PARALLEL NEURAL NETWORK TRAINING ON MULTI-SPERT

PHILIPP FÄRBER AND KRSTE ASANOVIĆ

International Computer Science Institute,
Berkeley, CA 94704

Multi-Spert is a scalable parallel system built from multiple Spert-II nodes which we have constructed to speed error backpropagation neural network training for speech recognition research. We present the Multi-Spert hardware and software architecture, and describe our implementation of two alternative parallelization strategies for the backprop algorithm. We have developed detailed analytic models of the two strategies which allow us to predict performance over a range of network and machine parameters. The models' predictions are validated by measurements for a prototype five node Multi-Spert system. This prototype achieves a neural network training performance of over 530 million connection updates per second (MCUPS) while training a realistic speech application neural network. The model predicts that performance will scale to over 800 MCUPS for eight nodes.

1 Background and Motivation

The Spert-II board is a general purpose workstation accelerator based on the T0 vector microprocessor[1]. Originally, it was designed for the training of large artificial neural networks used in continuous speech recognition. For phoneme classification, we use 3-layer perceptrons with 100–400 input, 200–4000 hidden, and 56–61 output units. The speech databases used in our experiments typically contain millions of patterns, and training requires 6–8 passes through this data to converge. For this task, Spert-II surpasses the performance of today's workstations by factors of 3–10, reducing training time from several weeks to just a few days. To further reduce training time and to make use of even larger volumes of input data, we have developed the Multi-Spert system, which combines several Spert-II boards to form a small-scale multicomputer. Multi-Spert enables us to run large experiments in less than a day.

We train our networks using a variant of the error backpropagation algorithm, and so in this paper we present the parallelization and resulting performance of backprop on Multi-Spert. We concentrate on the two components which dominate the runtime of our training experiments, the *training* routine, which consists of forward pass, error back propagation, and weight update, and the *forward pass* alone, which is used for cross-validation of the training process and in recognition.

2 Multi-Spert Architecture

2.1 Spert-II Board

A Spert-II board comprises a 40 MHz T0 vector microprocessor with 8 MB of memory mounted on a double-slot SBus card[1]. T0 is capable of sustaining 320 million multiply-accumulate operations per second, with 16-bit fixed-point multiplies and 32-bit fixed-point accumulates. The T0 processor has a byte-serial port that allows the host to access T0 memory at a peak bandwidth of 30 MB/s. The Spert-II board contains a field programmable gate array (FPGA) that transparently maps SBus read and write transactions to T0 serial port transactions. The current FPGA design supports data transfer rates of up to 10 MB/s for writes from the host and 4 MB/s for reads.

For a realistic training run using our optimized training code, a single board achieves the performance listed in Table 1. The second column contains results for *online* training, where weights are updated after every pattern presentation. The third and fourth columns contain results for *bunch mode* training, where weights are updated only once after several patterns, or a *bunch*, have been presented.

Bunch mode allows more frequent weight updates than a fully off-line, or *batch mode*, training where weights are updated only once after all patterns have been presented. Compared to the matrix-vector operations used in online training, bunch mode allows use of more efficient matrix-matrix operations[2]. Performance is approximately doubled and reaches over 100 MCUPS. Note that a small bunch size of 12 is sufficient to reap most of the performance benefits. Experimental results show that moderate bunch sizes of about 100 patterns do not significantly change training convergence or accuracy. For larger bunch sizes of several thousand patterns, we've found that the total data set size must be roughly 100 times larger than the bunch size to achieve satisfactory convergence and final classification performance. Practical bunch size can also be limited by other factors, such as buffer memory size and numeric overflow for accumulated errors.

The online training performances presented here are somewhat lower than those we reported earlier for a single board[1]. The routines measured in this paper store weights to 32-bit precision but use only the most significant 16 bits for forward pass and error backpropagation, whereas previously[1] we used 16-bit weights throughout. Although 16-bit weights have frame-level classification equivalent to single precision floating-point, we found that word-level utterance recognition performance suffered in some cases. Training with 32-bit weight updates yields word-level recognition equivalent to single precision floating-point.

MLP Size	Single Spert-II Board		
Input-Hidden-Output	Online	Bunch = 12	Bunch = 96
153-200-56	37	74	72
153-400-56	41	63	69
153-1000-56	45	95	107
153-2000-56	47	102	–

Table 1: Single Spert-II board backprop training performance in MCUPS.

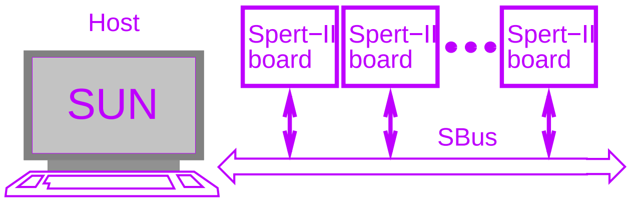

Figure 1: Multi-Spert system.

2.2 Multi-Spert Hardware

To construct a Multi-Spert system, we use commercial SBus expander boxes to enable a moderate number of Spert-II boards to be connected to the same Sun workstation host, as shown in Figure 1. The result is a shared bus master/slave architecture. The Sun host acts as master, controlling all data transfer and synchronization.

Multi-Spert required little additional hardware and system software development over that for Spert-II, but the system has two main limitations. First, SBus only supports transactions with one slave device at a time, so broadcasts from the host have to be repeated to each Spert-II board. Second, a board cannot overlap host communication and T0 computation, so the host must ensure the board's T0 processor is idle before initiating a data transfer.

2.3 Multi-Spert Programming Environment

Our software environment supports any number of Spert-II boards, either plugged into expander boxes or plugged directly into host SBus slots. A Multi-Spert program consists of a host master program, and one or more slave programs, all typically written in C++. The Multi-Spert system software provides routines that allow a master program to allocate a Spert-II board and then to download a slave application executable to the board. Once the slave program

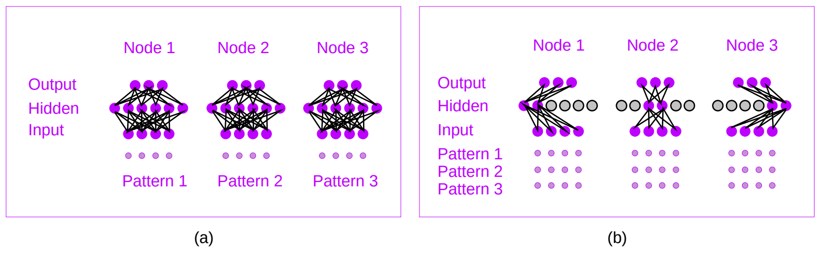

Figure 2: Pattern Parallel (a) and Network Parallel (b) distribution strategies.

is running, the master program can use the C++ *Spert Procedure Call* library to transfer data and parameters and to invoke remote routines on the slave. Slave routines can be called asynchronously, allowing the host to invoke concurrent routines on other slave boards. Synchronization commands allow the host to wait for slaves to complete execution.

3 Parallelizing Error Backpropagation

We have implemented two different strategies for parallelizing backprop training, *pattern parallel* (PP) and *network parallel* (NP).

3.1 Pattern Parallel Training

The PP strategy replicates the entire network on each node and presents different patterns in parallel, as illustrated in Figure 2 (a). The PP version does not impose any constraints on the network topology, as long as the weight updates of different patterns may be accumulated independently and combined. After each bunch of patterns, nodes synchronize by exchanging weight update information. Smaller bunch sizes require more frequent weight communication, but larger bunch sizes can affect training convergence. Because the network is replicated on each board, the size of trainable networks is limited to that which will fit into a single board's memory.

In our PP implementation, we overlap the communication of patterns to one node with computation on other nodes. To reduce the buffer storage required for intermediate values, we only process 24 patterns at a time on each node. Updates are accumulated until the end of the bunch. We overlap reading back weight updates from one node with computation finishing on other nodes. All slaves then wait for the host to broadcast new weight values.

3.2 Network Parallel Training

The NP strategy splits the network across all the boards, as shown in Figure 2 (b). Compared with the PP strategy, the NP strategy has the disadvantage that every pattern must be sent to every board. In our application, we use 3-layer networks with many more hidden units than output units. We take advantage of this in our NP implementation by only communicating the final output activations. Hidden unit computations and all weight updates are entirely local to each node. Because the connection weights are now distributed across the slaves, the maximum network size scales with the number of boards, enabling us to train very large networks.

As in the PP implementation, we overlap communication of patterns to each node with computation on other nodes, but for NP we use a bunch size of 12 patterns. In addition, we overlap the calculation of output errors on the host. When a slave finishes computing the output activations for a bunch, we immediately send the error values for the *previous* bunch along with the input values for the *next* bunch. This means each bunch is calculating errors based on the last but one bunch's weight updates. We found this delayed weight update did not affect convergence.

4 Performance Evaluation

We have developed analytic models that accurately predict the performance of our two strategies for a given number of slave boards. Full details of the models are given in a separate technical report[3]. We validated the performance models with timing measurements made on a Multi-Spert prototype consisting of a Sun Ultra-1/170 workstation host and five Spert-II nodes.

Figure 3 shows model predictions and measurements of training performance for the two strategies. Overall, behaviour of the Multi-Spert system coincides well with the models. As the number of hidden units grows, scalability improves because computation increases relative to communication. For PP, larger bunch sizes reduce overhead and increase performance, as can be seen comparing the two top curves which are for 2000 hidden units with bunch sizes of 5000 and 10000. For NP with a given network size, increasing the number of nodes increases the number of times each pattern is communicated but decreases the size of each node's subnetwork. Performance drops rapidly once communication time overtakes computation time. PP is more efficient for smaller networks, but NP is more efficient for larger networks. The crossover point is at just over 1000 hidden units for a four node system. The model predicts that NP performance should scale to around 600 MCUPS for 3000 hidden units with six nodes.

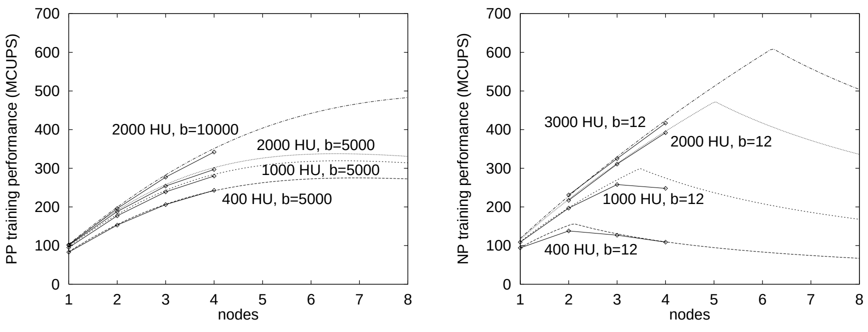

Figure 3: Predicted and measured Multi-Spert training performance. Networks have 153 input units, 56 output units, and varying numbers of hidden units. Bunch sizes are indicated on the graphs.

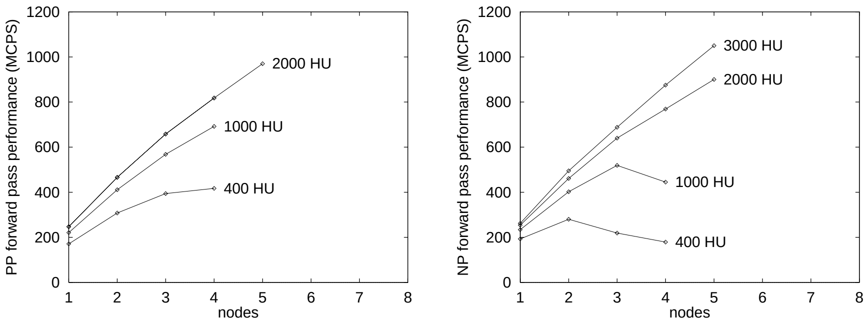

Figure 4: Multi-Spert performance for forward pass. Networks have 153 input units, 56 output units, and varying numbers of hidden units.

Figure 4 shows forward pass performance. Forward pass is mostly communication bandwidth limited. The NP strategy has the disadvantage that all patterns are sent to all nodes, so parallel efficiency is poor on smaller networks. The PP strategy scales much better, since every pattern is sent once only regardless of the number of processors. The one advantage of the NP approach is that it can handle networks that will not fit on a single board using PP. We attain around 1 GCPS with five nodes.

We can take advantage of the structure of the patterns used in the speech application to reduce pattern communication overhead in the NP strategy. Each pattern represents a sliding time window over 9 consecutive frames of spectral features. By keeping the pattern database on the slave nodes, we need only send each frame once and can then reuse it 9 times at different

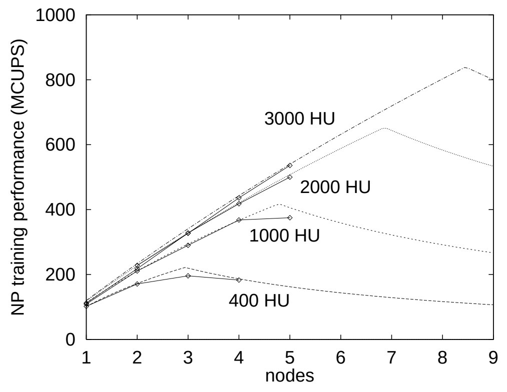

Figure 5: Multi-Spert NP training performance with optimized pattern presentation.

window alignments. Figure 5 shows the predicted and measured performance of this optimization of the NP algorithm. Performance is measured at over 530 MCUPS for 5 nodes. The model predicts that scalability improves to around 800 MCUPS on eight nodes for 3000 hidden units.

5 Related Work

Modern workstations achieve training performances of up to 20–30 MCUPS on this task, but several special-purpose machines achieve performance levels competitive with Multi-Spert. The MUSIC and GF11 systems report training performances of 247 and 901 MCUPS, using 60 and 356 processors, respectively[4]. The full-custom CNAPS with 512 processors on 8 chips achieves over 1 GCUPS peak performance, but using only 16-bit weights throughout. We have not found any published backpropagation performance numbers for the Synapse-1[5].

Compared with these other machines, Multi-Spert is more flexible, lower cost, and easier to program, yet it sustains high performance on parallel neural network training for real-world applications. T0 has a general purpose vector architecture based on the industry standard MIPS instruction set[1], and has been used to implement a wide variety of algorithms. Multi-Spert supports multiple simultaneous jobs each running on a subset of the available nodes; this feature increases system utilization when a variety of network sizes are being trained. The Multi-Spert system is a low-cost system consisting of a few boards plugged into a host workstation. The system is programmed in C++ with a simple master/slave model.

6 Summary and future work

We presented the Multi-Spert system and described two strategies for parallelizing the error backpropagation training algorithm. We presented performance predictions and measurements for neural networks taken from a speech recognition task. The measurements validate our models, and demonstrate over 530 MCUPS performance with five nodes for a 3000 hidden unit network. The model predicts a performance of over 800 MCUPS for eight nodes.

We are working to extend measurements and predictions for larger systems with up to 16 Spert-II nodes. Future work includes improvements to the FPGA SBus interface to increase communication bandwidth. We are also planning to port other applications to Multi-Spert.

Acknowledgments

We would like to thank Jim Beck who designed the Spert-II board and Xilinx interface, John Hauser who wrote the device driver software for the Spert-II board, and David Johnson who wrote the original uniprocessor neural network training code on which the Multi-Spert version was based. Primary support for this work was from ONR URI Grant N00014-92-J-1617, ARPA contract number N0001493-C0249, and NSF Grant No. MIP-9311980. Additional support was provided by the International Computer Science Institute.

References

1. John Wawrzynek, Krste Asanović, Brian Kingsbury, James Beck, David Johnson, and Nelson Morgan. Spert-II: A Vector Microprocessor System. *IEEE Computer*, 29(3):79–86, March 1996.
2. Davide Anguita and Benedict A. Gomes. MBP on T0: mixing floating- and fixed-point formats in BP learning. Technical Report TR-94-038, International Computer Science Institute, August 1994.
3. Philipp Färber. QuickNet: Neural Network Training on Multi-Spert. Technical report, International Computer Science Institute, 1997. To appear.
4. Urs A. Muller, Bernhard Baumie, Peter Kohler, Anton Gunzinger, and Walter Guggenbuhl. Achieving supercomputer performance for neural net simulation with an array of digital signal processors. *IEEE Micro*, 12(5):55–64, October 1992.
5. 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.