# Part IV

# IMPLEMENTATIONS

Implementation of neural networks can be divided into three categories:

- software simulation;
- (hardware) emulation[2];
- hardware implementation.

The distinction between the former two categories is not clear-cut. We will use the term *simulation* to describe software packages which can run on a variety of host machines (e.g., PYGMALION, the Rochester Connectionist Simulator, NeuralWare, Nestor, etc.). Implementation of neural networks on general-purpose multi-processor machines such as the Connection Machine, the Warp, transputers, etc., will be referred to as *emulation*. *Hardware implementation* will be reserved for neuro-chips and the like which are specifically designed to run neural networks.

To evaluate and provide a taxonomy of the neural network simulators and emulators discussed, we will use the descriptors of table 9.1 (cf. (DARPA, 1988)).

1. **Equation type**: many networks are defined by the type of equation describing their operation. For example, Grossberg's ART (cf. section 6.4) is described by the differential equation

$$\frac{dx_k}{dt} = -Ax_k + (B - x_k)I_k - x_k \sum_{j \neq k} I_j, \tag{9.12}$$

   in which $-Ax_k$ is a decay term, $+BI_k$ is an external input, $-x_kI_k$ is a normalisation term, and $-x_k \sum_{j \neq k} I_j$ is a neighbour shut-off term for competition. Although differential equations are very powerful, they require a high degree of flexibility in the software and hardware and are thus difficult to implement on special-purpose machines. Other types of equations are, e.g., difference equations as used in the description of Kohonen's topological maps (see section 6.2), and optimisation equations as used in back-propagation networks.
2. **Connection topology**: the design of most general purpose computers includes random access memory (RAM) such that each memory position can be accessed with uniform speed. Such designs always present a trade-off between size of memory and speed of access. The topology of neural networks can be matched in a hardware design with fast local interconnections instead of global access. Most networks are more or less local in their interconnections, and a global RAM is unnecessary.
3. **Processing schema**: although most artificial neural networks use a synchronous update, i.e., the output of the network depends on the previous state of the network, asynchronous update, in which components or blocks of components can be updated one by one, can be implemented much more efficiently. Also, continuous update is a possibility encountered in some implementations.
4. **Synaptic transmission mode**: most artificial neural networks have a transmission mode based on the neuronal activation values multiplied by synaptic weights. In these models, the propagation time from one neuron to another is neglected. On the other hand, biological neurons output a series of pulses in which the frequency determines the neuron output, such that propagation times are an essential part of the model. Currently, models arise which make use of temporal synaptic transmission (Murray, 1989; Tomlinson & Walker, 1990).

Table 9.1: A possible taxonomy.

The following chapters describe general-purpose hardware which can be used for neural network applications, and neuro-chips and other dedicated hardware.

[2]The term *emulation* (see, e.g., (Mallach, 1975) for a good introduction) in computer design means running one computer to execute instructions specific to another computer. It is often used to provide the user with a machine which is seemingly compatible with earlier models.

# 10 General Purpose Hardware

Parallel computers (Almasi & Gottlieb, 1989) can be divided into several categories. One important aspect is the *granularity* of the parallelism. Broadly speaking, the granularity ranges from *coarse-grain* parallelism, typically up to ten processors, to *fine-grain* parallelism, up to thousands or millions of processors.

Both fine-grain and coarse-grain parallelism is in use for emulation of neural networks. The former model, in which one or more processors can be used for each neuron, corresponds with table 9.1's type 2, whereas the second corresponds with type 1. We will discuss one model of both types of architectures: the (extremely) fine-grain Connection Machine and coarse-grain Systolic arrays, viz. the Warp computer. A more complete discussion should also include transputers which are very popular nowadays due to their very high performance/price ratio (Group, 1987; Board, 1989; Eckmiller, Hartmann, & Hauske, 1990). In this case, descriptor 1 of table 9.1 is most applicable.

Besides the granularity, the computers can be categorised by their operation. The most widely used categorisation is by Flynn (Flynn, 1972) (see table 10.1). It distinguishes two types of parallel computers: SIMD (Single Instruction, Multiple Data) and MIMD (Multiple Instruction, Multiple Data). The former type consists of a number of processors which execute the same instructions but on different data, whereas the latter has a separate program for each processor. Fine-grain computers are usually SIMD, while coarse grain computers tend to be MIMD (also in correspondence with table 9.1, entries 1 and 2).

| | | Number of Data Streams | |
|---|---|---|---|
| | | single | multiple |
| Number of Instruction Streams | single | SISD (von Neumann) | SIMD (vector, array) |
| | multiple | MISD (pipeline?) | MIMD (multiple micros) |

Table 10.1: Flynn's classification.

Table 10.2 shows a comparison of several types of hardware for neural network simulation. The speed entry, measured in interconnects per second, is an important measure which is often used to compare neural network simulators. It measures the number of multiply-and-add operations that can be performed per second. However, the comparison is not 100% honest: it does not always include the time needed to fetch the data on which the operations are to be performed, and may also ignore other functions required by some algorithms such as the computation of a sigmoid function. Also, the speed is of course dependent of the algorithm used.

| | HARDWARE | WORD LENGTH | STORAGE (K Intcnts) | SPEED (K Int/s) | COST (K$) | SPEED / COST |
|---|---|---|---|---|---|---|
| WORKSTATIONS | | | | | | |
| Micro/Mini | PC/AT | 16 | 100 | 25 | 5 | 5.0 |
| Computers | Sun 3 | 32 | 250 | 250 | 20 | 12.5 |
| | VAX | 32 | 100 | 100 | 300 | 0.33 |
| | Symbolics | 32 | 32,000 | 35 | 100 | 0.35 |
| Attached | ANZA | 8–32 | 500 | 45 | 10 | 4.5 |
| Processors | $\Delta - 1$ | 32 | 1,000 | 10,000 | 15 | 667 |
| | Transputer | 16 | 2,000 | 3,000 | 4 | 750 |
| Bus-oriented | Mark III, IV | 16 | 1,000 | 500 | 75 | 6.7 |
| | MX/1–16 | 16 | 50,000 | 120,000 | 300 | 400 |
| MASSIVELY | CM–2 (64K) | 32 | 64,000 | 13,000 | 2,000 | 6.5 |
| PARALLEL | Warp (10) | 32 | 320 | 17,000 | 300 | 56.7 |
| | Warp (20) | | | 32,000 | | |
| | Butterfly (64) | 32 | 60,000 | 8,000 | 500 | 16 |
| SUPER-COMPUTERS | Cray XMP | 64 | 2,000 | 50,000 | 4,000 | 12.5 |

Table 10.2: Hardware machines for neural network simulation.
The authors are well aware that the mentioned computer architectures are archaic... current computer architectures are several orders of magnitute faster. For instance, current day Sun Sparc machines (e.g., an Ultra at 200 MHz) benchmark at almost 300,000 dhrystones per second, whereas the archaic Sun 3 benchmarks at about 3,800. Prices of both machines (then vs. now) are approximately the same. Go figure! Nevertheless, the table gives an insight of the performance of different types of architectures.

## 10.1 The Connection Machine

### 10.1.1 Architecture

One of the most outstanding fine-grain SIMD parallel machines is Daniel Hillis' Connection Machine (Hillis, 1985; Corporation, 1987), originally developed at MIT and later built at Thinking Machines Corporation. The original model, the CM–1, consists of 64K (65,536) one-bit processors, divided up into four units of 16K processors each. The units are connected via a cross-bar switch (the *nexus*) to up to four front-end computers (see figure 10.1). The large number of extremely simple processors make the machine a *data parallel* computer, and can be best envisaged as active memory.

Each processor chip contains 16 processors, a control unit, and a router. It is connected to a memory chip which contains 4K bits of memory per processor. Each processor consists of a one-bit ALU with three inputs and two outputs, and a set of registers. The control unit decodes incoming instructions broadcast by the front-end computers (which can be DEX VAXes or Symbolics Lisp machines). At any time, a processor may be either listening to the incoming instruction or not.

The router implements the communication algorithm: each router is connected to its nearest neighbours via a two-dimensional grid (the *NEWS* grid) for fast neighbour communication; also, the chips are connected via a Boolean 12-cube, i.e., chips $i$ and $j$ are connected if and only if $|i - j| = 2^k$ for some integer $k$. Thus at most 12 hops are needed to deliver a message. So there are 4,096 routers connected by 24,576 bidirectional wires.

By slicing the memory of a processor, the CM can also implement virtual processors.

The CM–2 differs from the CM–1 in that it has 64K bits instead of 4K bits memory per processor, and an improved I/O system.

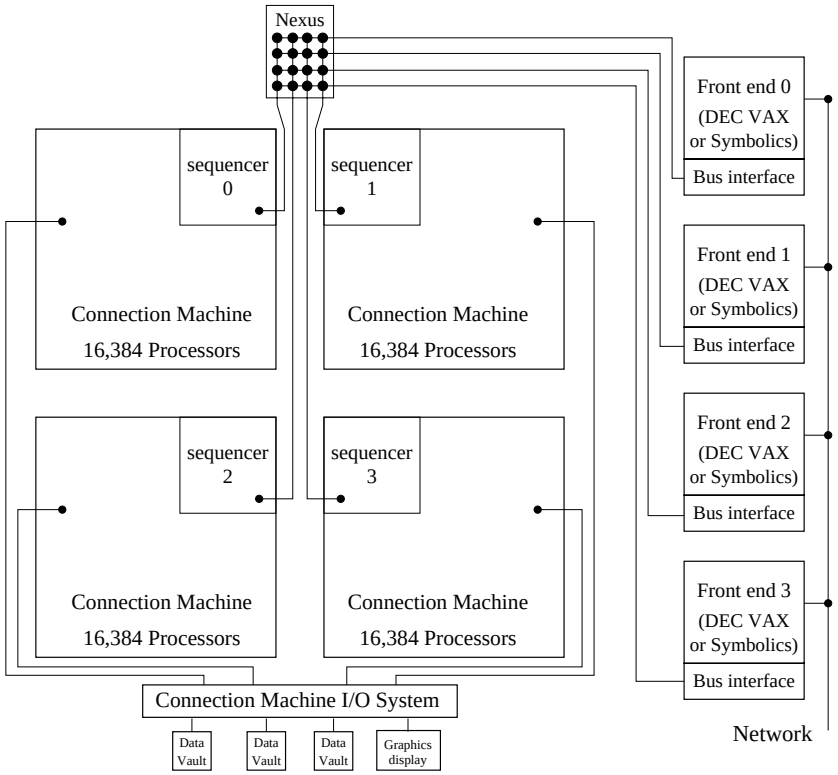


Figure 10.1: The Connection Machine system organisation.

### 10.1.2 Applicability to neural networks

There have been a few researchers trying to implement neural networks on the Connection Machine (Blelloch & Rosenberg, 1987; Singer, 1990). Even though the Connection Machine has a topology which matches the topology of most artificial neural networks very well, the *relatively* slow message passing system makes the machine not very useful as a general-purpose neural network simulator. It appears that the Connection Machine suffers from a dramatic decrease in throughput due to communication delays (Hummel, 1990). Furthermore, the cost/speed ratio (see table 10.2) is very bad compared to, e.g., a transputer board. As an effect, the Connection Machine is not widely used for neural network simulation.

One possible implementation is given in (Blelloch & Rosenberg, 1987). Here, a back-propagation network is implemented by allocating one processor per unit and one per outgoing weight and one per incoming weight. The processors are thus arranged that each processor for a unit is immediately followed by the processors for its outgoing weights and preceded by those for its incoming weights. The feed-forward step is performed by first clamping input units and next executing a *copy-scan* operation by moving those activation values to the next $k$ processors (the outgoing weight processors). The weights then multiply themselves with the activation values and perform a *send* operation in which the resulting values are sent to the processors allocated for incoming weights. A *plus-scan* then sums these values to the next layer of units in the network. The feedback step is executed similarly. Both the feed-forward and feedback steps can be interleaved and pipelined such that no layer is ever idle. For example, for the feed-forward step, a new pattern $\mathbf{x}^p$ is clamped on the input layer while the next layer is computing on $\mathbf{x}^{p-1}$, etc.

To prevent inefficient use of processors, one weight could also be represented by one processor.

## 10.2 Systolic arrays

Systolic arrays (Kung & Leierson, 1979) take the advantage of laying out algorithms in two dimensions. The design favours compute-bound as opposed to I/O-bound operations. The name *systolic* is derived from the analogy of pumping blood through a heart and feeding data through a systolic array.

A typical use is depicted in figure 10.2. Here, two band matrices $A$ and $B$ are multiplied and added to $C$, resulting in an output $C + AB$. Essential in the design is the reuse of data elements, instead of referencing the memory each time the element is needed.

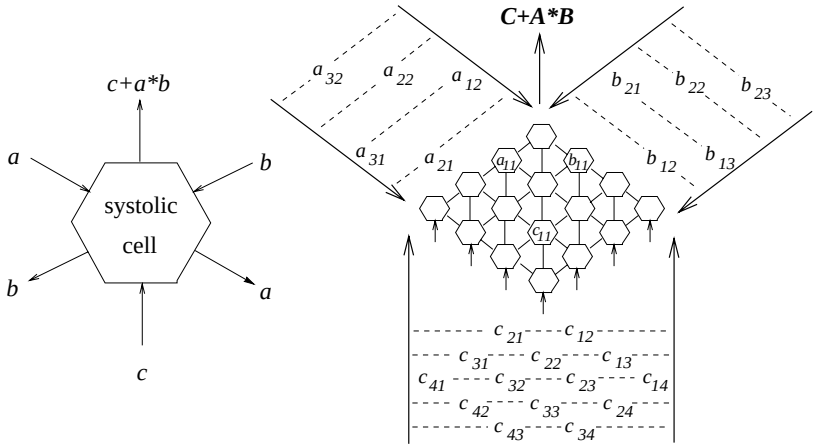


Figure 10.2: Typical use of a systolic array.

The Warp computer, developed at Carnegie Mellon University, has been used for simulating artificial neural networks (Pomerleau, Gusciora, Touretzky, & Kung, 1988) (see table 10.2). It is a system with ten or more programmable one-dimensional systolic arrays. Two data streams, one of which is bi-directional, flow through the processors (see figure 10.3). To implement a matrix product $W\mathbf{x} + \boldsymbol{\theta}$, the $W$ is not a stream as in figure 10.2 but stored in the memory of the processors.

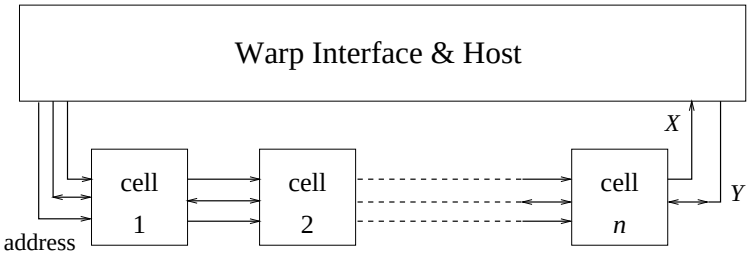


Figure 10.3: The Warp system architecture.