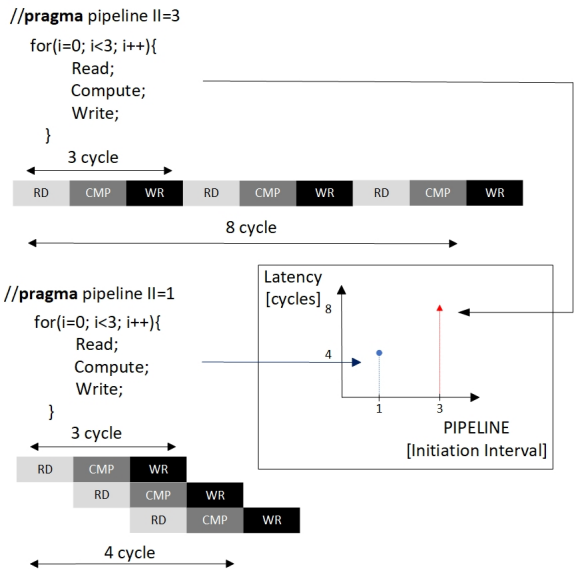

Fig. 1 The pragma of pipeline on hardware circuit generated by HLS

expanded into independent trigger circuits. Now there are numerous cases [2], [3] have demonstrated that algorithms implemented on FPGAs can be efficiently accelerated using Vitis HLS.

The snow ablation optimizer (SAO) [4] is a newly proposed innovative meta−heuristic algorithm. In comparison with other heuristic algorithms, it demonstrates a superior ability to balance the exploration and exploitation processes. This enables it to effectively avoid premature convergence and achieve more favorable outcomes. Moreover, it exhibits remarkable global search capabilities and strong general applicability. Nevertheless, the SAO has a relatively high time complexity because it uses dual−population mechanism and the dependence on random number generation for convergence. When addressing high−dimensional or large−scale optimization problems, it consumes the longer processing time for handling such problems because it has the substantial number of iterations and the extended convergence time. Consequently, the reduction of the SAO's execution time has emerged as an urgent issue.

In this work, we have proposed an effective optimization method to achieve better performance. The main contributions of this paper include:

*Co-first authors with equal contribution.
†Corresponding author.
This work is supported in part by National Key R&D Program of China (2022YFB2901100) and Beijing Natural Science Foundation under Grant 4244107.

- A pragma integration model designed based on the latency of the objective function.
- A random number generation model based on HLS for LFSR combined with the pragma integration model
- Based on the optimization of the previous two points, the experiment shows that the latency has decreased by 44.50%.

The rest of the paper is organized as follows: Section II introduces the preliminary knowledge. Section III presents the proposed methodology. Section IV gives the experimental results. Section V concludes the paper.

II. Preliminaries

A. Vitis HLS

To shorten the execution time of SAO, we contemplated implementing SAO on an FPGA. In contrast to the conventional RTL design approach, HLS enables rapid algorithm deployment. Consequently, we decide to utilize HLS for this deployment. There are numerous HLS tools available. ROCCC HLS tool can transform C programs into hardware accelerators. GAUT can convert C programs into a pipeline structure that satisfies specific constraints. LegUp HLS can take C programs as input and automatically convert them into a hybrid system. Vitis HLS is capable of automatically converting C/C++/OpenCL code into an optimized RTL implementation. Through the comprehensive comparative analysis, we determined that Vitis HLS provides a more extensive library set in comparison to other HLS tools. Its framework integrates C simulation, C/RTL co-simulation, and waveform analysis within a unified IDE, enabling accelerated hardware implementation.

Vitis HLS, an advanced HLS tool developed by Xilinx, supersedes Vivado HLS and features enhanced IDE integrations. Its architecture comprises a front-end and a back-end. The open-source front-end, available on GitHub, supports custom optimization plugins and deep integration with the Vitis AI and Vitis vision libraries. The back-end implements FPGA device-specific resource and timing path optimizations. The operation process of the Vitis HLS tool is shown in Fig. 2.

B. Snow Ablation Optimizer

The Snow Ablation Optimizer (SAO) is a novel metaheuristic algorithm inspired by snow melting phenomena. It simulates two physical processes: snow formation and melting, as shown in Fig. 3. During the exploration phase, SAO employs Brownian motion to model the stochastic diffusion of water vapor. During exploitation, it utilizes the degree-day method [5] to update individual positions relative to the swarm centroid. It incorporates a dual-population mechanism to maintain a dynamic equilibrium between global exploration and local exploitation throughout the optimization process.

The position update equation of the entire SAO algorithm is shown as Equation (1):

$$Z_i(t+1) = \begin{cases} \text{Elite}(t) + BM_i(t) \otimes \big[\theta_1(G - Z_i) \\ + (1-\theta_1)(\bar{Z} - Z_i)\big], i \in index_a \\ M \times G(t) + BM_i(t) \otimes \big[\theta_2(G - Z_i) \\ + (1-\theta_2)(\bar{Z} - Z_i)\big], i \in index_b \end{cases} \tag{1}$$

where $index_a$ and $index_b$ denote a set of indexes that include the line numbers of individuals in P_a and P_b throughout the position matrix.θ_1 and θ_2 denote a number randomly produced in $[0, 1]$. $Elite(t)$ denotes the elite pool. $Z_i(t)$ denotes the ith individual during the tth iteration. $\overline{Z}(t)$ denotes the centroid position of the whole swarm.

Although SAO exhibits exceptional global search capabilities and strong universality, it has high time complexity, as shown in Equation (2). It leads to scalability challenges. When applied to high-dimensional or large-scale optimization problems, the algorithm requires extensive iterations and prolonged convergence time. Consequently, the efficient hardware acceleration of SAO becomes imperative.

$$O\left(N * Dim + N * t_{\max} * (\log N + Dim + 1)\right) \tag{2}$$

where N denotes the number of search agents. Dim denotes the dimension.

C. Linear Feedback Shift Register

The convergence of SAO depends on random number generation. Compared to alternatives like the linear congruential generator [6], mersenne twister [7], and wichmann-hill [8] algorithms, the linear feedback shift register (LFSR) offers a key FPGA implementation advantage. It requires only shift registers and XOR gates. This eliminates the necessity for complex arithmetic units such as multipliers and dividers, thereby significantly reducing resource consumption. Therefore, we employ LFSR as the random number generation algorithm for the implementation of SAO.

The linear feedback shift register (LFSR) generates pseudorandom numbers. It generates random numbers from initial seeds via linear feedback and tap-selected XOR operations. At the same time, it will create a new seed for future use. This feedback can shifts the register states, as formalized in Equation (3). This architecture enhances the algorithm's performance longevity while maintaining superior optimization efficiency on FPGAs.

$$s_n = s_1 \oplus s_2 \oplus ... \oplus s_m \tag{3}$$

where s denotes the data of each bit. $\oplus$ denotes the XOR operation.

III. Methodology

A. Pragma Integration Model

The SAO implementation on FPGA achieves accelerated execution and enhanced resource efficiency via pragma-directed HLS optimizations. However, algorithm-specific

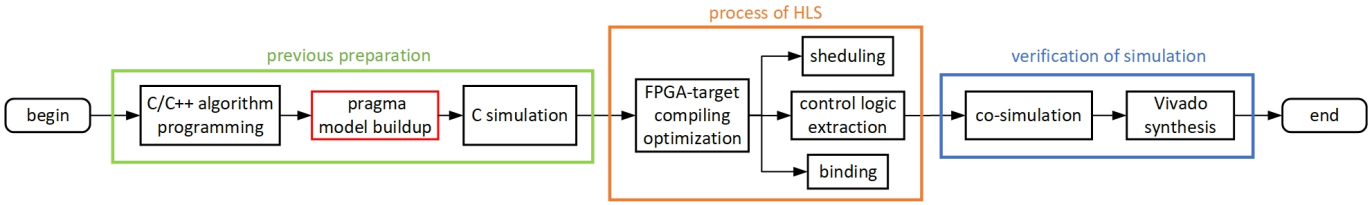

Fig. 2 The process of Vitis HLS

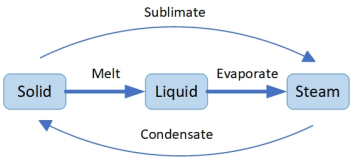

Fig. 3 Three-state change diagram

characteristics dominate pragma optimization outcomes. Systematic exploration of pragma interdependencies is therefore critical for optimizing SAO hardware performance.

In Vitis HLS, diverse pragmas optimize algorithm implementations. Directly applying dataflow pragmas to the main function of SAO is infeasible, because SAO has the prevalent imperfect loop nesting and its subsequent loop iterations depend on prior computations. Although temporary arrays can decouple dependencies, they incur prohibitive resource overhead. To resolve this, we propose a hierarchical pragma integration model that can significantly reduce latency in SAO.

Taking init_loop as an example, the init_loop needs to call the objective function during its operation. Each objective function has the different running cycles. For high-dimensional or large-scale optimization problems, this variability leads to significant latency overhead. The entire process of the init_loop algorithm is summarized in Algorithm 1.

Algorithm 1 init_loop process

Require: $function$,$current\_best\_score$,$current\_best\_pos$

for $i = 0 \rightarrow N-1$ **do**
  obj_val = $objective\_function(X[i])$
  $objective\_function[i] = obj\_val$
  if $obj\_val < current\_best\_score$ **then**
    $current\_best\_score = obj\_val$
    for $j = 0 \rightarrow dim$ **do**
      $current\_best\_pos[i]$=$X[i][j]$
    end for
  end if
end for

Based on the above pseudo-code, we significantly accelerate the overall running speed through the micro-combination of pipeline and unroll. Taking the three-dimensional sphere function as the objective function as an example, analysis shows that the clock cycle for calling the objective function once is shown in Equation (4)

$$L_y = L_m + \log_2(\text{dim}) \cdot L_a \tag{4}$$

where L_y represents the total cycles, L_m represents the cycle of each level of multiplication, L_a represents the cycle of each level of addition, dim represents the dimension of the function.

In this example, we use the pragma directive inline to specify inlining of the target function and use the pragma directive unroll to expand the entire for loop. We obtained the total latency for one cycle is 3. In the SAO algorithm, multiple data need to be read simultaneously. Since each block of RAM has only two data ports at most, the loading operation cannot be completed within a single cycle. We propose a model that can minimize the initiation interval of the pipeline in SAO. This model can significantly reduce the latency. Taking the three-dimensional sphere function as an example, in the init_loop, we set the initiation interval of pipeline to 3, which is the total latency number. The unroll factor should be divisible by the number of search agents. We control throughput at 1.6 data per cycle. In thie case, this approach partitions combinational logic depth to prevent single-cycle path timing violations and reduce critical path delay. It can also avoid the explosion of fully expanded resources, resulting in a significant decrease in latency.

B. LFSR Based on HLS

The SAO algorithm mainly reduces the solution space through random generation. It generates random numbers to search for the approximate path and conducts detailed search through population segmentation. The entire process can consume a large amount of latency during the generation of random numbers. Therefore, the processing of random number generation is crucial for reducing latency.

The initial implementation of the LFSR algorithm generates integers ranging from 0 to 2^{16} through a linear feedback shift register and then divides by 2^{16} to obtain decimal numbers between 0 and 1. However, this method consumes a large amount of multipliers during repeated calls. It will significantly reduce the operational efficiency.

To further reduce latency, we optimize the LFSR algorithm. The optimization is summarized in Fig. 4. To reduce the risk of overflow and truncation, we generate a 16-bit fixed-point number as the required random number. We initially generate a 32-bit binary number using LFSR, where the first 16 bits represent the integer part and the last 16 bits represent the fractional part. Setting the upper 16 bits to zero, we obtain a random number in the range [0,1]. Based on the

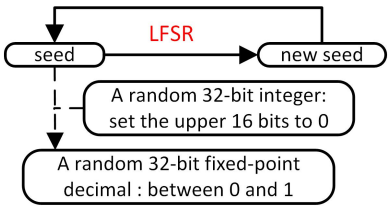

Fig. 4 The process of LFSR

analysis, the operation of generating random numbers can be completed in just 3 clock cycles. To adapt LFSR for HLS, we employ inline directives to enable function inlining. Through the analysis of the LFSR function implementation, we have developed a random number generation algorithm that can be executed within a single cycle in Vitis HLS. This algorithm exclusively employs operations such as shifting and concatenation. These operations are inherently free from additional latency. It does not involve multiplication or division operations that are unsuitable for FPGA implementation. By integrating this algorithm with the pragma integration model, we have achieved a significant reduction in latency.

IV. Evaluation

A. Experimental Environment

We implemented the FPGA deployment of the SAO algorithm based on Vitis HLS, and achieved another FPGA acceleration by modifying its code and fine-tuning pragmas. All test cases are written in C/C++. All experiments were conducted on an Ubuntu 20.04.6 LTS system, with the CPU being an Intel (R) Core (TM) i5-12500H @ 3.1GHz.

B. Experimental Setting

We use the Ackley function [9] as a test case. The Ackley function is a multi-modal test function often used to evaluate the performance of optimization algorithms. It has one global minimum point and multiple local minimum points, which is shown in Equation (5)

$$f(\mathbf{x}) = -a \cdot e^{-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(c \cdot x_i)} + a + e \quad (5)$$

where a,b.c denotes the constant, n denotes dimension, x_i denotes input parameter.

Because the global minimum point of the arkley function is at the origin (0, 0, 0), we use the distance from the optimal convergence point to the origin as the evaluation criterion, which is shown in Equation (6)

$$score = x^2 + y^2 + z^2 \quad (6)$$

where x, y, z denotes the coordinate of the outcome.

C. Experimental Result

We set the dimension of the SAO algorithm to 3, the number of search agents to 50, and the maximum number of iterations to 2000. To minimize data precision loss, all data are represented as 32-bit fixed-point numbers. The convergence results obtained before and after optimization are shown in Fig. 5. The final result is shown in TABLE I.

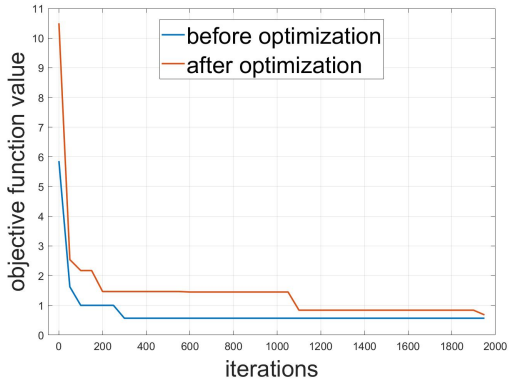

Fig. 5 The result of the snow ablation optimizer

TABLE I result of accelerator

Result	Best Score	Latency(s)	Improvement (%)
before optimization	0.57	0.1827	—
after optimization	0.65	0.1014	44.5%(Latency ↓)

According to TABLE I, the error of its best score is 14.04%, which is within the acceptable range. Compared with previous optimizations, the optimization method can reduce latency by 44. 50%.

V. Conclusion

In this work, we propose an implementation and optimization of the SAO algorithm based on HLS, which will be beneficial for the future optimization of algorithms involving random number generation using the Vitis HLS tool. By proposing the pragma integration model and optimizing the LFSR function, we achieve a reduction in its latency.

References

[1] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An introduction to high-level synthesis," *IEEE Design & Test of Computers*, vol. 26, no. 4, pp. 8–17, 2009.

[2] M. A. Elhewehy, K. O. Abbass, and O. A. Nasr, "Hardware-software co-design implementation of fixed-point googlenet on soc using xilinx vitis," in *5th Novel Intelligent and Leading Emerging Sciences Conference (NILES)*, 2023, pp. 274–278.

[3] M. Zou, L. Ma, and J. Li, "Implementation and optimization of polyphase channelization using vitis hls," in *IEEE 12th International Conference on Information, Communication and Networks (ICICN)*, 2024, pp. 42–46.

[4] L. Deng and S. Liu, "Snow ablation optimizer: A novel metaheuristic technique for numerical optimization and engineering design," *Journal of Expert Systems with Applications*, vol. 225, p. 120069, 2023.

[5] G. Zhou, M. Cui, J. Wan, and S. Zhang, "A review on snowmelt models: progress and prospect," *Sustainability*, vol. 13, no. 20, p. 11485, 2021.

[6] I. Borosh and H. Niederreiter, "Optimal multipliers for pseudo-random number generation by the linear congruential method," *BIT Numerical Mathematics*, vol. 23, no. 1, pp. 65–74, 1983.

[7] M. Matsumoto and T. Nishimura, "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998.

[8] B. McCullough, "Microsoft excel's 'not the wichmann–hill'random number generators," *Computational Statistics & Data Analysis*, vol. 52, no. 10, pp. 4587–4593, 2008.

[9] D. Ackley, *A connectionist machine for genetic hillclimbing*. Springer science & business media, 2012, vol. 28.