Flexible Platform for Neural Network Based on Data Flow Principles

Liberios Vokorokos, Norbert Ádám, Anton Baláž

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Slovakia
Liberios.Vokorokos@tuke.sk, Norbert.Adam@tuke.sk, Anton.Balaz@tuke.sk

Abstract: Multiprocessor systems are possible to use for parallel computation related to association and classification tasks. For efficient solution above described tasks are used neural networks in which computation is processed in sense of data-driven executing. Similar model is used in data flow computers also. This article describes abilities of transformation neural networks to computation model applied in Neural DF data flow system assigned for solution of classification tasks with neural networks.

Keywords: data flow, neural network, parallelism, Neural DF, operand matching

1 Introduction

Multiprocessor systems are possible to use for parallel computation related to association and classification tasks. For efficient solution above described tasks are used neural networks in which computation is running in sense of data-driven executing. Processor/neuron executes given instruction as soon as there are all operands available. In data flow systems processors for execution single arithmetic, logic and control operations use message sending mechanism, interruption and synchronization signal and for exchanging those information is used shared memory space. Communication/synchronization problems which can be seen in multiprocessor systems during execution parallel tasks come from the physical structure (topology of network) of systems. It is possible to remove this problem by transformation task into Data flow graph. (Data Flow Graph, DFG)[1].

Utilization of implicit paralelism in neural networks, pipelined pattern processing in learning phase of network described in chapters 2 and 3 of this arcticle is part of research of the DF KPI system developed on the Department of computers and informatics, Faculty of Electrical Engineering and Informatics on Technical University of Košice, Slovak Republic supported by grand project: Simulation parallel architectures of computer systems, methods of its specification,

developing technologies and implementation (VEGA 1/1064/04). Next chapters deal with parallelism in computation of neural networks, ability to increase inherent parallelism in networks by implementation pipelined processing of pattern in learning phase, design of pipelined units and conception of direct operands matching in the Neural DF architecture adapted to classification tasks.

2 Computation Parallelism in Neural Networks

In spite of fact that the neural networks dispose with strong inherent parallelism for implementation of artificial neural systems, sequential computers are still used for it. In [2] were designed two restrictions of algorithm formulation for the neural networks with plan of increasing inherent parallelism. These restrictions are:

1 No global „programming language“ control constructs. All components should be self-controlling.

2 No random access memory. Data is either found in the state of the components or in explicit communication between them.

Suitable computer systems which match these restrictions are the computer systems which consist from list of interconnected components with the distribution control and the memory for a data to which data flow computers are assigned [3].

Characteristic property of data flow computer is that DF program instructions wait passive for arrival some combination of the arguments, availability of which is organized as data stream in sense of data-driven. Waiting interval of instruction for approaching operands represent select phase during which the computing resources are allocated. Basis of DF computing model is the mapping tasks (Fig. 1) to processor's elements (CEs – Computing Elements).

In general the task is necessary to divide into smaller communication processes which are represented by the data flow program [4]. DF computing model (DF program) enables to detect parallelism on the lowest level, computer instructions (fine grained parallelism; ILP – Instruction-level Parallel) and provides effective mechanism to utilize inherent parallelism hidden in artificial neural networks. Scheduling of computing is based on availability of the data, similar as it is in neural networks. Data flow computing model or DF program is represented by the DFG in which nodes represent asynchronous active members (operators, instructions, tasks) and edges represent communication paths for transferring and routing messages (data packages, operands), generated by nodes during their activation or received from external enviroment during computation initialization. The carrier information about state of computation is activation token displayed on node's edges of DFG by solid ring.

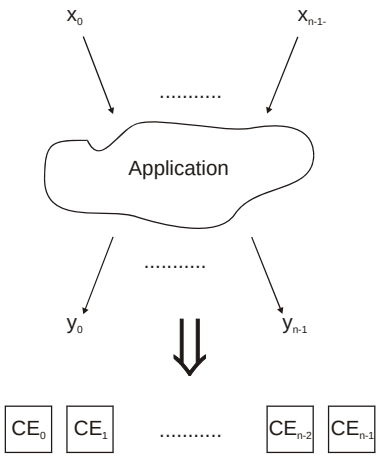

Figure 1
Tasks mapping to CEs

Its position on input node's edges (operators) requires availability of operands and permition to define executive instruction. Placement of activation token on output edge of node requires availability of the result operation defined by operator of executed instruction [1]. As DFG represent universal paradigm for description a flow in given system, formal mapping of neural networks on DFG is possible to use on implementation of neural networks based on various computer architecture. Data flow graph is oriented graph [5]. Its nodes represent instructions and edges interconnect nodes represent operands of these instructions. Instruction can be executed if there are available all input tokens on its input edge of node so if all operands are available.

Generic structure of neural networks is possible to describe with theory of graph. Neuron can be seen as node and synaptic connection among neurons as edges of the graph. States of graph represent information coded into neural network into form of outputs, weights, threshold values. Definitions are expressed in [5] shows that there is strong dependence within DFG and neural networks.

During mapping neural networks into DFG it is necessary to take in reflection following facts. At first we have to map neural network into acyclic DFG.

Every time when new process of learning starts it is necessary to construct new DFG (Fig. 2b). Second, we can use Batch Mode learning for better utilization of inherent parallelism in neural network. In this case pattern enters into network synchrony and at the end of every learning process are the weights updated for next learning. (Fig. 2c)

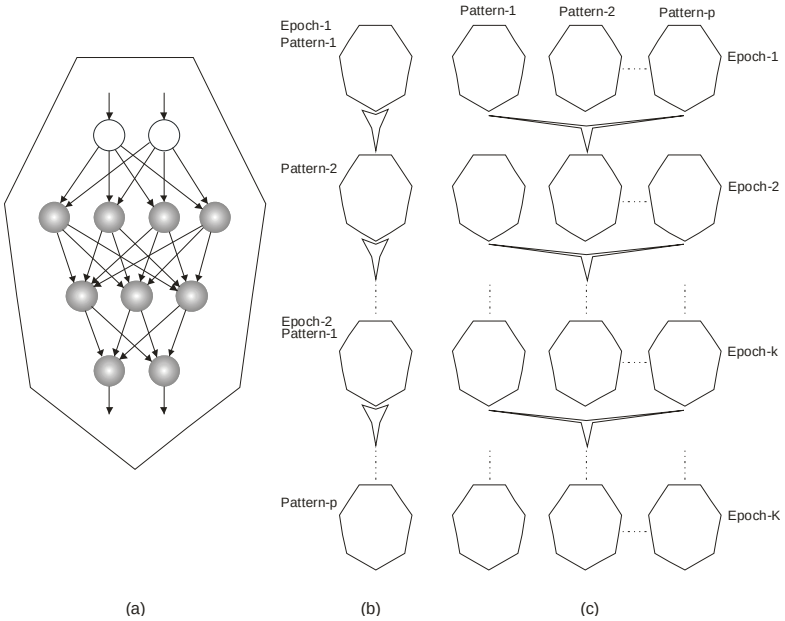

Figure 2
(a) Multi-layer neural network trained in mode (b) Pattern (c) Batch

3 Implicit Parallelism in Neural Networks

The basic equations for error back-propagation for on-line learning, using a logistic activation function and a momentum term, are formulated as shown below:

$$
\begin{aligned}
o_j &= \frac{1}{1+\exp\left(-\sum_k w_{jk} o_k\right)} \\
\delta_j &= \begin{cases} \left(t_j - o_j\right) \\ o_j\left(1-o_j\right)\sum_k\left(\delta_k w_{kj}\left(t\right)\right) \end{cases} \\
w_{ji}\left(t+1\right) &= w_{ji}\left(t\right) + \Delta w_{ji}\left(t+1\right) \\
\Delta w_{ji}\left(t+1\right) &= \eta\left(\delta_j o_j\right) + \alpha \Delta w_{ji}\left(t\right)
\end{aligned}
\qquad (1 - 4)
$$

Where o_j is the output of neuron j;

δ_j is the back propagated error at neuron j;

t_j is the "training value" for output neuron j;

w_{kj} (t) is the synaptic weight from neuron j to neuron k at time t;

η is the learning rate;

α is the momentum term.

Additional, define sequence of values o_j for every neuron in case of watching this neuron. Computing consists of two phases: forward phase, during which are computed values o_j and backward phase when are computed values δ_j. These two phases are expressed implicit, through data dependency. Values o_j depend on values w_{jk}, which depend on values δ_j which depend on values o_j (for all neurons j and k).

Formal networks are usually organized into layers in which in general do not exist synaptic connection among elements in layer. Number of layers is not usually very big, in general 3 or 4, but in some cases as network with time back-propagation, numbers of layers can increase dramatically.

While every layer is dependent on its predecessor layer in the forward phase and its successor layer in the backward phase, all neurons on layer can be processed in parallel and because there does not exist data dependency among neurons on the same layer. The layers of neural network can be processed in a pipelining also.

As soon as results from first layer is accessible in second step of pipelined execution, output of second layer can be computed on the output of second layer and first part of pipelined executing can simultaneously compute output of first layer for next input pattern. Advantage of pipelined execution is based on that pipelining reduces number of messages and increases efficient of the whole network. In the backward phase is possible to compute and modify weights synapse in parallel and pipelined execution can be applied within network layers.

Data dependency resulted from the basic equations obstruct implementation of data pipelining executing in both phases. We have to consider the fact that every calculation of backward phase depends on result of o_j computed in the forward phase.

Application of pipelined executing in the forward phase leads into the lack of o_j values in every calculation of new o_j.

Data dependency results from the basic equations obstruct implementation of pipelined executing in both phases together. In principle, this problem is connected through strong link between backward and forward phase.

It is evident that synchronization between layer j in forward phase and same layer in backward phase can not be simply eliminated. Come out from the fact that error signals are used for deriving new weights and these new weights are used immediately during computing new error signals, it is not necessary that results from backward phase are known before beginning of new forward phase.

Error signal computed in backward phase are used in computation of the output value in backward phase. Output value computed for one input pattern can be used in computation of error signal related with this pattern.

Presented synchronization problem can be overcome by modification of neuron in computation model. Every neuron needs to know its output computed in forward phase for weights modification in backward phase. In the modified model, every neuron is connected to stack for saving computed output values in the forward phase.

After weight calculation of all input synapses, neuron computes its output value based on its activation function. In the backward phase, neuron based on weight sum all input error values computes new synapse weights for its input applying the learning function.

Instead of using straight value o_j, values are saved into stack from which are removed when error function gets particular error value computed in backward phase.

Minimal capacity of neuron stack is designed by distance between given neuron and the output layer of the network. This distance of neuron is called the deep of neuron. The deep d of output neuron is equal to 0. Neurons of input layer have maximal deep d_{max} equal to deep of whole network. For hidden neuron, deep of neuron is equal to deep of post-neuron +1. Predict that neuron has deep d. Its output considering to any input pattern spreads on neurons output in d steps. Backward phase requires next d steps to spread error back on the input. For capacity c of neuron's stack is valid $c = 2d$.

4 Architecture Model of Neural DF

Based on the knowledge introduced in previous chapters within research of distributed / parallel systems [9] for classification tasks on the Department of computers and informatics, Technical University of Košice SR was developed Neural DF architecture. The Neural DF is designed as dynamic system with direct matching of operands [8] and is modified to solve classification tasks with neural network, support pipelined processing of pattern in learning phase and living phase of neural network. System level of architecture includes huge number of operation units which are mapped on single layers of neural network. Combination of local CF model (calculation of activation values is done sequencely) with global DF model enables to organize parallel implementation of neural network effective. Architecture model of Neural DF is part of complex DF system which includes supported components computing environment of DF for realization defined by application learning.

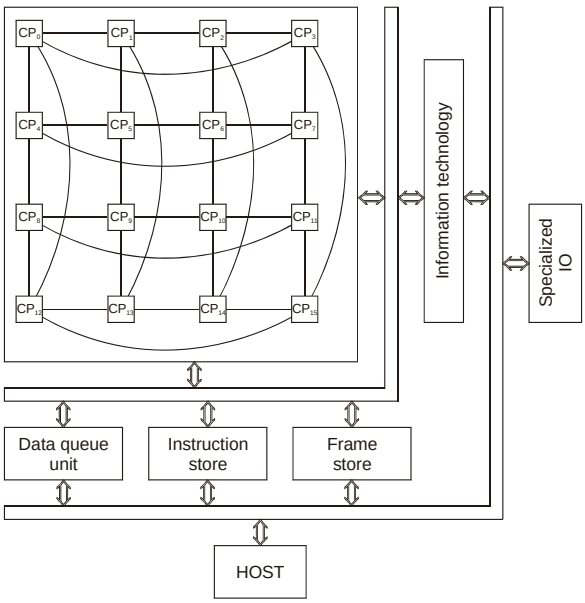

Figure 3
Neural DF architecture

Structure organization of Neural DF architecture model consists from following components (Fig. 3).

- *CP* – Coordinating Processors are assigned to control, coordinating and processing instruction of DF program. Based on presence of the operands
- availability of operands which come on the input port CP.DI from coordinating processor from it's output port CP.DO, from the output ports CP.DO other CP through interconnected network, from data queue DQU and from frame memory FS.
- *DQU* – *D*ata *Q*ueue *U*nit assigned for saving activation tokens (DT – data tokens), represent operands which are waiting for matching during program executing,
- *IS* – *I*nstruction *S*tore, instruction memory (DFI) DF program in form of graph DF (DFG),
- *FS* – *F*rame *S*tore memory of matching vectors by which CP locates presence of the operands for executing operation defined by operator (node) in DFG. Format of matching vector in FS is $\langle FS \rangle ::= \langle AF \rangle \langle V \rangle$, where AF is presence sign of operand (*A*ffiliation *F*lag) and $\langle V \rangle$ is value of this operand (*DT*).

Supported components of DF system are needed for construction real computing environment. In this architecture is formed by:

- *HOST* – main computer for executing standard function of computer system,
- *IT – Information Technology* unit for creating specialized application environment (virtual reality, diagnostic, e-learning),
- *Specialized I/O* – for fast direct input/output into DF module (standard I/O are realized by main computer).

Structure organization of CP is designed as dynamic multi-functional system [7] (Fig. 4) which consists from two pipelined preprocessor units (Fig. 4, Fig. 5), execution unit and communication unit. Preprocessor units have in common separation forward and backward phase of learning network with BP algorithm. Execution unit concerns about processing of given instruction and communication unit takes part in sending result into the communication network.

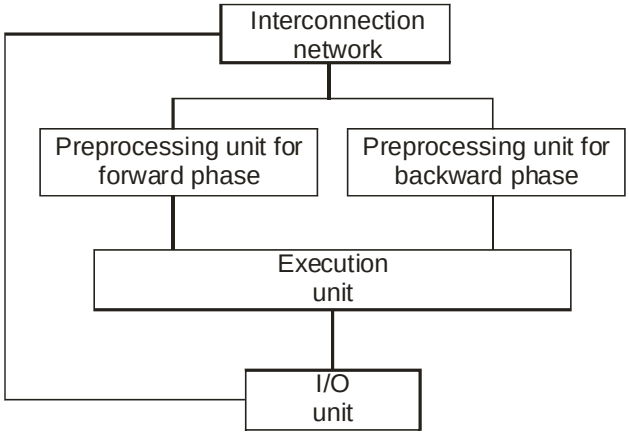

Figure 4
Structure of CP

This model uses mechanism of pipelined processing [6], on edges (represent synaptic connection) DFG is represented as DF program, at same time more than one token (represents weight) can be on edges, this is dynamic data flow computer. One of the problem of this model is efficient operand matching.

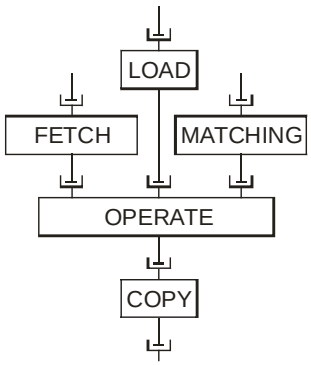

Figure 5
Pipelined levels of preprocessor unit

5 Operands Matching

Program for implementation in data flow environment is compiled into assembler (machine language) which is mostly represented in form of DF graph. The Neural DF architecture support DFG processing. Information carrier about state of computing is activation token represented on edges of nodes DFG in form of solid circle. Its placing on input edges of nodes (operands) express operands presence and define executable instruction, instruction which has tokens on output node's edge and express presence of result operation defined by operator of executed instruction.

Operand has following format:

$$\langle DT \rangle ::= \langle P \rangle \langle D \rangle \langle MVB \rangle \langle DST \rangle \langle [IX] \rangle, \tag{5}$$

where P is operator priority; $\langle D \rangle ::= \langle T, V \rangle$ input data; T data type; V data value; MVB is base address of matching vector MV in frame memory.; DST operand's destination ; IX index of item in matching vector.

DF graph is processed by rules of execution (instruction is executable when all operands are accessible) and data flow program instruction activations (instruction is fireable when resources for its activation are disposal).

The most significant step, coming from dynamic model DF is direct operand matching. Concept of direct operands matching resides in elimination computing expensive process connected with associative searching operands. Operand matching is necessary with operands which enter into double-input and multi-

input operands. In scheme of direct operands matching items in frame memory FS are allocated dynamically for every token generated during DFG processing. Actual position of item in FS is determinated with program compilation, while location of FS is determinated after running DF program. In scheme of direct operands matching every calculation is possible to describe by pointer to instruction IP and with pointer to matching vector in frame memory (MVB). Pair <MVB, IP> is part of activation token header. Typical task is searching operand's pair in frame memory. Searching same labeled tokens is task of matching function. After coordination processor CP gets operand, based on index IX and matching function, system determinate if there is partner of operand in frame memory. If there is not searched partner, operand is saved into matching vector determinated by base address of operand MVB into item by index IX.

Operands matching process [8] is affected on its input and output by execution process and by result generation. Program compiler to DFG and forward phase enables to detect and eliminate redundant computation, changing order of token processing, control is possible to define as transaction of activation tokens on DFG edges (Fig. 6) between operator „generating" (P- producer) and operand „consumer" (K- consumer) token.

Producer produces token which are received be u-single input or v-double input. In the first case there does not come to activation matching function in second case but yes. Process of operands matching is controlled by coordination processor CP.

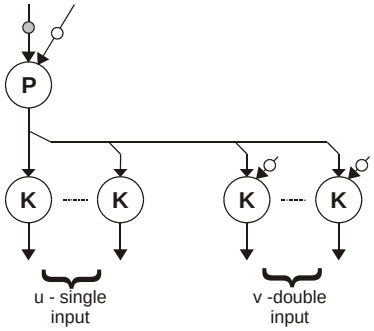

Figure 6
Operands matching process

Coordination processor represents dynamic pipelined system which enables to go through states (L – load, M – matching, C – copy, F – fetch, O - operate) pipelined unit in different order. Transition between states during control are described on Fig. 7, where are control signals also, CP_free – indicates busy or availability of CP, Get DQ – reading token from DQU, Put DQ – record token into DQU; Init – initialization of pipelined stages.

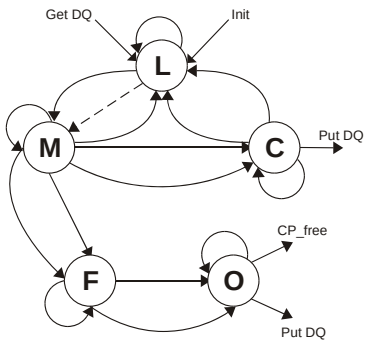

Figure 7
Control diagram of matching operands

In parallel implementation of microprogram for matching operands between single stages of CP are inserted interstages memories with following specification:

- Between stage L a M $\rightarrow$ LMP
- Between stage M a F $\rightarrow$ MFP
- Between stage F a O $\rightarrow$ FOP

If partner's operand is available (affiliation flag AF = 1) is its position recorded in MV and this determines its place in FS, both are loaded and processed in elementary unit (PEU). Result of operation (token) is accessible for processing in next operator, if CP is not busy with other computation. If CP is busy than operator is sent through interconnected network to another available CP. If there is not available CP, token is saved in DQU. State OPERATE can be divided into several smaller steps. To do this CP uses state COPY.

Conclusions

Base model of artificial neural network has several features which require massive parallel implementation. These features include high parallel operation, simple execution unit (neuron), local memory for neuron with small capacity (shared memory) and error tolerance of connected neurons. The target architecture for artificial neural network implementation can be high parallel computer systems with simple execution unit. However there exists strong analogy between neural networks and Data Flow graphs (mainly control of computing in sense data-driven) this architecture represents suitable platform for implementation of neural networks. Intention of releasing and formulation synchronization problem in learning process of neural network was classic model of neuron extended by the stack which enables to minimize data dependencies between forward and backward phase. This model enables to establish pipelined data processing with utilization of draft-grained parallelism resulting from multi-layer forward network architecture. Pipelined processing in the Neural DF architecture developed on

Department of computers and informatics is attractive alternative to fine-grained parallelism every time when number of layers in the network is approximately the same as number of processors (in cases of small parallel systems) and any time when number of layers is significant.

Supported by VEGA project No. 1/1064/04

References

[1] Jelšina M. et al: Architecture of Data Flow KPI (in Slovak). Elfa s.r.o., 2004 Košice. ISBN 80-89066-86-0

[2] Moore, R.; Klauer, B.; Waldschmidt, K.: What computer architecture can learn from computational intelligence—and vice versa. In 23rd Euromicro Conference, Budapest, Hungary, Sept. 1997

[3] Vokorokos, L.: Data Flow computer principles (in Slovak). Copycenter, s.r.o., Košice, 2002, ISBN 80-7099-824-5

[4] Heath, J. R.: „Development, Analysis, and Verifiation of a Parallel Hybrid Dataflow Computer Architectural Framework and Associated Load-Balancing Strategies and Algorithms via Parallel Simulation“ *ProQuest Science Journals*, Jul 1997; 69, 1, pp. 7

[5] Yuceturk, A. C.; Klauer, B.; Zickenheiner, S.; Moore, R.; Waldshmidt K.: Mapping of Neural Networks onto Data Flow Graphs. Proceedings of the 22nd EUROMICRO Conference, 1996, ISSN 089-6503/96

[6] Jelšina Milan, Šuba Stanislav, Dzuriak Miloš: Parallel Control Unit of the Data Flow Pipeline Processors, Proceedings of the sixth international scientific conference Electronic Computers and Informatics ECI 2004, Košice-Herľany, September 22-24, 2004, Košice, VIENALA Press, 2004, 55, pp. 332-337, ISBN 80-8073-150-0

[7] Jelšina Milan, Ádám Norbert: Multipipelined and Multithreaded Architectures Approache – Over View of Data Flow Architectures, Proc. of the 6th international scientific conference – ECI´2004, Košice-Herľany, September 22-24, Košice, VIENALA Press, 2004, pp. 241-246, ISBN 80-8073-150-0

[8] Vokorokos, L., Ádám, N., Petrík, S.: Operators Matching In Dynamic Data Flow Architectures. In: 2th International Conference on Computational Cybernetics (ICCC) 2004, August 30 – September 1, 2004, Vienna, Austria, pp. 77-81, ISBN 3-902463-02-03

[9] Vokorokos, L., Petrík, S.: Proposed Parallel and Distributed Architectures for Behavioral Animation. In: 9th IEEE International Conference on Intelligent Engineering Systems (INES) 2005, Cruising on Mediterranean Sea, September 16-19, 2005, pp. 199-203, ISBN 0-7803-9474-7