Embedded Machine Learning on a Programmable Neuromorphic Platform

Indar Sugiarto, Agustinus Bimo Gumelar, and Astri Yogatama

Abstract This paper presents an implementation of k nearest neighbor (k-NN) algorithm on SpiNNaker. SpiNNaker is a programmable neuromorphic platform well-known for its very low power energy consumption which is suitable to be used as an embedded system. By utilizing the SpiNNaker communication protocols, we are able to efficiently distribute jobs across SpiNNaker cores for performing the k-NN algorithm. From the experiments, we observed that the k-NN program runs smoothly and performs basic classification task on Irish dataset correctly. From the investigation, we found that the k-NN program reports not only the accuracy and the best k-value to get that accuracy, but also the reason why that k-value should be chosen for the corresponding classification task. We also found that increasing the potential number for k-values, the k-NN program was able to find better accuracy. For example, the accuracy of 91.67% was achieved when we increased the range of k between 30 to 60.

1 Introduction

Machine learning (ML) is a technique for exploring and exploiting data that can be used to improve performance of a system or an application. It is considered as the most progressive aspect of artificial intelligence, since many applications nowadays use machine learning algorithms or at least interact with a cloud service provider that provides machine learning capabilities [1].

I. Sugiarto (✉)
Electrical Engineering, Petra Christian University, Surabaya, Indonesia
e-mail: indi@petra.ac.id

A. B. Gumelar
Computer Science, Narotama University, Surabaya, Indonesia
e-mail: bimogumelar@narotama.ac.id

A. Yogatama
Communication Science, Petra Christian University, Surabaya, Indonesia
e-mail: astri@petra.ac.id

Z. Zakaria and S. S. Emamian (eds.), *Recent Advances in Electrical and Electronic Engineering and Computer Science*, Lecture Notes in Electrical Engineering 865, https://doi.org/10.1007/978-981-16-9781-4_13

Machine learning algorithms can also be used in various applications the embedded world. The term edge computing now becomes increasingly popular especially in the context of industrial revolution 4.0 related topics such smart city, smart transportation, and so on. In this sense, many embedded systems are equipped with hardware modules capable of computing or implementing certain machine learning algorithm [2].

One requirement and also the benefit of an edge computing system is the low power consumption whilst maintaining simple operability due to its direct access to sensors and actuators of the associated cyber physical systems [3].

On the other side of emerging technologies, now exist a new computing paradigm called neuromorphic computing. This paradigm exhibits superiority on low power requirement of an artificial intelligence (AI) based computing [4]. This neuromorphic computing is supported by various neuromorphic devices. Along with their software architecture, these neuromorphic devices comprise the so called neuromorphic platform. The presence of neuromorphic platforms seems to be inline with the advancing trend of edge computing. Together, these technologies may offer hugh benefits in term of inter-operability as well as reliability of an AI-driven ecosystem in the future [5].

This paper presents preliminary results of experiments on implementing AI concept in a programmable neuromorphic platform. In this study, some ML algorithms were implemented and tested on a SpiNNaker platform. SpiNNaker, which stands for Spiking Neural Network Architecture, originally was designed for implementing the third generation of artificial neural network [6]. By implementing ML algorithms on SpiNNaker, we gain not only the low power energy consumption, but also high configurability of ML-based applications, since the SpiNNaker offers many options to optimize the program running on it [7]. In this paper, only the k-NN (k nearest neighbor) algorithm is reported.

This paper is organized as follows. After giving some introductory background, we present our implementation method in Sect. 2. The experiment results and the discussion are presented in Sect. 3. Finally, this paper is closed by conclusion presented in Sect. 4.

2 Methods

2.1 Brief Overview of SpiNNaker

SpiNNaker is originally a neuromorphic platform dedicated for the study of human brain in digital form. It was conceived by the University of Manchester in the UK, where computer scientists work together with computational neuroscientists to create a system capable of simulating millions of digital neurons in real time [8].

SpiNNaker ecosystem consists of various level of neuromorphic computing, from the chip level neuromorphic devices to the system level brain simulator. The lowest

level of this system is the SpiNNaker chip. This chip is a multicore system that has 18 ARM968 cores. These cores are packed along with 256 MB dynamic RAM during the chip production. In addition to this low-power SDRAM, each SpiNNaker core is also equipped with two types of static memory, namely 32 KB ITCM (instruction tightly-coupled instruction memory) exclusively for storing program instruction, and 64 KB DTCM (tightly-coupled data memory).

From these chips, higher level SpiNNaker boards are produced. Currently, there are two type of SpiNNaker boards available for research (see Fig. 1): 4-chips SpiNNaker boards called Spin3 and 48-chips SpiNNaker boards called Spin5.

Since each SpiNNaker chip consists of 18 cores, thus in total, Spin3 will have 72 cores, whereas Spin5 will have 864 cores. Unfortunately, not all cores are usable for normal user's program since SpiNNaker system requires at least one core for internal use.

On both SpiNNaker boards, there is a special SpiNNaker chip (usually designated as chip <0,0> in the SpiNNaker datasheet) that has a special function as to provide the communication access to the outside world through a standard TCP/IP connection. In this paper, as in other our previous papers, we use this special chip as the master that coordinates all SpiNNaker chips during algorithm/program deployment.

There are several modes of communication within the SpiNNaker board. Two of the most important ones for user program deployment are so called SpiNNaker

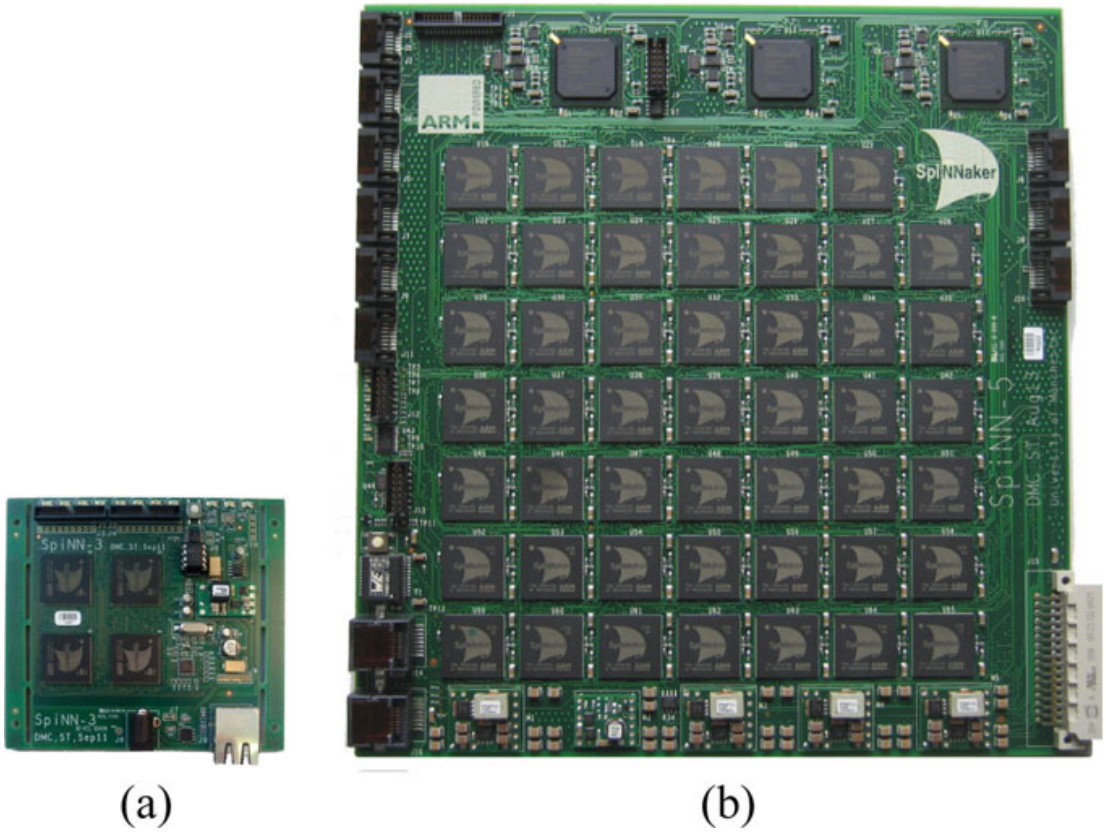

Fig. 1 Two types of SpiNNaker boards: **a** Spin3 with 4 SpiNNaker chips, and **b** Spin5 with 48 SpiNNaker chips

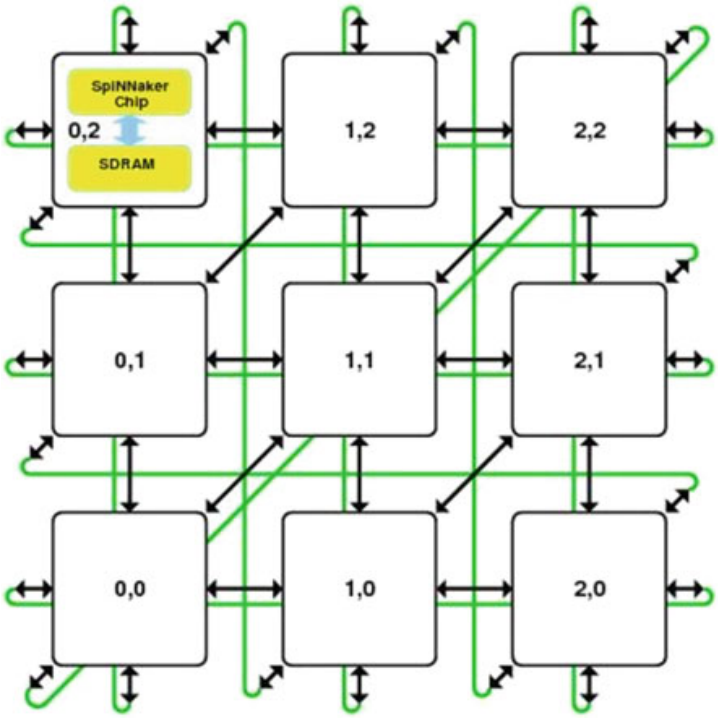

Fig. 2 Each SpiNNaker chip has six bidirectional links that can be connected to other chips. Using these links, 2D triangular mesh topology can be created that makes SpiNNaker an efficient high-throughput many-core system

multicast packet (MC) and SpiNNaker datagram packet (SDP). These two packets are used in this study for different purpose: the MC packets are used for communication and coordination between cores of SpiNNaker, whereas SDP packets are used for delivering data from/to computer host via TCP/IP. The MC packets have higher throughput than SDP packet since they have lower overhead and can optimally make use of the SpiNNaker six channels torus topology as shown in Fig. 2.

In the following subsection, we describe how two machine learning algorithms are implemented on SpiNNaker. In both implementation strategies, we designated one core as a master core. This master core resides in chip <0,0> and has the core virtual number 1; hence, it is often called as core <0,0,1> in the SpiNNaker ecosystem (Fig. 3).

2.2 K-NN Implementation

k-NN is a basic ML algorithm that is used commonly in regression or classification tasks. It works pretty simple and can be explained intuitively since it is a non-parametric statistical algorithm that processes k closest training examples.

k-NN on SpiNNaker is implemented by dividing tasks between master core and slave core. The master core is tasked with delegating calculation functions to other cores (referred to as slave cores). In addition, the master core will receive the distinct k-ID of the corresponding slave, and the accuracy value of that slave core. The slave core is in charge of running the k-NN core function and calculating the accuracy

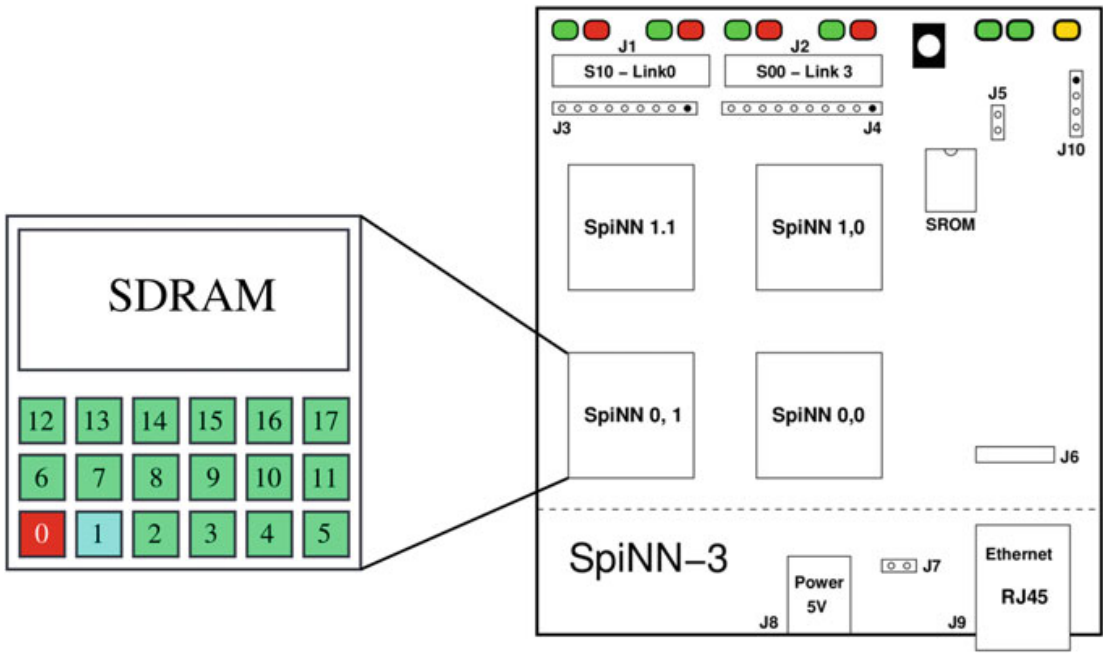

Fig. 3 Each SpiNNaker chip has 18 cores (see Fig. 1 for the real/physical appearance of the Spin3 board). Core-0 is dedicated for SpiNNaker kernel, and core-1 will be used as the master core in our ML implementation strategy. The rest of the cores will be used as worker/slave cores. The master core in chip <0,0> also operates as the master core for the entire system since it has responsibility to communicate with external devices through the ethernet port

value of the given k-value. After computing the accuracy value, that value will be sent to the master core.

The algorithm of the k-NN program on SpiNNaker is designed as follows:

1. Set the routing table to deliver MC packets from the master to slave cores.
2. Set a timer to send a multicast packet from the master core to the slave core. Here the timer is set to run every 1 (one) second and will stop after receiving stop comand from the host PC.
3. Master core sends multicast packet to slave core with parameter k-value.
4. Slave cores listen and receive multicast packets from master.
5. Slave cores perform k-NN calculations based on its predefined region, and get the accuracy value for the k-value obtained.
6. Slave cores sends accuracy and k-value back to the master as a MC packet.
7. Master core receives the accuracy value and k-value from all other slave cores. If the accuracy value received is better than the previous accuracy value, the accuracy value and k-value will be updated.
8. If the timer has stopped (after some iteration or by request), then the process on the slave cores will stop and the master core will send the stored accuracy and k-value to the external device (i.e., host-PC) via SDP.

Figure 4 represents the flowchart of k-NN program running on SpiNNaker. In this study, a standard host-PC is used to send and retrieve data to/from the SpiNNaker

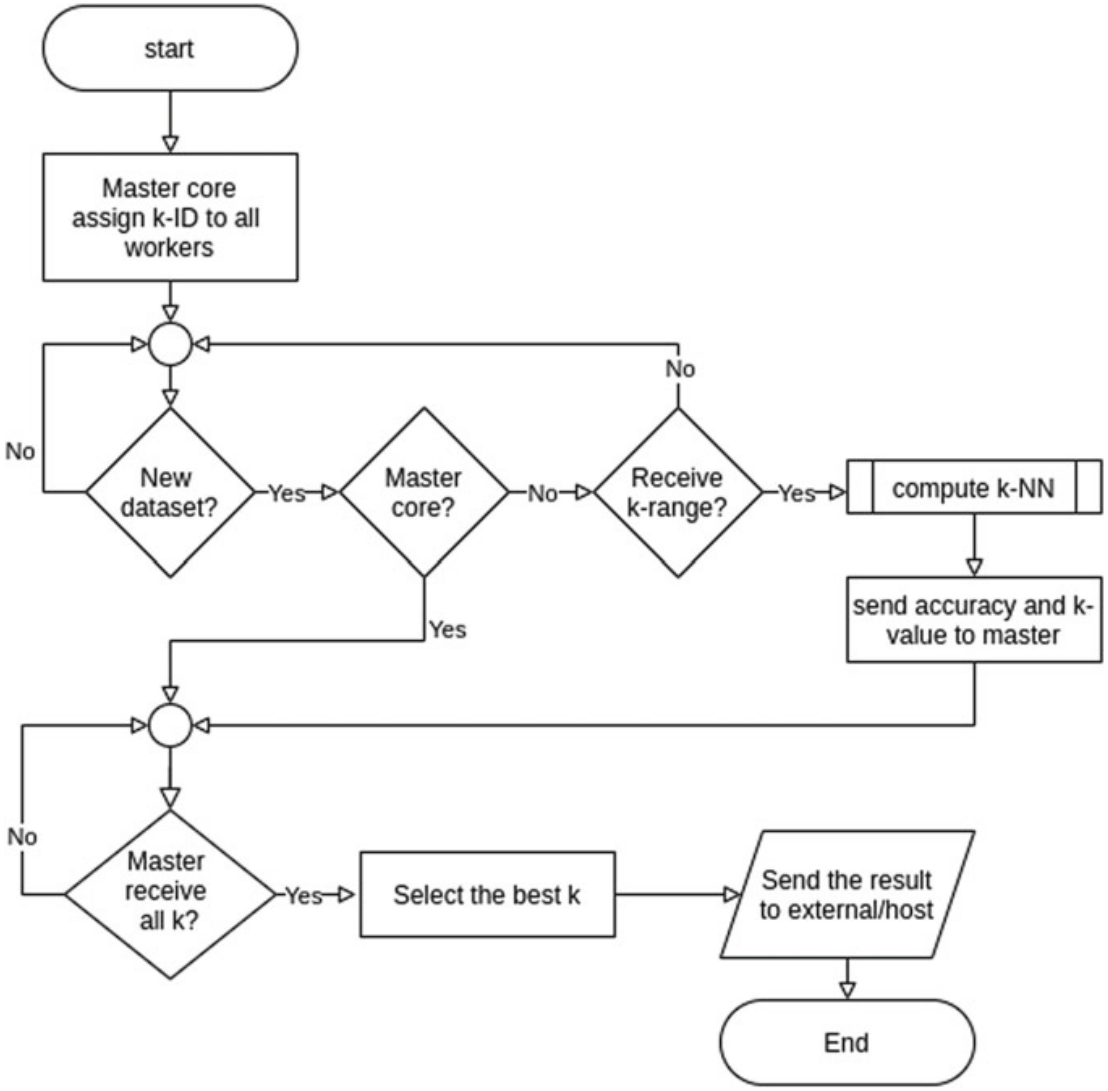

Fig. 4 The main flowchart of k-NN program running on SpiNNaker

board. This host-PC can be replaced by any edge computing hardware in real applications as long as that hardware has a TCP/IP connection to the SpiNNaker board. To receive the results from the SpiNNaker, a Python program is used and it will listen to the datagram socket coming from the SpiNNaker board.

3 Result and Discussion

3.1 K-NN Experiment

For evaluation purpose, we used k-NN for classification of Iris flower using dataset available from the University of California, Irvine. This is a well-known multivariate dataset used by statistician that originated from Ronald Fisher paper in 1936. The

dataset consists of 50 species of Iris flower, namely: Iris setosa, Iris virginica and Iris versicolor. In the dataset, four features are used to distinguish the species: the length and width of the sepals and petals, in centimeters.

After deploying the k-NN program onto the SpiNNaker board, we inspected the SpiNNaker to see if everything runs properly. The initial output of the SpiNNaker program shows that the k-NN program was deployed successfully (see Fig. 5).

A Python script has been developed to control how the k-NN program on the SpiNNaker should work. This Python script is also used to notify the k-NN program on the SpiNNaker how many k-values should be used by the k-NN program. This in

```
# ybug - version 3.4.1
SPIN3:0,0,0 > @ knn_spinnaker.ybug
@SPIN3:0,0,0 > iptag 1 set . 17894
@SPIN3:0,0,0 > app_load knn_spinnaker.aplx all 1-17 22
@SPIN3:0,0,0 > sleep 1
@SPIN3:0,0,0 > app_sig all 22 sync0
SPIN3:0,0,0 > sp
SPIN3:0,0,0 > ps
Core State  Application       ID   Running  Started
---- -----  -----------       --   -------  -------
  0  RUN    scamp-3            0   0:24:07   1 Nov 17:33
  1  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
  2  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
  3  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
  4  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
  5  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
  6  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
  7  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
  8  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
  9  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
 10  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
 11  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
 12  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
 13  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
 14  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
 15  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
 16  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
 17  RUN    knn_spinnaker     22   0:00:07   1 Nov 17:57
SPIN3:0,0,0 >
```

Fig. 5 The result of deploying k-NN program onto the SpiNNaker board. Here core-1 operates as the master, and the remaining cores run as the worker (except core-0, which is the kernel of the SpiNNaker system called SCAMP or SpiNNaker Control and Monitor Processor)

turn will determine how many cores will be involved in the k-NN computation. For example, if the number of k was specified as 40, then only 40 SpiNNaker cores will be activated. This mechanism is depicted in Fig. 6.

After feeding the training and testing dataset to the k-NN program on the SpiNNaker, we collected the accuracy values. Table 1 shows the result.

As we can see from Table 1, the accuracy increases as the k-value also increases. However, the increase in accuracy is not monotonically linear and tends to increase step-wisely. The first column in Table 1 represents the range of k values that will be evaluated. This column also represents the total number of SpiNNaker cores involved in the k-NN computation. The second column in Table 1 represents the k-value among the range of active k values that produces the highest accuracy. As we can see in that table, when we varied k values between 6 and 25, the highest accuracy was found at ke equals 6. It means that the ideal k for the given dataset (Irish flower dataset) is 6. However, we also see that when we increased the potential number of k between 30 and 60, we found that the highest accuracy was achieved when the k-value is 29. For the lower range of k-values (between 3 and 5), we see that the optimal number of k tends to the the maximal number of k available in that range.

To understand this behavior further, we look into the distribution of accuracy for the corresponding k-values in the given range. Figure 7 shows how the accuracy was achieved by each k-value in that range.

As we can see on Fig. 7, the highest accuracy was achieved for the k-value number 29. On that figure, we also plot the approximated trending line of the average accuracy across all possible k-values. We found that the knee (or elbow) phenomenon happens

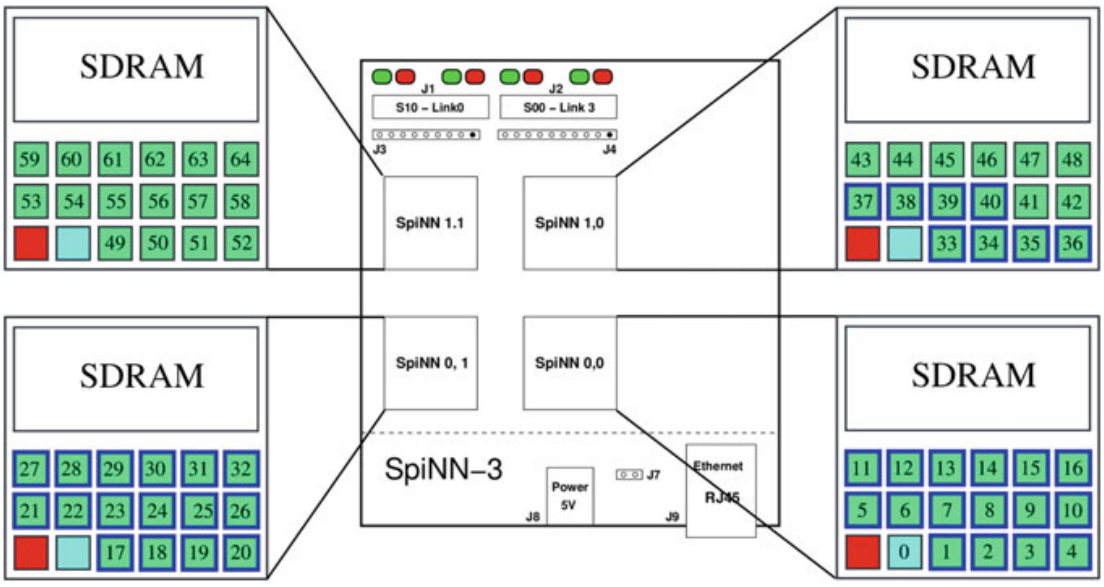

Fig. 6 Example of running k-NN program on SpiNNaker with k = 40. In this case, only 40 cores will be activated as working cores (see the cores with thick blue border above) and the remaining cores will be idle. Core-0 (at chip <0,0>) will act as the master that coordinates all of active working cores

Table 1 The accuracy of k-NN program on SpiNNaker

Max k-range	Optimal k-value	Accuracy (%)
3	3	77.08
4	3	77.08
5	5	85.42
6	6	87.5
7	6	87.5
8	6	87.5
9	6	87.5
10	6	87.5
15	6	87.5
20	6	87.5
25	6	87.5
30	29	91.67
40	29	91.67
50	29	91.67
60	29	91.67

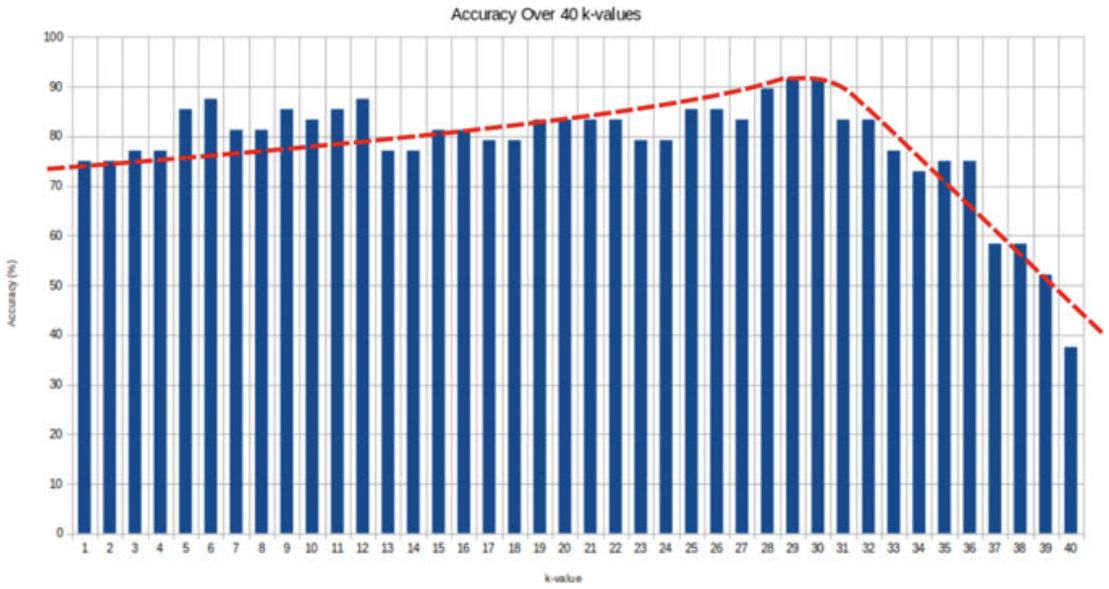

Fig. 7 Achieved accuracy for each possible k-values in the range of 40 k-values. This also corresponds to the distributed k-value across worker cores

at k-29. In ML literature, this corresponds to the elbow method for finding optimal k-value [9].

4 Conclusions

This paper presents a preliminary but novel work on machine learning on SpiNNaker. SpiNNaker is a programmable neuromorphic platform which is basically a very low power many core system. We implemented a basic machine learning algorithm called k nearest neighbor (k-NN) that is commonly used in classification and regression tasks. By utilizing two type of SpiNNaker communication packets, namely MC-packets and SDP, we are able to efficiently distribute jobs across SpiNNaker cores for performing k-NN algorithm. From the experiments, we observed that the k-NN program runs smoothly and performs basic classification task on Irish dataset correctly. From the investigation, we found that the k-NN program reports not only the accuracy and the best k-value to get that accuracy, but also the reason why that k-value should be chosen for the corresponding classification task. We also found that increasing the potential number for k-values, the k-NN program was able to find better accuracy. For example, the accuracy of 91.67% was achieved when we increase the range of k between 30 to 60. For our future work, we will implement other ML algorithms such as SVM, MLP (multilayer perceptron), etc., as well as testing those algorithms on real world application scenarios.

Acknowledgements This research was supported by the Directorate of Higher Education of the Ministry of Research and Technology of the Republic of Indonesia (Ristekdikti) through the grant Penelitian Dasar (PD) no: 002/AMD-SP2H/LT-MULTI-PDPK/LPPM-UKP/2021.

References

1. Neloy AA, Alam S, Bindu RA, Moni NJ (2019) Machine learning based health prediction system using IBM Cloud as PaaS. In: 2019 3rd international conference on trends in electronics and informatics (ICOEI), pp 444–450
2. Branco S, Ferreira AG, Cabral J (2019) Machine learning in resource-scarce embedded systems, FPGAs, and end-devices: a survey. Electronics 8(11):1289
3. Shi W, Dustdar S (2016) The promise of edge computing. Computer 49(5):78–81
4. Marković D, Mizrahi A, Querlioz D, Grollier J (2020) Physics for neuromorphic computing. Nat Rev Phys 2(9):499–510
5. Mead C (2020) How we created neuromorphic engineering. Nat Electron 3(7):434–435
6. Furber S (2016) Large-scale neuromorphic computing systems. J Neural Eng 13(5):051001
7. Sugiarto I, Furber S (2021) Fine-grained or coarse-grained? Strategies for implement-ing parallel genetic algorithms in a programmable neuromorphic platform. TELKOMNIKA Telecommun Comput Electron Control 19(1):182–191
8. Furber SB, Galluppi F, Temple S, Plana LA (2014) The spinnaker project. Proc IEEE 102(5):652–665
9. Park CH, Kim SB (2015) Sequential random k-nearest neighbor feature selection for high-dimensional data. Expert Syst Appl 42(5):2336–2342