PARNEU: Scalable Multiprocessor System for Soft Computing Applications

PASI KOLINUMMI, TIMO HÄMÄLÄINEN AND JUKKA SAARINEN
Digital and Computer Systems Laboratory
Tampere University of Technology
P.O. Box 553, FIN-33101 Tampere, FINLAND

Abstract:- Many soft computing applications have inherent parallelism in the form of synapses and neurons. Exploitation of parallelism will bring computers more close to their biological counterparts as well as improve the performance. Modularity and good communication capabilities are essential for practical implementations. Our multiprocessor system called PARNEU is designed according to general requirements found in artificial neural networks. PARNEU is a parallel co-processor system, which includes a bus, ring and reconfigurable partial tree communication networks. Programming is done using C-language primitives, which hide the communication topology. PARNEU is directly connected to the host computer, which includes a TCP/IP server for remote programming over Internet connection. Scalability of communication topology and artificial neural network applications, like Multilayer Perceptron (MLP), Self-Organizing Map (SOM) and Sparse Distributed Memory (SDM) are shown. Parallel efficiency with eight boards configuration is generally more than 75%.

Keywords:- Neurocomputer, scalable parallel implementation, partial tree shape architecture, artificial neural networks, soft computing, hardware implementation.

1 Introduction

Soft computing applications like artificial neural networks (ANN) and genetic algorithms require much computation especially in the learning phase, where the number of iterations is usually very high. In the production phase, the computational requirements are much smaller, which makes the algorithms attractive in embedded and portable systems. The applications being for example speech recognition, pattern recognition and different classification tasks.

The soft computing ideas, inspired by biological findings are usually inherently parallel like their biological counterparts. A massive number of synapses and neurons work simultaneously and affect to each others via the communication mechanism. This parallelism can not be utilized in sequential processing on PCs or workstations.

The complexity of parallel system design and usage can be alleviated by a modular and scalable structure consisting of small testable units. In general, replication seems to be a reasonable choice to use the increased amount of computational resources and transistors in the future system-on-chip implementations.

Our PARNEU (Partial tree shape neurocomputer) system was implemented to speed up algorithm development in research environment where the algorithms continuously change. Thus it was important that the system is flexible and allows many parallel mapping possibilities. PARNEU system is a continuation of successful TUTNC (Tampere University of Technology Neural Computer) [1] computer, which has a complete tree topology where processing units are at the outer leaves of a binary tree. In contrary to TUTNC, PARNEU has more flexible communication topology consisting of bus, ring and tree networks, which allows practical expandability without any needs to redesign any part of the system.

Other examples of similar systems based on digital signal processors (DSPs) are RAP [2], MUSIC [3], RENNS [4], and Manitoba's reconfigurable computer [5]. RAP has a ring topology consisting of 10 boards at maximum, each of which have four Texas TMS320C30 DSPs. MUSIC has been implemented with Motorola 96002 DSPs and Inmos T805 transputers and it also utilizes a ring topology. RENNS uses TMS32025 DSPs in each system module while Manitoba's system use Motorola 96002 DSPs and bus topology.

REMAP [6] is a SIMD-type linear vector with bit serial computing elements using only Xilinx Field Programmable Gate Arrays (FPGAs). Commercial neurocomputer systems like CNAPS [7] and Synapse [8] use application specific integrated circuits (ASICs) as processing units. A more thorough review of neurocomputers are found in [9,10,11].

In this paper, we present PARNEU topology and show practical expandability with results obtained from several ANN applications. In addition to soft computing, PARNEU is currently used in digital video and image processing which is another field of computationally intensive data processing.

In the next section, we briefly review the general requirements of ANN algorithms implemented in PARNEU system. In Section 3, the hardware and software implementation of PARNEU is presented. Performance analysis is given in Section 4. Section 5 concludes the paper.

2 Requirements of ANN algorithms

An artificial neuron, which is a basic block in most of the ANN algorithms, requires numerous multiplication and addition operations as well as a computation of an activation function. A support for fast addition and multiplication operations are therefore the most important common requirements for different ANNs. Multilayer Perceptron networks (MLP) [12] use generally sigmoidal or hyperbolic tangent as an activation function, which requires exponent and division operations. Sometimes the computation of the activation function is improved by using look-up tables.

The main computational operations in Self-Organizing Map (SOM) [13] and other winner-take-all type networks are comparison operation, which is used to find the winner neuron, as well as multiplication, substruction and addition, which are used in distance computation and weight update. The operations in distance computation depend on the used distance metrics. The squared Euclidean norm requires only subtraction, multiplication and addition. The dot-product metric requires normalization, but after that the weight matrix can be multiplied with the input vector as in many other neural networks.

The computational needs for the basic Sparse Distributed Memory (SDM) [14] are Hamming distance computation and bit manipulations. The operations are simple and fast, but an enormous number of them are executed when mimicking biological systems. The improved reading method requires more computation and higher precision even though 8 bits is enough for the counter matrix [15]. In contrast to other ANNs, SDM requires a large memory area to store the counter matrix.

The communication requirements of each ANN depend on the mapping style. In *fine-grain* implementations and *on-line learning* [17] where weight values are updated continuously after each input pattern, the communication is much more frequent than in *large-grain* and *batch mode learning* [17] where weights are updated only after a complete learning set or a subset of it. The batch mode allows pipelined operations and block data transfers, for which reason it is used in many hardware implementations to achieve good performance values. However, the required algorithm modifications slow down the convergence and more iterations are needed in the application level [17].

Especially in the on-line learning, a fast broadcast and global reduction operations like global addition or comparison are needed. A fast broadcast operation is achieved for example with a bus or tree topology [16]. Bus topology does not allow effective data gathering, which limits its usability as an only communication media. Instead, the tree topology gives logarithmic gathering time as a function of the number of processing units (PUs) [16], which is important to achieve practical scalability. A local communication between the PUs can be best arranged in a mesh or ring topology, where point-to-point connections are available.

A tree topology alone would be a good solution, but it is hard to scale without external cabling. Bi-directional communication is also difficult without separate communication channels. A broadcast bus combined with an uni-directional partial tree network is a good compromise between easy scalability, modular implementation, and good performance. This topology allows pipelined uni-directional data flow, which decreases the effect of initial latency time. A local communication, required in some ANNs, can be best fulfilled with ring or mesh type communication topology.

3 PARNEU System

PARNEU is a modular and scalable multiprocessor system. It consists of identical, modular processing cards connected to a master board and further to a host computer, as depicted in Fig. 1. Currently, the host computer is a PC with WindowsNT operating system and it is used for initialisation and monitoring.

The *master processing unit* (MPU) controls the operations inside the PARNEU system as well as executes some sequential parts of an application. Master board consists of internal bus that connects DRAM memory banks, MPU and communication interfaces together.

Processing boards also have a local bus connecting four *processing units* (PUs) and a global bus interface together. Each PU has a 256 kilobytes internal SRAM memory. Program initialization, data transfer and synchronization are done with the communication network. The PUs can also reconfigure the partial tree FPGAs.

An effective communication network is the most important part of PARNEU system. In addition to scalability and modularity requirements, it should support different communication methods and thus different type of algorithm mappings. Buffered and physically similar, synchronous processing board interfaces with board wide clock signals allows physical scalability. Modular boards cause only a fixed extra cost per additional board and thus cost scalability is achieved. Performance scalability is analysed in Section 4.

According to previous analysis, data broadcast to the PUs, direct data exchange between PUs, and global reduction operations for PUs data are required. In PARNEU, a *Global bus* (GB) serves the broadcast requirement and has the highest data transfer bandwidth. The global bus is formed by buffered, point-to-point segment between each board [18]. The pipelining through First-In First-Out (FIFO) memories increase communication delay, but allows unlimited

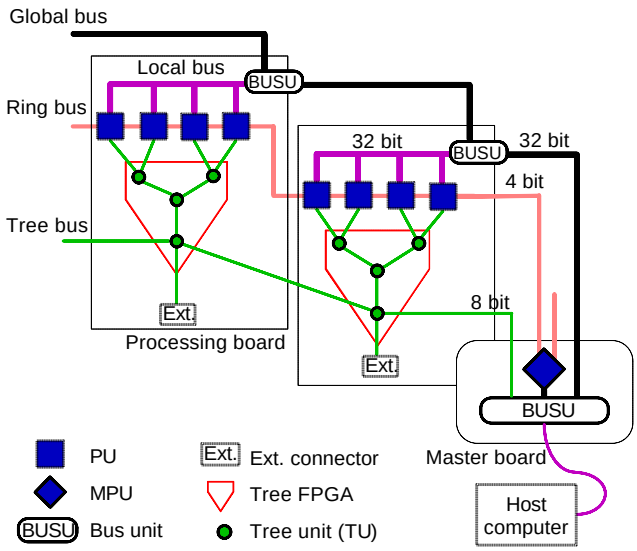

Fig. 1. PARNEU system architecture.

Fig. 2. PARNEU hardware implementation.

physical scalability without clock skew or other timing problems. Buffering is done in Bus Units (BUSU) as depicted in Fig. 1.

A reconfigurable *partial tree* is formed from a complete subtrees located in each processing board. Subtrees are connected together to form the overall partial tree structure that ends up at the master board. Data can be transferred from the PUs to the MPU in a pipelined manner and the partial tree can perform comparison, addition or other simple global reduction operations. With different FPGA configuration, data can also be transferred from an external connector to each set of four PUs.

To support local communication, two of the link ports of each PU are connected to the previous PU and two link ports to the next PU to form a bi-directional *ring* network. Local data transfer and systolic operations as well as synchronization of the PUs are the main operations. Synchronization between the MPU and the PUs are done with two physical wired-OR/AND signals which inform the MPU whether none, some or all the PUs have finished the current operation.

A good example of effectiveness and capability of data pipelining is a circle that starts from a data broadcast via the global bus to all the PUs. The PUs perform a local computation according to their own program. After that PUs send the data to the partial tree network, which does a global reduction operation such as addition or comparison. Finally the MPU receives the result via the tree network. Thus, the global bus, the partial tree bus, the PUs and the MPU operate simultaneously during the whole operation. Data transfer latencies are hidden and PU idle times are reduced.

All the communication paths are implemented using point-to-point connections and synchronous data transfer between boards, which guarantees constant timing characteristics and physical scalability.

PARNEU hardware is implemented with Analog Device ADSP-21062 DSPs [19] working as processing units (PUs) and Xilinx Field Programmable Gate Arrays that perform the communication operations. All boards are mounted in a standard rack, as shown in Fig. 2. The modular structure was chosen for convenient replacement of possible faulty board and practical scaling of the computational power. The backplane is truly passive containing only connectors and routing.

3.1 Programming Environment

Programming environment is one of the most important part in application development, especially in research, where different algorithms and parallel mapping methods are tested and analysed. The programming environment should hide the communication network and allow simple and fast primitives for programmers. Currently, a support for high level languages like C-language is almost mandatory. Our work with programming environment has concentrated on easy usability and fast C-language primitives that keep the software overhead for basic data transfers very low.

A simplified diagram of PARNEU software interfaces is illustrated in Fig. 3. PARNEU is connected to the host computer via PCI card using WindowsNT device driver. PARNEU system can be used directly from the host computer or via a special TCP/IP server. With TCP/IP server, PARNEU can be used from an internet connected PC in a similar way as in the host computer. At the remote computer, a TCP/IP client is running to give handles to PARNEU applications.

The host and client computers have a special graphical user interface (GUI) for system initialization, debugging, monitoring, and execution of simple user programs. The debugger GUI helps in program

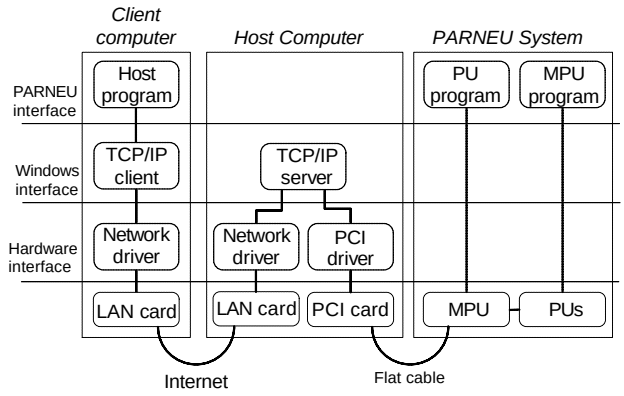

Fig. 3. PARNEU software interfaces.

development and fault diagnosis. In multiprocessor systems, the debugging interface is more important than in sequential systems because of synchronization and locking difficulties. PARNEU has exception messages of the low level errors to inform users of the failed operation. Many times the reason for the exception is caused much earlier than the exception arrives. To find the actual stage of program execution it is important that the host computer can access all PU's memory and register space directly even though they are not physically connected to the same bus.

PARNEU allows MIMD (multiple-instruction, multiple-data) type operations, but for scalability and easy programming the operations are usually limited to master-slave configuration, in which the MPU controls the overall operation and the PUs execute the parallel program. Each PU has its own memory space and thus the PUs can execute the same code but they neither need to be synchronized in each operation nor depend on each other's memory contents. This, restricted MIMD structure, is called SPMD (Same Program, Multiple Data) model [20]. For a programmer this means that separate programs are needed, one for the MPU and one for the PUs. A programmer has possibility to use only a specific PU for a certain task by using conditional branches with a PU identification number. Each PU has a differentiable order number for this purpose. However, this is an exceptional situation and should be avoided, since it may decrease parallel speedup.

Application programs are written completely in C-language and a programmer can choose the most reasonable communication primitive from a C-library. Communication primitives work similarly even though the number of processors changes.

Communication is performed using a message-passing interface (MPI) that gives users a consistent programming platform. In addition to basic MPI primitives, more complicated operations such as global reduction operation is supported. This is very effective in the reconfigurable hardware because the primitives hide the way operations are performed. Operations can be done either in the hardware or software level, but the programmer has always the same functions even though the hardware configuration changes.

The parallel algorithm mapping to the PARNEU architecture is done manually. Compilation and linking are done using ADSP-2106x development tools and the development library primitives. PC programs are written in C++-language under the Windows NT operating system. The programmer can also write an application specific GUI that uses the host computer library functions. The application program downloads the program files and initiate PARNEU as well as configure the FPGAs by calling the library primitives. The ready-made FPGA configuration files are stored in the disk of the host computer from which the programmer can select the most suitable configuration for each application.

4 Performance Measurements

The most important hardware performance metrics are plain processor performance, communication throughput and data transfer latencies. The application performance depends on the parallel mapping strategy and algorithm properties, which makes comparison between different systems quite difficult.

To achieve reliable and predictable external signal timings, all communication buses operate synchronously. Flip-flops cause one clock cycle delay in both input and output pads but allows zero wait state implementation for 40 MHz system clock. All DSPs work with the same clock frequency.

Latency time in each node of the global bus is three clock cycles and the achieved throughput is 160 Mbyte/s. A cluster of four PUs need three clock cycles to receive one 32 bit wide data word. Latency time in the ring bus is about 13 clock cycles and the throughput is limited to 20 Mbyte/s because of nibble wide communication links and 40 MHz system clock. The tree network also uses DSPs' link ports but additional delays are caused by the active tree nodes that combine two data values. In four PUs configuration, the latency time from the PUs to the MPU is about 50 clock cycles. Operation pipelining gives a sustained throughput of 20 Mbyte/s.

One of the critical operations to achieve good performance in ANN is a global reduction operations. In small systems, a common bus can work reasonable well, but as the number of processors is increased the communication times increase remarkable as illustrated in Fig. 4. *GB sync all* means an operation using the global bus where synchronization is done between each operation i.e. the order of data is important. *GB sync add* operation is synchronized only after each addition operation. However, both methods scales much worse than similar operations using either the

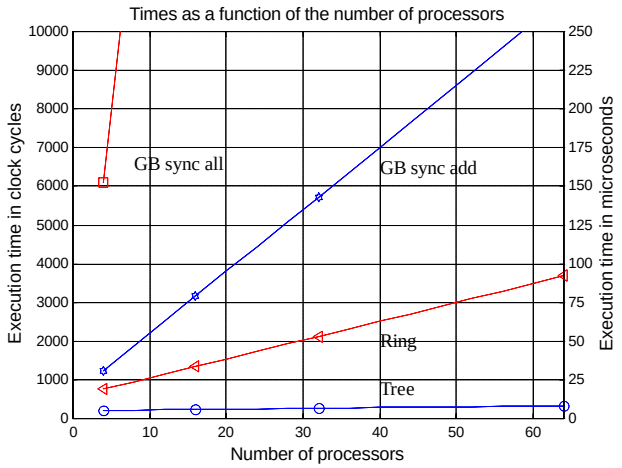

Fig. 4. Comparison of the performance of 20 pipelined global addition operations using PARNEU communication methods.

ring or partial tree networks. In practical implementation, the partial tree network does not limit the scalability.

Next we analyse the performance of ANN applications, like MLP, SOM and SDM. Performance is given either in wall clock time, in million connections per second (MCPS) or in million connection updates per second (MCUPS). All implementations use on-line learning and execute the same functionality as the sequential implementations.

MLP implementation is based on *continuous weight update* and it uses back-propagation (BP) learning with momentum [12]. Both *neuron* and *weight parallel mappings* can be effectively used in PARNEU architecture, but a mixed neuron and weight parallel mapping minimizes communication without increasing memory consumption or computation time [21].

Table 1 illustrates performance in the forward pass (production) of the MLP network for a problem with 203 inputs, 80 hidden neurons and 26 output neurons. The measurement is done with one processing board configuration (4 PUs). The estimated speedup, based on exact equations found out by the actual measurements, for 8 board configuration in the learning phase is about 6.2, which gives efficiency of 78%. The number of neurons in the MLP network affects the performance values. For a good parallel performance, at least one neuron is mapped per PU. However, this fine-grain parallelism does not give the best performance due to intensive communication. However, more significant improvements would be achieved by optimizing the program code in an assembly level, but the purpose is to show the practical performance values achieved with normal C-language.

In SOM, the weight and neuron parallel mappings are both suitable for an on-line execution. Measurements are done with the neuron parallel mapping using Euclidean distance metric and squared neighborhood area. The presented mapping only needs broadcast communication and global comparison operation, which are both well supported in PARNEU. Global comparison is done in the partial tree network that is reconfigured to transmit simultaneously the coordinates of the winner neuron.

Table 1: Production and learning performance of MLP, SOM and SDM applications. Estimated speedup values are given for 8 board configuration.

	Production read	Learning write	Speedup, 8 boards
MLP	5.5 MCPS	1.5 MCUPS	6.2
SOM	30.1 MCPS	25.7 MCUPS	5.9
SDM	51 ms	12.4 ms	6.4

Table 1 gives the performance of SOM with 32 inputs and 32x32 two-dimensional output plane. The neighbourhood radius is 4. The performance of the learning phase is computed using all the output neurons even though all the neurons are not updated in each cycle. The neighbourhood radius has a large effect on the learning performance, because the number of updated weights increase as a function of neighbourhood radius. Computations are performed in 32 bit precision and measurements are done with a configuration of four PUs. Program codes are mainly written in C-language, but the local winner search in each PU is optimized using an assembly level subroutine. The speedup value for 8 board configuration in the learning phase is 5.9, while the speedup in the read operation is about 7.5. The extremely good speedup value in read operation is based on the partial tree network that is used to find out the best matching unit.

A neuron and weight parallel mappings can also be applied to the SDM model, where a row-wise mapping corresponds to the neuron parallel mapping and a column-wise mapping corresponds to the weight parallel mapping. In the column-wise mapping, the columns of the counter matrix are divided into PUs. The writing operation is fast in both mappings even though the row-wise mapping benefits from parallel search of activated locations. However, the column-wise mapping requires significantly less communication in the reading operation and is thus selected for implementation.

The measurements shown in Table 1 are done for column-wise mapped SDM with a mask based activation mechanism and with an improved reading method [15]. The number address bits is 256 and the mask size is five. The contents matrix used in the measurement is 4096 (number of memory locations) times 512 (data width). The speedup values for 8 board configuration in both reading and writing oper-

ations are about 6.4. The program is written in C-language. The number of data bits, the number of memory locations and the number of mask bits to select an active position mostly affect the total execution time.

5 Conclusions

The presented PARNEU architecture combines the most useful communication topologies: the tree is optimal for global reduction operations and the bus can be used for broadcast. Local communication is supported by the ring network. However, the reconfigurable and scalable implementation of the active partial tree is the most significant topological contribution. The flexibility of the architecture makes it a very good platform for parallel system research and allows effective implementations for different algorithms and different mappings. Scalability and modularity are achieved by placing the functional units into processing boards, which can be connected together and increased in number. A separate master board makes the control path shorter and faster than would be achieved with external controllers.

In the future, the application development will continue. In addition to soft computing algorithms, a parallel H.263 low bit-rate video encoder was implemented. New DCT-based motion estimation methods and wavelet based image processing algorithms have also been implemented, and will be further developed. Parallel compilers and run-time performance monitors will also be studied.

In the hardware development, the next phase is to apply the architectural ideas to System-On-Chip design. The current board level implementation gives a good reference for a more integrated chip scale systems.

References:

[1] T. Hämäläinen, J. Saarinen and K. Kaski, “TUTNC: A General Purpose Parallel Computer for Neural Network Computations”, *Microprocessors and Microsystems*, Vol. 19, No. 8, 1995, pp. 447-465.

[2] N. Morgan, J. Beck, J. Kohn, J. Bilmes, E. Allman and J. Beer, “The Ring Array Processor: A Multiprocessing Peripheral for Connectionist Applications”, *Journal of Parallel and Distributed Computing*, Vol. 14, No. 3, March 1992, pp. 248-259.

[3] U. Müller, A. Gunzinger and W. Guggenbühl, “Fast Neural Net Simulation with a DSP Processor Array”, *IEEE Transactions on Neural Networks*, Vol. 6, No. 1, January 1995, pp. 203-213.

[4] J. G. Solheim, G. Myklebust, “RENNS - A Reconfigurable Computer System for Artificial Neural Networks”, Proceedings of the First International Conference on Algorithms and Applications of Parallel Processing, ICA3PP’95, Brisbane, Australia, 1995.

[5] H. Card, G. Rosendahl, D. McNeill, and R. McLeod, “Competitive Learning Algorithms and Neurocomputer Architecture”, *IEEE Transactions on Computer*, Vol. 47, No. 8, August 1998, pp. 847-858.

[6] T. Nordström and B. Svensson, “Designing and Using Massively Parallel Computers for Artificial Neural Networks”, *Journal of Parallel and Distributed Computing*, Vol. 14, No. 3, March 1992, pp. 260-285.

[7] D. Hammerström, “A Highly Parallel Digital Architecture for Neural Network Emulation”, *VLSI for Artificial Intelligence and Neural Networks*, J. G. Delgado-Frias and W. R. Moore (Eds.), Plenum Publishing Company, New York, USA, 1990.

[8] U. Ramacher, “SYNAPSE - A Neurocomputer that Synthesizes Neural Algorithms on a Parallel Systolic Engine”, *Journal of Parallel and Distributed Computing*, Vol. 14, No. 3, March 1992, pp. 306-318.

[9] P. Ienne, G. Kuhn, “Digital Systems for Neural Networks”, In P. Papamichalis, R. Kerwin, (*eds.*), Digital Signal Processing Technology, Vol. CR57 of Critical Reviews Series, SPIE Optical Engineering, USA, 1995, pp. 314-345.

[10] P. Kolinummi, T. Hämäläinen, and K. Kaski, “Designing a Digital Neurocomputer”, *IEEE Circuits and Devices Magazine*, March 1997, pp. 19-27.

[11] T. Nordström, “Highly Parallel Computers for Artificial Neural Networks”, Ph.D. Thesis, Division of Computer Science and Engineering, Luleå University of Technology, Sweden, 1995.

[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Company, NY, USA, 1994.

[13] T. Kohonen, The Self-Organizing Maps, Springer-Verlag, Berlin, 1995.

[14] P. Kanerva, *Sparse Distributed Memory*, The MIT Press, 1988.

[15] G. Sjödin, “Getting More Information Out of SDM”, International Conference on Artificial Neural Networks (ICANN'96), Bochum, Germany, July 1996, pp. 477-482.

[16] D. P. Bertsekas, J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall International Inc., USA, 1989.

[17] N. Sundarajan and P. Saratchandran (*Eds.*), *Parallel Architectures for Artificial Neural Networks: Paradigms and Implementation*, IEEE Computer society Press, USA, 1998.

[18] P. Kolinummi, T. Hämäläinen, and J. Saarinen, “Chained Backplane Communication Architecture for Scalable Multiprocessor Systems”, to be published in *Journal of Systems Architecture*.

[19] ADSP-2106x SHARC User’s Manual, second edition, Analog Devices Inc., USA, July 1996.

[20] D. E. Culler, J. P. Singh, A. Gupta, *Parallel Computer Architecture, A Hardware/Software Approach*, Morgan Kaufman Publishers, Inc., CA, USA, 1999.

[21] P. Kolinummi, P. Hämäläinen, T. Hämäläinen and J. Saarinen, “PARNEU: General-Purpose Partial Tree Computer”, to be published in *Microprocessors and Microsystems*, Vol. 24, No. 1, March 2000, pp. 23-42.