

THE CONNEX ARRAY™ AS A NEURAL NETWORK ACCELERATOR

Răzvan Andonie
Computer Science Department
Central Washington University
Ellensburg, WA, USA
email: andonie@cwu.edu

Mihaela Malița,
Computer Science Department
Saint Anselm College
Manchester, NH, USA
email: mmalita@anselm.edu

ABSTRACT

We discuss the parallel implementation of neural networks on a Connex Array™ circuit. By estimating the number of memory cycles, we approximate the real performance of the machine. We show how to implement a basic neural learning algorithm. Our preliminary study suggests that the Connex Array™ is a good neural network accelerator, due to its high performance for vector operations. The execution time obtained is significant: 5 GCUPS, at 2.5 watts, on less than 1 cm^2 area.

KEY WORDS

Hardware neuro-computing, parallel computing, parallel architecture.

1 Introduction

Since the very beginning of the neural network era, there has been the belief that to fully exploit the potential of this paradigm it would be necessary to also develop efficient hardware implementations [1]. Hardware neural network components are available in a number of different forms. The main types are:

- Neurocomputers, which provide a complete system based on neural techniques.
- PC accelerators and other cards, which are generally made for a common bus standard such as ISA (for PCs).
- Chips, which can be used to build either of the preceding forms or can be included along with other devices to make a complete application unit.
- Cell libraries, which are now becoming available to allow an appropriate level of neural functionality to be included within a dedicated chip alongside other necessary functions.
- Embedded microcomputers, which can be thought of as general purpose computers implementing a software neural network on dedicated hardware.
- Accelerator cards which contain general-purpose programmable processors. Their increased performance

is gained by speeding up the repetitive multiply-and-add steps, which are required in software simulations of parallel operations.

An overview of the commercial hardware available in 2004 for neuro-computing can be found in [2]. There is no consensus on how to implement artificial neural networks on parallel machines. During the last years, researchers have been trying to achieve maximal performance on their favorite (or available) parallel machine. Most frequently, they have customized the parallel algorithm to fit the hardware architecture. Customization means also less generality and more effort in implementing different neural models. Our goal is to focus on direct (brute-force) parallel implementation. In this case, only vector related operations are parallelized. There are two reasons why we proceed like this. First, such implementations are easy to obtain by non-specialists in parallel programming. Second, in neural computation, vector operations are the computational bottleneck.

In our paper, we will refer to implementations on an accelerator card which is a general purpose programmable scalable digital VLSI architecture: the Connex Array™ (CA). The CA is a new type of Single Instruction Multiple Data (SIMD) parallel in-memory device with thousands of cells that permits fast general purpose computations [3]. It contains standard RAM circuitry at the higher level of the hierarchy, and a specialized memory circuit at the lower level, the Connex Memory, that permits parallel search at the memory-cell level and shift operations. A controller oversees the exchange of data between the two levels. Just as regular memory circuits, the operations supported by the CA can be performed in well-defined cycles whose duration is controlled by the current memory technology, which in today's technology is in the 5 ns range. Several computational intensive applications have been developed on this machine: data compression [5], alignment of DNA sequences [4], DNA search [7], massive computation of polynomials [8], and real-time packet filtering for detection of illegal activities [6].

The CA can perform vector operations in a small number of cycles. Addition, subtraction, multiplication, search can be performed in one cycle. The resulted execution times are very competitive. For instance, the present CA implementation can perform 7 million scalar products of 1024-tupled vectors per second. Since neural network

computations are mostly vector-matrix, it is interesting to investigate the performance of CA based neural computation.

We aim to evaluate the performance of the CA as a neural network accelerator. We analyze the execution time (in terms of CPU cycles) of the most characteristic neural network operations, implemented on a CA. This is the first neuro-computing approach on the CA architecture. The implementations are executed on a simulation of the CA. However, the CA hardware is already available on the market.

Section 2 is an overview of the CA hardware architecture. In Section 3, we analyze the performance of vector operations on the CA. The description we provide here is not available elsewhere. Section 4 provides the CA-implementation of a basic neural learning algorithm, with the aim to illustrate how easy it can be obtained. In Section 5, we compare our results to implementations on similar parallel architectures.

2 The Connex Array™ Circuit

The CA is a parallel programmable VLSI chip. Physically, it consists of an array of processors. Functionally, it is an array/vector processor. It is not a dedicated, custom-designed (ASIC) chip, but a general purpose architecture. It is a fast, very cheap device, with low power consumption. For a programmer, the CA is a SIMD mesh architecture.

The first CA chip with 1024 cells (CA1024, see Fig. 1) was implemented in 2006 by Connex Technology¹, a startup in Silicon Valley, now called BrightScale. The CA can be conceptualized by the programmer as 256 vectors of 1024 words that can be manipulated as single entities, and engaged in simple vector manipulations such as basic vector arithmetic, or searching for the occurrences of a pattern in a vector.

The Connex design [3, 12, 13] is an integral parallel architecture [9], centered on the CA, a one-dimension array of simple processing elements and on a Speculative Accelerator (SA), an array of eight 16-bit simple processors. CA is a data-parallel engine operating on 1024-component vectors, and SA is a speculative pipeline stream-machine. The CA is packaged for scalability, with a chip design allowing several CA chips to coexist on the same board and interface with each other to extend the length of the vectors, while receiving instructions and data from only one controller.

The 1024 cells are individually addressable as in a regular RAM, but can also receive broadcast/instructions or data on which they operate in parallel at a peak rate of 1 operation per cycle. This general concept fits the Processor-In-Memory paradigm. The cells are connected by a linear chain network, allowing fast shifting of data between the cells, as well as the insertion or deletion of data from cells while maintaining the relative order of all the data. All these operations are performed in one memory cycle only.

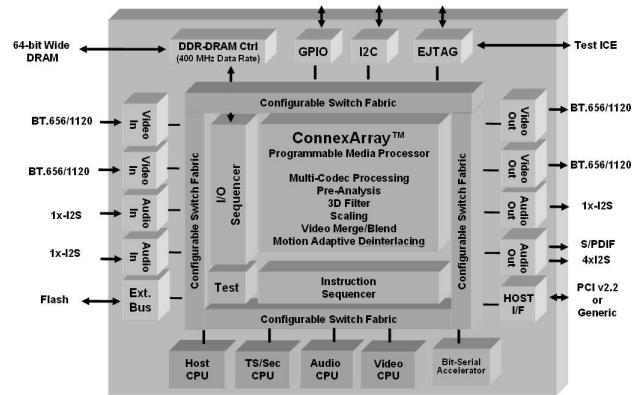


Figure 1. The CA1024 chip [12, 13].

Using a 16-bit arithmetic, the performances of the CA1024 chip are:

- memory cycle: 5 ns
- scalar product for 1024-toupled vectors: 140 ns (7.1 million scalar products/sec)
- matrix multiplication for 1024×1024 matrices: 150 ms (6.7 operations/sec)

Adding up to 1024 numbers is done in 18 cycles. This is a very special feature of this machine. Multiplication is done in 10 cycles. The power consumption for this chip is 2.4 watts and the area of the Connex Array is under 1 cm^2 . Compared with a common PC computer (100 W) we can say that power is 40 times less for the CA chip.

3 Basic Vector Operations on the CA1024

By nature, the neural network paradigm is a massively parallel computational model. The main computational kernel is scalar vector product, both during learning and recognition. So it is crucial that the scalar vector product has an excellent computation time.

Let N be the number of components of a vector of real numbers each represented on 16-bits. On the CA1024 chip, the maximum value for N is 1024. The Connex architecture is scalable and this limit can be easily extended with increments of 1024. In order to simplify notations, we will assume in the following that $N \leq 1024$.

After being stored in the 1024 cells, the components of a vector

$$\mathbf{x} = [x(1), x(2), \dots, x(N)]^t,$$

can be selectively activated to perform any operation. The major vector operations can be computed in parallel within very few memory cycles:

1. Add a scalar to a vector (one cycle):

$$\mathbf{x} = a + \mathbf{x}$$

¹<http://www.connextechnology.com>

Scalar a is added in parallel to the array cells which contain \mathbf{x} .

2. Add two vectors (one cycle):

$$\mathbf{x} = \mathbf{y} + \mathbf{z}$$

3. Sum of all elements of a vector can be performed in 18 cycles.

4. Multiplication of a vector x by a scalar a (10 cycles):

$$\mathbf{x} = a * \mathbf{x}$$

5. Multiply two vectors component-by-component (10 cycles):

$$\mathbf{x} = \mathbf{y} * \mathbf{z}$$

6. Scalar vector product (28 cycles):

$$\mathbf{x} = \mathbf{y}^t \cdot \mathbf{z}$$

The products $y(i) * z(i)$, $i = 1, \dots, N$, are computed in parallel in 10 cycles. The sum of these products is computed in 18 cycles.

Observation: These performances hold for any value of N , as long as $N \leq 1024$, and are determined by the number of bits used for scalar data representation (our chip uses 16 bits precision).

4 A Simple Neural Implementation

As a first example, we will implement the perceptron learning rule for a single discrete neuron [10, 11] on the CA, and analyze its performance. The parallel design is immediate, since only the vector operations are parallelized. The algorithm is described by the following steps:

We have the P training pairs (\mathbf{y}_p, d_p) , $p = 1, \dots, P$, where

$$\mathbf{y}_p = [y_p(1), y_p(2), \dots, y_p(N-1), y_p(N) = 1]^t$$

is an N -dimensional input vector and $d_p \in \{-1, 1\}$ is the corresponding target value.

1. Initialize \mathbf{w} , the vector of weights, with small random values.

Initialize counters: $k = 1, p = 1$.

Initialize output error: $E = 0$.

2. $out = \text{sgn}(\mathbf{w}^t \cdot \mathbf{y}_p)$.

Time: The scalar vector product $\mathbf{w}^t \cdot \mathbf{y}_p$ is evaluated in parallel in 28 cycles.

3. $\mathbf{w} = \mathbf{w} + c/2 * (d_p - out) * \mathbf{y}_p$, where c is the learning constant.

Time: The sub-expression $a = c/2 * (d_p - out)$ is computed sequentially in 4 cycles, $\mathbf{y}_p = a * \mathbf{y}_p$ is computed in parallel in 10 cycles, $\mathbf{w} = \mathbf{w} + \mathbf{y}_p$ is computed in parallel in one cycle. The total number of cycles is $4+10+1 = 15$.

$$4. E = E + 1/2 * (d_p - out)^2$$

5. **If** $p < P$ **then** $p = p + 1, k = k + 1$, **go to** Step 2.

6. **If** $E > 0$ **then** $E = 0, p = 1$, **go to** Step 2;

else we found the final \mathbf{w} and k .

This algorithm can be also extended for multiple one-layer perceptrons and is convergent if and only if the input vectors are linearly separable, according to the well-known Perceptron Convergence Theorem [10, 11]. Matrix-vector products are the major bottleneck of this neural learning procedure.

In steps 2 and 3, we use parallelism to compute the vector operations. On the CA, one iteration of the algorithm does not depend on the size of the vectors, since vector operations are executed in constant time. In one iteration, the parallel vector operations in steps 2 and 3 are performed in $(15 + 28) * 5 = 215$ ns ($1 \text{ ns} = 10^{-9}$ seconds). This results in $1024 * 10^9 / 215 \cong 5$ Giga Connection Updates per Second (GCUPS). Execution time for storing an input vector in the 1024 cells can be omitted since it is performed in parallel with processing the previous vector. We also omit the overhead from sequential operations and steps 4-6. The estimated execution time for this algorithm is in the order of 5 GCUPS, independent of the value of N , $N \leq 1024$. By adding new CA chips, we can increase the maximum value of N .

To implement the algorithm on the Connex architecture we can use the CPL language (a C-like parallel language) designed for the CA. In CPL, the perceptron training algorithm for a very small example can be coded as follows:

```

int P = 3;
int N = 2;
int p;           // counter
short E = 0;    // output error
short a, s;     // temporary scalars
int out;        // current output value
int i;          // counter

int k = 1;
// total number of executed iterations

short vector <0> y[P];
// vector of P N-toupled input vectors

int vector <0> d[P];
// vector of target values

short vector <0> w[N];
// vector of weights

short vector <0> z[N];
// temporary vector

```

```

d = {1,1,1};
y[1] = {0, 2, 4, 6, 8, 1};
y[2] = {0, 2, 4, 6, 8, 1};
y[3] = {0, 2, 4, 6, 8, 1};
w = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6};

public void main ()
{
  while (E > 0)
  {
    p = 1; E = 0;
    for(i = 0; p < P; i++, k++)
    {
      z = w * y[i];
      // vector component-by-component
      // multiplication

      s = sum(z);
      // sum is a library function
      // executed in 18 cycles

      if (s > 0)
        out = 1;
      else
        out = -1;

      a = c/2 * (d[i]-out);
      y[i] = a * y[i];
      w = w + y[i]; // vector addition
      E = E + (d[i]-out)*(d[i]-out)/2;
    }
  }
} // end ; output w and k

```

The final value of k is the total number of iterations performed. The execution time of the whole algorithm depends on the final k , but this does not affect the estimate of the number of weight updates per second.

5 Conclusion

We have investigated the suitability of a commercially available chip (CA) for neural computations. The CA is a circuit with great potential, mainly because of its high level of parallelism, its scalability, its programmability, and its low power dissipation. Our preliminary study suggests that it is a good neural network accelerator, reaching maximum performance without specialization.

It is difficult to compare the CA to other commercial devices because of several reasons. Sometimes, it is even hard to obtain information from the manufacturers [2]. Meanwhile, there are many criteria to consider in such comparisons. We will only mention some of the most competitive dedicated neurochips at the 2004 year level [2] and compare them to the CA, which is a general purpose architecture. We have selected only digital chips which can implement neural learning because the CUPS we use mea-

sures only the learning phase of a neural model.

*Inova N64000*²: This programable chip contains 64 processing elements (PEs) and a speed of 220 Mega CUPS (MCUPS). It is easy to combine multiple chips.

*HNC 100-NAP*³: The Hecht-Nielsen Computers 100 programable Neurocomputer Array Processor is a 32-bit floating point processor with a maximum number of 4 PEs and a speed of 64 MCUPS.

Hitachi WSI: It has a built in learning model (Hopfield or Backpropagation) and a speed of 300 MCUPS.

The fastest of these chips, Hitachi WSI, is about 17 times slower than the CA! A very important CA feature is also its cost, which makes it largely affordable.

Our example was implemented directly, without any other customization, which can certainly increase performance. The obtained execution acceleration is due exclusively to the hardware, not to the algorithm design. In the future, it would be interesting to implement on the CA a standard benchmark, like the one in [16]. The parallel design will certainly perform better than the direct implementation, where only the vector operations are parallelized. Our research is an ongoing project and we are experimenting with more complex neural models.

During the last years, the hardware market for neural-computing had a slow development because of several reasons [2]:

1. Building dedicated neurochips is expensive.
2. The variety of neural network paradigms is still increasing rapidly.
3. There is no clear consensus on how to exploit the current available hardware implementations.

We believe that this situation will change in the near future with the appearance of new hardware solutions based on general purpose multi-processors, like the CA. During a recent talk, Dave Patterson made the following statement [17]:

The sequential processor era is now officially over, as the IT industry has bet its future on multiple processors per chip. The new trend is doubling the number of cores per chip every two years instead the regular doubling of uniprocessor performance.

After several decades of experimenting with neurochips, our conjecture is that only general purpose, cheap and easy to program architectures, like the CA, have chances to be commercially successful for computational intelligence applications. For the first time, parallel computers are available (and affordable) as personal computers. For this reason, parallel programming should be made easy.

²Which seems to be no longer available [2].

³No longer available [2].

References

- [1] B. Linares-Barranco, A. G. Andreou, G. Indiveri, T. Shibata, Guest editorial - special issue on neural networks hardware implementations, *IEEE Trans. on Neural Networks*, 14(5), 2003, 976–977.
- [2] F. M. Dias, A. Antunes, A. M. Mota, Artificial neural networks: a review of commercial hardware, *Engineering Applications of Artificial Intelligence*, 17, 2004, 945–952.
- [3] M. Thiebaut, G. Ştefan, Memory engine for the inspection and manipulation of data, *U.S. Patent No. 6,760,821*, July 2004.
- [4] D. Thiebaut, G. Ştefan, Local alignment of DNA sequences with the Connex Engine, *The First Workshop on Algorithms in BioInformatics WABI 2001*, BRICS Univ. of Aarhus, Denmark, August 2001.
- [5] D. Thiebaut, G. Ştefan, Ziv-Lempel compression with the Connex Engine, *Tech. Rep. 077*, Dept. Computer Science, Smith College, Northampton, MA, 01063, January 2002.
- [6] D. Thiebaut, M. Maliţa, Real-time packet filtering with the Connex Array, *International Conference on Complex Systems*, Boston, MA, June 25-30, 2006.
- [7] D. Thiebaut, G. Ştefan, M. Maliţa, DNA search and the Connex technology, *International Multi-Conference on Computing in the Global Information Technology (ICCGI'06)*, August 1-3, Bucharest, Romania, 2006.
- [8] D. Thiebaut, M. Maliţa, Fast polynomial computation on Connex Array, *Technical Report 303*, Smith College, November 2006.
- [9] G. Ştefan, M. Maliţa, M. Stoian, Complex vs. intensive in parallel computation, *International Multi-Conference on Computing in the Global Information Technology (ICCGI'06)*, August 1-3, Bucharest, Romania, 2006.
- [10] J. Zurada, *Introduction to artificial neural systems*, West Publishing Company, St. Paul, 1992.
- [11] R. Andonie, A. Caţaron, *Computational intelligence (in Romanian)*, Transylvania University Press, Brasov, Romania, 2002.
- [12] G. Ştefan, The CA1024: SoC with integral parallel architecture for HDTV processing, *4th International System-on-Chip (SoC) Conference & Exhibit*, November 1-2, Radisson Hotel Newport Beach, California, 2006.
- [13] G. Ştefan, A. Sheel, B. Mitu, T. Thomson, D. Tomescu, The CA1024: a fully programmable system-on-chip for cost-effective HDTV media processing, *Hot Chips: A Symposium on High Performance Chips*, Memorial Auditorium, Stanford University, August 20-22, 2006.
- [14] J. Liu, D. Liang, A survey of FPGA-based hardware implementation of ANNs, *Proceedings of the International Conference on Neural Networks and Brain (ICNN&B '05)*, 2005, 915–918.
- [15] F. P. Farber, K. Asanovic, Parallel neural network training on Multi-Spert, *Proceedings of the IEEE Third International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'97)*, Melbourne, Australia, December 1997.
- [16] R. Andonie, A. T. Chronopoulos, D. Grosu, H. Gálmeanu, Efficient concurrent implementation of a neural network algorithm, *Concurrency and Computation: Practice and Experience*, 18, John Wiley & Sons, 2006, 1559-1573.
- [17] D. Patterson, Computer architecture is back. The Berkeley view of the parallel computing research landscape. *Stanford EE Computer Systems Colloquium*, January 31, 2007.