DOI: 10.21917/ijme.2023.0259

ANALYSING THE PERFORMANCE TRADEOFFS OF PARAMETERIZED VLSI ARCHITECTURE USING TREE-LOGIC MACHINE LEARNING SYSTEM

T. Gobinath[1], A. Sumalatha[2], M. Kumar[3] and M. Dharani[4]

[1]*Department of Computer Science and Engineering, Chettinad College of Engineering and Technology, India*
[2]*Department of Computer Science, Kristu Jayanti College, India*
[3]*Department of Electronics and Communication Engineering, Chettinad College of Engineering and Technology, India*
[4]*Department of Electronics and Communication Engineering, KSR Institute for Engineering and Technology, India*

Abstract

This paper deals with the design of a tree logic machine learning system agent that is based on a hybrid technique that blends graph handmade properties with graph neural network embeddings. It is a combination of traditional reinforcement learning and deep learning that involves the substitution of tree logic machine learning system in the learning process. The proposed agent uses a new EMAC-TLML system that is compatible with the inference made by deep learning networks that use 8 bits of precision, and we demonstrate that this compatibility works quite well. It has been demonstrated that the proposed research uses resources and produces energy delay products in a manner that is analogous to that of their floating-point counterparts. The solution that has been suggested provides a greater maximum working frequency in contrast to the floating-point method.

Keywords:

Machine Learning, Tree Logic, Performance Trade-off, VLSI

1. INTRODUCTION

The quality of the VLSI placement has significant repercussions on the quality of the design as well as the completion of the design in the procedures that come after it. These procedures are referred to as the physical design. However, recent research [1] has shown that the placers that are now in use are unable to produce solutions that are nearly ideal. This is the case because the placers cannot find the optimal solution [2].

The placement job is to search the chip for regions that are suitable for the placement of the various types of cells and then find those areas. The primary goal of today modern placers is to reduce the overall interconnect length, which is defined by the estimated half perimeter wire length (HPWL) between each adjacent pair of cells. This measure is used to characterise the total interconnect length [3].

The users of the EDA tools can modify the parameters of the algorithms that are used within the tools so that they can obtain the power performance area (PPA) that they require. The authors have made the startling discovery that the time necessary to adjust and operate a commercial placer after it has been initially created takes far longer than the time necessary to develop the system itself [4]. The ever-increasing complexity of both the tools themselves and the flows that they are meant to support is demonstrated by the fact that contemporary location and route tools contain more than 10,000 distinct parameter options. The most modern nodes of technology are not only expensive but also dangerous, and as a result, their implementation requires the assistance of qualified professionals [5].

When tuning, one will frequently rely on experience and domain knowledge because the design space of the parameters is typically much too wide and complex to be explored by a single human engineer working alone [6]. On the other hand, it is not impossible for the interactions between the many different components and the PPA that is produced as a result to be convoluted or to run counter to what one would anticipate.

As a result of the fact that placement engines are dependent on the rules and metaheuristics that were developed by humans, it is feasible for these systems to exhibit tendencies that are nondeterministic. Additionally, it is not always the case that the aim that is specified for a parameter is the same as the actual metric that is being measured. This is something else to keep in mind when working with parameters [7].

It is possible to improve the quality of results (QoR) in EDA tools like [8] and [9] for FPGA and high-level synthesis (HLS) compilation flows with the assistance of a cutting-edge tool known as auto-tuner [10]. It does this by coordinating a few conventional approaches to optimisation and by employing a method to carry out an effective exploration of the design space.

These methods are based on heuristics that are way too generic, and therefore, they ignore the details that make each netlist one of a kind. The parameter needs to begin all over again from the very beginning with each netlist.

We can avoid this limitation in our RL agent by employing a hybrid technique that blends graph handmade properties with graph neural network embeddings. This allows us to overcome the issue. As a result of this, you will be able to reduce costs and save time during the placement phase of the project by adopting a tuning method that is more adaptable and applying it across several different netlists [11].

By training and inferring the network using a numerical representation that has a low level of precision, it is possible to lessen the weight of a trained network without having to resort to quantization as a solution [12]. This is made possible by the fact that the weight can be reduced. In earlier studies, comparisons and contrasts were made between low-precision and high-precision floating-point implementations of DNN inference.

This research compares a numerical representation with varying bit-widths, which leads to an inaccurate picture of the effectiveness of the network. To become an expert in an optimisation strategy for identifying placement parameters that, when applied, result in the minimum wire length that is feasible after placement is the ultimate objective of our framework.

2. KERNEL ADAPTIVE FILTERS

Traditional adaptive filter methods such as LMS, RLS, and their variants take it as a given that there is linearity between the input un and the goal output d_n when they are used to activities

such as system identification and regression. This is because these approaches assume that there is a relationship between the two variables. When we apply this formula to the facts that have been presented to us, we get the result:

$$d_n = u_n^T h^* + v_n, \quad (1)$$

where $h^* \in R^L$ is the amount that is unknown to us. It is necessary to compute the parameter vector h, and $u_n = [u_n, u_{n-1},...,u_{n-L+1}]^T$. T is a vector that represents the input signal, and n is the variance-squared noise σ_v^2 that is produced because of the observations; this noise has a mean value of zero. Nonlinear models are typically found in a wide variety of engineering problems in applications that take place in the real world. The equation can be used to model input-output interactions,

$$d_n = f(u_n) + v_n \quad (2)$$

where the function $f : R^L \rightarrow R$ is a continuous nonlinear function.

The ability of linear adaptive filters to deal with input-output interactions that are as complex as these is severely restricted. As a result of the study that has been conducted on this subject, a number of different approaches to estimating $f(\cdot)$ from the data pairs $\{u_i\}_{i=1:n}$ have been proposed. These approaches have been discussed.

The kernel adaptive filters that operate in the replicating kernel Hilbert space (RKHS) is becoming increasingly common as a direct result of the relative simplicity that is required to put them into mathematical implementation.

Kernel approaches involve mapping the input regressors u_i = 1:n onto a high-dimensional feature space that is denoted by the symbol $\varphi(u_i)$. This is done so that the inner products of a nonlinear function can be calculated. For a kernel function to fulfil the requirements of Mercer criterion, it must be continuous, have symmetric behaviour, and have a positive-definite value.

$$\kappa(\cdot,\cdot) : R^L \times R^L \rightarrow R \quad (3)$$

Even if the mapping $\varphi(\cdot)$ is unknown, it is possible to construct inner products in higher-dimensional space by evaluating kernel functions and using those results in the construction. If a kernel can satisfy the requirements and then it is referred to as a replicating kernel.

$$\kappa(u_i,u_n) = \varphi^T(u_i)\varphi(u_n). \quad (4)$$

Given that H provides the specification of the replicating kernel, we shall refer to the corresponding inner product as H.

$$\{u_i, d_i\}_{i=1}^{n-1} \bigcup \{u_n\}. \quad (5)$$

The u_i is provided for the user interface evaluation. This work is solely concerned with the Gaussian kernel, which is a well-known Mercer kernel due to the universal approximation capabilities that it provides. This kernel is the only item that this work concentrates on. When we consider the pair u_i, we get the result, which is the assessment of the output of the nonlinear model. The representer theorem is given as below:

$$d_n = \sum_{i=1}^{n-1} \alpha_i \kappa(u_n, u_i) \quad (6)$$

Since the model order is increasing, kernel approaches are becoming increasingly inappropriate for usage in real-time applications. Several different sparsification processes, which can be used to acquire a lexicon of constant size, can be put into effect to get over this issue. These procedures can be used to learn words.

These sparsification algorithms make use of a similarity metric that compares the candidate to the current vocabulary to decide whether to include the candidate regressor in the existing dictionary.

These methodologies are not appropriate for usage in dynamic situations since dictionary training needs to be performed every time the underlying system experiences a change. This makes it impossible to employ these approaches.

2.1 TLML

The method of machine learning known as multi-agent reinforcement learning (MARL) needs a group of agents to collaborate with one another to maximise an expected reward signal that is produced by the agent interactions with a certain environment. This signal is formed because of the agent interactions with the environment.

The Markov game (MG) model is one that is used rather frequently to describe scenarios such as the one that we are looking at now. The agents start off knowing nothing at all about this model; hence, they make the decision to learn as much as they possibly can about it to improve their chances of effectively obtaining incentives and carrying out the policies defined by the model.

In an n-agent Markov game, the value of M is the tuple S,

$$M = \langle S, A_1,\ldots,A_n, P, R_1,\ldots,R_n, \gamma \rangle \quad (7)$$

where S represents the state space, $A_1,\ldots,A_n$, P, $R_1,\ldots,R_n$, represent the initial circumstances, and Rn represents the ultimate payout and S is the state space.

Ai denotes the action space of any agent i, where i can have any value between $i \leq n$; this is because i can have any value between $i \leq n$.

P is the transition function, where $P_a \in ss'$ is the probability of moving from state $s \in S$ to state $s' \in S$ via the joint action a = $\langle a_1,\ldots,a_n \rangle$, where all $a_i \in A_i$ are the simultaneous actions done by the agents, and where P is the chance of shifting from state s to state s'.

Every single MARL agent works in accordance with the procedure that she has laid out for herself. Deterministic policies are utilised throughout the entirety of this body of work, and the policy of agent i is defined as $\pi_i : S \rightarrow A_i$. Every agent is always looking for the optimal strategy that will maximise the rewards are anticipating obtaining and has found the right one. To sum everything up, the representation of a group policy is the tuple $\pi = \langle \pi_1, \pi_2, ..., \pi_n \rangle$.

The research provides a significant emphasis on fully cooperative problems, which are situations in which all the agents work together to increase the ultimate payoff for the group. We recommend utilising the Independent Q-Learning (IQL) method since it is straightforward to construct as well as the fact that it is a extension of the tabular Q-Learning that was usually utilised in previous single-agent shielding-based techniques.

According to IQL, the values of the state-action pairs for agent i ought to be modified to reflect the following changes: The learning rate is between 0 and 1 is defined as below:

$$Q_{i,t+1}(s,a_i) \leftarrow Q_{i,t+1}(s,a_i) + \alpha \begin{bmatrix} r_i, t+\gamma \cdot \max a_i' \in A_i Q_{i,t}(s',\alpha_i') \\ -Q_{i,t}(s,a_i) \end{bmatrix} \quad (8)$$

where $0 < \alpha \leq 1$ is the learning rate.

When an agent that uses IQL has finished the learning process, the optimum policy for that agent will be to return to the state-action pair in her Q-table that has the greatest Q-value. This will be the agent default state. Deep reinforcement learning, which is a blend of traditional reinforcement learning and deep learning, is utilised in this work. The scalability of conventional tabular RL algorithms is going to be significantly improved because of our work.

Independent deep Q-learning, commonly known as IDQL, is an extension of IQL that involves the substitution of neural networks for the Q-tables used by the agents in the learning process. Safeguard RL. Our method makes use of a shield to make certain that it abides by all the relevant safety rules, both in actual use and when being tested. The agents are protected by a shield, which stops them from taking any acts that could put them in danger if the shield ever malfunctions.

The shield is frequently viewed as a go-between for the agents and the environment, and its primary objective is to restrict the agents to actions that are safe. This is since the shield one and only function is to shield the agents from the surrounding environment. It is crucial to bear in mind that the ideal shield would have a minimal level of intrusiveness and would not hinder the agent capacity to freely study their surroundings.

If this is the case, the shield may make it difficult for the agents to determine which course of action will have the greatest impact. As a result of this, a wide variety of various kinds of protection measures incorporate a mechanism that punishes agents for trespassing in the system.

2.2 EXACT MULTIPLY-AND-ACCUMULATE (EMAC)

The operation that is known as multiply-and-accumulate (MAC) is an action that is carried out by each neuron in a DRL to carry out a weighted sum of the inputs that it receives. This is an imprecise operation that, in most implementations, leads in an increase in inaccuracy of rounding or truncation. This is because most implementations round or truncate the numbers.

The EMAC can get around this problem by utilising a modified version of the Kulisch accumulator to carry out an aggregation of the products of all layers in advance of the insertion of an error. This allows the EMAC to avoid the occurrence of the problem. When precision is low, there is an increased need to cut down on the number of errors that crop up in a particular location. A fixed-point value is accumulated in a broad register throughout the processing of each EMAC module. After that, during a delayed stage, the value is rounded to the nearest integer and stored. Calculating the accumulator width, also known as W_a for k multiplications can be done as follows:

$$W_a = \lceil \log_2(k) \rceil + 2 \times (l/\log_2 \max(\min(m))) + 2 \quad (9)$$

where, max and min refer to the maximum and minimum value magnitudes for a particular numerical system, while l refers to the length of the accumulator. The value that stands for the width of the accumulator is denoted by W_a.

Multiplication, accumulation, and rounding are operations that are carried out in turn on each EMAC as part of a pipeline that operates in a sequential fashion. The fourth step of development involves the creation of a fundamental activation function for the neurons in the buried layer. This function may be represented as:

$$ReLU(x) = \max(x, 0). \quad (10)$$

3. EXPERIMENTS

As can be seen in Table 5, our agent training and evaluation will centre on the following benchmark designs: 15 from OpenCores, two from the ISPD 2012 contest, and two RISC-V single cores. The first eleven are utilised for teaching purposes, while the remaining four are submitted to an examination. The Synopsys Design Compiler is utilised to facilitate the process of doing the synthesis of the RTL netlists. The 28nm technology node that TSMC must provide is the one that we employ.

The placements are performed with the help of the software version Cadence Innovus 17.1. The floor layouts have an aspect ratio of 1, and the proper fixed clock frequencies have been chosen for the various clocks. The manual placement of memory macros in advance is required by both RocketTile and OpenPiton Core. The parameter max density, which represents the ratio of the total cell area to the floorplan area, has a minimum value that has been assigned to it. This is done to ensure that the best possible location is obtained. The IO pins will be positioned in the intermediate layer, which is located between layer 4 and layer 6 and will be done automatically by the tool.

The ability of our agent to generalise is evaluated by observing how well it performs on four different test networks that it has never been acquainted with previously. The agent will eventually enhance a random starting parameter set through the process of reinforcement learning. This will be accomplished by the agent repeatedly selecting the action that would result in the greatest anticipated value.

The actions are deterministic, the parameters that were generated are already known and have been transmitted back to the network. This information can no longer be changed. We are going to continue to iterate until the estimated value has reduced for three iterations in a row, at which point we are going to revert to the settings that produce the best results. It is feasible to discover the set of parameters that define a good candidate by making use of this technique. This allows one to avoid the obligation of carrying out an actual placement. Following the selection of a particular set of parameters, we carry out a single placement and make a notation of the wirelength that is generated as a direct result of this action.

On the other hand, for the TLML to be able to acquire the reward signal and suggest a new set of parameters, it is necessary for actual placements to be finished. This is one of the requirements. The TLML has been provided with the option to carry out a total of 50 rounds of iteration to discover the optimal wire length.

We have made the surprising discovery that a single installation of our TLML agent is all that is required to produce dramatically improved wire lengths. The Table.1-Table.4

provides TLML recommendations for the best possible values that can be used for each of many parameters. It should come as no surprise that the two optimizers, WL, and congestion, used different approaches to reduce the amount of HPWL that was produced as in Table.1-Table.3.

Table.1. Hardware Complexity

Component	Area (mm^2)	Complexity (kGE)	Percentage (%)
10	0.0765	59.9613	44.0872
20	0.0446	34.6053	25.4528
30	0.0358	27.7288	20.3874
40	0.0048	3.8838	2.8571
50	0.1617	126.1792	92.7845
60	0.0010	0.9782	0.7167
70	0.0048	3.7579	2.7603
80	0.0019	1.2785	0.9395
90	0.0077	6.0145	4.4165
10	0.1695	132.1937	96.8523

Table.2. Total Instruction per Cycle

Round	Number of Instructions ($\times 10^{-7}$)	Deviation ($\times 10^{-7}$)	Deviation (%)
10	7.719	6.935	0.000
20	7.913	1.598	0.019
30	9.598	1.501	0.019
40	4.378	5.627	0.010
50	1.801	3.826	2.063
60	2.024	8.174	3.913
70	1.104	1.201	0.107
80	3.632	1.356	0.039
90	0.988	5.559	0.058
100	2.354	8.591	0.039
120	1.153	4.320	0.039
140	1.230	4.630	0.039
160	4.610	6.818	0.010

Table.3. Area and Energy

Design	Area	Delay	Power	Energy	ADP ($\times 10^{-7}$)	EDP ($\times 10^{-8}$)
10	157.48	2.20	44.75	101.60	3.57	2.31
20	91.72	1.13	24.21	28.38	1.08	3.32
30	98.60	1.14	24.70	29.06	1.16	3.44
40	87.36	1.14	23.34	27.51	1.03	3.24
50	91.14	1.15	24.12	28.67	1.08	3.41
60	90.17	1.14	24.12	28.57	1.07	3.36
70	109.44	1.14	27.89	32.83	1.29	3.88
80	109.73	1.08	26.34	29.44	1.23	3.31
90	94.53	1.08	23.63	26.44	1.06	2.96
100	83.68	1.15	22.86	27.22	1.00	3.23
120	92.01	1.08	23.63	26.44	1.03	2.96
140	102.13	1.08	24.985	27.94	1.145	3.135
160	87.845	1.115	23.245	26.83	1.015	3.095

4. CONCLUSION

We show that the new EMAC-TLML system is compatible with the inference made by deep learning networks that use 8 bits of precision, and we demonstrate that this compatibility works quite well. It has been demonstrated that the proposed research uses resources and produces energy delay products in a manner that is analogous to that of the floating-point counterparts. The solution that has been suggested provides a greater maximum working frequency in contrast to the floating-point method.

The performance reduction that is brought on by direct quantization to ultra-low precision is substantially less severe as compared to that which is brought on by fixed-point. In addition, EMAC-TLML consistently matches the floating points across a broad variety of datasets. The capabilities of the underlying technology have a direct consequence on the possibilities that can be realised by different kinds of learning algorithms, and these possibilities are directly influenced by the capabilities of the underlying technology.

REFERENCES

[1] C. Rao, Y. Zhang and X. Lou, "An Energy-Efficient Accelerator for Medical Image Reconstruction from Implicit Neural Representation", *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 78, No. 2, pp. 1-14, 2022.

[2] C. Kumar and K. Naik, "Smartphone Processor Architecture, Operations, and Functions: Current State-of-the-Art and Future Outlook: Energy Performance Trade-Off: Energy Performance Trade-Off for Smartphone Processors", *The Journal of Supercomputing*, Vol. 77, pp. 1377-1454, 2021.

[3] A. Guler and N.K. Jha, "McPAT-Monolithic: An Area/Power/Timing Architecture Modeling Framework for 3-D Hybrid Monolithic Multicore Systems", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 28, No. 10, pp. 2146-2156, 2020.

[4] Y. Shen and V. Chen, "Class-E Power Amplifiers Incorporating Fingerprint Augmentation with Combinatorial Security Primitives for Machine-Learningbased Authentication in 65 nm CMOS", *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 69, No. 5, pp. 1896-1909, 2022.

[5] K.G. Devi and N.T.D. Linh, "*Artificial Intelligence Trends for Data Analytics using Machine Learning and Deep Learning Approaches*", CRC Press, 2020.

[6] S. Zheng and S. Yin, "An Ultra-Low Power Binarized Convolutional Neural Network-Based Speech Recognition Processor with On-Chip Self-Learning", *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 66, No. 12, pp. 4648-4661, 2019.

[7] C. Niu and D. Liu, "Detector Processor for a 5G Base Station", *Sensors*, Vol. 22, No. 20, pp. 7731-7739, 2022.

[8] M.K. Adimulam and M.B. Srinivas, "A 12-Bit, 1.1-GS/s, Low-Power Flash ADC", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 30, No. 3, pp. 277-290, 2022.

[9] Z. Zhang and M. Zhang, "A Fast Parameter Tuning Framework via Transfer Learning and Multi-Objective Bayesian Optimization", *Proceedings of ACM/IEEE Conference on Design Automation*, pp. 133-138, 2022.

[10] G. Sapone and G. Palmisano, "A 3-10-GHz Low-Power CMOS Low- Noise Amplifier for Ultra- Wideband Communication", *IEEE Transactions on Microwave Theory and Techniques*, Vol. 59, No. 3, pp. 678-686, 2011.

[11] V.R.G.D. Silva and S. Xavier-De-Souza, "Analytical Energy Model Parametrized by Workload, Clock Frequency and Number of Active Cores for Share-Memory High-Performance Computing Applications", *Energies*, Vol. 15, No. 3, pp. 1213-1219, 2022.

[12] Y. Parmar and K. Sridharan, "A High-Performance VLSI Architecture for a Self-Feedback Convolutional Neural Network", *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 68, No. 1, pp. 456-460, 2020.