DOI: 10.21917/ijme.2023.0252

AI BASED MACHINE LEARNING ALGORITHM IN VLSI TECHNOLOGY APPLICATION

G.R. Thippeswamy[1], R. Jayadurga[2] and Suresh Kumar Sharma[3]
[1]*Department of Computer Science and Engineering, Don Bosco Institute of Technology, India*
[2]*Department of Computer Science, Soundarya Institute of Management and Science, India*
[3]*Department of SMCS, Sri Karan Narendra Agriculture University, India*

Abstract

Estimating power consumption in CMOS VLSI circuits using supervised learning is the focus of this investigation. Unlike more conventional approaches like the SPICE circuit modelling that has been recommended, the proposed model does not assume a predetermined set of empirical equations or parameters. Unlike other technologies, and it doesn't require the user to pay attention to the circuit topology or the connectivity to provide precise results. An alternative interpretation with improved efficiency is suggested by the proposed design, but it will require a large amount of additional data for proper implementation. The proposed architecture has certain qualities that can improve power estimation for CMOS VLSI circuits.

Keywords:

AI, Machine Learning, VLSI, NoC

1. INTRODUCTION

The requirements of the platform in terms of data transmission are satisfied by the numerous communication lines that are included in the system-on-a-chip (SoC) [1]. Globally Asynchronous, Locally Synchronous Systems (GALS), is becoming increasingly popular because of the difficulties that are inherent in the design of these on-chip communication cables for sub-micron technology. This way of designing divides a platform up into many different synchronous areas, each of which can run a different application task in parallel.

This sphere is synchronous just inside itself, and to communicate with other synchronous zones, it would employ asynchronous ways. The network-on-chip, often known as NoC, offers a new paradigm for the communication that takes place within a chip when applied to a particular topology design. Because of this, the GALS-based system enables communication to proceed without any problems. In addition to this, what emerges as a result is a layout that is efficient and extensible. Furthermore, the NoC is a scalable method for meeting application communication requirements [2] in heterogeneous CPUs with multiple cores.

It is necessary to map diverse activities of a target application to different cores to achieve better performance from a GALS-based SoC architecture. Fixing this issue, which is the root cause of the difficulty in mapping applications to the NoC architecture, may result in further quality of life enhancements. Therefore, it is vital to select the most effective solution for the performance of the NoC [3].

The computer industry will eventually shift its focus to artificial intelligence to manage the vast volumes of data that are required by cutting-edge programs. In the field of artificial intelligence (AI), one example of a rapidly evolving application that requires a high degree of parallelism to meet the application processing deadline is neural networks (NNs) [4]. In the case of NNs, it is possible to make use of this parallelism by distributing neurons throughout the various components of the NoC design.

Throughout the years, there have been a great number of researchers who have focused on developing various application processes suitable for a variety of uses. It delves into the process of mapping multiple applications onto the architecture of the NoC. The mapping of AI algorithms onto a NoC infrastructure is not taken into consideration by this, though. In [5], a multi-objective algorithm is constructed by considering the various temporal limits of the intended applications. This was done to optimise the algorithm performance. Due to the difficulty of reconfiguring, the way that was suggested did not demonstrate any evidence of progress, which is unfortunate.

The article in [6] outlines the procedure that has been suggested as a means of locating this optimal zone for a certain application. The events took place in a sequential order, which may have contributed to the lack of noticeable results. Another paper [2] focused on developing a fault-tolerant method for application mapping and advocated giving healthy cores higher priority when constructing programmes. The concept was called application mapping.

Mapping algorithms on heterogeneous multi-core processors with distinct features are discussed in both [5] and [6], while discusses a NoC application mapping to balance packet latency with other performance aspects. It also discusses mapping algorithms on heterogeneous multi-core processors. A technique that is based on rectangle analysis is presented by the authors of [4] as a means of selecting NoC zones for use in multi-application mapping. With the use of this design space exploration (DSE) method, we can zero in on the sweet spot for the efficient operation of an application about both its latency and its power consumption.

2. RELATED WORKS

This article will explore the various mapping strategies currently in use for NoC architectures. Mapping is a key stage in the process of developing a NoC since different parts of the programme need to be distributed over different processor cores. Several different objectives might be decided upon, depending on the use case. Latency in application processing, energy consumption, meeting of real-time deadlines, and throughput are all examples. The implementation of these algorithms has involved the use of a variety of different optimization approaches [7].

For NoC-based real-time application mapping, the branch-and-bound (BB)-based exact mapping (BEMAP) technique is explained in full in reference [8]. The method reduces the amount

of power consumed and the amount of latency while simultaneously boosting the throughput for a given NOC bandwidth. The solution is associated with the multi-goal optimization of the NoC application mapping. The approach offers a reduction in network latency of up to 61.10% and an overall reduction in energy consumption of up to 19.93% for mesh and torus topologies, respectively. Although the article discusses certain applications that may be developed with AI, it does not go into detail regarding how these applications could be implemented.

In reference [4], it is shown how to map several applications onto the NoC by first doing an analysis of the various regions that make up the NoC. A genetic algorithm is used to evaluate the effectiveness of the application tasks after they have been mapped into the prospective zones. In the second step of the process, you will derive a new task mapping by employing a simulated annealing strategy that is based on a B* tree. Based on the results of the experiments, the authors claim that the selected apps experienced a reduction in latency of 24.42 percent and a decrease in power usage of 23.45 percent. However, the method does not take into consideration the existence of an AI.

It is advise that demanding real-time applications use a precompute phase to partially map the application, with the remaining mapping being executed on the fly during runtime. This strategy is intended to improve performance. Because of this, programmes are given the freedom to adjust their mapping in accordance with the resources that are now available on the hardware. The researchers claim that this strategy can cut the amount of energy needed by 13 percent when compared to the best practises that are currently in use. The solution that has been proposed, on the other hand, has the potential to be expanded to accommodate AI-based input programmes.

In mapping of many applications onto the NoC while imposing latency constraints is explored. To discover a solution to the issue, a method that is based on heuristics is applied, and this method increases performance all around while simultaneously decreasing the maximum average packet delay by 10.42%. The method that has been proposed, on the other hand, does not take into account mapping AI-based applications to this framework. As a result, this topic is open to additional research.

3. PROPOSED MODELLING

The investigation of AI-based application mapping across a wide variety of NoC platforms while considering latency and energy consumption is the primary objective. This algorithm is carried out in steps as it is being executed. In the first part of the process, which is referred to as region analysis, we evaluate which regions are ideal for deploying the various components of an application.

In the second stage, you will map the entirety of the programme onto the architecture of the processor by allocating the neurons for each individual task to the cores of the region that you selected in the previous step. Because, to the best of our knowledge, there are very few studies of the task mapping of AI applications on NoC architecture, we will investigate a mesh based NoC as a possible solution for application mapping. On the other hand, further research will investigate network-centric design patterns other than mesh networks.

Before settling on a particular area of the programme to map, the model divides the application into discrete tasks, which are then mapped into the various layers of the neural network. In addition to this, the cores of the currently available processors are being investigated. In the second phase, we will map specific neurons to respective cores to optimise latency as well as power consumption. In the sections that follow, we will discuss the two stages that are required to accomplish these goals.

3.1 REGION MAPPING

We divide the mesh based NoC into a few different zones so that we can find open zones more quickly. The number of layers in the neural network determines which portions of the network are loaded onto the CPU. We divide the result by one to get the total number of regions in the neural network. This is accomplished by first dividing the number of cores by the total number of layers in the neural network.

$$n_r = n_c / n \quad (1)$$

where

n_r is the number of regions,

n_c is the number of cores, and

n is the number of neural network layers.

We select neighbouring cores to combine into a single area by making use of the region count that was provided in the prior algorithm. After this step has been completed, any cores or clusters of cores that are still present are merged with those that are located nearby.

Consider a predetermined occupancy level to find out which areas are currently unoccupied. This is done to select regions that do not currently have any inhabitants. This guarantees that only less-crowded places are selected, with a primary focus on those that have a lower occupancy rate. This would modify the pattern of the processor energy consumption, which would be to its advantage.

The method that has been offered, which takes a game-theoretic approach to maximise the achievement of global goals. A layer of a neural network must be mapped into some physical area to properly represent the activities that need to be carried out, which are represented by the neurons in that layer. Using this approach, the work that needs to be done on one layer will be finished before moving on to the work that has to be done on another layer. At the level that has been planned, using this strategy will be effective in reducing the time lag in communication.

3.2 CORE - NEURONS MAPPING

Neuron Mapping at the Core 3.2 Secondly, to map neurons to a particular core of the processor, we make use of the NoC region that was assigned during the level-1 mapping process. During this stage, we will select a core that has the lowest possible energy requirements and the fastest possible network speeds that are possible. The overall objectives are optimised using a game-theoretic framework with the help of the recommended algorithm. The number of problematic places in the NoC can be reduced, which will have a beneficial effect and allow for the achievement of a better solution.

We show that the neurons associated with a job are assigned to a particular processing unit to reduce the energy-draining communication overhead associated with the NoC. It is also crucial to distribute the neurons in an even manner overall the processing cores to achieve a consistent power profile.

It is possible that neurons may be assigned to a single processor to cut down on the energy required for communication; however, doing so would result in the production of energy hot spots. On the other hand, if the neurons are dispersed evenly, this will increase the amount of energy available for communication while simultaneously lowering the influence of any energy hot spots.

4. RESULTS

The training dataset and the testing dataset for the model originate from previously published research. The proposed model is educated using the data from a set of 20 benchmark ISCA89 sequential circuits, and then 5 of those circuits are utilised for testing purposes which is shown in Table.1-Table.4. A sequential circuit number of inputs and outputs, as well as its D flip-flops, inverters, and total number of gates (AND gates, NAND gates, OR gates, and NOR gates), are used as attributes during instruction and evaluation. Other attributes include the total number of gates.

Training and testing the newly developed model both make use of ten-fold cross-validation. Python, a high-level programming language, is used to create the actual job, and the application is run on a machine with a 3.4 GHz Intel Core i7-6700 central processing unit and 16 gigabytes of random-access memory.

The number of input neurons in a particular neural network is directly proportional to the number of input attributes that were employed throughout the training process. The number of attributes is reduced from 9 to 7 because of the removal of OR gates and AND gates from the input. In a few specific instances, like traincgf and traincgp, the output of 7 input attributes was superior to the output of 9 input attributes. The learning rate, the momentum constant, the activation function, and the training technique are the four primary parameters that make up a BPNN.

Adjustments can be made to the momentum constant, which can range from 0.1 to 0.9, the number of epochs, which can range from 150 to 2700, the number of neurons in the hidden layer, which can range from 10 to 17, the learning rate, which can range from 0.3 to 0.8 across 11 unique training algorithms, and the activation functions, which can range from tansig to logsig. In the tables that follow, comparisons of the results for the ISCA89 benchmark circuits are presented.

Table.1. Delay (ms)

	Benchmark circuit	**LSTM**	**BPNN**	**RF**	**LR**	**SVM**	**Proposed**
Training	S344	0.427094	0.574875	0.929611	0.712334	0.839982	0.719795
	S382	0.717444	0.952811	0.655102	1.008714	1.058588	0.814023
	S386	0.475332	0.402055	0.664198	0.613507	0.755258	0.587343
	S400	0.617799	0.991442	0.507116	0.863488	0.963337	0.696595
	S641	0.317024	0.382228	0.573444	0.807789	0.886381	0.680243
Testing	S344	1.131456	1.124098	1.12604	1.127981	1.131763	1.131456
	S382	0.873299	0.921231	1.014335	1.066764	1.082298	0.918471
	S386	1.064311	1.165284	1.507961	1.080152	1.09538	2.021107
	S400	0.546872	0.739315	0.543193	0.853779	0.929611	0.764865
	S641	0.510693	0.738599	0.717035	0.948314	0.980405	0.992566

Table.2. Computational time (ms)

	Benchmark circuit	**LSTM**	**BPNN**	**RF**	**LR**	**SVM**	**Proposed**
Training	S344	3.060379	7.03647	1.123893	1.076984	1.185418	1.8582
	S382	0.99829	1.136464	1.053989	0.918574	0.946065	3.027471
	S386	1.007385	1.108052	1.09027	1.105906	1.049287	1.080561
	S400	1.142494	1.134318	1.65145	1.100898	1.077801	3.121495
	S641	0.708859	1.14556	0.944124	0.877489	1.058894	0.865327
Testing	S344	1.002991	1.079232	0.984186	0.9709	0.976419	0.978258
	S382	1.442246	2.242677	1.085262	1.156291	1.092314	3.843435
	S386	0.773245	1.134624	2.021516	1.095891	1.135544	0.937889
	S400	1.070954	1.126857	1.135749	1.018832	1.1335	0.926954
	S641	0.928691	1.259308	0.992158	0.946474	1.329111	1.001049

Table.3. Throughput (kbps)

	Benchmark circuit	LSTM	BPNN	RF	LR	SVM	Proposed
Training	S344	0.721532	0.971309	1.571223	1.192265	1.397483	1.216487
	S382	1.341068	1.803217	1.215669	1.920747	2.017939	1.535453
	S386	0.61739	0.535937	0.902733	0.842843	1.04336	0.792765
	S400	0.68893	1.111629	0.566188	0.971615	1.083933	0.778355
	S641	0.398069	0.479829	0.672374	1.005035	1.106519	0.852552
Testing	S344	1.305912	1.297531	1.299882	1.301824	1.30632	1.305912
	S382	1.02803	1.084342	1.194003	1.255731	1.274025	1.081174
	S386	1.164262	1.279544	1.682008	1.190937	1.203405	2.231333
	S400	0.612689	0.828229	0.608703	0.950971	1.03825	0.857049
	S641	0.552289	0.858991	0.842946	1.123791	1.166409	1.171416

Table.4. Communication Delay (ms)

	Benchmark circuit	LSTM	BPNN	RF	LR	SVM	Proposed
Training	S344	3.843538	6.486736	1.376327	1.315314	1.456043	2.318305
	S382	1.212296	1.380415	1.280157	1.113776	1.141778	3.680426
	S386	1.374488	1.514911	1.49212	1.516444	1.440407	1.475972
	S400	1.322979	1.313168	1.923404	1.272697	1.242752	3.666732
	S641	0.928896	1.556097	1.263907	1.199726	1.454919	1.179797
Testing	S344	1.14791	1.234883	1.111425	1.096708	1.106519	1.119499
	S382	2.117891	3.834544	1.158437	1.375816	1.188279	6.961046
	S386	0.906923	1.337185	2.388823	1.291093	1.338207	1.103147
	S400	1.232532	1.297736	1.308058	1.171621	1.305401	1.063289
	S641	1.18552	1.554462	1.243263	1.206369	1.779404	1.278522

5. CONCLUSION

Within the scope of this study is a method for estimating power consumption in CMOS VLSI circuits that makes use of supervised learning. The RF model that has been suggested does not rely on the assumption of any set of empirical equations or parameters, in contrast to traditional methods such as the SPICE circuit modelling that has been suggested. RF produces accurate results without requiring the user to pay attention to the circuit structure or the connectivity.

A design that has been offered will offer an alternate interpretation that has enhanced performance, but for it to be properly executed, it will require a substantial amount of additional data. The power estimation of CMOS VLSI circuits can benefit from the one-of-a-kind characteristics of the suggested design.

REFERENCES

[1] V. Govindaraj and B. Arunadevi, "Machine Learning Based Power Estimation for CMOS VLSI Circuits", *Applied Artificial Intelligence*, Vol. 35, No. 13, pp. 1043-1055, 2021.

[2] M. Bansal, "Machine Learning Perspective in VLSI Computer-Aided Design at Different Abstraction Levels", *Proceedings of International Conference on Mobile Computing and Sustainable* Informatics, pp. 95-112, 2021.

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

[4] S. Bavikadi and S.M. Pudukotai Dinakarrao, "A Review of In-Memory Computing Architectures for Machine Learning Applications", *Proceedings of International Symposium on Great Lakes on VLSI*, pp. 89-94, 2020.

[5] F.M. Aswad and S.A. Mostafa, "Tree-Based Machine Learning Algorithms in the Internet of Things Environment for Multivariate Flood Status Prediction", *Journal of Intelligent Systems*, Vol. 31, No. 1, pp. 1-14, 2021.

[6] A. Sebastian, "Computational Memory-based Inference and Training of Deep Neural Networks", *Proceedings of International Symposium on VLSI Technology*, pp. 168-169, 2019.

[7] A. Site and E.S. Lohan, "Systematic Review on Machine-Learning Algorithms used in Wearable-Based eHealth Data Analysis", *IEEE Access*, Vol. 9, pp. 112221-112235, 2021.

[8] C. Venkatesan and R. Kumar, "ECG Signal Preprocessing and SVM Classifier-Based Abnormality Detection in Remote Healthcare Applications", *IEEE Access*, Vol. 6, pp. 9767-9773, 2018.