Parallelization of SINNs for Shift and Rotation Invariant Pattern Recognition

Sabine Kröner[1], Michael Nölle[2], Gerald Schreiber[3]

[1]Technologie-Zentrum Informatik, Abteilung Bildverarbeitung, Universität Bremen,
Universitätsallee 21-23, D-28359 Bremen, e-mail: kroener@informatik.uni-bremen.de
currently on leave to: Image Processing Laboratory, DEEI, University of Trieste,
Via Valerio 10, I-34127 Trieste, Italy

[2]Institute for Software Technology and Parallel Systems, Liechtensteinstraße 22,
A-1090 Vienna, Austria
now: Institut für Informatik, Lehrstuhl für Mustererkennung und Bildverarbeitung,
Universität Freiburg, Am Flughafen 17, D-79085 Freiburg i.Br.,
e-mail: michael.noelle@informatik.uni-freiburg.de

[3]bbcom Broadband Communications, Harburger Schloßstr. 6-12, D-21079 Hamburg,
e-mail: g.schreiber@bbcom-hh.de

Abstract

In this paper an algorithm is presented for the parallel implementation and data distribution of structured invariant neural networks (SINN) for invariant pattern recognition.

Structured invariant neural networks have shown convincing results for the task of shift and rotation invariant pattern recognition. They perform the invariant feature extraction and adaptive classification simultaneously in a sparsely connected architecture with shared weights, thus leading to a fast invariant recognition of gray scale images.

Additional speed up of the image processing system can be achieved by parallelization. Image capturing and preprocessing with parallel computers can lead to an enormous acceleration. Here we present a parallel implementation of the SINN that is necessary to meet real time demands also for the task of invariant pattern recognition. First we give a reordering algorithm for the highly irregular and complex data access patterns of the SINN minimizing the communication overhead. This algorithm accepts a wide class of possible input distributions determined by the implementation at the preceeding preprocessing step. The resulting data distribution allows the local computation of parts of the network architecture in parallel on a large number of processors. Our experimental results show a significant speed up of this parallelization compared to a sequential computation of the SINN.

1 Introduction

During the last 10 years neural networks have shown convincing results in pattern recognition applications. For the task of invariant pattern recognition where an object has to be recognized regardless of its position and orientation in the image plane structured multilayer feedforward neural networks gained considerable success [2, 3, 9, 17]. Their characteristic is that a-priori knowledge about the task to be performed is already built into the architecture by use of nodes with shared weight vectors. In [8] a class of structured invariant neural networks (SINN) has been proposed for shift and rotation invariant pattern recognition that performs the invariant feature extraction and the adaptive classification simultaneously. The architecture of the SINN is regularly but sparsely connected and uses layerwise shared weights. Experiments have shown [8] that the SINNs allow a fast invariant recognition of gray scale images at considerably lower computational costs compared to other methods for invariant pattern recognition, e.g. higher order neural networks or the method of invariant integration.

Neural networks, especially the class SINN presented here, are ideally suited for parallel evaluation. By constructing a special data distribution it is possible to divide the whole problem into equally sized independent sub-problems, the merging of the sub-results requires only minimal communication effort. Consequently almost linear speedup can be achieved for the SINN evaluation.

Knowing **what** distribution is required for the parallel SINN evaluation does only solve half the problem of parallelization. We need to know **how** that particular distribution can be generated. This issue is often neglected, since the input data is assumed to be stored on a disk and any data permutation can be performed off-line. In practice, the input image will be taken by a camera and fed into the parallel computer. Before the SINN is set to work, typically some preprocessing such as normalization of brightness and contrast, noise reduction, etc. is done. In a parallel environment of course, all those preprocessing steps are parallelized. Thus, the input to the SINN is a distributed image, where the distribution type is determined by the preceeding operation. We introduce a class of so-called *linear* data distributions that cover not only distributions suitable for SINNs and various image preprocessing algorithms but also many distribution schemas proposed in the literature [4, 5, 13]. Further, we propose algorithms for the efficient redistribution of linear distributions.

The paper is organized as follows. After some general definitions about neural networks with shared weights in Section 2, in Section 3 we introduce the architecture of the SINNs for shift and 90-degree rotation invariant pattern recognition. Since for the implementation of the SINN a parallel computer with de Bruijn connection topology is used in Section 4 we give some definitions on de Bruijn graphs and describe how trees can be embedded in such a topology. The issue of data distribution and redistribution is covered in Section 5. Experimental results about the speedup that can be achieved by a parallel implementation with respect to a sequential one are given in Section 5.4. The paper finishes with a conclusion.

2 Definitions on multilayer feedforward neural networks with shared weights

A *layered feedforward network architecture* $\mathcal{N}^r_{e,a}$ is a directed acyclic graph with a sequence of e input nodes which feed the input patterns into the rest of the network. They are followed by $r-1$ ($r \in \mathbb{N}$) intermediate (*hidden*) layers of nodes, and a final output layer with a nodes where the result of the computation is read off. Every node is connected only to nodes in the next layer. Nodes that are neither input nor output nodes are called *hidden* nodes.

The *indegree* $fan_{in}(k)$ of a node k is the number of incoming edges for this node. The *outdegree* $fan_{out}(k)$ is defined analogously.

To every node k with indegree $n \in \mathbb{N}$ a triplet $(\vec{s}_k, \vec{w}_k, \theta_k)$ is assigned, where $s : \mathbb{R} \to \mathbb{R}$ is a connecting function combining the input vector $\vec{x}_k$ into the node k with the weight vector $\vec{w}_k \in \mathbb{R}^n$, and $\theta_k : \mathbb{R} \to \mathbb{R}$ is the activation function. The connecting function, the weight vector, and the activation function define the parameters of the node function $F(\vec{x}_k) = y_k$ which determines the mapping of the node k from the input vector $\vec{x}_k = (x_1, \ldots, x_{fan_{in(k)}})$ to the activation x_k. Usually the connecting function is defined as $s_k = \vec{w}_k^T \cdot \vec{x}_k$, but also combinations of higher order like in sigma-pi units [12] are in use. For all input nodes (indegree 1) the weight vectors and the activation functions are fixed: $\vec{w} := (1)$, $\theta :=$ Id. The activation functions for all other nodes can be distinguished in four different types: linear functions, hard limiter functions, sigmoidal functions, and gaussian activation functions. Figure 1 shows the characteristics of these functions. The

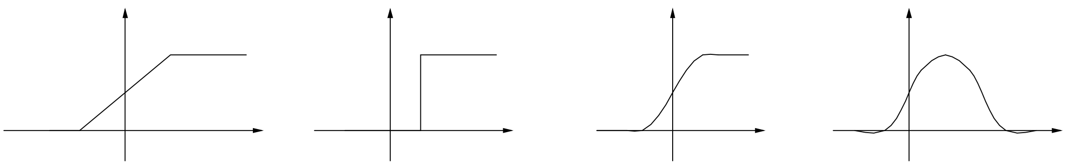


Figure 1: Characteristic of four different activation functions: linear, hard limiter, sigmoidal, and gaussian

range is usually limited to the interval $[0,1]$ or $[-1,1]$. In most network architectures the activation functions for all nodes in the network or at least for all nodes in the same layer are identical.

An *architecture* $\mathcal{N}^r_{e,a}$ with given triplets $(s, \vec{w}, \theta)$ for all nodes defines a *net* $N^r_{e,a}$. With the net itself a function $F : \mathbb{R}^e \mapsto \mathbb{R}^a$ is associated. A set of $m \in \mathbb{N}$ input vectors $\vec{x}^l \in \mathbb{R}^e$ $(l = 1, \ldots, m)$ arranged as m rows in a $(m \times e)$-matrix S is denoted *input matrix* for a net $N^r_{e,a}$. An input-matrix S is mapped to an $(m \times a)$*-output-matrix* T by a net $N^r_{e,a}$.

Consider all nodes of the architecture $\mathcal{N}^r_{e,a}$ within one layer (except the input layer) with the same indegree $d \in \mathbb{N}$ and the same activation function. All nodes k for whose components of their weight vectors $\vec{w}_k \in \mathbb{R}^d$ exists a permutation $\pi_k : \mathbb{R}^d \to \mathbb{R}^d$ so that $\pi_k(\vec{w}_k) = \vec{w}\ \forall k$ for some vector $\vec{w} \in \mathbb{R}^d$, are elements of the same *node class* $C_{\vec{w}}$. An architecture $\mathcal{N}^r_{e,a}$ is called *structured* if at least one node class has more than one element. An example of a one-layer architecture $\mathcal{N}^1_{e,a}$ with nodes of two node classes is shown in Fig. 2.

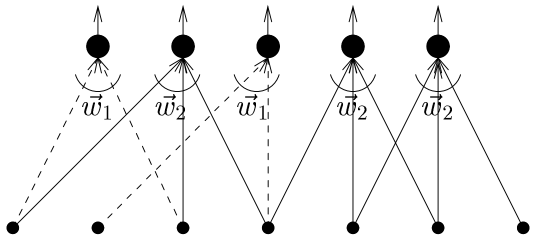


Figure 2: A one-layer architecture $\mathcal{N}^1_{7,5}$ with nodes belonging to two different node classes $C_{\vec{w}_1}$ and $C_{\vec{w}_2}$

3 Structured invariant neural networks

3.1 The architecture

The structured invariant neural network (SINN) is used for the recognition of patterns invariant with respect to their position and orientation. Some results are shown in [6]. Here the architecture is first described for the cyclic shift invariant recognition of a one-dimensional input vector of $e = K$, $K \in \mathbb{N}$ elements. These input elements can be gray values directly taken from a line of image pixels or higher order combinations of pixel values.

Definition 1 (SINN-architecture)
A structured invariant neural network (SINN)-architecture *is defined as a n-layer feed-forward architecture with $e = K$, $K \in \mathbb{N}$, input nodes and one output node ($a = 1$). K is factorized as $K := p_0 \cdot p_1 \cdot p_2 \cdot \ldots p_n$, $p_i \in \mathbb{N}$, with $p_0 := 1$ and $1 < p_i \leq K$ for $1 \leq i \leq n$. The cardinality of the sets $H^j := \{h^j_0, \ldots, h^j_{a_j-1}\}$ of nodes in the layers $j = 0, \ldots, n$ is given as $a_j = \prod_{i=1}^{j} p_i$. The layer H^n is also referred to as input layer, and the layer H^0 as output layer. The outdegree of all nodes of the sets $H^j := \{h^j_0, \ldots, h^j_{a_j-1}\}$, $j = 0, \ldots, n$, is defined as*

$$fan_{out}(h^j_{\alpha_j}) = 1 \ , \quad for\ \alpha_j = 0, \ldots, a_j - 1 \ .$$

For the indegree of all nodes of the sets $H^j := \{h^j_0, \ldots, h^j_{a_j-1}\},\ j = 0, \ldots, n-1,$

$$fan_{in}(h^j_{\alpha_j}) = p_{j+1}\ , \quad for \ \ \alpha_j = 0, \ldots, a_j - 1$$

holds. The connectivity between the sets H^{j+1} *and* H^j, $j = 0, \ldots, n-1$, *of nodes of two succeeding layers is described as*

$$\bigcup_{\alpha_j=0}^{a_j-1} \{h^{j+1}_{\alpha_j+i\cdot a_j} | 0 \le i < p_{j+1}\} \times \{h^j_{\alpha_j}\}\ .$$

The node functions F^j *are defined to be identical for all nodes in one layer:*

$$F^j_0 = \ldots = F^j_{a_j-1} \quad for\ j = 0, \ldots, n-1\ , \tag{1}$$

and to produce the same result for every cyclic permutation $\tau(\vec{x}_{\alpha_j})$ *of the elements of the input vector* $\vec{x}_{\alpha_j}$ *into a node* $h^j_{\alpha_j}$*:*

$$F^j(\tau(\vec{x}_{\alpha_j})) = F^j(\vec{x}_{\alpha_j})\ .$$

To the nodes of the input layer H^n *no node functions are assigned.*

Fig. 3 shows the parameter assignments in an SINN-architecture.

Depending on the factorization of K, different SINN-architectures can be generated to a given dimension of the input vector. Fig. 4 shows the SINN-architecture for an input pattern of length 12, factorized as $K = 2 \cdot 3 \cdot 2$. It has to be noted that the factors need not be prime.

With a group theoretic approach it can be proved that a SINN produces an output that is invariant with regard to cyclic shifts of the input elements [7]. On each succeeding layer invariance with respect to a subgroup of the group of cyclic shifts of K elements is achieved finally leading to invariance with respect to the full group. The degree of completeness of the invariance of the final network output depends on the degree of completeness of the node functions with respect to cyclic shifts of the elements of the input vector $\vec{x}_{\alpha_j}$ into a node $h^j_{\alpha_j}$, and on the number of layers. Completeness of the mapping is achieved for a one-layer network with a complete node-function.

The recognition of 2D patterns invariant with respect to cyclic shifts is performed in analogy to the one-dimensional case by factorizing both dimensions independently from each other and constructing a similar 2D-architecture. As an alternative, based on the chosen factorization the 2D pattern elements can be reordered to a one dimensional vector that can be evaluated with the one-dimensional SINN-architecture defined above.

Moreover the shift invariant recognition of 2D patterns can be extended to additional invariance with respect to rotations of multiples of 90°. The connection structure between two succeeding layers of a SINN for this case is shown in Fig. 5. It is recursively repeated for all layers of the network architecture.

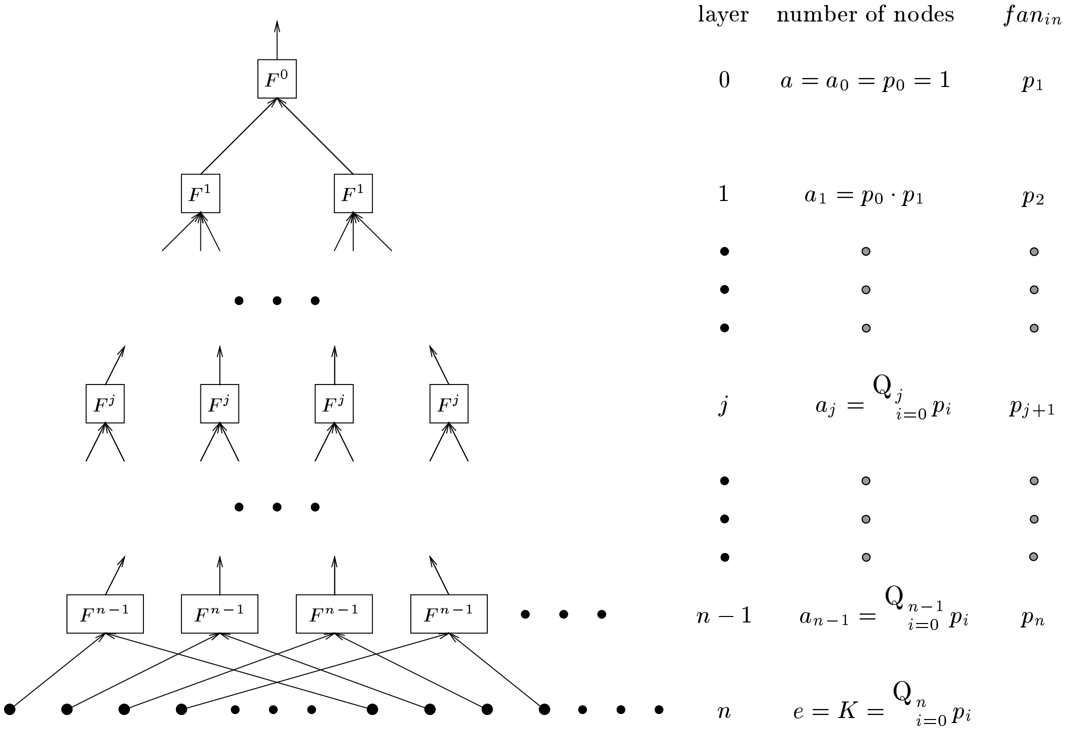


Figure 3: SINN-architecture

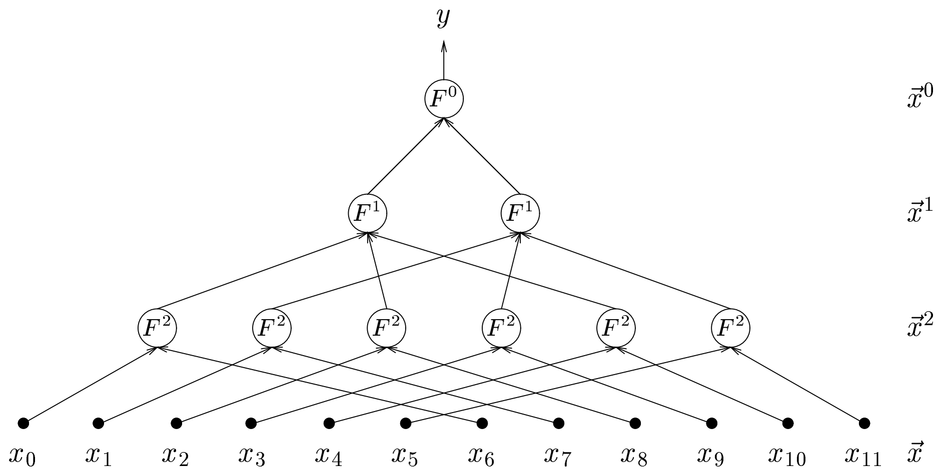


Figure 4: SINN-architecture for an input pattern of length 12, factorized as $K = 2 \cdot 3 \cdot 2$

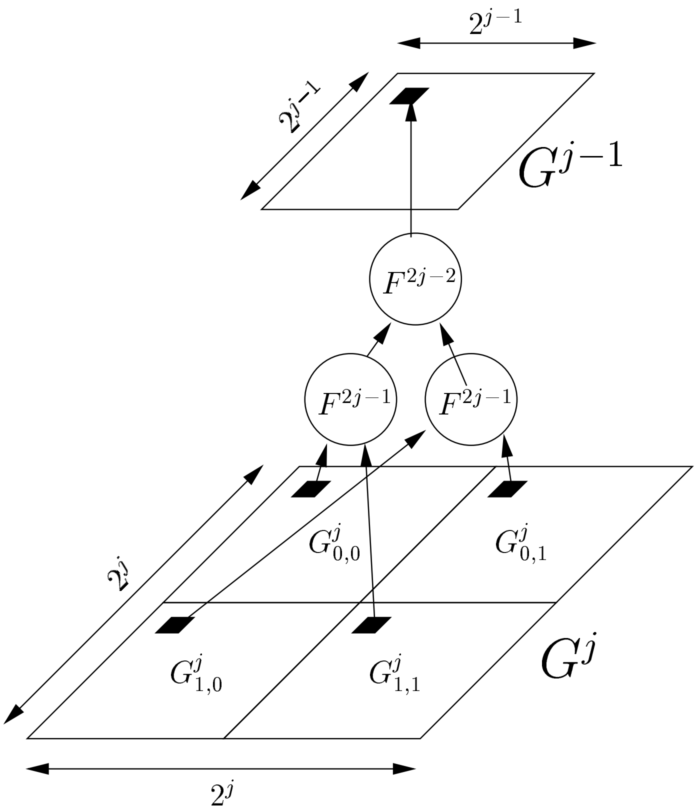


Figure 5: Recursive structure of a SINN invariant with respect to cyclic shifts and rotations of multiples of 90° of gray scale images.

Also for this 2D shift and rotation invariant architecture a recursive reordering algorithm of the 2D pattern elements to a one-dimensional vector exists. Its parallel realization is given in chapter 5.2. The extension to rotations of multiples of 90° as well as the extension to invariance with respect to general rotations which is described in [6] is limited to the application on square images of size $2^n \times 2^n$ factorized in prime factors. Factorizations of other dimensions with odd factors in the recursive algorithm would not lead to invariance with respect to the transformation group of cyclic shifts and rotations of multiples of 90°.[8].

3.2 Realization of the node functions

Since there are no further constraints on the realization of the node functions unless those mentioned in Definition 1 a large variety of neural nets exists even for a fixed factorization of the input vector. The incoming values in a node can be connected via a weighted sum or a multiplication, thus including sigma-pi neurons. The activation functions can be any function.

A simple realization of a node function for a node k invariant with respect to cyclic shifts of the input elements is given by weighting all inputs with the same weight w_1^j:

$$F^j = \theta^j \left(w_1^j \sum_{i=1}^{fan_{in}(k)} x_i + w_0^j \right) . \tag{2}$$

After adding a threshold value w_0^j the output is determined by the activation function θ^j. A similar realization uses the multiplication as connecting function:

$$F^j = \theta^j \left(w_1^j \prod_{i=1}^{fan_{in}(k)} x_i + w_0^j \right) . \tag{3}$$

Obviously both these mappings are invariant but not complete with respect to the group of cyclic shifts acting on the input elements into the node. Completeness can be achieved with the following realization for a node with indegree two:

$$F^j = \theta^j \left(w_3^j \left(t^j (w_1^j x_0 + w_2^j x_1 + w_0^j) \right) + w_3^j \left(t^j (w_2^j x_0 + w_1^j x_1 + w_0^j) \right) \right) . \tag{4}$$

Here w_1^j, w_2^j, and w_3^j are the weights on the connectional links, w_0^j is a threshold value, and θ^j and t^j are two nonlinear activation functions. Fig.6 shows the node function F^j in the nodes of layer j.

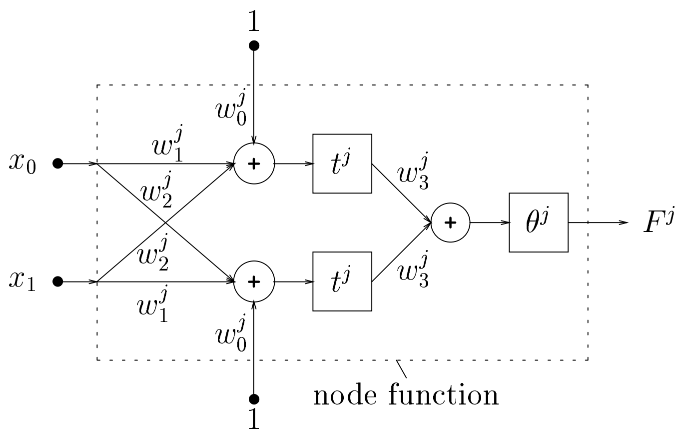


Figure 6: Node function F^j

This node function is symmetric and discriminative between the two values x_0 and x_1. The completeness property is achieved by the use of shared weights. Although a lot more realizations for the node functions are feasible in the following the investigations are limited to these three representative possibilities.

4 Parallelization of SINN

The definition of the SINN-architecture from the previous section gives - at first glance - a straight forward parallelization strategy. According to the factorization of the SINN-architecture the resulting data flow may be described by an appropriate evaluation tree (cf. Fig.4).

We have to perform the evaluation of the node functions $F^j, j = 0, \ldots, n-1$ in correspondence with the connectivity of any two nodes in succeeding layers (see 1)

$$\bigcup_{\alpha_j=0}^{a_j-1} \{h^{j+1}_{\alpha_j+i\cdot a_j} | 0 \leq i < p_{j+1}\} \times \{h^j_{\alpha_j}\} \ .$$

Therefore the resulting evaluation structure may be regarded as a level n tree. The degree of the tree corresponds in every level to the fan_{in}

$$fan_{in}(h^j_{\alpha_j}) = p_{j+1} \ , \quad \text{for } \alpha_j = 0, \ldots, a_j - 1 \ .$$

A data parallel implemention of the evaluation tree requires a distribution of the input data over the processors of the parallel system. The choice of any specific data distribution data maps the data dependencies expressed by the evaluation tree of the SINN-architecture onto data exchange requirements between the processors of the system. In parallel distributed memory systems these data exchange requirements are decisive for the efficiency of the implementation. This results from the fact that accessing data in local memory is by far faster than the access of non local data items.

For a good (i.e. fast) parallel implementation we naturally want to reduce the overheads of parallelization. The major overheads result from communication demands. These will mainly be determined by the total amount and number of data exchanges necessary and by the cost of each communication.

To analyze the impact of communication on our implementation we have to define a communication model. Here we use the graph family of de Bruijn graphs. Interpreting the vertices of the graph as processors and the edges as communication lines the de Bruijn graph serves as a description of the target system frequently addressed as **host**. We will briefly describe embeddings of trees as they are required for the SINN-architecture into de Bruijn graphs. With this preparation we analyze and optimize the communication demands of the consideredalgorithms in Section 5.

4.1 De Bruijn graphs

The de Bruijn graph or de Bruijn network is frequently used as a host for the implementations in the area of signal and image processing [13, 16]. The network is known to have logarithmic diameter and a constant node degree [1]. Moreover it can simulate any other bounded degree network with at most logarithmic slowdown [10].

Throughout the paper $\mathbb{Z}_B^n$ indicates a set of length n strings over the (digit) alphabet $\mathbb{Z}_B$. The concatenation of strings is written as the concatenation of the respective symbols (e.g. αu denotes the concatenation of the string u and the digit α). $|u|$ denotes the length of the string u.

Definition 2 (De Bruijn graph)
The directed base B de Bruijn graph $D_B(n) = (V_D, E_D)$ *of dimension n is given by the node set $V_D = \mathbb{Z}_B^n$ and the edge set*

$$E_D = \{(\alpha u, u\beta) | u \in \mathbb{Z}_B^{n-1} \wedge \alpha, \beta \in \mathbb{Z}_B\}.$$

The given definition of the base B de Bruijn graph is the natural extension of the commonly known de Bruijn graphs of fixed base $B = 2$.

In order to exchange data between two processors of a parallel system with de Bruijn communication network we have to indicate a route or path along which the data may travel.

To find a path from node x to node y of a de Bruijn graph we can simply concatenate the addresses $z = xy = x_{n-1} \dots x_0 y_{n-1} \dots y_0$. The edges to use for a travel from x to y are given by length n substrings of z: $(e_i, e_{i-1}) = (z_{2n-i-1} \dots z_{n-i}, z_{2n-i-2} \dots z_{n-i-1})$, $i = 0, \dots, n-1$. [1] [2] [3]

There are several cases where we can find a shorter route for a given source and destination. If we consider a pair $x = uv, y = vw \in \mathbb{Z}_B^n$, where $|u| = k$, the k length n substrings $(z_{k+n-i-1} \dots z_{k+n-i}, z_{k+n-i-2} \dots z_{k+n-i-1})$, $i = 0, \dots, k-1$, of

$$z = uvw = z_{k+n-1} \dots z_0 \tag{5}$$

describe a route of length $|u| = |w| = k$. A detailed discussion on routing in de Bruijn graphs may be found in [11].

4.2 Embedding B-ary trees in de Bruijn graphs

The embedding of a B-ary tree into de Bruijn graphs can be established in several ways. We will present an embedding that is especially suited for the SINN-architectures. It can be observed that the calculation tree of SINN-architecture requires only one active tree

[1] This is a modification of the well known destination tag routing algorithm [10].

[2] It is easy to see that the shortest route from 0 to $B^n - 1$ uses n intermediate nodes. That gives the worst case and establishes the logarithmic diameter.

[3] Note that we find a path from y to x using the reversed edge direction in the undirected de Bruijn graph.

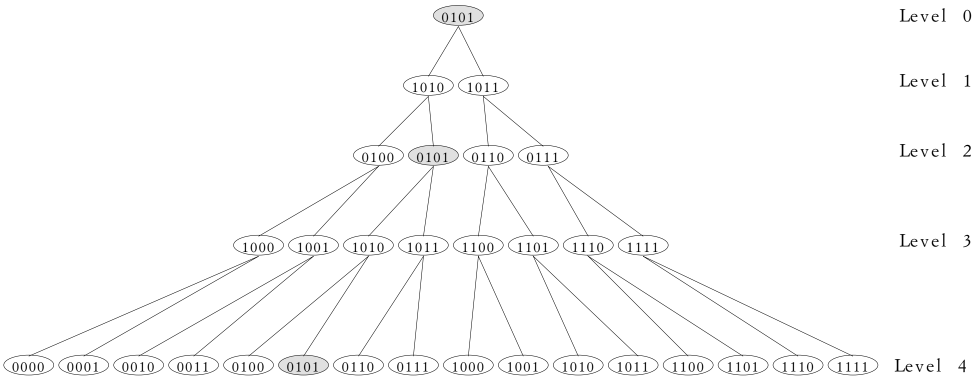


Figure 7: Embedding of the level 4 binary tree in the de Bruijn graph. The numbers indicate the de Bruijn node where the corresponding tree node is mapped to. The same de Bruijn node may be used on different levels of the tree (see the darker shaded nodes).

level at a time[4]. Therefore we embed the tree in an iterative manner and use it from the leaves to the root for the computation of the global evaluation.

Let the de Bruijn node $y^0 \in \mathbb{Z}_B^n$ be an arbitrary root address of a B-ary tree. The address of the β^{th} successor node of a level k node y^k is given by $y^{k+1} = y^k_{n-1,0}\beta = (y^k_{n-2}\dots y^k_0\beta),\ \beta \in \mathbb{Z}_B$. The superscript indicates the level of the tree. We have to assure that a de Bruijn node is used by at most one active tree node. By construction $(y^k, y^{k+1}) \in E_{D_B(n)}$. An example tree embedding is given in Figure 7. The addresses of the successors of the root $y^0 = (0\mathbf{101})$ within the de Bruijn graph are given by $(\mathbf{101}0)$ and $(\mathbf{101}1)$.

For any node $y^k \in \mathbb{Z}_B^n$ with $y^k_{n-1}\dots y^k_k = y^0_{n-k-1}\dots y^0_0$ a length k path from y^0 is given by Eq. (5).

The k^{th} level of a B-ary tree consists of B^k nodes. Using all possible settings for $y^k_{k-1}\dots y^k_0$ we can reach B^k different de Bruijn nodes. Thus, the given mapping of the level k tree nodes to de Bruijn nodes is onto. If $t^k \in \mathbb{Z}_B^k$ denotes a numbering of the level k nodes of the tree we map t^k to the node y^k with $t^k = y^k_{k-1}\dots y^k_0$. On the n^{th} level all de Bruijn nodes become the leaves of the tree.

Using this embedding tree nodes from different levels may be mapped to the same de Bruijn node (i.e. the root t^0, $t^2 = (01)$ and $t^4 = (0101)$ are mapped to the de Bruijn node $y^0 = y^2 = y^4 = (0101)$). Nevertheless, for any de Bruijn node there will be no interference between messages from different tree levels as long as we can distinguish between the subtrees where this message comes from. To see that, we assume two nodes $t^k, t^l, k < l$ are mapped to the de Bruijn node $y \in \mathbb{Z}_B^n$. Then $y_{k,0} = t^k$ and $y_{l,0} = t^l$ holds. Since we are using the tree from bottom to top it is necessary for t^k to be active that all subtrees of t^k starting at level $k+1$ are finished. The same is true for all subtrees of t^l. The addresses

[4]To be precise this is only true for a synchronous computation. However, in any path from a leaf to the root only one tree node is active at a time.

within the de Bruijn graph of all subtree root nodes are given by $y_{n-1,0}\beta, \beta \in \mathbb{Z}_B$ and are identical for t^l and t^k. Within the subtrees of t^l and t^k the same argumentation as before can be applied. Finally we reach the leaf level of the tree starting in t^l. That is we find a node that serves as a leaf for the subtree starting in t^l and as an inner tree node for the subtree starting in t^k. Since the leaf level of the tree consists of all de Bruijn nodes every node must have been a leaf before it can be an inner node. This guarantees that the sequence of all paths from the leaf to the root is preserved by our embedding.

5 Data distribution for SINNs

In this section we investigate data distributions, i.e. mappings of input data onto processors, for the efficient parallelization of SINN architectures on de Bruijn networks. In Section 4.2 it was shown that a complete B-ary tree with B^p leaf nodes (i.e. $\frac{B^{p+1}-1}{B-1}$ nodes in total) can be embedded into the de Bruijn graph $D_B(p)$ such that

- every node of the de Bruijn graph hosts exactly one leaf of the tree,
- the edges of the tree are mapped onto edges of the de Bruijn graph,
- and for every level of the tree, any node of the de Bruijn graph hosts at most one tree node of that level.

From the above it becomes clear, that any computation in the tree can be emulated by the de Bruijn graph without time loss, provided that only one level of the tree is active at a time.

When we attempt to use this embedding to map a SINN architecture onto a parallel computer with a de Bruijn interconnection network, we have to deal with two problems. First, the number of input nodes K will exceed the number of processors P by far. And second, a SINN architecture is not given by a complete B-ary tree in general.

In order to solve the first problem, we will develop an appropriate data distribution such that every node of the de Bruijn graph hosts the subset of the input data which is required to compute one node on level p of the SINN. This enables us to start the computation by locally (independently) evaluating levels $n, \ldots, p$ of the SINN. From that, we let the de Bruijn Graph emulate the remaining levels $p-1, \ldots, 0$ of the tree. However, the constraints are always met if we consider SINNs for shift and 90°-rotation invariant feature extraction, see Section 3.1.

We will evade the second problem by assuming that the number of input nodes $K := p_0 \cdot p_1 \cdot p_2 \cdot \ldots p_n$, is given by B^m, where B is the base of the de Bruijn graph (cf. Definition 1), further, let B be prime. At first, this seems a rather harsh restriction, but it is possible for an arbitrary SINN to be embedded into a larger SINN that matches the constraint. This is done by choosing $p'_0, \ldots p'_n$ such that p'_i is a power of B and $p'_i \geq p_i > p'_i/B$, for $i = 0, \ldots, n$.

It follows that the factors p_i are powers of B. If there exists i with $p_i = B^r$ and $r > 1$, the SINN architecture is still not identical to a complete B-ary tree. In this case we replace the nodes in the i-th level of the SINN by a r-level complete B-ary tree. The inner nodes of this replacement trees do only collect data from the leaves up to the root, where the node function is finally evaluated[5].

Summarizing, we can assume $p_i = B$ for $i = 1, \ldots, n$ resulting in $K = B^n$ without loss of generality. The input vector can thus be indexed over $\mathbb{Z}_B^n$.

In Section 5.1 we will define a class of so-called *linear* data distributions. These cover a wide range of data distribution concepts found in the literature, e.g. distributions defined by bit permutations [13] and an important subset of the regular distributions defined by *High Performance Fortran* (HPF) [4]. The choice of the "right" data distribution is decisive for the performance of a parallel implementation. In Section 5.2 we will construct a linear data distribution that is optimized for two-dimensional rotation and shift invariant SINNs. Having fixed a suitable distribution, the question arises how that distribution can be generated. An easy answer is to perform any data permutations offline, while the data is still stored on disk or on a single processor. However in reality, the input data is already distributed after image capturing via parallel IO or after some parallel image preprocessing has been done. Moreover, in most cases the data distribution required for the SINN computation will differ from any distribution suitable for image preprocessing. Thus, we investigate the redistribution of data in Section 5.3.

5.1 Linear data distributions

We describe a data distribution by a permutation $\phi : \mathbb{Z}_B^n \to \mathbb{Z}_B^n, \phi(x) = y$. The argument x is the index of an element of the input vector. The image y of x under ϕ is interpreted as follows: the most significant p digits form the index of the de Bruijn node which stores the data element with index x. The least significant $n-p$ digits determine the according memory location on that node.

data element B-ary index	ϕ	B-ary index		node index		memory index
$x_{n-1}, \ldots, x_0$	$\to$	$y_{n-1}, \ldots, y_0$	$=$	$y_{n-1}, \ldots, y_{n-p}$	,	$y_{n-p-1}, \ldots, y_0$

As B was assumed to be prime, $(\mathbb{Z}_B, \oplus, \odot)$ is a field, where $\oplus$ and $\odot$ denote addition and multiplication modulo B. Thus, $\mathbb{Z}_B^n$ is a n-dimensional finite vector space. For the purpose of this paper, $\mathbb{Z}_B^n$ can be treated similarly to the known vector spaces over the real or complex numbers (however, there are differences, e.g. in $\mathbb{Z}_B^n$ the sum of the component wise multiplication of two vectors does not define an inner product) [15].

[5]Depending on the node function F it might be possible to perform some data accumulation for the inner nodes of the replacement trees. If, for example, F^j is chosen as in Eq. (2) or (3), the inner nodes can add or multiply the input values, where the root nodes have additionally to evaluate the activation function.

We define the class of *linear data distributions*, where ϕ is given by a linear mapping in $\mathbb{Z}_B^n$:

Definition 3 (Linear data distribution)
Let $D \in \mathbb{Z}_B^{n \times n}$ be a $n \times n$ matrix where the entries are taken from $\mathbb{Z}_B$. D describes the distribution of a data object with B^n elements indexed over $\mathbb{Z}_B^n$:

$$xD = (x_{n-1}, \ldots, x_0) \begin{pmatrix} d_{n-1,n-1} & \cdots & d_{n-1,0} \\ \vdots & & \vdots \\ d_{0,n-1} & \cdots & d_{0,0} \end{pmatrix} = \overbrace{y_{n-1}, \ldots, y_{n-p}}^{\text{processor}}, \overbrace{y_{n-p-1}, \ldots, y_0}^{\text{memory}} .$$

We call D a **linear data distribution** *if D is one-to-one.*

With this schema, only balanced distributions are covered (i.e. all nodes host the same number of data elements). The extension to unbalanced distributions is described in [16].

5.2 Linear data distribution for two-dimensional SINNs

We consider a SINN architecture for the extraction of shift and 90°-rotation invariant features of two-dimensional gray scale images. For the rotation invariance the image dimensions must be of size $2^m \times 2^m$, see Section 3.1. Thus, the evaluation tree is isomorphic to a complete $2m$-level binary tree. Clearly, the optimal parallel evaluation is to compute as many levels of the tree locally (without any data interchange between processors). The aim of this section is to construct an optimal data distribution onto a 2^p-processors-de-Bruijn-network $D_2(p)$, such that all but the top p levels of the SINN can be evaluated locally[6].

Let $G^m = \{g(y,x) | x, y \in \mathbb{Z}_2^m\}$ be a two-dimensional image (input data object), where the pixels (data elements) are indexed over $\mathbb{Z}_2^m \times \mathbb{Z}_2^m$. (The parameters of the SINN as in Definition 1 are thus given by $n = 2m$, $K = 2^{2m}$, and $p_i = 2$ for $i = 1, \ldots n$.) We split G^m in four subimages $G^m_{\alpha,\beta} = \{g_{\alpha,\beta}(y,x) = g((\alpha,y),(\beta,x)) | x, y \in \mathbb{Z}_2^{m-1}\}$ for $\alpha, \beta \in \mathbb{Z}_2$. The SINN can then be described by the following recursion

$$F(G^j) = \widetilde{G^j} = \left\{ \begin{array}{ll} \widetilde{F^{2j-2}(F^{2j-1}(G^j_{0,0}, G^j_{1,1}), F^{2j-1}(G^j_{0,1}, G^j_{1,0}))} & : \ j > 0 \\ g(0,0) & : \ j = 0 \end{array} \right\}. \tag{7}$$

where the node functions F^{2j-1} and F^{2j-2} are to be applied on corresponding pixels of their argument images (see Figure 5).

As mentioned above, for an efficient parallel implementation on a binary de Bruijn graph with 2^p nodes, $D_2(p)$, we want to compute the first (lower) $2m - p$ levels of the recursion

[6] We do only consider balanced and non-redundant distributions here. Other distributions would lead to load imbalance or increase in workload.

locally, without invoking any communication. Thereby we reduce the size of the parallel problem by a factor of 2^{2m-p}. The remaining p levels of the SINN form a complete binary tree, whose embedding in $D_2(p)$ was studied in Section 4.2. We construct the appropriate data distribution by introducing permutations π_j, such that

$$F(G^j) = \widetilde{\pi_j G^j} = \left\{ \begin{array}{lcl} \widetilde{F^{2j-2}(F^{2j-1}(\pi_j G^j))} & : & j > 0 \\ g(0,0) & : & j = 0 \end{array} \right\} . \qquad (8)$$

Here the node functions F^{2j-1} and F^{2j-2} are now to be applied on successive pixels (rather than corresponding pixels, as it was the case in Equation 7).

Consider corresponding pixels in the four subimages $G^j_{0,0}$, $G^j_{0,1}$, $G^j_{1,0}$, and $G^j_{1,1}$ at coordinates $x, y \in \mathbb{Z}_2^{j-1}$. A comparison between Equations (7) and (8) reveals that π_j must map the pixels $g_{0,0}(x,y)$, $g_{1,1}(x,y)$, $g_{0,1}(x,y)$, and $g_{1,0}(x,y)$ consecutively. π_j turns out to be a linear mapping given by

$$\begin{aligned} \pi_j : \quad \mathbb{Z}_2^{2j} \to \mathbb{Z}_2^{2j} : \quad & \pi_j(y_{j-1}, \ldots, y_0, x_{j-1}, \ldots, x_0) = \\ & (y_{j-2}, \ldots, y_0, x_{j-2}, \ldots, x_0, x_{j-1} \oplus y_{j-1}, y_{j-1}). \end{aligned}$$

By unwrapping the recursion we get

$$\begin{aligned} F(G^m) &= F^0(F^1(\pi_1 \ldots F^{2j-2}(F^{2j-1}(\pi_j \ldots F^{2m-2}(F^{2m-1}(\pi_m G^m)) \ldots)) \ldots)) \\ &= F^0(F^1(\ldots F^{2m-1}(\pi(G^m)) \ldots)). \end{aligned}$$

with the linear one-to-one mapping

$$\begin{aligned} & \pi(y_{m-1}, \ldots, y_0, x_{m-1}, \ldots, x_0) \\ = \;& \pi_1 \pi_2 \cdots \pi_m (y_{m-1}, \ldots, y_0, x_{m-1}, \ldots, x_0) \\ = \;& \pi_1 \pi_2 \cdots \pi_{m-1}(y_{m-2}, \ldots, y_0, x_{m-2}, \ldots, x_0), x_{m-1} \oplus y_{m-1}, y_{m-1} \\ = \;& (x_0 \oplus y_0, y_0 \,,\, x_1 \oplus y_1, y_1 \,,\, \ldots \,,\, x_{m-1} \oplus y_{m-1}, y_{m-1}) \\ = \;& (y_{m-1}, \ldots, y_0, x_{m-1}, \ldots, x_0) D_\pi \,, \end{aligned}$$

$$D_\pi = \left(\begin{array}{cccccccc} & & & & & & 1 & 1 \\ & 0 & & & & \iddots & & \\ & & & & 1 & 1 & & \\ & & 1 & 1 & & 0 & & \\ 1 & 1 & & & & & & \\ \hline & & & & & & 1 & \\ & 0 & & & & \iddots & & \\ & & & & 1 & & & \\ & & 1 & & & 0 & & \\ 1 & & & & & & & \end{array} \right).$$

Thus, π is a linear data distribution. An example for the distribution of a 4×4 image and the parallel evaluation of the SINN is illustrated in Figure 8.

5.3 Linear data *re*distribution

In order to describe the conversion of a data distribution in de Bruijn networks, we will specify for every data element a path from its source to its destination node. For the redistributions constructed here all paths are of identical length r, $2p \geq r \geq p$, and all B^{p+1} edges of the de Bruijn graphs have to carry exactly B^{n-p-1} data elements in each of the r communication steps. Our data redistribution method was inspired by [5] and extends the results presented there.

The paths of length r in a de Bruijn graph can be indexed over $\mathbb{Z}_B^{p+r}$. Let $w \in \mathbb{Z}_B^{p+r}$ be such a path. Then the j-th node on this path is given by $w_{j+p-1}, \ldots, w_j$. Thus, the p least significant digits of w form the starting node, and the end node of the path is given by the p most significant digits of w. Consequently, we describe a r-step data redistribution by a mapping $V : \mathbb{Z}_B^n \to \mathbb{Z}_B^{p+r}$. V assigns to every data element the index of a path of length r. Let V be linear.

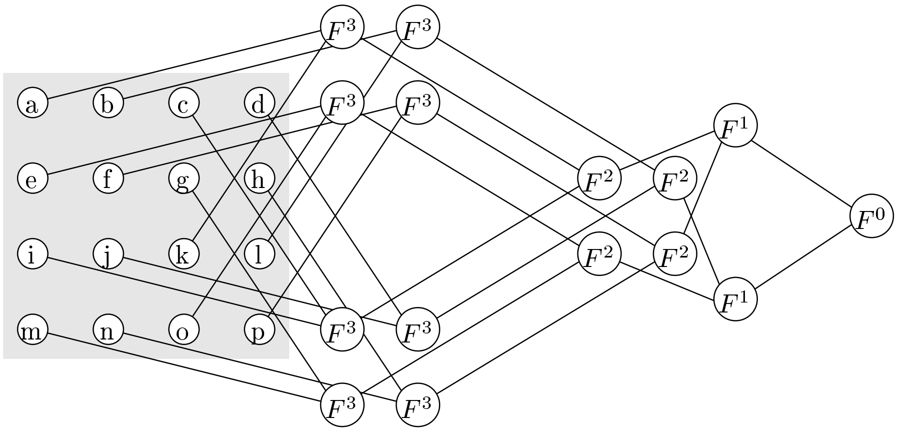
Consider two linear data distributions S and D, where S is the source distribution and D is the distribution that is accomplished by the redistribution V. V can again be described by a matrix. Obviously, S and D are part of V, as the start and end nodes of all paths have to match the source and destination distribution.

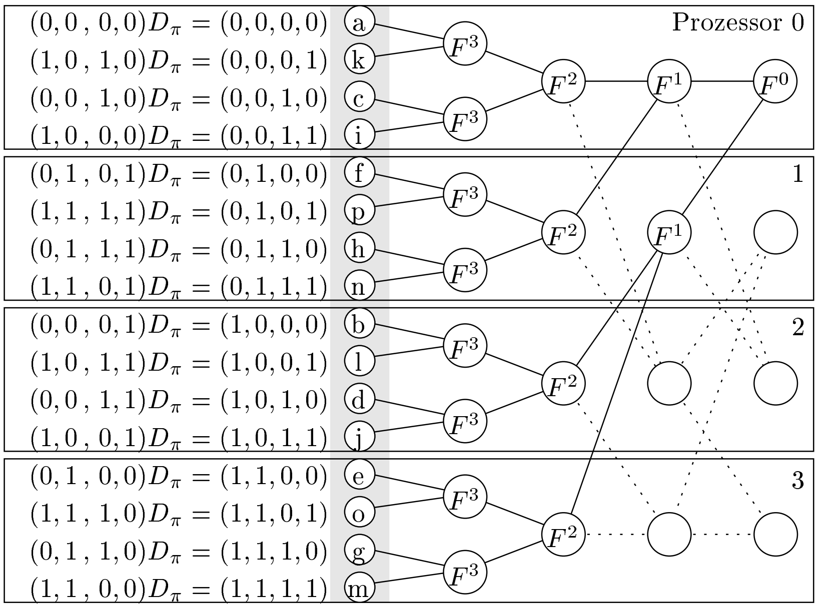
$$V = \left(\begin{array}{ccccccccc} \multicolumn{3}{c}{\overbrace{\begin{matrix} d_{n-1,n-1} & \cdots & d_{n-1,n-p} \end{matrix}}^{\text{destination } D}} & x_{n-1,r-p-1} & \cdots & x_{n-1,0} & \multicolumn{3}{c}{\overbrace{\begin{matrix} s_{n-1,n-1} & \cdots & s_{n-1,n-p} \end{matrix}}^{\text{source } S}} \\ \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots \\ d_{0,n-1} & \cdots & d_{0,n-p} & x_{0,r-p-1} & \cdots & x_{0,0} & s_{0,n-1} & \cdots & s_{0,n-p} \end{array} \right)$$

The data element with index $i \in \mathbb{Z}_B^n$ travels on the path iV, in the j-th communication step the data element with index i uses the de Bruijn edge with index iV^j, where

$$V^j = \begin{cases} d_{\cdot,n+j-r} \ldots d_{\cdot,n-p} \quad x_{\cdot,r-p-1} \ldots x_{\cdot,j-p} & : \; r > j \geq p \\ d_{\cdot,n+j-r} \ldots d_{\cdot,n-p} \quad x_{\cdot,r-p-1} \ldots x_{\cdot,0} \quad s_{\cdot,n-1} \ldots s_{n-p+j} & : \; p > j \geq r-p \\ x_{\cdot,j} \ldots x_{\cdot,0} \quad s_{\cdot,n-1} \ldots s_{n-p+j} & : \; r-p > j \geq 0 \end{cases}$$

What condition must be imposed on V^j such that exactly B^{n-p-1} data elements use the same edge in the j-th step? It follows from basic linear algebra that the column vectors of V^j must be linearly independent. Our task of constructing a data redistribution V is thus reduced to finding r and $X \in \mathbb{Z}_B^{n \times r-p}$ such that V^j has no defect in column vectors, i.e. $\mathrm{defect}(V^j) = 0$ for $j = 0, \ldots, r-1$. A closer examination reveals that for finding r, it





$$D_\pi = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Figure 8: top: Translation and 90°-rotation invariant SINN-architecture for a 4×4 image.
bottom: Data distribution and parallel evaluation of the SINN on 4 nodes of $D_2(2)$.

is sufficient to consider only the defect of

$$\widehat{V}^j = V^j \setminus X \quad = \quad \begin{cases} d_{\cdot,n+j-r} \dots d_{\cdot,n-p} & : \; r > j \geq p \\ d_{\cdot,n+j-r} \dots d_{\cdot,n-p} \quad s_{\cdot,n-1} \dots s_{n-p+j} & : \; p > j \geq r-p \\ \qquad\qquad\qquad\quad s_{\cdot,n-1} \dots s_{n-p+j} & : \; r-p > j \geq 0 \end{cases}$$

We will show that it is always possible to find a suitable X once r has been determined such that all $\widehat{V}^j$ are linearly independent. The following algorithm GetMinR serves that purpose:

Algorithm GetMinR

$r := p$
$j := 0$
while $j < p$
  if defect$(\widehat{V}^j) > 0$
    $r := r + 1$
  $j := j + 1$

Lemma 1 *Let D and S be linear data distributions. The algorithm GetMinR computes $r \geq p$, such that* defect$(\widehat{V}^j) = 0$ *for all $j < r$. There exists no smaller $r \geq p$ with that property.*

The easy proof is left to the reader. Please note that at most $2p$ communication steps are required. The algorithm GetX computes X for the number of communication steps r given by GetMinR. The usage of the various variables is illustrated in Figure 9.

Algorithm GetX

$x_{\cdot,r-p-1} := d_{\cdot,n-p-1}$
for $i = 2, \dots, r-p$
  $x_{\cdot,r-p-i} := d_{\cdot,n+1-i}$
for $j = p-1, \dots, 0$
  if defect$(V^j) > 0$
    $Y := (V^j \cap V^{j+1}) \cap X$
    $Z := (V^j \cap V^{j+1}) \setminus X$
    $i := V^j \setminus V^{j+1}$
    $o := V^{j+1} \setminus V^j$
    find $y \in Y$ with defect$(V^j \setminus y) = 0$ and replace y in X by $y \oplus o$.

Lemma 2 *Let $D, S \in \mathbb{Z}_B^{n \times n}$ be linear data distributions, let $p < n$, and let further r be given such that* defect$(\widehat{V}^j) = 0$ *for all $r > j \geq 0$. The algorithm GetX computes $X \in \mathbb{Z}_2^{n \times r-p}$ such that* defect$(V^j) = 0$ *for all $r > j \geq 0$.*

Proof (Sketch)
For the purpose of this proof we employ the set operations $\subseteq$, $\cup$, $\cap$, and $\setminus$ on matrices.

$$d_{n-1},\dots,\overbrace{\underset{o}{d_{n-p+k}},\underbrace{\underbrace{d_{n-p+k-1},\dots,d_0}_{Z},\underbrace{x_{r-p-1},\dots,x_0}_{Y},\underbrace{s_{n-1},\dots,s_{n-p+j+1}}_{Z}}_{V^{j+1}\cap V^j}}^{V^{j+1}},\underset{i}{s_{n-p+j}},\dots,s_{n-p}$$

Figure 9: Usage of the variables in algorithm GetX.

The matrices are interpreted as sets of column vectors. In case of the $\cap$ and $\setminus$ operation, the order of the columns in the resulting matrix is determined by the left argument. The operator $\propto$ is used to indicate that the column vectors of the left argument are linearly dependent on the right argument.

For $j = r-1,\dots,p$ we have $\mathrm{defect}(V^j) = 0$ because of the initialization of X.

We proof by induction: Let $j = 0,\dots,p-1$ and assume $\mathrm{defect}(V^{j+1}) = 0$.

If $\mathrm{defect}(V^j) > 0$ then $Y \cup Z \subseteq V^j$ and $Y \cup Z \subseteq V^{j+1}$. Because of $\mathrm{defect}(V^{j+1}) = 0$, Y is linearly independent of Z and we have $\mathrm{defect}(Y \cup Z) = 0$. Thus, $\mathrm{defect}(V^j) = 1$ and $i \propto (Y \cup Z)$. Moreover, $\widehat{V}^j = Z \cup i$. From that we conclude that $i \not\propto Z$. Writing i as a linear combination of $(Y \cup Z)$, we need at least one vector of Y. There exists a $y \in Y$, such that $(Y \setminus y) \cup Z \cup i = \boxed{V^j \setminus y \text{ is linearly independent}}$ i.e. we can always fulfill line 10 of the algorithm.

Assume the replacement fails and we still have $\mathrm{defect}(V^j) = 1$, i.e. $i \propto Z \cup (Y \setminus y) \cup (y \oplus o)$. Then there exist $a, b \in \mathbb{Z}_2^{p-1}$ and $\alpha, \beta \in \mathbb{Z}_2$:

$$\begin{array}{lrl} i \propto Z \cup (Y \setminus y) \cup (y \oplus o) & \Longrightarrow & i = (Z \cup Y \setminus y)a \quad \oplus (y \oplus o)\alpha \\ i \propto Z \cup Y & \Longrightarrow & i = (Z \cup Y \setminus y)b \quad \oplus y\beta \\ \hline & \ominus & 0 = (Z \cup Y \setminus y)(a \ominus b) \oplus y(\alpha \ominus \beta) \oplus o\alpha \end{array}$$

$Z \cup Y \cup o = V^{j+1}$ is linearly independent, thus $\alpha = \beta = 0$ and $i = (Z \cup Y \setminus y)a$.
Hence $(Y \setminus y) \cup Z \cup i = \boxed{V^j \setminus y \text{ is linearly dependent}}$ — a contradiction.

The linear independence of $V^{j'}$ for $j' > j$ is not affected by the replacement: If $V^{j'}$ does contain y it contains o too. Thus, $V^{j'}$ is linearly equivalent before and after the replacement. The linear independence is untouched. ∎

We define the cost of a redistribution as the sum over all communication steps of the maximum number of data elements sent via an edge.

$$\mathrm{cost}(V) = \sum_{j=0}^{r-1} \max_{e \in E(D_B(n))} |\{i \in \mathbb{Z}_B^n | iV^j = e\}|$$

We can show that the redistribution constructed by the algorithms GetMinR and GetX achieves minimum cost [16].

5.4 Experimental results

The algorithms presented in Section 5.2 and 5.3 allow us to efficiently generate a special data distribution for the optimized parallel evaluation of shift and 90°-rotation invariant SINNs from any given linear distribution of the input image.

For the evaluation of the theoretic results presented here, we have integrated our results into the parallel image processing system PIPS [14]. Figure 10 illustrates the speedup achieved by our implementation on a transputer-based system with up to 64 processors. We compare two implementations: the one titled "double arithmetic" is based on the 64 bit floating point evaluation of the node functions as given in Figure 6 with sigmoidal activation function. The results for an implementation based on bit respectively integer arithmetic and with the hard limiter activation function is titled "bit/integer arithmetic". As the latter node function is faster to evaluate, the time for the redistribution outweighs the computation time and consequently we get a lower speedup.

Summarizing, it can be said that the parallel evaluation of the SINN (without redistribution) achieves almost linear speedup. The time for the data redistribution has to be considered as overhead, since only little or no reordering is required for a sequential implementation.

6 Conclusion

In this paper we present an algorithm for the parallelization of structured invariant neural networks for shift and rotation invariant pattern recognition. We show how the architecture of SINNs can be embedded into a host with de Bruijn communication topology. To achieve an efficient embedding with minimum communication overhead we introduce the class of linear data distribution and find an optimized embedding for SINNs. Further we give algorithms for data redistribution so that the parallel evaluation of the SINN can directly proceed on the output of a large variety of preceeding parallel preprocessing steps.

Our experimental results show a significant speedup of the proposed parallelization of the SINN compared to a sequential computation. The speedup is almost linear for SINNs with sigmoid transfer functions.

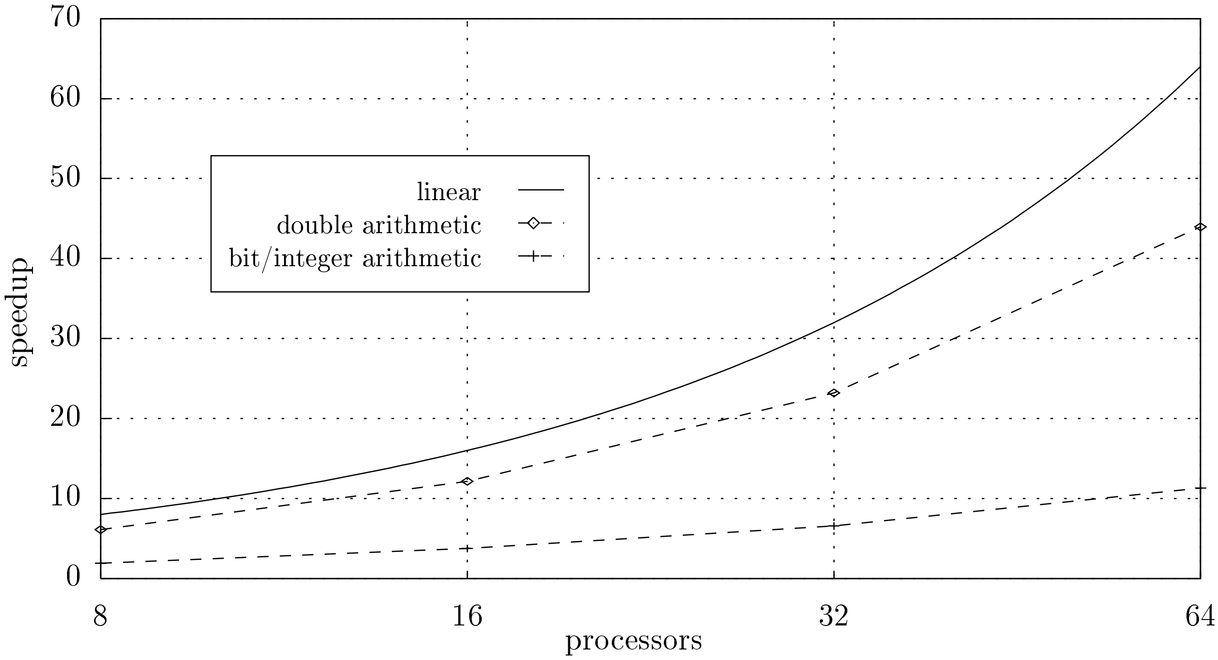


Figure 10: SINN evaluation speedup. Base of calculation is the sequential runtime on one transputer. The parallel time includes the time for redistribution.

7 Acknowledgement

This work has been partially supported by two TMR research grants of the European Commission.

References

[1] N.G. de Bruijn. A Combinatorial Problem. *Koninklijke Nederlandsche Akademie van Wetenschappen Proc.*, 49:758–764, 1946.

[2] Minoru Fukumi, Sigeru Omatu, Fumiaki Takeda, and Toshihisa Kosaka. Rotation-invariant neural pattern recognition system with application to coin recognition. *IEEE Transactions on Neural Networks*, 3(2):272–279, March 1992.

[3] K. Fukushima and N. Wake. Handwritten alphanumeric character recognition by the Neocognitron. *IEEE Transactions on Neural Networks*, 2(3):355–365, May 1991.

[4] High Performance Fortran Forum. High performance fortran language specification. version 2.0. Technical report, Rice University, Houston,Texas, January 1997.

[5] Hu, Shen, and Liang. Optimally routing LC permutations on k-extra-stage cube-type networks. *IEEE Transactions on Computers*, 45:97–103, 1996.

[6] S. Kröner. A structured neural network invariant to cyclic shifts and rotations. In G. Sommer et al., editors, *Proceedings of CAIP'97, 7. Int. Conf. on Comp. Anal. of Images and Patterns*, Lecture Notes in Computer Science No.1296, pp. 384-391, Kiel, Germany, September 1997.

[7] S. Kröner and H. Schulz-Mirbach. Fast adaptive calculation of invariant features. In G. Sagerer, S. Posch, and F. Kummert, editors, *17. DAGM - Symposium "Mustererkennung"*, pages 23–35, Bielefeld, 1995. Reihe Informatik aktuell, Springer.

[8] Sabine Kröner. *Neuronale Netze zur lageinvarianten Mustererkennung.* PhD thesis, Technische Universität Hamburg-Harburg, August 1997. Ersch. als VDI-Fortschr.-Bericht, Reihe 10, Nr. 516, VDI-Verlag, 1997.

[9] Y. LeCun. Generalization and network design strategies. In R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Amsterdam, 1989. Elsevier. Proceedings of the International Conference Connectionism in Perspective, University of Zürich, 10. – 13. October 1988.

[10] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufman Publishers, San Mateo, California, 1992.

[11] Z. Liu. Optimal Routing in the De Bruijn Networks. In *IEEE 10th Int. Conf. on Distributed Computing Systems*, pages 537–544, Paris, 1990. IEEE.

[12] James L. McClelland, David E. Rumelhart, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1987.

[13] M. Nölle. *Konzepte zur Entwicklung paralleler Algorithmen der digitalen Bildverarbeitung.* PhD thesis, Technische Universität Hamburg-Harburg, December 1994. Ersch. als VDI-Fortschr.-Bericht, Reihe 10, Nr. 410, VDI-Verlag, 1996.

[14] M. Nölle, G. Schreiber, and H. Schulz-Mirbach. PIPS–a general purpose Parallel Image Processing System. In W.G. Kropatsch and H. Bischof, editors, *16. DAGM - Symposium "Mustererkennung"*, pages 609–623, Wien, September 1994. Reihe Informatik XPress, TU-Wien.

[15] H. Niederreiter R. Lidl. *Introduction to Finite Fields and their applications.* Cambridge University Press, 1986.

[16] Gerald Schreiber. *Modulare Parallelisierung von Algorithmen der digitalen Bildverarbeitung.* PhD thesis, Technische Universität Hamburg-Harburg, 1997. in preparation.

[17] Bernhard Widrow, Rodney G. Winter, and Robert A. Baxter. Layered neural nets for pattern recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1109–1118, July 1988.