Programmable Neurocomputing

Krste Asanović
MIT Laboratory for Computer Science
200 Technology Square
Cambridge, MA 02139
krste@mit.edu

Appears in *The Handbook of Brain Theory and Neural Networks, 2nd edition*,
(M.A. Arbib, Ed.), Cambridge, MA: The MIT Press, 2002. (c) The MIT Press
http://mitpress.mit.edu

1 Introduction

General-purpose personal computers and workstations are the most popular computing platforms used by researchers to simulate artificial neural network (ANN) algorithms. They provide a convenient and flexible programming environment and technology advances have been rapidly increasing their performance and reducing their cost. But large ANN simulations can still overwhelm the capabilities of even the most powerful workstations. For example, computations may require greater than 10^{15} arithmetic operations and operate on data sets containing several gigabytes of data [5].

Many neural net algorithms are highly parallelizable, allowing simulation speed to be improved by employing a network of workstations (NOW) [2]. Compared with more specialized hardware, a NOW can be an expensive solution. Fast network hardware increases the cost per node and parallelization overheads reduce the performance per node.

For constrained application domains, fixed-function neural computing circuits can provide extremely high compute speeds at low cost, but these do not provide the flexibility required to support experimentation with ANN algorithms.

To meet the need for high performance on large ANN simulations with flexible software control and reasonable cost/performance, several groups have proposed and built *programmable neurocomputers.* Programmable neurocomputers (hereafter abbreviated to "neurocomputers") attempt to maintain most of the flexibility of a general-purpose computer system while improving the cost/performance ratio by specializing processors for neural computation.

This article first reviews the most significant neurocomputer architectures, discusses their design and use, before concluding with predictions of future trends.

2 Survey of Neurocomputer Architectures

Programmable neurocomputers can be classified into four major categories. The first category uses commercial digital signal processors (DSP) while the last three categories are based on custom-designed silicon.

2.1 Commercial DSP Arrays

Several neurocomputers have been built using arrays of commercial DSPs. Two notable examples are the RAP (Ring Array Processor) [11] developed at the International Computer Science Institute and the MUSIC system [12] developed at the Swiss Federal Institute of Technology. Both of these systems connect the DSPs in a unidirectional ring topology with communication circuitry built from field-programmable gate arrays (FPGAs). The RAP supports up to 40 Texas Instruments TMS320C30 floating-point DSPs, with a peak performance of 32 MFLOPS per node. The MUSIC system connects up to 45 Motorola DSP96002 floating-point DSPs, with a peak performance of 60 MFLOPS per node.

Both of these systems have distributed memories and are programmed using a Single Program Multiple Data (SPMD) model, where all nodes run identical programs but operate on different portions of the data. A separate host computer manages the overall program flow and handles data input and output.

2.2 SIMD Processor Arrays

Another popular approach in neurocomputer design is a SIMD (Single Instruction Multiple Data) array of processors with some limited form of interprocessor interconnect. In these SIMD designs, a central sequencer broadcasts instructions that are executed simultaneously by all processors. The processors in a SIMD system can be much simpler than those in a SPMD system because they do not have to fetch and decode instructions. Also, SIMD processing elements do not require separate synchronizing operations because all processors work in lockstep.

Example SIMD neurocomputers include the CNAPS systems [6] from Adaptive Solutions and the SNAP [10] system from HNC. The CNAPS system is built around large custom chips containing an array of 64 SIMD processing elements. Each processing element contains a fixed-point 16-bit$\times$8-bit multiplier, a 24-bit accumulator, a set of 32 16-bit registers, and 4 KBytes of local on-chip memory. The processing elements are connected by two 8 bit broadcast busses and a 2 bit inter-processor ring connect. The HNC system is built from SNAP chips each of which contains four 32-bit floating-point multiply-add datapaths with access to local off-chip memory. Both of these systems allow multiple chips to be interconnected and controlled by the same central sequencer.

2.3 Systolic Processor Arrays

Several neurocomputers have been built around systolic processor arrays that perform the matrix operations at the heart of most neural algorithms. A systolic processor contains an array of interconnected pipelines through which operands flow in a regular rhythmic fashion.

The most advanced of these systems is the Synapse-1 constructed and sold by Siemens [13]. This system is based on a custom systolic multiply-accumulate chip, the MA-16, which integrates sixteen 16-bit fixed-point multipliers. The Synapse-1 system employs multiple chained MA-16 chips to give higher throughput. Large quantities of off-chip memory are provided, split into several disjoint memory areas. Operands must be located in the correct memory region before performing a given systolic matrix operation. Additional special-purpose fixed-point datapath hardware is provided to support ANN node activation functions not provided by the MA-16 chips, and Motorola 68040 CISC processors are used to perform all other operations. The entire system is controlled by a host workstation.

Another example of a systolic neural net engine is the Mantra machine [7] built at EPFL, Switzerland. Mantra can have up to 1600 bit-serial processing elements arranged in a 40×40 systolic mesh. A commercial DSP acts as system controller.

2.4 Vector or SIMD Coprocessor

The previous machines all rely on some form of off-chip control sequencer or host computer to manage the matrix computations occurring on the parallel processor arrays. An alternative approach is to tightly integrate the control processor with the parallel execution units on the same die. Two examples of this type of design are the T0 vector microprocessor [15] and the L-Neuro 2.3 multi-DSP [4].

The T0 vector microprocessor integrates an industry-standard MIPS-II 32-bit integer scalar RISC processor with a tightly-coupled fixed-point vector coprocessor. The vector coprocessor contains a central vector register file with 16 vector registers each holding 32 elements of 32 bits, two vector arithmetic units, and a vector memory unit. The two vector arithmetic units each contain 8 parallel pipelines and can produce up to eight 32-bit results per clock cycle. One of the arithmetic units contains 16-bit fixed-point multipliers but otherwise the two pipelines are identical. The memory unit connects to off-chip memory over a 128-bit data bus. T0 has a single flat memory space equally accessible by the scalar unit and any element in the vector unit. T0 is similar in design to vector supercomputers [14], and scalar and vector instructions can be freely intermixed in the single instruction stream. The instruction set was designed to enable object-code compatibility with future higher performance implementations.

The L-Neuro 2.3 design contains a 16-bit RISC controller plus an array of 12 DSP datapaths. The DSP datapaths are controlled via a writable microinstruction store indexed by the RISC controller macroinstructions. The wide microinstruction words allow pairs of DSP datapaths to perform different functions, and a flexible inter-DSP communication network is provided. The L-Neuro 2.3 supports an off-chip memory connection for each DSP datapath.

3 Neurocomputers versus General-Purpose Processors

Neurocomputers are distinguished from general-purpose processors by their specialization for neural computations. If we examine the range of neurocomputers above, we find that three features specific to neural computation have been exploited to improve cost/performance:

- Limited numeric precision. Many neural algorithms can be coded to require only 8–16 bits of fixed-point arithmetic precision [3]. The reduced precision allows reductions in the area required for arithmetic circuits, particularly multipliers, and also reduces the bandwidth required to transfer operands.
- Data parallelism. Most neural algorithms are inherently highly data parallel, where the same operation is performed across large arrays of data. Data parallelism is the simplest form of parallelism to exploit because a single block of control hardware can be shared over many datapaths.
- Restricted communication patterns. Broadcast busses or unidirectional rings are sufficient to support parallel execution of many common neural network algorithms. These simplified communication networks reduce the cost of interconnecting large numbers of parallel processing elements.

All neurocomputers aim to achieve high performance on the matrix computations at the heart of many neural algorithms by exploiting these features. But real world neural net programs require operations other than these basic matrix operations. For example, an ANN training run may require significant disk I/O to retrieve training patterns and to checkpoint trained weights. Also, the training patterns may need preprocessing before being presented to the network and the network outputs may require post-processing to obtain results. If the neurocomputer is too slow at performing these other non-matrix tasks, then the overall system performance will be dominated by these non-neural components. This result is well known in general-purpose computing as Amdahl's law [1]:

$$S = \frac{1}{(1-f) + f/E}$$

where E is the factor by which performance is improved by some new technique, f is the fraction of the program execution time for which the new technique is applicable, and S is the resulting overall speedup. For example, if 90% of the execution time of a computation is taken by matrix arithmetic, then even an infinitely fast matrix computation engine will never achieve an overall speedup greater than 10. Some neural computer designs have exacerbated this problem by imposing a *slowdown* for non-neural computations by requiring slow communication to a remote host processor to implement the required functionality. Ideally, fast general-purpose computing should be tightly integrated with the special-purpose processing units.

Another related issue is that the existing neurocomputers have been primarily designed to accelerate neural algorithms with dense connectivity (e.g., back-propagation), which can be expressed using dense matrix-vector operations. However, researchers are also interested in algorithms that explore sparse connectivity and sparser activation. These require fast scatter/gather memory operations and support for rapid irregular communications.

Flexible software support is perhaps the most important aspect of a successful neurocomputer design. Most neurocomputers are intended to be programmed using libraries of optimized matrix-vector functions. This approach is adequate for the computation portion of the code provided the libraries are extensive and are easy to compose in arbitrary ways. In particular, difficulties can arise if the libraries place constraints on the location of operands when the machine has multiple distributed memories.

To support experimentation with new ANN models, it is important to provide tools to allow users to extend the libraries to provide missing functionality. Ideally, this would consist of an optimizing high-level language compiler but in practice assembler or microcode programming is usually required. Some of the neurocomputers have extremely complicated microarchitectures which are difficult to program efficiently at this low level. The features which complicate the task of the low level programmer include:

- **Multiple distributed memories.** These require the programmer to carefully position data and to manage movement of data between memories.
- **Deep exposed execution pipelines.** These require the programmer to explicitly schedule operations occurring over many clock cycles.
- **Multiple levels of control flow.** Some systems have several levels of control flow (e.g., controller macroinstructions, microcontroller microinstructions, and nanocontroller nanoinstructions) that must be jointly scheduled for peak performance.

Architectures which expose many details of an implementation to a programmer incur a significant programming overhead. If the same programmer-visible architecture is not preserved in subsequent machines, the software investment in low-level library code cannot be carried forward to new technology.

It is also important to provide libraries for I/O functions as well as computation, as often the amount of code required to manage data input and output dwarfs that required for matrix computation.

4 Discussion

The development of programmable neurocomputers was based on the premise that neurocomputing was significantly different from "general-purpose" computing and thus that a new type of computer architecture was warranted. But the previous sections outlines many concerns shared with conventional computer architectures, namely the need for flexible software development, high-performance library code, reasonable performance on arbitrary code, and fast I/O.

The primary distinguishing characteristics, namely limited numeric precision and large-scale data parallelism, are not only features of ANN algorithms but also many other algorithms in the areas of digital signal processing and multimedia. In recent years, many general purpose microprocessors have added multimedia processing extensions [9] which provide support for data-parallel fixed-point processing. Typically, these multimedia extensions partition an existing 64-bit wide datapath into a short vector of lower precision subword components, e.g., 4×16-bit values, with new instructions that operate on all subword components simultaneously, e.g., adding two vectors of 4×16-bit operands to produce a vector of 4×16-bit results. Microprocessors with multimedia extensions are very similar to the vector or SIMD coprocessor-based neurocomputer architectures, and share the same advantage of a tightly-coupled general-purpose scalar unit.

Although the multimedia extensions implemented to date provide only a limited boost to the performance of general-purpose processors on fixed-point matrix code, they signal an intent by commercial microprocessor manufacturers to perform well on these types of code. As commercial design teams incorporate multimedia-style kernels into the workloads they consider during the design of new microprocessors, we can expect performance to increase rapidly also for ANN algorithms. The continuing tremendous investment placed in high-volume microprocessors ensures that these devices will use the most advanced fabrication technologies and the most aggressive circuit design styles yielding the highest clock rates. Given these trends, there will be greatly reduced interest in future special-purpose neurocomputers.

In attempting to optimize microprocessors for these multimedia codes, microprocessor architects will face many of the same challenges that confronted neurocomputer architects. Perhaps the greatest limitation on performance of highly data-parallel codes is sustainable memory bandwidth. Sustaining high bandwidth to off-chip memory requires both high raw memory bandwidth and the ability to tolerate long memory latencies by overlapping many concurrent memory requests. New off-chip memory architectures and packaging techniques will help improve raw memory bandwidths, but significant improvements in on-chip processor architecture will be required to provide sufficient parallelism to tolerate large off-chip memory latencies. The current multimedia extensions only provide very limited data parallelism of four or eight elements at a time. It is likely that future designs will exploit much longer vectors to increase the level of parallelism supported without incurring additional instruction bandwidth costs. In addition, although current microprocessor multimedia extensions have no support for scatter/gather operations, it is likely these will eventually be added to accelerate the large set of applications that rely on sparse matrix calculations.

A promising future direction is to integrate processor and main memory together on the same die, as in the Berkeley IRAM project [8]. The IRAM project is placing a vector processor similar to T0 on the same die as a large DRAM-based main memory.

The vector processor is a simple hardware scheme for controlling a large degree of parallelism, while the on-chip memory both reduces latencies and dramatically increases available memory bandwidths. The vector unit provides fast scatter/gather operations from multiple on-chip memory banks. The combination should provide high sustained performance at low cost for data-parallel codes, including both dense and sparse ANN algorithms.

References

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, pages 483–485, Montvale, New Jersey, 1967. AFIPS Press.

[2] T. E. Anderson, D. E. Culler, D. A. Patterson, and the NOW team. A case for NOW (Networks of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[3] K. Asanović and N. Morgan. Experimental Determination of Precision Requirements for Back-Propagation Training of Artificial Neural Networks. In *Proceedings 2nd International Conference on Microelectronics for Neural Networks*, Munich, October 1991. Kyrill & Method Verlag.

[4] M. Duranton. Image processing by neural networks. *IEEE Micro*, 16(5):12–19, October 1996.

[5] D. Ellis and N. Morgan. Size matters: An empirical study of neural network training for large vocabulary continuous speech recognition. In *Proceedings International Conference on Acoustics, Speech, and Signal Processing*, Phoenix, Arizona, 1999. IEEE Press.

[6] D. Hammerstrom. A VLSI architecture for High-Performance, Low-Cost, On-Chip Learning. In *Proceedings International Joint Conference on Neural Networks*, pages II–537–543, San Diego, CA, 1990. IEEE Press.

[7] P. Ienne and M. A. Viredaz. Implementation of Kohonen's self-organizing maps on MANTRA-I. In *Proceedings Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, pages 273–279, Turin, Italy, September 1994. IEEE Press.

[8] C. Kozyrakis, S. Perissakis, D. Patterson, T. Anderson, K. Asanović, N. Cardwell, R. Fromm, J. Golbus, B. Gribstad, K. Keeton, R. Thomas, N. Treuhaft, and K. Yelick. Scalable Processors in the Billion-Transistor Era: IRAM. *IEEE Computer*, 30(9):75–78, September 1997.

[9] R. B. Lee and M. D. Smith. Special issue on media processing. *IEEE Micro*, 16(4), August 1996.

[10] R. Means and L. Lisenbee. Extensible linear floating-point SIMD neurocomputer array processor. In *Proceedings of the International Joint Conference on Neural Networks*, pages I–587–592, New York, 1991. IEEE Press.

[11] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, and J. Beer. The Ring Array Processor (RAP): A multiprocessing peripheral for connectionist applications. *Journal of Parallel and Distributed Computing*, 14:248–259, April 1992.

[12] U. A. Muller, B. Baumie, P. Kohler, A. Gunzinger, and W. Guggenbuhl. Achieving supercomputer performance for neural net simulation with an array of digital signal processors. *IEEE Micro*, 12(5):55–64, October 1992.

[13] U. Ramacher, J. Beichter, W. Raab, J. Anlauf, N. Bruls, M. Hachmann, and M. Wesseling. Design of a 1st generation neurocomputer. In *VLSI Design of Neural Networks*. Kluwer Academic, 1991.

[14] R. M. Russel. The CRAY-1 Computer System. *Communications of the ACM*, 21(1):63–72, January 1978.

[15] J. Wawrzynek, K. Asanović, B. Kingsbury, J. Beck, D. Johnson, and N. Morgan. Spert-II: A vector microprocessor system. *IEEE Computer*, 29(3):79–86, March 1996.