NEURAL COMPUTING ON A ONE DIMENSIONAL SIMD ARRAY

Stephen S. Wilson
Applied Intelligent Systems
110 Parkland Plaza,
Ann Arbor, MI 48103

ABSTRACT

Parallel processors offer a very attractive mechanism for the implementation of large neural networks. Problems in the usage of parallel processing in neural computing involve the difficulty of handling the large amount of global communication between processing units and the storage of weights for each of the neural processor connections. This paper will discuss how massive parallelism in the form of a one dimensional SIMD array can handle indefinitely large networks in near real time by efficiently organizing the memory storage of weights, and input and output signals. Very little overhead time is used in communication of signals between processing units, and there is no idle time for any of the units. An advantage of SIMD array systems is that the arithmetic processing is done bit serially, with the result that trade-offs can be easily be made between the processor speed and precision of the signals and weights.

1 Introduction

Fine grained massive parallelism is an ideal method to enhance the speed of neural processing for real time applications. In these systems, which are also called SIMD (Single Instruction, Multiple Data) architectures there are many processing elements each having a very simple bit serial arithmetic unit with local memory. In a given clock cycle, all units receive the same instruction, and all local memories receive the same address. The SIMD architecture works well in neural computing because one set of instructions corresponding to the operation of a neural cell is applied to a large data set corresponding to the various signals and neural weights in the network. The most common examples of SIMD architectures are mesh connected systems, such as the Massively Parallel Processor [Potter, 1985], with inter-processor communication in two dimensions, or a hypercube system such as the Connection Machine [Hillis, 1985] with processing units at the vertices of an N-dimensional hypercube and inter-processor communication along the hypercube edges. However they are generally too expensive to be considered as a vehicle for dedicated applications. A SIMD architecture which is low cost and commercially available, is a one-dimensional processor array system, the AIS-5000, which has up to 1024 processing elements and uses the PIXIE chip, [Schmitt and Wilson, 1988, Wilson, 1985]. The system has a wide base of real-time industrial applications in vision processing. A next generation one dimensional array system under development uses the Centipede chip [Wilson, 1988]. In the following sections, the one dimensional architecture will be described in more detail, and a discussion of the memory organization of signals and weights will illustrate how this architecture can efficiently handle neural computing. Programming a SIMD machine is similar in many ways to programming any serial RISC computer. A serial program is written for a simple processor which accesses and operates on data in its memory. The major difference is that when an instruction is executed, a large number of processing units are active. Data dependent operations cannot be handled in the same way as for a serial processor, and algorithms must be specifically organized to operate correctly for an ensemble of processing units with multiple data paths.

2 One-dimensional SIMD Architectures

The basic model of the one dimensional SIMD architecture is shown in Fig. 1 where the individual processing elements (PEs) are connected to their nearest east and west neighbors. In addition, each PE is connected to its own bit-wide memory. One dimensional array architectures have been designed or implemented by a number of people [Fisher, 1986, Fountain, et al., 1988]. Some work has indicated that this architecture has a good cost/performance trade-

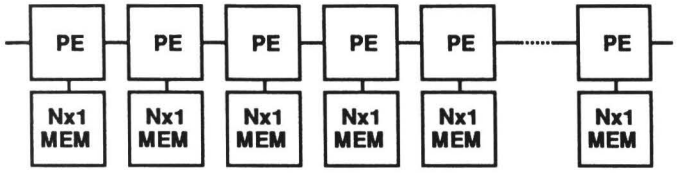


Figure 1 The one dimensional SIMD array

off for many imaging algorithms [Fountain, 1986]. The Applied Intelligent Systems AIS-5000 is shown in Fig. 2 where three major components are designated: a general purpose host processor, special purpose high speed I/O hardware for the parallel processor hardware, and the SIMD array with a controller.

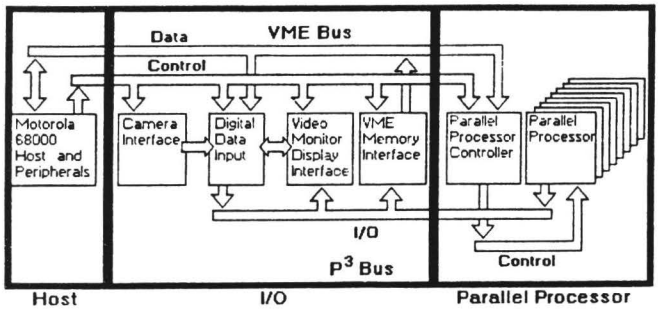


Figure 2. The AIS-5000 SYSTEM

Fig. 3 shows the physical arrangement of the parallel processing units where the fundamental building block is a pair of chips: a static byte wide memory, coupled to the PIXIE chip which contains eight PEs. Each data line from a memory is connected to a single PE. The processors handle neighborhood transformations (N), boolean operations with 16 bit truth tables (B), and bit serial arithmetic operations (A). Processor instructions and memory addresses are common and are broadcast to all units by the central controller.

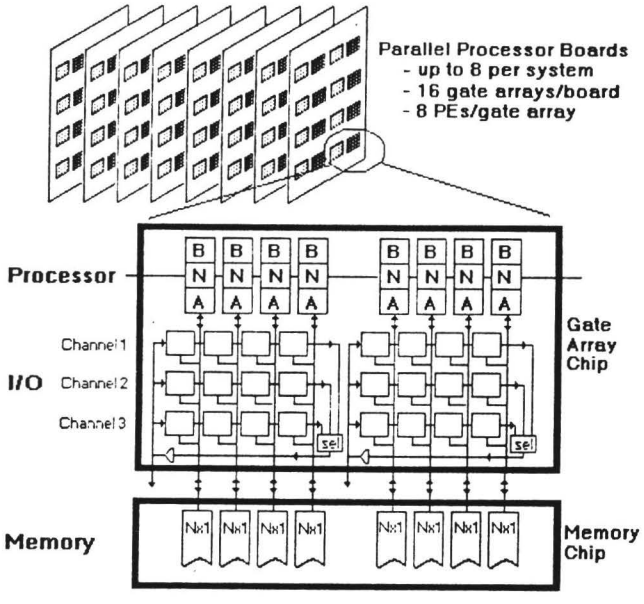


Figure 3. The parallel array.

A new custom chip under development called the Centipede [Wilson, 1988] will be used in one dimensional array systems with an architecture very similar to the AIS-5000. The boolean, neighborhood, and arithmetic processing sections are similar to the PIXIE chip, but the major enhancements are in data communications and transformations, and a hardware multiply-accumulator (MAC). The Centipede will also be capable of indirect addressing of the local memory, 8x8 transposes on sub-arrays, and bit-serial or word-parallel modes of arithmetic. A block diagram of the processing units are shown in Fig. 4. There will be 32 processing units on an integrated circuit chip with 25,000 gates, and one processor for each column in the data matrix. The entire memory associated with the parallel array is mapped so that it can be directly read or written by either the host or controller. However, the I/O shift registers are a more efficient means for data transmission.

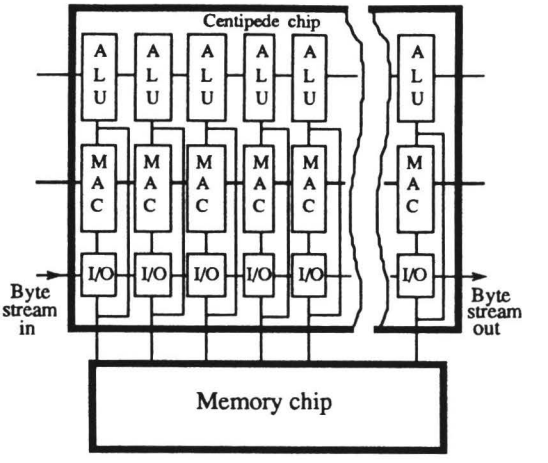


Figure 4. The CENTIPEDE chip

Data bytes from one row of memory across the processor array can be shifted up in 8 cycles to the I/O registers in the centipede chip shown in Fig. 4. This row can then be shifted east in a number of clock cycles equal to the number of processing units. Meanwhile a new row of data can simultaneously be shifted in from the west into the same I/O shift registers, and then stored into the local memory. The east I/O shifting operations do not interfere with processing operations, and are thus very efficient. Although these shift registers were made for camera input for vision processing, they also serve efficiently for the types of block data moves needed in neural processing. This I/O architecture easily and naturally scales as more processing units are added.

A simplified diagram of the MAC circuit for a single processing unit is shown in Fig. 5. The MAC contains hardware for bit-serial multiplications. The design of the bit-serial arithmetic unit allows it to perform operations at the same rate as bits can be read from memory. The U register is an 8 bit register

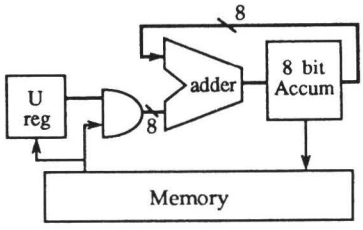


Figure 5. The multiply-accumulate circuit.

which holds the multiplicand. The 8 bit adder can sum the multiplicand with the contents of the accumulator. The multiplier is read from the memory, one bit at a time and can control the operation of the adder through the AND gates shown in Fig. 5, so that multiply operations are done by a succession of shift and conditional add operations where the condition is controlled by whether a one or a zero multiplicand bit is read from the memory to the AND gates. The product is read out of the lowest bit of the accumulator to memory as the accumulator is shifted down during the multiply process. In this manner, the MAC circuit associated with every unit is maximally efficient when performing fixed-point arithmetic where one of the operands (multiplicand) is 8 bits or less, and the other operand (multiplier) has any precision.

3 Neural Processing

The neural model to be adapted to the SIMD array is the general form given by Rumelhart, et al., [1986]. Using their notation the various components of a single processing unit i include a

net: $\quad net_i = \sum_j w_{ij} O_j$,

activation function: $\quad a_i(t) = F_i(a_i(t-1), net_i)$, and

output function: $\quad O_i = f_i(a_i)$,

where W_{ij} are weights from unit j to unit i, O_i is the output from unit i, and F_i and f_i are functions which specify the nature of the neural processing units. The network model for a particular layer is shown in Fig. 6, where there are M input signals S_i and N outputs O_j, where M is less than or equal to N. For computations on the SIMD array, it will be easier to separate the net and weights into two parts, one for input signals, and one for output signals:

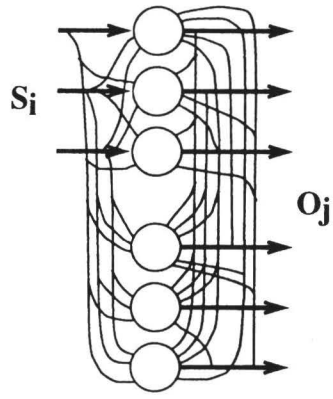


Figure 6. The connection model.

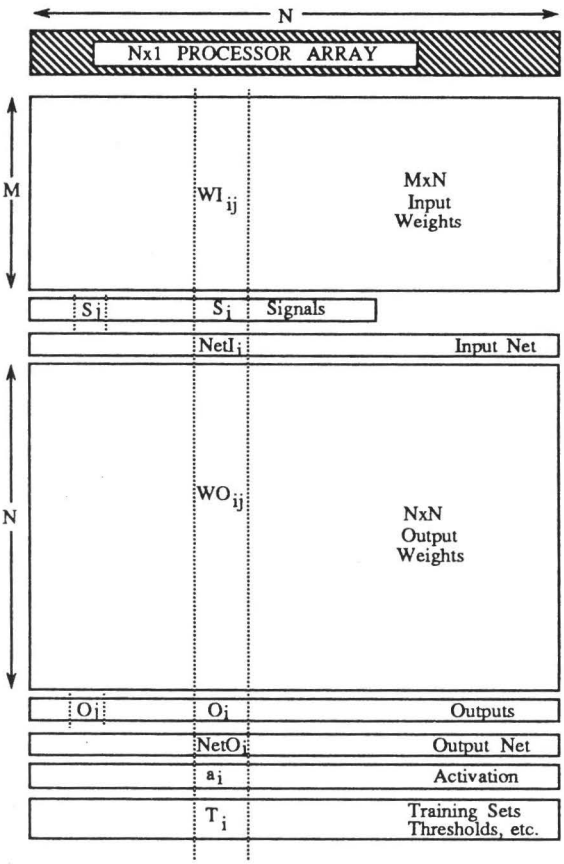
$$net_i = netI_i + netO_i \text{ , where}$$

$$netI_i = \Sigma WI_{ij} S_j \text{ , and } \quad netO_i = \Sigma WO_{ij} O_j .$$

The activation state a_i at time t is a function of the net and the previous activation state at time t-1. With this definition, temporal decay of neural activity can be provided.

3.1 SIMD Processing

The organization and memory allocation of a single neural layer in a one dimensional SIMD computer system is shown in FIG. 7. Each neural cell is assigned to one processing unit. More than one cell could be assigned, or, if memory storage capacity is too low, a single neuron cell could be assigned to more than one unit. These extensions will not be covered here. The weights WI_{ij} for unit i will be stored in a vertical single bit wide stack in the memory assigned to that unit. Subscripts i and j respectively denote column and row indices. The number of storage bits required are bM where b is the number of bits in the weight, and M is the number of input signals. The output weights WO_{ij} are stored in a similar manner, where bN bits are required for each column. All neurons are completely interconnected. Signals S_i, and the input net, $netI_i$, are each stored in single rows of the memory, each row comprising a number of bits equal to the size of the word. Other rows in the memory can store other vector variables such as output signals O_j, the output net $netO_i$, activation states, training patterns, thresholds, function parameters, and perhaps random variables for statistical models. Since the outputs of all neurons are fully connected to inputs of all other neurons in a specific layer in this model, a relaxation process must be used. A number of iterations of computations must be performed in order for



MEMORY ORGANIZATION FOR NEURAL PROCESSING

Figure 7

the outputs to converge to stable states. For the first iteration, the outputs start out at some initial state provided by the particular neural model. Since the inputs are assumed to be stable, no iterations are needed to compute the input net. The total net was partitioned into a separate input net and output net so that unnecessary iterations could be avoided for the stable input signals.

To compute the input net, all signals, S_i are first sent to the controller via the fast I/O path discussed earlier. In the controller, the signals are treated as row multiplicands, and should now be denoted as S_j. Assume that the signals are eight bits. The controller then broadcasts, and loads the first signal S_1 into all U registers in all the MACs to act as multiplicands. The first row of weights will be the multipliers. Using the MAC hardware as described earlier, all weights WI_{i1} are multiplied by the first signal S_1, making 100% efficient usage of the parallelism. These products are accumulated in the netI row of the memory. Next S_2 is loaded and multiplies weights WI_{i2}, and accumulates them in netI. This process continues until all products are summed into the netI row.

The same procedure occurs for the output weights: outputs O_j are multiplied by the output weights WO_{ij} and accumulated in $netO_i$. The input and output nets are summed to get the total net. The activation state and output functions are computed next. Since they are only functions of the previous activation state and the total net, the computation does not require any variables or parameters which are stored outside the local memory for each processing unit. It is assumed that those parameters such as decay constants, and sigmoid shape parameters, that are used in the computations of these functions are common to all neural units and would be incorporated in the SIMD instructions broadcast to all units. Thus the activation and outputs for all neural units can be computed in parallel with no inter-processor data communication overhead. Since they are simple vector operations they proceed much faster than the matrix computations of the nets.

For the next iteration in the relaxation process, the output net is computed again by multiplying the output weights by the new output states to generate a new output net. This new output net is added to the input net to form a new total net. New activation states, and new outputs are computed. These operations can be iterated until adequate convergence occurs according to some relaxation scheme given by the model. Finally the output row which is assumed to be eight bits will be sent to the I/O registers in the processing units and rapidly transferred out while the new input states are simultaneously being transferred in on the same I/O shift registers, as in Fig. 4.

3.2 Learning

The general form of Hebbian learning is given by Rumelhart, et al., [1986] as

$$w_{ij} = g(a_i(t),t_i(t))\, h(O_j(t),w_{ij}),$$

where $t_i(t)$ is a teaching input, and g and h are functions particular to the model. Computations of this form are very efficiently implemented on the one dimensional SIMD architecture. The g function is simple because the arguments involve row operations only, and are not functions of the column dependent variables. Computation of the h function is similar in concept to methods previously discussed, where the O_j vector is first read out to the controller. However, rather than a global multiplication of O_j with a row of weights, the operation is much simpler. For example in simple Hebbian learning [Rumelhart et al. 1986], or the Widrow-Hoff rule [Sutton and Barto, 1981], h is equal to O_j. In rules used by Grossberg [1976], h is a simple difference: $h = O_j - w_{ij}$.

With the above implementation of neural computing in both learning and network operation, the global inter-processor communication of output states is done without a great deal of data movement. When the output state of a unit is to undergo operations by other units, the value of that state is transmitted to all other units by instructions broadcast by the controller, and not as a series of separate data movements. It is for this reason that the SIMD architecture is very efficient for large fully connected networks.

4 Performance

Benchmarks for the evaluation of performance of the one dimensional SIMD array compared to other computation architectures are difficult because there are so many neural models; however a few general remarks can be made. Fixed point addition operations are much faster than floating point additions on SIMD systems because of the difficulty in handling the alignment of the fraction parts of two floating point numbers when the exponent parts are not equal. The alignment and subsequent renormalizing of the floating point numbers in a bit serial machine takes more time than the addition operation itself. If the input and output signals are limited to eight bit fixed point numbers, and the weights are floating point, then the multiplication and addition operations can proceed much faster than if the signals and outputs were also floating point. This is not generally a restriction since strict constraints on the dynamic range of these signals is common. For example in thermodynamic models [Hinton and Sejnowski, 1986] the signals can be single bit variables. In connectionist modeling the signals can have a limited number of discrete states, for example, 0 to 9, [Feldman and Ballard, 1982].

Floating point operations are used in neural computing because a wide dynamic range is needed for the weights. The precision is less important. The time to compute the sums of products for the input or output net is dependent on the clock period, number of clock cycles for a single row operation (signals times weights + accumulation into the net), and the total number of rows, or the time is given by:

$t = T(\text{time/cycle}) \times C(\text{cycles/row}) \times R(\text{rows})$.

If we choose an 8 bit fixed point representation for the input and output signals; a 14 bit floating point representation (8 bit fraction with 6 bit exponent) for the weights; and a 24 bit representation (16 bit fraction with 8 bit exponent) for the net, then C = 530 SIMD clock cycles are needed to compute the product of signals times a single row of weights on the Centipede chip. Suppose there are R = 512 neurons, and the processor clock rate is 20 MHz, then the time to compute the input or output net using this partial floating point representation is t = .05 usec x 530 x 512 = 13.6 msec. This time scales linearly with the number of neurons assuming that there is one processing element per neuron.

Suppose the signals are 8 bits, the weights are 32 bits, and the net is 50 bits, all in fixed point. Then 180 clock cycles per row operation are needed, and the time t to compute the net is 4.6 msec. For single bit signals, no multiplications are needed. Then C = 78, and the time for 32 bit weights and a 41 bit net is 2 msec. Performance in terms of equivalent connections per second is given by the total number of connections, 512x512, divided by the processing time per net. These results for 512 neural units are summarized in the table below.

Processing time/net	Connections per second	Signal range	Weight range	Net range
13.6 msec	19 million	8 bit fixed	8 bit frac. 6 bit exp.	16 bit frac. 8 bit exp.
4.6 msec	57 million	8 bit fixed	32 bit fixed	50 bit fixed
2 msec	131 million	1 bit fixed	32 bit fixed	41 bit fixed

The above processing times must be multiplied by the number of times that a relaxation iteration is performed. If 100 iterations are needed for convergence, the total time is still reasonable, ranging from 0.2 seconds to 1.36 seconds.

For highest speeds in processing, a number of Centipede arrays can be joined in a multiple SIMD pipeline as shown in Fig. 8, where communication is provided by the I/O registers. This architecture expansion will allow any number of neural layers to be handled as quickly as one. The system shown in Fig. 8 for example, can handle three neural layers. The first P.C. board receives the external signals, and processes the first layer. Output states are sent to the second board while new input signals are being received by the first board. While the second board

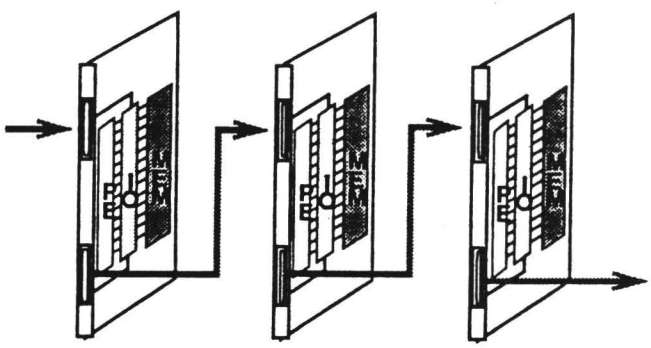


Figure 8. A multiple SIMD pipeline.

is processing the second layer, the first board is processing the new first layer. The third board processes the third layers in a similar manner and transmits the final output states.

5 Conclusion

The bit serial nature of the processing units in a SIMD computer allows a wide range of trade-offs in speed vs. precision. Usage of various representations of fixed and floating point variables for a 512 x 512 array of weights correspond to an equivalent execution rate of 19 to 131 million neural connections per second. For a larger number of neurons, the number of computations will grow quadratically, but the computation time will only grow linearly.

There are a number of other favorable properties of this architecture. One main intent of the design of the current SIMD arrays was to provide an I/O subsystem which was capable of rapidly moving images in and out of the memory local to the processing units without interfering with the parallel processing. This I/O mechanism also works well with the type of data movement needed for neural processing. Rows of signals, outputs, and weights can be quickly transferred in and out of memory with very little time penalty. For multiple layered networks, several SIMD systems can be connected in a pipeline architecture, where the I/O subsystem transfers data between each SIMD system in the pipeline.

It is generally understood that for a large degree of parallelism to be feasible in neural networks there is also a requirement for a capability of massive global communication because of the obvious intimate interconnection of all neural cells. The one dimensional SIMD array does not explicitly have the degree of inter-processor communication that a hypercube or mesh connected system contains. Yet inter-processor unit communication is not a bottle-neck. The reason the one dimensional system works well is that the inter-processor communication is rather subtly imbedded in the nature of the SIMD concept. For example, when the system controller has an input signal value, it can cause all processor units to operate with that value simultaneously via the "Single Instruction" aspect of SIMD. It is the full broadcasting of an instruction, and not the actual movement of data that results in a global communication of information. For that reason SIMD architectures have an advantage over other parallel architectures where data movement is the primary means of providing communication in a network. Furthermore, the one dimensional SIMD architecture has a very strong economic advantage over other SIMD architectures which have intercommunication schemes which are unnecessarily complex for neural networks.

References

[Feldman and Ballard, 1982] J.A. Feldman and D.H. Ballard. Connectionist models and their properties, *Cognitive Science*, Vol. 6, pp. 205-254, 1982.

[Fisher, 1986] A.L. Fisher. Scan line array processors for image computation. In *13th Annual Symposium on Computer Architecture*, Tokyo, Japan, pp. 338-345, 1986.

[Fountain, 1986] T.J. Fountain. Array architectures for iconic and symbolic image processing. *Proc. 8th Int. Conf. on Pattern Recognition*, Paris, France, pp. 24-33, Oct. 1986.

[Fountain, et al., 1988] T.J. Fountain, K. N. Matthews, and M. B. J. Duff. The CLIP7A Image Processor. *IEEE Tran. Pattern Anal. Machine Intell.* Vol. 10, No. 3, pp. 310-319, May, 1988.

[Grossberg, 1976] S. Grossberg. Adaptive pattern classification and universal recoding: Parallel development and coding of neural feature detectors. *Biological Cybernetics*, Vol. 23, pp. 121-134, 1976.

[Hillis, 1985] W.D. Hillis. *The connection Machine*. The MIT Press, Cambridge Mass., 1985.

[Hinton and Sejnowski, 1986] G.E. Hinton and T.J. Sejnowski. Learning and relearning in Boltzman machines. In *Parallel Distributed Processing*, (ed. D.E. Rumelhart and J. L. McClelland), The MIT Press, 1986.

[Potter, 1985] J.L. Potter, ed. *The Massively Parallel Processor*. The MIT Press, Cambridge, Mass., 1985.

[Rumelhart, et al., 1986] D.E. Rumelhart, G.E. Hinton, and J. L. McClelland. A General framework for parallel distributed processing. In *Parallel Distributed Processing*, (ed. D.E. Rumelhart and J. L. McClelland). The MIT Press, 1986.

[Sutton and Barto, 1981] R.S. Sutton and A.G. Barto, Toward a modern theory of adaptive networks: expectation and prediction. *Psychological Review*, Vol. 88, pp. 135-170, 1981.

[Schmitt and Wilson, 1988] L.A. Schmitt, and S.S. Wilson. The AIS-5000 parallel processor. *IEEE Tran. Pattern Anal. Machine. Intell.*, Vol. 10, No. 3, pp. 320-330, May 1988.

[Wilson, 1985] S.S. Wilson. The Pixie-5000, a systolic array processor. *Proc. IEEE Workshop Computer. Architecture Patt. Anal. Image Database Management.*, Miami Beach, FL, pp. 477-483, Nov. 1985.

[Wilson, 1988] S.S. Wilson. One dimensional SIMD architectures -the AIS-5000, In *Multicomputer Vision*, Ed. S. Levialdi, Academic Press, London, 1988.