2010 Ninth International Symposium on Parallel and Distributed Computing

# Algorithm for Mapping Multilayer BP Networks onto the SpiNNaker Neuromorphic Hardware

X. Jin, M. Luján, M.M. Khan, L.A. Plana, A.D. Rast, S.R.Welbourne and S.B. Furber
*The University of Manchester*
*Manchester, UK e-mail: jinxa@cs.man.ac.uk*

***Abstract*—This paper demonstrates the feasibility and evaluates the performance of using the SpiNNaker neuromorphic hardware to simulate traditional *non-spiking* multi-layer perceptron networks with the backpropagation learning rule. In addition to investigating the mapping of checker-boarding partitioning scheme onto SpiNNaker, we propose a new algorithm called pipelined checker-boarding partitioning which introduces a pipelined mode and captures the parallelism within each partition of the weight matrix, allowing the overlapping of communication and computation. Not only does the proposed algorithm localize communication, but it can also hide a part of or even all the communication. The performance is evaluated with SpiNNaker configurations up to 1000 nodes (20000 cores).**

***Keywords*-parallel; perceptron; backpropagation; SpiNNaker; mapping**

## I. Introduction

As with other parallel applications, the challenge in the parallel simulation of multi-layer perceptron (MLP) networks with the backpropagation (BP) learning is to understand how to partition and distribute the computational tasks while minimizing communication requirements. A number of partitioning schemes have been developed to solve this problem and have been mapped onto a variety of parallel hardware with different topologies such as the hypercube machine, mesh connected multiprocessors, transputers, networks of workstations, and also dedicated neural hardware [1], [2], [3].

SpiNNaker machine was originally designed to simulate large-scale *spiking* neural network in real-time [4], [5], [6], [7]. Each SpiNNaker node is a bespoke multi-core chip with an on-chip router, and these nodes are interconnected through a two dimensional torus mesh and communicated using multi-cast mechanism [8], [9]. This paper demonstrates how SpiNNaker should operate when dealing with the traditional *non-spiking* multi-layer perceptron (MLP) networks with backpropagation (BP) learning rule, using the multi-cast communication system. We firstly investigate how to map the checker-boarding (or block-block) partitioning (CBP) scheme onto SpiNNaker. Based on such a study, a new algorithm called pipelined CBP (PCBP) is then proposed for partitioning and mapping MLP networks onto SpiNNaker. The performance evaluation of such mappings is also an important object of this paper.

The PCBP algorithm distributes the weight matrix using the checker-boarding (or block-block) partitioning (CBP) scheme. The CBP scheme cuts the whole weight matrix into small sub-matrices enabling the communication to be localized and reducing the number of communication packets. In the context of neural networks, the CBP scheme was used in [10], [11] and others, but none of them has addressed the parallelism *within* each individual submatrix. Based on the CBP scheme, we have developed the PCBP scheme. In the PCBP scheme, in addition to the traditional group of cores which do the vector-matrix computation, an extra two groups of cores are employed to compute the partial sums and outputs. The three groups of cores are able to work in parallel and produce a six-stage pipeline, allowing the overlap of computation and communication. Previous work has considered pipelined implementations, but it was either based on pipelining the work in each neural network layer [12] or based on pipelining between patterns [13]. They can not be applied to recurrent neural networks (RNNs), since all layers in RNNs are updated concurrently. Our algorithm, on the other hand, overcomes this barrier by considering pipelining within each partition. Although the pipelined checker-boarding partitioning (PCBP) scheme we proposed works for both feed-forward and recurrent networks, we focused mostly on recurrent networks in this paper, based on three reasons. 1. More and more neural models are built based on RNNs to simulate more complex situation. 2. RNNs are much more computationally demanding to train than FFNNs. 3. Less research work of parallel implementation has been taken on RNNs than on FFNNs.

The performance of PCBP on SpiNNaker is evaluated in a semi-experimental and semi-analytical way, and it is compared with the performance based on CBP scheme. The performance curves we produced show that with the PCBP scheme, we can achieve better speedup than with the traditional non-pipelined CBP scheme on SpiNNaker. The work shows that SpiNNaker not only can support spiking neural networks (as shown in previous papers), but is also able to deal with non-spiking MLP networks.

The rest of the paper is organized as follows: Section II describes the computations required by MLP networks learning using BP. Section III presents the traditional CBP partitioning that will be used as the base line for comparison.

978-0-7695-4120-4/10 $26.00 © 2010 IEEE
DOI 10.1109/ISPDC.2010.10

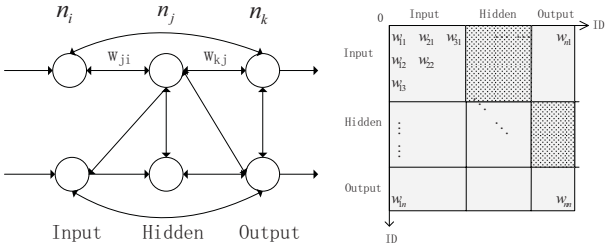


IEEE computer society

The new pipelined algorithm proposed is described and analyzed in Section IV. Sections V and VI present the analytical and simulation results, respectively. Section VII presents the discussion. Section VIII summarizes the paper.

## II. Neural network models

### A. Feedforward and recurrent neural networks




(a) A random recurrent network. (b) Weight matrix.

Figure 1. Neural network models and the weight matrix

A feedforward network (FFNN), as a standard form of multi-layer perceptron, consists of a number of simple neuron or units, organized in layers. Neurons in one layer are connected only to neurons in the next layer. There is no self-connection or connection within the same layer. In such networks, groups are updated in the order in which they appear in the network's group array (sequential updates). Only one layer need to be computed at a time.

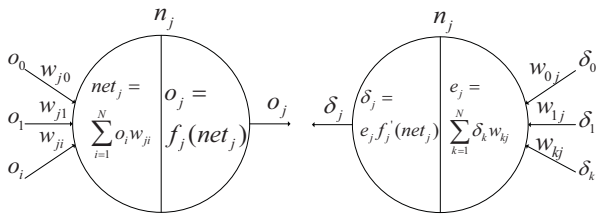
Recurrent Neural Networks (RNNs), as shown in Figure 1(a), not only have feedforward connections but also have feedback connections. A neuron in RNN may connect to any other neurons including itself. Further more, RNNs use concurrent updates and propagate error derivatives backwards through time. That is, in each time tick, all layers first update their inputs and then all layers update their outputs. Backpropagation will not start until the forward phase running for a given number of time ticks. Backpropagation then loops for the same number of ticks. In this case, all layer need to be computed at each time. The way neurons are connected and the way they are updated make RNNs more computationally intensive than FFNNs.

The weights can be seen as a matrix shown in Figure 1(b). Depending on the connectivity, the weight matrix may be dense or sparse. In FFNNs, elements are spread only in several areas indicated by dot-filled blocks in Figure 1(b), while in RNNs, weights are spread over the whole matrix. In this paper, we will discuss the partitioning scheme based on the weight matrix.

### B. BP learning rule

The BP algorithm was introduced in 1986 [14] and is the most common learning algorithm for training multi-layer perceptron (MLP) networks. At the core of the BP algorithm is a delta rule which computes weight changes in proportion to the difference between the actual output vector achieved and the target output vector provided by the training pattern [15]. We denote neurons in the input layer, hidden layer and output layer as $n_i$, $n_j$, $n_k$ respectively; $w_{ji}$ and $w_{kj}$ are the weight of connection from neuron $n_i$ to $n_j$ and $n_j$ to $n_k$ respectively.

**Forward Phase** — During the forward phase, information propagates from the input layer to the output layer (Figure 2(a)).




(a) Forward phase. (b) Backward phase.

Figure 2. Computational phases of a MLP network with BP learning

In each step of propagation, neuron $n_j$ receives an output vector $o_0$, ...,$o_i$ from the previous layer and produces a net input $net_j$ according to

$$net_j = \sum_{i=1}^{N} o_i w_{ji}. \tag{1}$$

The generated $net_j$ is then passed to a continuous non-linear activation function $f_j$ to generate the output $o_j$ according to

$$o_j = f_j(net_j). \tag{2}$$

The output $o_j$ will be passed to neurons in the next layer until the propagation reaches the output layer.

**Backward Phase** — the BP algorithm requires a backward phase in the neural network layers to propagate back a delta error $\delta$ that is used to generate weight changes. In the backward phase, firstly, the output $o_k$ produced in the output layer is compared with the target $t_k$ to generate an error $e_k$ which is $e_k = t_k - o_k$. This error $e_k$ is then used to produce the delta error $\delta_k$ given by

$$\delta_k = e_k f'(net_k) = (t_k - o_k) f'(net_k). \tag{3}$$

The delta error $\delta_k$ can be used to produced weight updates of $w_{kj}$ denoted by $\Delta w_{kj}$ according to $\Delta w_{kj} = \eta \delta_k o_j$, where $\eta$ is the learning rate. Then, $\delta_k$ is propagated back to the previous layer and is used to compute the error $e_j$ according to

$$e_j = \sum_{k=1}^{N} \delta_k w_{kj}, \tag{4}$$

as indicated in Figure 2(b). The delta error of this layer $\delta_j$ is given by

$$\delta_j = e_j f'(net_j) = \sum_{k=1}^{N} \delta_k w_{kj} f'(net_j). \tag{5}$$

Weight changes $\Delta w_{ji}$ for connections from $n_i$ to $n_j$ are given by

$$\Delta w_{ji} = \eta \delta_j o_i. \tag{6}$$

## III. Non-pipelined model

### A. Review of partitioning schemes

Depending on the level of parallelism, MLP networks with BP learning can be parallelized either by training on multiple patterns simulaneously or by parallelizing the computations of updating the weights within neural networks, or by a hybrid of these two [16]. Since most of the learning algorithms compute weight changes on a per-pattern basis, pattern parallelism can be easily applied on top of most network parallelism. As a result, in this paper, we focus on parallelism within neural networks.

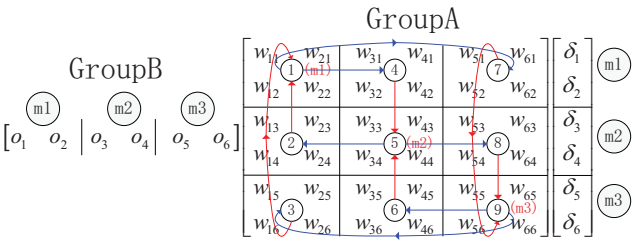
The multiplication-accumulation operations in a propagation tick (equations 1 and 4) can be parallelized. The weight matrix can be partitioned by element, row, column, or sub-matrix, corresponding to complete weight partitioning [17], inset grouping [18], outset grouping and *checkerboarding partitioning schemes* (CBP) [10], [11] respectively. Complete weight partitioning allocates one processor per weight to maximize the concurrency but it incurs too much communications. Inset or outset groupings are efficient in either the forward or the backward phase. Some schemes allocate both inset weights and outset weights to the same processor by duplicating weights [16], but this causes inefficiency during weight updating. The CBP scheme partitions the weight matrix into square sub-matrices and therefore optimizes both forward and backward phases.

### B. CBP scheme and non-pipelined model

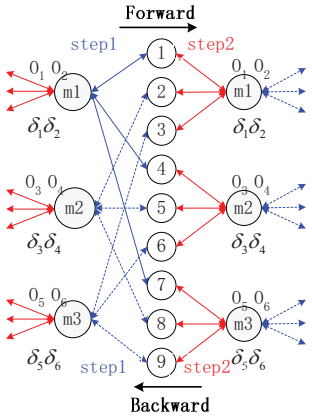
We use the CBP partitioning scheme to split the weight matrix as the starting point of our algorithm. The first step, for simplicity, is to analyse the mapping of RNNs onto a 2D torus topology with one processor per node. Figure 3(a) shows an example of a 6x6 weight matrix mapped onto 9 nodes (or processors) interconnected by a 2D torus. Each processor keeps a 2x2 sub-matrix of weights in its local memory. We assign processors $1-9$ to GroupA responsible for the vector-matrix multiplication. Among those 9 processors in GroupA, we select 3 processors in the main diagonal and name them $m1-m3$. Then we also assign processors $m1-m3$ to GroupB which is responsible for the partial result accumulation and output computation.

### C. Communication pattern analysis

For a more general discussion, we consider now an $n$-neuron network and a torus with $p$ processors (nodes), although still maintaining the one processor per node relation. In fully connected networks, the size of the weight matrix is $n$ by $n$ and each processor stores and updates a $\frac{n}{\sqrt{p}}$ by $\frac{n}{\sqrt{p}}$ sub-matrix. Assuming the transmission of a packet along a link takes $\alpha$ time units, while sending or receiving




(a) Checkerboarding partitioning.




(b) Communication patterns.

Figure 3. CBP partitioning

a packet takes $\beta$ time units, in each forward tick we can divide the operations into two discrete steps following the communication pattern illustrated in Figure 3(b).

In step1, for each column, the GroupB processors compute their outputs according to Equation 2 and send them to GroupA for a specific row. The communication is localized within each row of processors. For this communication, the $\sqrt{p}$ processors in one row of the 2D torus can be seen as connected in a ring topology (see Figure 4(b)). The row-wise communication requires a broadcast[1] in the ring topology with $\sqrt{p}$ processors. The diameter[2] of the ring is $\frac{\sqrt{p}}{2}$. So the packets traveling time is $\alpha\frac{\sqrt{p}}{2}$. There are also two other operations, one send and one receive, which take $2\beta$ time units. There are $\frac{n}{\sqrt{p}}$ columns of weights in each processor, so the communication time for step1 of a forward tick $T_{c1}$ (for any $1 < p < n^2$) is

$$T_{c1} = (\alpha\frac{\sqrt{p}}{2} + 2\beta)\frac{n}{\sqrt{p}}. \tag{7}$$

In step2, for each column, GroupA processors in one column do the vector-matrix multiplication according to equation 1. Note that each processor in the column only produces a partial result. Partial results are sent to the

[1]Sending the same packet from a single processor to every other processor [19].

[2]The maximum distance of the network, where the distance is the minimum number of links between any pair of processor [19].

GroupB processors in the same column. The communication is localized within each column of processors. The $\sqrt{p}$ processors in one column of the torus can again be seen as connected in a ring topology. The column-wise communication requires the accumulation in a single processor[3] in a ring with $\sqrt{p}$ processors. For each single node accumulation, the diameter of the ring is $\frac{\sqrt{p}}{2}$. It takes $\alpha\frac{\sqrt{p}}{2}$ time units for the packet transmission. However, in step2, the first packet will arrive very soon since there is only one link to travel. In most practical situations, $\beta$ is larger than $\alpha$. So a new packet usually arrives before the previous packet was processed. As a result, the communication time for packets, $\alpha\frac{\sqrt{p}}{2}$, is hidden. We only consider the time of one send, one link transfer and $\sqrt{p}$ receive operations, which take $\beta(\sqrt{p}+1)+\alpha$ time units. There are $\frac{n}{\sqrt{p}}$ columns of weights in each processor, so the communication time for step2 of a forward tick $T_{c2}$ (for any $1 < p < n^2$) is

$$T_{c2} = [\beta(\sqrt{p}+1)+\alpha)]\frac{n}{\sqrt{p}}. \tag{8}$$

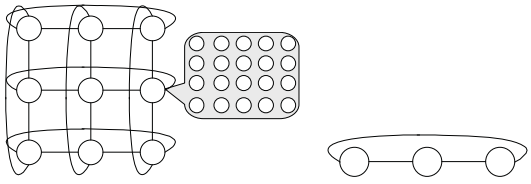
When $\sqrt{p}$ is large, $\beta(\sqrt{p}+1)+\alpha \approx \beta\sqrt{p}$. When $1 < p < n^2$, we obtain

$$T_{c1} = \frac{\alpha}{2}n + 2\beta\frac{n}{\sqrt{p}}, T_{c2} = \beta n. \tag{9}$$

In RNNs, BP starts when the forward phase has looped for a certain number of ticks. As you can find in Figure 3(b), the communication pattern and time in the backward phase are exactly the same as in the forward phase.

In this model, only the operation within the vector-matrix multiplication is parallelized. The operations between vector-matrix multiplication, output computation and communications cannot be parallelized and neither can they work in a pipelined mode, since the GroupB processors are selected from the processors in GroupA. This is also a common limitation we found in some other published implementations [10], [11]. Hereafter we refer to this model as the *non-pipelined model*.

### D. SpiNNaker



(a) Torus-connected CMPs (b) Ring

Figure 4. Topologies

What if each node in Figure 3(a) is replaced by a chip multiprocessor (CMP), as shown in Figure 4(a)? We use SpiNNaker as a paradigm for such an analysis. SpiNNaker is a system based on the torus-connected CMPs topology. It is a massively-parallel architecture originally designed for simulating large-scale spiking neural networks in real-time [8], [20]. It comprises multiple identical SpiNNaker chips connected in a 2D torus mesh topology (i.e., it also contains diagonal connections). Each SpiNNaker chip is a multi-core system containing 20 ARM968 processors and a router. Each processor has a 64KB local private memory called a data tightly-coupled memory (DTCM) storing performance essential data. All processors on one chip share an on-chip system RAM and an off-chip SDRAM for information exchange and extended data storage. In this paper, the diagonal connections are not required, therefore the system can be seen as a standard 2D torus topology (one SpiNNaker chip, or 20 cores, per node) as shown in Figure 4(a). The on-chip router supports multicast packets and handles both on-chip and off-chip traffic.

### E. Mapping onto SpiNNaker without pipeline

If the CBP is mapped onto SpiNNaker directly (without pipeline), all processors in a chip can be used for processing. Since there are 20 processors (in a 4x5 rectangle) in each chip, the diameter of the ring is reduced from $\frac{\sqrt{p}}{2}$ to $\frac{\sqrt{p}}{8}$, about $1/4$ of the original value. The time required on SpiNNaker without pipeline becomes:

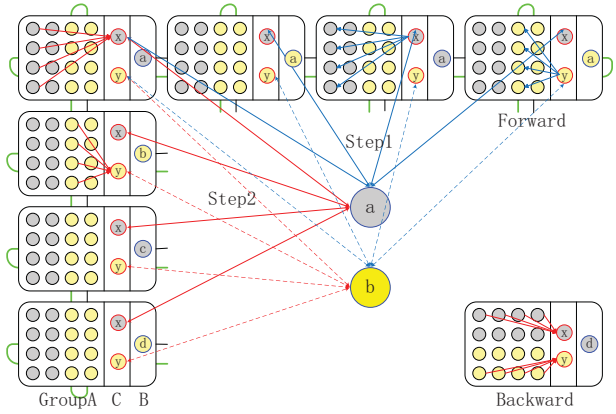
$$T_{c1} = \frac{\alpha}{8}n + 2\beta\frac{n}{\sqrt{p}}, T_{c2} = \beta n. \tag{10}$$

## IV. Pipelined model

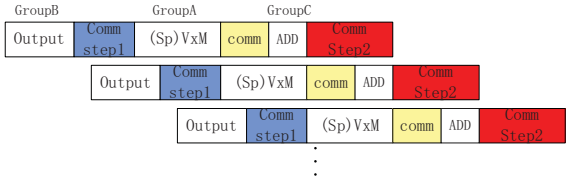
### A. Mapping onto SpiNNaker with the pipelined model

To map MLP networks on SpiNNaker or hardware with similar topology more efficiently, we propose a new pipelined model called PCBP. The mapping on SpiNNaker using PCBP scheme is illustrated in Figure 5(a). Each rectangle (with rounded corners) represents one SpiNNaker chip. Each circle in a rectangle denotes a processing core. We use 19 processors out of 20 in each chip. Among them 16 (4 by 4) processors are allocated to GroupA, 1 (a/b/c/d) processor is allocated to GroupB and the other 2 (x and y) are allocated to GroupC. In step1, GroupB processors produce outputs and send to GroupC processors. GroupC processors get single node broadcast packets from GroupB processors, and then forward to GroupA processors. In step2, GroupA processors do the vector matrix computation and send reults to GroupC processors with same color. Each GroupC processor receives packets from GroupA processors in two columns (2 by 4 processors) in turn and accumulate partial results, then forward the results to GroupB processors. Notice that what ever the number of chips are there in one column, we only require four GroupB processors in total, each responsible for one column, since there are only four columns of GroupA processors in total. GroupC processors need to send packets to two GroupB processors in the same color in turn (for example processor x sends to

[3]Sending a packet to a given processor from every other processor[19].




(a) Mapping on SpiNNaker using pipelined model (PCBP).




(b) The six-stage pipeline.

Figure 5. Mapping and the pipeline

processor a and c). The backward phase works exactly the same as the forward phase, but swaps the order of columns and rows. In each chip, the three groups of processors are working in parallel and produce a six-stage pipeline shown in Figure 5(b). Hereafter we refer to this mapping algorithm as the *pipelined model* (PCBP).

### B. Communication analysis

Compared to the CBP model shown in Figure III-C, in the PCBP model the on-chip communication between GroupA and GroupC processors is localized and in small-scale (4 packets per column); the accumulation operation performed by GroupC processors requires only four processor cycles per column. Both of them are fast enough to be hidden by the off-chip communication or other computation.

The off-chip row-wise/column-wise communication can be seen as communication in a ring topology with a diameter of $\frac{\sqrt{p}}{8}$; and with the help of the GroupC processors, the number of packets that go to a GroupB processor is reduced to $1/4$ of the original amount. We get the communication time in step1 $T_{pc1}$ and in step2 $T_{pc2}$ when $1 < p < n^2$ as

$$T_{pc1} = \frac{\alpha}{8}n + 2\beta\frac{n}{\sqrt{p}}, T_{pc2} = \frac{\beta}{4}n. \quad (11)$$

## V. Analytical comparison

With the basic analytical model of the mapping algorithms in hand, we can use these models to compare them. For that, we need to fix certain parameters. We assume: 1. there are a number $p$ of GroupA processors ($p = 20$ per chip in the non-pipelined model, $p = 16$ per chip in the pipelined model); 2. a multiply-accumulate operation of GroupA processors in a forward tick takes time $\theta_f$(including the average time of memory access), while the same operation takes $\theta_b$ in a backward tick ($\theta_f \neq \theta_b$ due to the data presentation); 3. the output computation of GroupB processors in the forward phase takes $o_f$, and the output computation in the backward phase takes $o_b$. Thus, computing a sub-matrix in a forward and backward tick takes $\theta_f\frac{n^2}{p}$ and $\theta_b\frac{n^2}{p}$ respectively; computing the outputs in a forward and backward tick takes $o_f\frac{n}{\sqrt{p}}$ and $o_b\frac{n}{\sqrt{p}}$ respectively. These can be summarized in Figure 6.

| Operations | Forward | | Backward | |
|---|---|---|---|---|
| | step1 | step2 | step1 | step2 |
| Comms. pipelined | $\frac{\alpha}{8}n + 2\beta\frac{n}{\sqrt{p}}$ | $\frac{\beta}{4}n$ | $\frac{\alpha}{8}n + 2\beta\frac{n}{\sqrt{p}}$ | $\frac{\beta}{4}n$ |
| Comms. non-pipelined | $\frac{\alpha}{8}n + 2\beta\frac{n}{\sqrt{p}}$ | $\beta n$ | $\frac{\alpha}{8}n + 2\beta\frac{n}{\sqrt{p}}$ | $\beta n$ |
| Comp. GroupA | $\theta_f\frac{n^2}{p}$ | | $\theta_b\frac{n^2}{p}$ | |
| Comp. GroupB | $o_f\frac{n}{\sqrt{p}}$ | | $o_b\frac{n}{\sqrt{p}}$ | |

Figure 6. Computation and communication cost

### A. On a single processor

Let $p = 1$, then no communication is required. We get the time required $T_{seri}$ for processing one forward tick plus one backward tick on a single processor

$$T_{seri} = (\theta_f + \theta_b)n^2 + (o_f + o_b)n. \quad (12)$$

### B. The non-pipelined model

In the non-pipelined mode, communication and computation are carried out sequentially. The time required in the non-pipelined model for one forward tick plus one backward tick $T^{np}_{para}$ is:

$$T^{np}_{para} = (\theta_f+\theta_b)\frac{n^2}{p}+(o_f+o_b)\frac{n}{\sqrt{p}}+(\frac{\alpha}{4}+2\beta+\frac{4\beta}{\sqrt{p}})n. \quad (13)$$

### C. The pipelined model

The pipelined model allows the overlap of communication and computation. In most practical problems, the link transmission time $\alpha$ is small compared to the packet send/receive time $\beta$, and the communication of step1 can be hidden behind the communication of step2 due to the pipeline (Figure 5(b)). $\theta_b$ is obviously larger than $\theta_f$ when using column order storage (CS) of the weight matrix. The computation times of GroupB processors are smaller than either the communication time or the computation time of GroupA processors (shown later), therefore computation of the GroupB processors is hidden behind either the communication or the computation time of GroupA processors. The time required in the pipelined model for one forward tick plus one backward tick $T^{p}_{para}$ relies on the relationship

between the computation and the communication because of the overlapping:

**Situation1** Communication takes more time than either forward or backward computation when $p \geq 4\theta_b \frac{n}{\beta}$. In this situation, computation is fully hidden behind communication.

$$T^p_{para} = \frac{\beta}{2} n \tag{14}$$

**Situation2** Communication takes more time than forward computation, but takes less time than backward computation, when $4\theta_f \frac{n}{\beta} \leq p < 4\theta_b \frac{n}{\beta}$. In this situation, communication is hidden partially.

$$T^p_{para} = \theta_b \frac{n^2}{p} + \frac{\beta}{4} n \tag{15}$$

**Situation3** Communication takes less time than forward or backward computation when $p < 4\theta_f \frac{n}{\beta}$. In this situation, the communication is hidden behind the computation.

$$T^p_{para} = (\theta_f + \theta_b) \frac{n^2}{p} \tag{16}$$

### *D. Speedup and efficiency*

We define the speedup $S$ and efficiency of an algorithm $E$:

$$S = \frac{T_{seri}}{T_{para}}, E = \frac{S}{p_{total}} \tag{17}$$

Where $p_{total}$ is the total number of processors in the system, including GroupA, GroupB and GroupC processors (For the sake of fairness, we use $p_{total} = 20$ per chip in both non-pipelined model and pipelined model when calculating the efficiency $E$. This highlights the improvement provided by the new method).

## VI. Simulation-based comparison

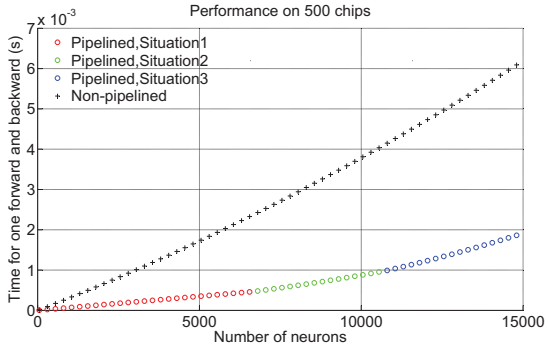
The performance of the system is evaluated analytically, since the real SpiNNaker chip is not available yet, and there is still quite a lot of software programming work remaining to be done to make the system fully functional. However, the analytical model presented helps us to determine the system scale and to predict the system performance.

We have run a medium size configuration up to 1000 chips to get first performance results on the SpiNNaker simulator built on the cycle accurate system level model [21]. By using 16-bit fixed-point arithmetic we have (in nanoseconds)[4]:

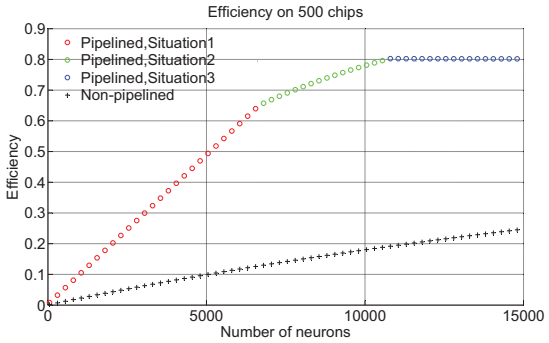
$$\theta_f = 26, \theta_b = 42, \alpha = 10, \beta = 140, o_f = 880, o_b = 1400 \tag{18}$$

The first evaluation we carried out is based on a scheme that runs a variety of scales of RNNs on a 500-chip

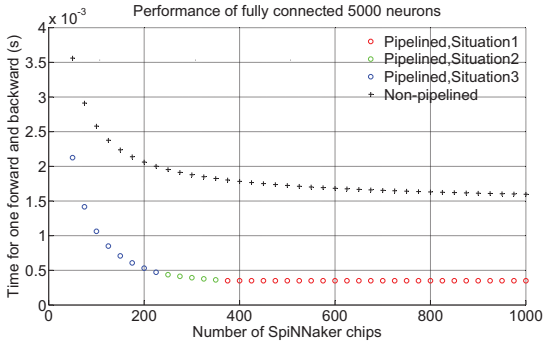
[4]According to Figure 6, when $p \geq 1600$, $\frac{\beta}{4} n \geq o_b \frac{n}{\sqrt{p}}$; when $p < 1600$ and $n > 1360$, $\theta_f \frac{n^2}{p} > o_f \frac{n}{\sqrt{p}}$ and $\theta_b \frac{n^2}{p} > o_b \frac{n}{\sqrt{p}}$. So on any network with a population of 1360 neurons and above, the computation time of GroupB processors is either hidden by the communication or hidden by the computation time of the GroupA processors.



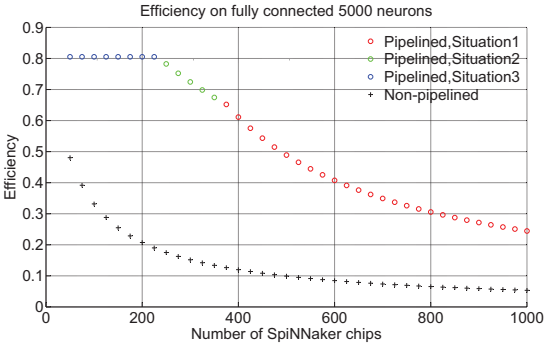

(a) A performance comparison between the non-pipelined (CBP) and the pipelined (PCBP) model.




(b) Efficiency comparison.




(c) Performance comparison.




(d) Efficiency comparison.

Figure 7. Performance analysis

SpiNNaker configuration. We show a performance comparison between the non-pipelined (CBP) and the pipelined (PCBP) model in Figure 7(a). Initially the performance of the pipelined model is dominated by the communication, shown in red color. When the scale of the network increases, the computation starts dominating the performance partially at the point of about 7,000 neurons, shown in green color. When the number of neurons are more than about 11,000, the computation is fully dominating, shown in blue. The efficiency comparison between the pipelined and non-pipelined model on a 500-chip SpiNNaker, modeling up to 15,000 fully-connected neurons, is shown in Figure 7(b). The pipelined model is more efficient than the non-pipelined model in most of the cases and its efficiency can reach as high as 0.8 when computation dominates. The efficiency of the non-pipelined model increases when the ratio of neurons to processors increases, getting close to 1 when the ratio is large. However, in that case, the system behavior is closer to serial computing than to parallel computing.

In the second simulation we considered the effect of running a fixed number (5000) of neurons on SpiNNaker, but changing the number of chips. As shown in Figure 7(c), it is very effective to use more chips initially, as the processing time drops dramatically. However, the gain in the performance becomes small at about 210 chips and above, where the communication becomes the bottleneck. The efficiency comparison is shown in Figure 7(d).

## VII. Discussion

**Partially-connected network.** We have restricted our discussion to fully-connected neuronal networks. The proposed pipelined model works also for partially-connected RNNs. Connection weights in a partially-connected network form a sparse matrix. The efficiency of computation is effected by the distribution of elements in the matrix. If the elements are evenly distributed, each processor carries out a similar amount of computation and the backpropagation algorithm can be parallelized well. Otherwise, the workloads in the processors are not balanced, in which case, some of the processors may be busy while others are idle. A partially-connected RNN is less computationally intensive due to the sparsity. However, the gain in the communication time by the PCBP partitioning is very little, as the communication can only be avoided when all of the elements in a column/row of the sub-matrix are non-zero (NZ) elements, which is a case very rare in a partial problem.

**Generalization.** There is no doubt that mapping schemes are mostly topology or architecture dependant. In this paper, we do analysis based on SpiNNaker (as a example of the torus-connected CMPs topology). The CBP mapping relies on a torus-like topology to partition the network into squares to achieve good balance between the forward and backward phases of computation. The main contribution of this paper is the new pipelined model which increases the efficiency of the original CBP mapping. The pipeline model may be applicable to other mapping schemes on other topologies.

**Workload balance.** There are three groups of processors. In each column/row of processing, the workloads of Groups B and C are fixed, while the workload of Group A is variable, depending on the scale of the neural network. We have shown in the paper that for any network with more than 1360 neurons on a system with less than 1600 processors, the processing time of Group A dominates. On a system with more than 1600 processors, the communication time dominates. As a result, the time cost of Groups B and C is always hidden in the pipeline when the system scale is large.

## VIII. Conclusion

This paper shows how to efficient implement MLP networks with the BP rule on SpiNNaker. An efficient pipelined mapping scheme is proposed over the traditional non-pipelined scheme. Compared to the traditional non-pipelined mode, our scheme is more efficient in most of the practical case. We also present a detailed performance analysis of such an implementation. The performance curves we have shown is significant and can be help to determine the ideal number of processors for a given scale of problems, providing a path to the further development of parallel solutions for the simulation of large-scale neural networks.

## Acknowledgment

We would like to thank the Engineering and Physical Sciences Research Council (EPSRC), Silistix, and ARM for support of this research. Dr. M. Luján holds a Royal Society University Research Fellowship. Prof. S.B. Furber is the recipient of a Royal Society Wolfson Merit Award.

## References

[1] R. Ayoubi and M. Bayoumi, "Efficient mapping algorithm of multilayer neural network on torus architecture," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 14, no. 9, pp. 932–943, Sept. 2003.

[2] S. K. Foo, P. Saratchandran, and N. Sundararajan, "Parallel implementation of backpropagation neural networks on a heterogeneous array of transputers," *Systems, Man, and Cybernetics, Part B, IEEE Transactions on*, vol. 27, no. 1, pp. 118–126, Feb 1997.

[3] S. Suresh, S. Omkar, and V. Mani, "Parallel implementation of back-propagation algorithm in networks of workstations," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, no. 1, pp. 24–34, Jan. 2005.

[4] X. Jin, F. Galluppi, C. Patterson, A. Rast, S. Davies, S. Temple, and S. Furber, "Algorithm and software for simulation of spiking neural networks on the multi-chip spinnaker system," in *Proc. 2010 International Joint Conference on Neural Networks*, 2010.

[5] X. Jin, A. Rast, F. Galluppi, S. Davies, and S. Furber, "Implementing spike-timing-dependent plasticity on spinnaker neuromorphic hardware," in *Proc. 2010 International Joint Conference on Neural Networks*, 2010.

[6] A. Rast, X. Jin, F. Galluppi, L. Plana, C. Patterson, and S. Furber, "Scalable event-driven native parallel processing: The spinnaker neuromimetic system," in *ACM International Conference on Computing Frontiers 2010*, 2010.

[7] A. D. Rast, F. Galluppi, X. Jin, and S. Furber, "The leaky integrate-and-fire neuron: A platform for synaptic model exploration on the spinnaker chip," in *Proc. 2010 International Joint Conference on Neural Networks*, 2010.

[8] X. Jin, S. Furber, and J. Woods, "Efficient modelling of spiking neural networks on a scalable chip multiprocessor," in *Proc. 2008 International Joint Conference on Neural Networks*, Hong Kong, 2008, inproceedings.

[9] X. Jin, "Parallel simulation of neural networks on spinnaker universal neuromorphic hardware," Ph.D. dissertation, Computer Science, The University of Manchester, 2010.

[10] V. Kumar, S. Shekhar, and M. B. Amin, "A scalable parallel formulation of the backpropagation algoritm for hypercubes and related architectures," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, 1994, inproceedings.

[11] T. Yukawa and T. Ishikawa, "Optimal parallel backpropagation schemes for mesh-connected and bus-connected multiprocessors," *Neural Networks, 1993., IEEE International Conference on*, vol. 3, pp. 1748–1753, 1993.

[12] A. Petrowski, G. Dreyfus, and C. Girault, "Performance analysis of a pipelined backpropagation parallel algorithm," *Neural Networks, IEEE Transactions on*, vol. 4, no. 6, pp. 970–981, Nov 1993.

[13] A. Petrowski, "Choosing among several parallel implementations of the backpropagation algorithm," in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, vol. 3, Jun-2 Jul 1994, pp. 1981–1986 vol.3.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*. Cambridge, MA, USA: MIT Press, 1986, ch. 8, pp. 318–362.

[15] D. Rumelhart, G. Hinton, and J. Mcclelland, *A General Framework for Parallel Distributed Processing*. MIT Press, 1986, ch. 2, pp. 45–76.

[16] V. Sudhakar and C. S. R. Murthy, "Efficient mapping of backpropagation algorithm onto a network of workstations," *Systems, Man, and Cybernetics, Part B, IEEE Transactions on*, vol. 28, pp. 841–848, 1998, article.

[17] G. Blelloch, "Network learning on the connection machine," in *In Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1987, pp. 323–326.

[18] X. Zhang, M. Mckenna, J. P. Mesirov, and D. L. Waltz, "An efficient implementation of the back-propagation algorithm on the connection machine cm-2," *Advances in neural information processing systems*, vol. 2, pp. 801–809, 1990.

[19] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[20] M. Khan, D. Lester, L. Plana, A. Rast, X. Jin, E.Painkras, and S. Furber, "Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor," in *In Proc. Intl. Joint Conf. on Neural Networks (IJCNN2008)*, Hong Kong, 2008, article.

[21] M. Khan, E. Painkras, X. Jin, L. Plana, J. Woods, and S. Furber, "System level modelling for spinnaker cmp system," in *Proc. 1st International Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO'09)*, 2009.