Proceedings, Industrial Electronic Seminar 1999(IES'99),
Graha Institut Teknologi Sepuluh Nopember, Surabaya, October 27-28, 1999

Design and Implementation of Parallel Batch-mode Neural Network on Parallel Virtual Machine

Adang Suwandi Ahmad[1, 2], Arief Zulianto[1], Eto Sanjaya[1]
[1]Intelligent System Research Group
[2]Department of Electrical Engineering, Institut Teknologi Bandung
E-mail: asa@ISRG.ITB.ac.id, madzul@ITB.ac.id, eto@ISRG.ITB.ac.id

Abstract

Artificial Neural Network (ANN) computation process is a parallel computation process that should run using parallel processor. Budget constraint drives ANN implementation using sequential processor with sequential programming algorithm. Applying specific algorithm to existing sequential-computers offer emulation of ANN computation processes executes in parallel mode. This method can be applied by building a virtual parallel machine that featuring parallel-processing environment. This machine consist of many sequential machine operated in concurrent mode utilize operating system capability to manage inter-process communication and resource computation process, although this will increase complexity of the implementation of ANN learning algorithm process.

This paper will describe the adaptation and development of sequential algorithm of feedforward learning into parallel programming algorithm on a **virtual parallel machine** based on **PVM** (**Parallel Virtual Machine** [5]) that was developed by **Oak Ridge National Laboratory**. PVM combines UNIX software calls to present a collection of high-level subroutines that allow the user to communicate between processes; synchronize processes; spawn, and kill processes on various machines using message passing construct. These routines are all combined in a user library, linked with user source code, before execution. Some modifications are made to adapt PVM into ITB network environment.

KEYWORDS: artificial neural network; backpropagation; parallel algorithm; PVM;

1 Introduction

Artificial Neural Networks (neural-nets) are used for the successful solution of many real world problems. However, training these networks is difficult and time consuming. One way to improve the performance of training algorithms is to introduce parallelism.

The most important factor has been the unavailability of parallel processing architectures. These architectures are very expensive, thus out of reach of nearly all neural-net researchers. Another limiting factor was the difficulty in porting code between architectures, as a large number of different architectural possibilities existed.

These and other factors have limited the research opportunities. Fortunately, technical and software advances are slowly removing the barriers, allowing access to parallel programming paradigm. An example of improving opportunity for researchers is the availability of mature parallel libraries such as **PVM** (Parallel Virtual Machine [5]). These libraries consist of a collection of macros and subroutines for programming a variety of parallel machines as well as support for combining a number of sequential machines into a large virtual architecture.

Successful parallel algorithms are normally associated with a decrease in execution time when compared with the implementation using sequential algorithm.

2 Conceptual Overview

2.1 Artificial Neural Network

Artificial Neural Network is a computation model that simulates a real biological nervous system. Neural-nets are commonly categorized in terms of their learning processes into:

1. Supervised learning, the training data set consists of many pairs of input-output training patterns.
2. Unsupervised learning, the training data set consist of input training patterns only, the networks learns to adapt based on the experiences collected through the previous training patterns.

Backpropagation (BP) is a well-known and widely used algorithm for training neural-net.

This algorithm involves two phases. During the first phase, an input is presented and propagated forward through the network to compute the output value (o_r) for r output neuron.

$$o_q = f\left(\sum_p w_{qp} o_p\right) \quad (1)$$

with o_q the output activation of each neuron in the hidden layer, w_{qp} the weight connection between the p-th input neuron and the q-th hidden neuron and o_p the p-th feature vector

$$o_r = f\left(\sum_q w_{rq} o_q\right) \quad (2)$$

with o_r the output activation for r-th output neuron, w_{rq} the weight connection between q-th hidden neuron and r-th output neuron, and o_q the hidden layer activation

The output is then compared with target output values, resulting in an error value (d_r) for each output neuron. The second phase consist of a backward pass, where the error signal is propagated from the output layer back to the input layer. The d 's are computed recursively and used as basis for weight changes.

$$\mathrm{d}_r = (1 - o_r)(t_r - o_r) \quad (3)$$

with d_r the r-th output neuron activation's and t_r the target value associated with the feature vector

$$\mathrm{d}_q = o_q(1 - o_q)\sum_r w_{rq}\mathrm{d}_r \quad (4)$$

with d_r the r-th output neuron's d and w_{rq} the weight connection between the q-th hidden layer neuron and r-th output layer neuron

And the gradient between input and hidden layer:

$$\nabla_{pq} = \mathrm{d}_q o_p \quad (5)$$

with d_q the hidden layer's d for the q-th neuron and o_p the p-th input feature vector

$$\nabla_{rq} = \mathrm{d}_r o_q \quad (6)$$

with d_r the output layer's d for the r-th neuron and o_q the q-th input feature vector

2.2 Parallel Processing

A variety of taxonomies uses to classify computer exist. **Flynn**'s taxonomy classifies architectures by the number of instruction and data streams the architecture can process simultaneously [6], [7]. The categories are:

1. **SISD** (Single Instruction Single Data stream);
2. **SIMD** (Single Instruction Multiple Data stream);
3. **MISD** (Multiple Instruction Single Data stream);
4. **MIMD** (Multiple Instruction Multiple Data stream)

For MIMD systems, there are two set models based on intensity level of processors' interaction i.e. tightly coupled architecture, and loosely coupled architecture.

Two different programming paradigms have evolved from the architectural models:

1. shared memory programming using constructs such as semaphores, monitors and buffers (normally associated with tightly coupled systems)
2. message passing programming using explicit message-passing primitives to communicate and synchronize (normally associated with loosely coupled systems)

In performance measurements, ***speedup*** is used as reference in determining the success of a parallel algorithm. Speedup is defined as the ratio between the elapsed time using m processors for the parallel algorithm, and the elapsed time completing the same task using the sequential algorithm with one processor.

The ***granularity*** is the average process size, measured in instructions executed. Granularity does effect the performance of the resulting program.

In most cases overhead associated with communications and synchronization is high relative to execution speed so it is advantageous to have ***coarse granularity*** (typified by long computations consisting of large numbers of instructions between communication points, i.e. high computation-to-communication ratio)

2.3 Possibilities for Parallelization

There are many possibilities method for parallelization [9], such as:

1. map each node to a processor
2. divide up the weight matrix amongst the processors

3. places a copy of entire network on each processor

Map each node to a processor so that the parallel machine becomes a physical model of the network. However, this is impractical for large networks on all but perhaps a massively parallel architecture since the number of nodes (and even nodes per layer) can be significantly greater than the number of processors.

Divide up the weight matrix amongst the processors and allow an appropriate segment of the input vector to be operated on at any particular time. This approach is feasible for an SIMD, shared memory architecture and suggests a data parallel programming paradigm.

Places a copy of the entire network on each processor, allowing full sequential training of the network for a portion of the training set.

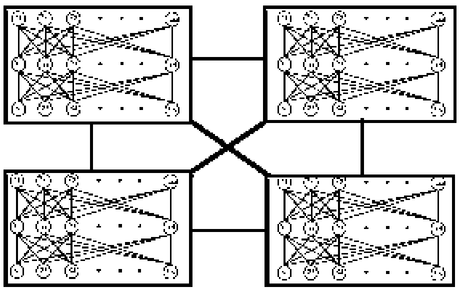


Figure 1. Places a copy of entire network on each processors

The results (i.e. the final weights) are then averaged to give the overall attributes of the network. This would result in near-linear speedup, and could be pushed to greater than linear speedup if the error terms were collected from the feedforwards and utilized for a single backpropagation step. Such a procedure is known as batch updating of the weights. However, as attractive as the potential speedups associated with these methods are, they tend to stray away from true parallelization of the sequential method and also have a tendency to taint the results.

2.4 PVM (Parallel Virtual Machine)

The UNIX operating system provides for inter process communication. Unfortunately, these routines are difficult to use, as available routines are at an extremely low level. PVM (Parallel Virtual Machine [5]), combines these UNIX software calls to present a collection of high level subroutines that allow users to communicate between processes; synchronize processes; spawn, and kill processes on various machines using message passing construct. These routines are all combined in a user library, linked with user source code, before execution.

The high-level subroutines are usable over a wide range of various different architectures, consequently different architectures can be used concurrently to solve a single problem. Therefore PVM usage can resolve huge computational problem by using the aggregate power of many computers.

PVM consists of a daemon process running on each node of virtual machine (*host*). The daemons are responsible for the spawning of tasks on host machines, communication, and synchronization between tasks ordered by the user process using PVM library and software constructs.

The daemons communicate with one another using UDP (User Datagram Protocol) sockets. A reliable datagram delivery service is implemented on top of UDP to ensure datagram delivery. TCP (Transmission Control Protocol) provide reliable stream delivery of data between Ethernet hosts. These sockets are used between the daemon and the local PVM tasks, and also directly between tasks on the same host or different hosts when PVM is *advised* to do so. Normal communication between two tasks on different hosts comprises a task talking to the daemon using TCP. The daemons communicate using TCP, and finally the daemon delivers the message to the task using TCP.

Direct communication task-to-task is possible if PVM *advised* to do so by using specific function in the source code. In this mode, TCP sockets are used for direct communications between various tasks.

The direct communication mode ought to be faster, but prevents *scalability* as the number of sockets available per task is limited by operating system.

3 Implementation

3.1 PVM Implementation,

The main step to implemented PVM in a *live* network is:

- ❑ hosts selection, with consideration in hosts' CPU resource utilisation

- porting the source code to host's architecture

The common problems in utilizing PVM are:

- Varying CPU-resource utilisation of PVM nodes. If one machine is severely loaded compared to other in virtual machine a bottleneck occurs, as PVM provides no dynamic load balancing.
- Varying on network loads. High network utilisation will decrease performance, as communication time will increase as a result of lower bandwidth available to the user.

3.2 Batch-Mode Training Algorithm

Instead of traditionally pattern-mode training, there is batch-mode training in terms of weights updating method. The difference between batch-mode and pattern mode training algorithms is the number of training examples propagated through the net before a weight update. Pattern-mode training algorithms propagate only one training pattern before weight update. Batch-mode algorithms propagate the complete training set before a weight update.

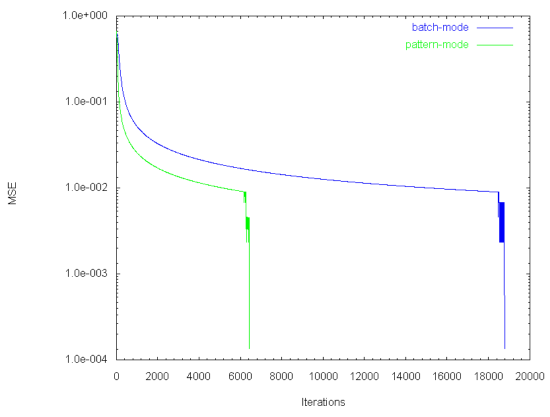


Figure 2. Comparison between batch-mode training and online mode training

In order to obtain a better performance of parallel algorithm, since in a message-passing construct cost of communication is a crucial issue, it is important to have bigger grain-size. For that matter, batch-mode algorithm is better than pattern-mode.

3.3 Parallelizing Scheme

Parallelizing scheme of batch-mode updating implemented with some forwardpasses operated concurrently in some PVM's slaves-process to obtain training errors, and then master-process adjust neural-net's weights using slaves' errors (figure 3.).

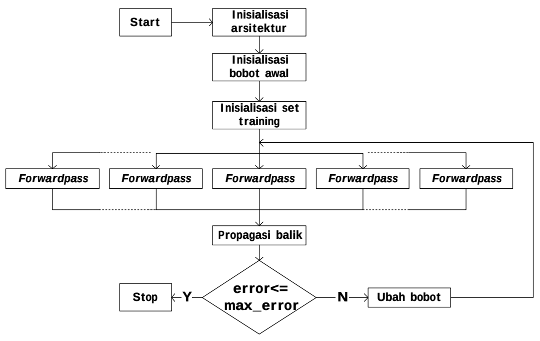


Figure 3. Parallelizing scheme using batch-updating

3.4 Putting it Together

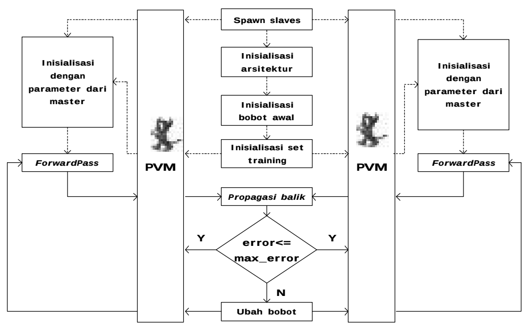


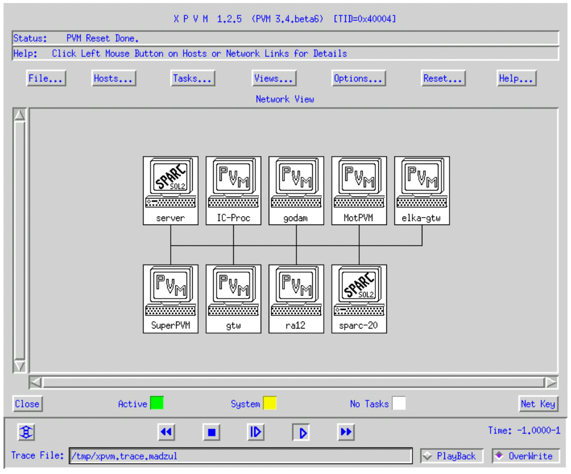
Figure 4. Implementation parallel batch-updating on PVM

Implementation of this parallel algorithm on PVM is based on standard master-slave model. The complete parallel algorithm using batch-mode-updating (figure 4.) is as follows:

1. Master initializes neural-net architecture, initial weights, and training set
2. Master spawns slaves and sends copy of neural-net architecture and initial weight values to each slaves
3. Master sends to each slave its part of the training set
4. Master waits output from slaves
   - Each slave propagates its parts of the training set forward through the net.
   - Each slave return its output to master
   - Master accumulate all partial values
5. Master backpropagate the errors and adapts the weight values
6. The algorithm is repeated from step 3 until the weights converge.
7. Master kill slaves

4 Results

Illustration of PVM on this experiment (figure 1.) depicted a parallel virtual machine consist of heterogeneous (various architectures and operating systems) computers connected in a network.



Status: PVM Reset Done.
Help: Click Left Mouse Button on Hosts or Network Links for Details
Trace File: /tmp/xpvm.trace.madzul

Figure 5. Parallel Virtual Machine

This implementation produces speedup as follows (figure 6.). The amount of speedup obtainable is influenced by:

- number of slaves,
- number of iterations,
- size of training set

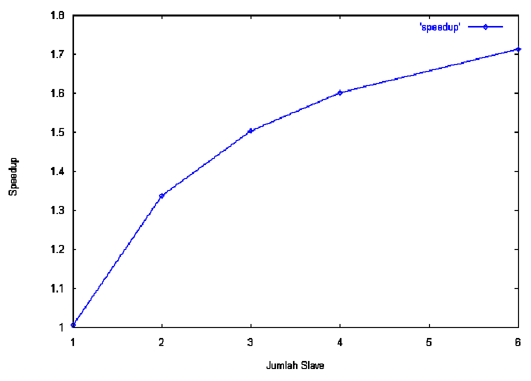


Figure 6. Speedup vs. number of slaves

5 Conclusions

- PVM can be implemented in a *live*-network, with some customizing regarding to resource sharing in multi-user environment, security reasons, and load balancing
- Parallelizing of batch-mode neural-net on PVM gives improved performance over sequential implementations.
- Optimum performance obtained for neural-net with large enough training set, since it will have a bigger grain-size.

References:

[1] Freeman, J.A, and Skapura, D.M, "Neural Networks: Algorithms, Applications, and Programming Techniques", Addison Wesley, 1991

[2] http://www.GNU.org/, "GNU's Not UNIX", The GNU Project and The Free Software Foundation (FSF), Inc., Boston, August 27, 1998

[3] Hunt, Craig, "TCP/IP Network Administration", O'Reilly and Associates, Second Edition, December 1997

[4] Geist, A., A., Begeulin, J., Dongarra, W., Jiang, R., Manchek, and V., Sunderam, "PVM: Parallel Virtual Machine —A Users' Guide and Tutorial for Networked Parallel Computing", Oak Ridge National Laboratory, May 1994

[5] Coetzee L., "Parallel Approaches to Training Feedforward Neural Nets", A Thesis Submitted in Partial Fulfillment for the Degree of Philosophiae Doctor (Engineering), Faculty of Engineering University of Pretoria, February 1996

[6] Lester, Bruce P., "The Art of Parallel Programming", Prentice Hall International Editions, New Jersey, 1993

[7] Hwang, Kai, "Advance Computer Architecture: Parallelism, Scalability and Programmability", McGraw-Hill International Editions, New York, 1985

[8] Purbasari, Ayi, "Studi dan Implementasi Adaptasi Metoda Eliminasi Gauss Parallel Berbantukan Simulator Multipascal", 2:1-39, Tugas Akhir Jurusan Teknik Informatika Institut Teknologi Bandung, 1997

[9] http://www.mines.colorado.edu/students/f/fhood/pllproj/main.htm, "Parallelization of Backpropagation Neural Network Training for Hazardous Waste Detection", Colorado School of Mines, December 12, 1995

[10] http://www.tc.cornell.edu/Edu/Talks/PVM/Basics/, "Basics of PVM Programming", Cornell Theory Center, August 5, 1996