Synthesis of Multiple Bus/Functional Unit Architectures
Implementing Neural Networks

Baher Haroun and Elie Torbey
Department of Electrical and Computer Engineering, Concordia University
1455 de Maisonneuve Blvd. W., Montreal, Quebec H3G 1M8
e-mail: [haroun,elie]@vlsi.concordia.ca

Abstract
This paper presents an automated architectural synthesis methodology for implementing digital neural networks. The synthesis approach uses heuristics and is based on VLSI multiple-bus/functional unit architectures with internal parallelism. The synthesis methodologies and trade-offs as well as the features of the architectures are presented. The resulting architectures from the synthesis tool outperform other architectures for the same applications.

1. Motivation for neural network synthesis

The use of ANNs (Artificial Neural Networks) for real-time applications requires the design of efficient architectures satisfying hard performance requirements. Existing digital neurocomputers use large and expensive general purpose processors in a single or multi-processor configuration which provide more flexibility than is required in specific applications [11,5]. Systolic array implementations using simple processing elements with limited internal parallelism are also proposed [8], but do not investigate the use of parallelism within each processing element. Similar problems in DSP hardware were overcome by the use of automated ASIC design and architectural exploration techniques [1]. Neural networks are special signal processing applications with a high degree of connectivity, massive data transfers, special activation function considerations, and large memory storage requirements. The parallelism inherent in ANN algorithms should be investigated in the design of neural processors. A synthesis tool is therefore presented that explores different degrees of parallelism of proposed architectures and generates one that satisfies the real time constraints and fulfills the VLSI silicon area limitations.

2. Multiple bus/functional unit architecture

The architecture proposed, shown in Figure 1, has been successfully used for the synthesis of DSP systems [3]. It is a bus style datapath with \( nb \) busses, \( nfu \) functional units (FUs) (adders, multipliers, etc...), and \( nr \) storage registers. A similar architecture was used by Vlontzos and Kung [10] as the processing element (PE) in a systolic implementation of ANNs, but without investigating internal parallelism.

The architecture has one register file connected to each bus. The number of RAMs used is less than the number of busses and prefetching is used. FU inputs are connected to some or all of the busses through multiplexers, their outputs through tristate bus buffers. The architecture uses a two phase clock cycle resulting in an efficient use of the busses and in the grouping of registers into one register file per bus. A microcode controller (generated by the synthesis tool) produces one control word per clock phase.

The vector-matrix operation in ANN algorithms

\[
y_j(l+1) = \sum_{i=1}^{N} x_i(l) w_{ij} (l+1) + \theta (l+1)
\]

make it necessary to provide a special FU for these computations. Pipelined multiply-accumulator units (MACs) such as the ones shown in Figure 2 are therefore used. The MAC in Figure 2-a can perform multiplications,
additions, truncations (used for activation function implementations) as well as combinations of these operations. The MAC in Figure 2-b is a simple multiply-accumulator while the one in Figure 2-c has the additional advantage of allowing two operations to be initiated at the same time, thus reducing the constrictions on scheduling and increasing the utilization of the whole unit. A typical MAC has a set of local storage registers \( lr \) (accumulators) used to store the intermediate values of different multiply-accumulate operations (MA). Using the local storage registers accumulates the partial sums on the same MAC and saves on data transfers and temporary storage of intermediate values. Using more than one \( lr \) results in substantial savings when the number of buses is constrained. Different degrees of pipelining of the adder can result in reduced utilization of the unit executing an MA and increase the delay. Using a number of \( lr \) equal to the number of stages in the adder can enhance the performance by allowing interleaving of operations with minimum delay (Section 3.2). Adding the threshold unit in the MAC rather than providing a separate \( FU \) results in a more efficient transfer function implementation such as a linearized sigmoid. Separate FUs, such as Comparators and Lookup Tables are also supported. The weights are not stored in a separate RAM connected to FU inputs in order to allow for parallel access to the weights in the learning phase. Having a separate weight RAM will increase the speed of the recall phase but restrict the weight update learning phase. This architecture is then particularly suited for adaptive real time applications. Both the architecture and the synthesis methodology support a wide range of ANN algorithms [9].

3. High-level synthesis methodology

3.1. Synthesis of neural network algorithms

The high performance requirements of ANNs add to the complexity of their silicon implementations. In order to meet the high computational power demand, design aids have to be developed. SPAID-N is an architectural synthesis tool, written in Prolog, based on the multiple-bus/FU architecture described above. It is an extension of SPAID (Signal Processing computer AIded Design) [3], and customized for ANN applications. The tool is interactive and uses user defined heuristics to obtain an optimal design. It is divided into three major synthesis phases.

Network description: The network description, which is modelled after the Axon language presented by Hecht-Nielsen [6], specifies the type, architecture and interconnectivity of the ANN and is transformed into a SFG as an intermediate form (which can also be used as input). The SFG is formed of the operation set and the connections set. In the case of backpropagation (BP) networks, the dependence is imposed by the layered structure of the network. For a \( N \)-layer network \( L_0, L_1, ..., L_N \) the scheduling time of the tasks corresponding to layer \( L_i \) \( (L_1, ..., L_N) \) should satisfy \( O(S_i) < O(S_{i+1}) \). The set of operations for each layer associated with each neuron is further divided into classes of operations performed by all neurons as shown in Figure 3 for the generalized delta rule. Those sets are pre-ordered to achieve better reservation schemes based on the number of interconnections of a neuron \( I(n_i) \) which is assumed to correspond to its execution time \( T(n_i) \). The ordering of the operations within a class ensures a good scheduling scheme while the ordering of the classes ensures correct data dependency.

Scheduling and reservation: The SFG is the input to the second phase where the operation set is scheduled on the available FUs and buses using a list scheduling approach. The operations list is pre-ordered as explained above. The reservations corresponding to two consecutive classes can overlap without violating the data dependencies. Greedy algorithms using binpacking heuristics [2] are employed to maximize the utilization of all FUs. Modified First fit algorithms are used for MAS. Each MA operation \( ma \in O \) consists of \( n \) operations \( o_1[ma], o_2[ma], ..., o_n[ma] \). Once a MAC has been
bound to that MAC where binding of pipelined operations is performed (Section 3.2). These operations do not have to be consecutive and can be interleaved with other operations by using the local storage registers of the MAC. All other operations are scheduled using a modified best fit algorithm. An operation to be scheduled will have to be reserved on a bus allowing for duplication of storage and resulting in a flexible scheduling scheme. Optimizations are then performed in the last stage.

Architecture Optimization: The third phase, performs several optimizations on the architecture. It involves a modified bipartite edge coloring algorithm to assign the registers used into a RAM for each bus, minimizing duplication of storage. Minimizations are also performed on each RAM by having temporary registers re-using the same hardware registers. The algorithm used colors a circular arc graph of the registers lifetimes [4]. Initially each link in the SFG a register. Even though the bus scheduling approach allows for duplication, the optimizations consistently reduce the number of registers (for BF in recall) to within 10% of the lower bound:

\[ I_b = M + W + \sum_{i=1}^{K} C_i \quad \text{where} \quad M = \max_{i=1}^{K} \left( \sum_{j=i}^{K} N_j \right) \]

where \( I_b \) is the number of the neurons in layer \( i \), \( W \) the number of all weights and \( C_i \) the number of constants in layer \( i \).

3.2. Scheduling on pipelined functional units

Figure 4: Pipeline reservation example

Pipelined units such as the MACs shown in figure 1 add to the complexity of the synthesis procedures. A special hazard avoidance algorithm is used to ensure that no conflicts occur in the allocation of hardware resources. The first part of this algorithm uses the reservation tables obtained from the hardware descriptions of the FUs augmented by information on resource limitations to generate initiation lists for all pairs of operations. One operation is used as reference and the other is shifted in time appropriately generating all the possible initiation times of the second operation when the first is being executed on a FU. This part is done prior to the start of the synthesis procedures. The second part of the algorithm is invoked when an operation is to be scheduled on a pipelined FU. For an operation to be scheduled at time \( T \), the set of initiation vectors, for the operations executing on the FU relative to the new operation is:

\[ V_i = \{ T_p, \ldots, T \} \]

where \( T_p = T - \delta \cdot S \times C_i \)

\( \delta \) being the number of clock cycles used by a stage and \( S \) being the number of stages of the FU. The initiation vector of the new operation will be:

\[ V_i = \{ T_i - \delta \} \]

\( T_i \) is the delay between existing operations and the new one. Thus, if a TRUNC operation whose inputs are available at clock cycle \( T \) is to be scheduled on the FU of Figure 4 (This corresponds to the MAC of Figure 2(a)) where 2 MAs and an addition are executing as shown, the resulting initiation vector becomes \([-1,0,\ldots], or [T+1,T+2,\ldots,\infty].

In the case of multi-stage adders within the MAC, special consideration need to be addressed in order to prevent Read before Write hazards. This is shown in figure 5 where the cases for one or two local registers are shown. For one \( l_r \), an extra delay is needed. This is guaranteed by a dynamic resource limitation list that scans the FU scheduling lists backwards by the total FU stage delay to adjust the initiation list. For the case of two \( l_r \), however, other MA operations can be executed on the same FU. Since a MA is divided into small atomic MAs which can be interleaved with those of another operation, the delay is not needed.

Figure 5: Multi-stage adder with local storage
3.3. Network partitioning and loop Folding

In order to reduce the controller size, the network is divided into similar partitions and looped execution of the network is implemented. Vertical folding is done by executing one layer at a time. This is accomplished by the addition of feedback edges to the SFG (This adds to the complexity of the controller). The addition of these edges does not modify the datapath scheduling. Horizontal folding divides each layer into similar partitions and translates into saving partial sums of the partitions in the local registers. This necessitates the division of ANN algorithms such as the BP in recall in sets of MAS followed by a class of activation function calculation after all partitions have been executed.

4. Performance evaluation

4.1. Architectural trade-offs of typical networks

In order to evaluate the performance of synthesized networks and investigate the architectural trade-offs, the MAC organization of Figure 2(a) (1 clock cycle per stage) is used in the proposed datapath. The activation function is implemented as a simple hard limiter on the threshold unit. Three fully connected BP networks are presented. The first network is a robotic Flexible Joint Manipulator with 6,21,12 and 1 neurons respectively [12], the second is an EKG signal processing network with 40,10, and 1 neurons respectively and the third is a pattern recognition with 16, 5, 9 and 4 neurons [7]. These networks are typical of the ANNs (in size, topology and algorithm) used in adaptive real time applications.

The percentage increase in area relative to the reference architecture (An initial minimum configuration with one MAC, one local register and one bus) is based on the area assumptions which are used to indicate cost trends and not absolute cost values as these depend on layout details and technology:

\[ A = 30 \cdot \text{nmac} + nr + 20 \cdot \text{nmac} \cdot nb \]

The design space search involves different combinations of the number of MACs, busses and local registers.

Figures 6 and 7 show the synthesis results for the robotic and EKG networks in the recall and training modes respectively based on \( AT^2 \) where \( A \) is determined as shown above and \( T \) is the total execution delay in clock cycles. The optimum architectures are circled and the number of busses and local registers used in each is indicated. The architectures reach their optimum values using well below the limit of two busses per FU. It is clear that the optimum architectures for different networks are not necessarily identical. The two networks (with similar size) achieve different performances in the recall and learning modes due to the difference in their topologies.

While the performance evaluations as presented above are helpful in investigating architectural trade-offs, better area and delay estimates from actual VLSI implementations are required to optimize ANN chip designs. VLSI speed and area measurements from standard library cell designs of arithmetic and memory units for a CMOS 1.2 \( \mu \)m technology were used to give an indication of the area-delay trade-offs. The speed is based on a 45 ns processor clock (15 ns datapath stage delay and 15 ns interconnection delay per clock phase) while the area includes that of the datapath, the RAMs used for the register files connected to each bus and the controller as well as estimates of the areas used by the busses and other interconnects. Figure 8 shows the area delay trade-offs for the pattern recognition network.

Table 1 contains a sample of the utilization percentages for the MAC stages for the Robotic network. The utilization of the threshold is low since it is performed once for every neuron calculation which includes a large number...
of multiplications and additions. The multiplier and the adder have the same utilizations in recall since they are only being used in MAs. In learning, the number of additions and subtractions is larger than multiplications (even though multiplication classes outnumber addition classes) resulting in better utilization of the adder.

4.2. Cycles per connection comparisons

The Delta II ANS processor achieves 4 CPC (Cycles Per Connection) [11]. The Balboa HNC board uses an i860 with 4.4 CPC (learning) and 1.5 CPC (recall). The GCN [5], a network of 128 PEs each being implemented by an Intel 80860 achieves 6.4 CPC. The optimum architectures presented in this paper achieve consistently less than 1 CPC (Recall). The best architectures produce results as low as 0.31 CPC (Recall) and 0.63 CPC (Learning). Our architectures outperform the above mentioned systems since they have been optimized for the specific applications using the synthesis tool. Furthermore, special FUs targeting ANNs have been used. Also, implementing networks as straight line codes eliminates all the overhead associated with branching. Therefore, the synthesis methodology and the architecture presented here result in a better performance than general purpose neural processors by avoiding the overhead induced by the generality of those systems and optimizing the architectures for specific applications.

5. Conclusion and extensions

The synthesis methodology presented results in architectures that can achieve high performance for ANN implementations used in adaptive real time applications. We have described the scheduling approach and the effect of architectural trade-offs on ANN system performance. These architectures are simple and lend themselves to efficient VLSI implementations because of regularity. In consequence, customized ANN chips can achieve execution speeds unequaled by other general purpose processors. Also, we have shown that exploring the parallelism within a processor is important for optimal use of silicon area. We have also demonstrated the importance of exploring architectural trade-offs in order to optimize the architectures. Extensions to this work includes multiprocessors and systolic embedding of ANNs in the same synthesis framework.

References