Malaysian Journal of Computer Science, Vol. 15 No. 2, December 2002, pp. 1-16

PARALLEL IMPLEMENTATION ON IMPROVED ERROR SIGNAL OF BACKPROPAGATION ALGORITHM

Teh Noranis Mohd Aris
Department of Computer Science
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
43400 UPM Serdang, Selangor
email: nuranis@fsktm.upm.edu.my

Md. Nasir Sulaiman
Department of Computer Science
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
43400 UPM Serdang, Selangor
email: nasir@fsktm.upm.edu.my

Md. Yazid Mohd Saman
Kolej Universiti Terengganu
Universiti Putra Malaysia
email: yazid@uct.edu.my

Mohamed Othman
Department of Communication Technology and Networking
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
43400 UPM Serdang, Selangor
email: mothman@fsktm.upm.edu.my

ABSTRACT

Improved error signal of the backpropagation (BP) algorithm on single processors has shown a tremendous result compared to its counterpart [1]. Further study on the improved BP algorithm is carried out on many processors, which is implemented using the Sequent Symmetry SE30 machine. Data partitioning method, with columnwise block striped and batch mode weight updating strategy, is applied on the BP algorithms. Twenty-six patterns consisting of uppercase letters from 'A' to 'Z' are tested in terms of speed and recognition rates. The parallel version of the BP algorithm produces good speedup as the numbers of processors are increased and a 100% recognition rate for trained and untrained data is achieved.

Keywords: ***Standard BP, Improved BP, Mean Squared Error (MSE), Speedup, Data Partitioning, Columnwise Block striped, Batch Mode Weight Updating***

1.0 INTRODUCTION

Execution speed is very important in computational processing. A fast computational process is considered as an effective and efficient system. Scientific and engineering problems deal with large volumes of data. These data can be processed faster using parallel computers.

Neural network (NN) is one of the artificial intelligence areas that attempts to imitate the computational power of the human brain [2]. Basically, NNs are mathematical models of information processing. NN has the ability to perform numerous learning tasks such as recognition, feature extraction, statistical clustering and prediction. BP is one of the popular NN algorithms [2]. The BP training process is time consuming. The two approaches used to speed up the BP training process are improving the BP algorithm and implementing the BP algorithm using parallel machines.

Shamsuddin [1, 3] has proposed an improved error signal for the BP algorithm running on a single CPU system. A modified error function has been generated to increase the convergence rates of the BP training, replaced by the MSE used in standard BP. The epoch size of the improved BP is less than the epoch size of the standard BP. As a result, the execution time of the improved BP is much faster than the standard BP. However, as the input data sets become larger, the execution time of the improved BP is slower. In this study, the improved BP is implemented on parallel processors to produce a much faster version of the BP algorithm.

1.1 Standard BP and Improved BP

BP is a multilayer feed forward network. It is also known as the gradient descent method to minimise the MSE of the output computed by the network. Typically, the network consists of an input layer, one or more hidden layers of computation nodes, and an output layer of computation nodes as illustrated in Fig. 1.

BP is a type of supervised learning network or is also known as learning with a teacher. In supervised learning, the process of adjusting the weights in a NN is given a learning algorithm to follow. The target output for each training set of input vectors is presented to the network. This process involves many cycles through the training data. Besides, unsupervised learning is a type of learning that takes place without a teacher. In supervised learning network, a learning algorithm may be given but target outputs are not.

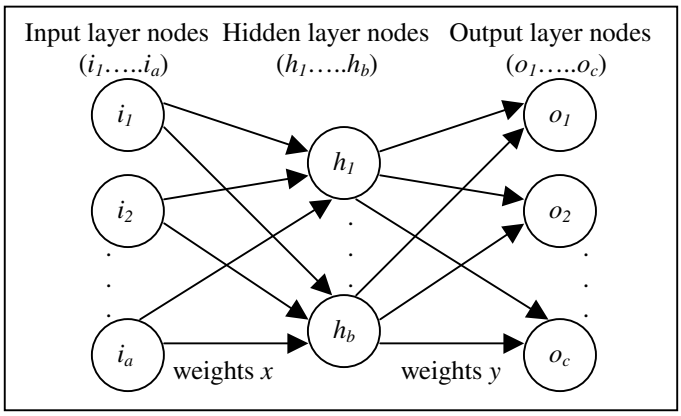

Fig. 1: BP Architecture

The BP training algorithm is as follows [1]:

Step 1: Set up the BP network by defining the number of input, hidden and output nodes.

Step 2: Read the input pattern file and the target file.

Step 3: Randomise weights and biases.

Step 4: Calculate activation function for the hidden and output layer by using the following formula:

$$O_j = f\left(\sum_j w_{kj} o_i - \theta_j\right)$$

where, w_{kj} is the weight from output O_i, θ_j is the bias and f is the sigmoid function.

Step 5: Calculate the weight correction term Δw_{ji} from unit *j* to unit *i* using:

$$\Delta w_{ji} = \alpha \delta_j o_i$$

where, α is the learning rate and δ_j is the error gradient at unit *j*.

Update the weights from output to hidden layer using the following formula:

$$w_{ji}(t) = w_{ji}(t-1) + \Delta w_{ji}$$

where, $w_{ji}(t)$ is the weight from unit *j* to *i* at iteration *t*.

If a momentum term is added, the above equations change as follows:

$$w_{ji}(t+1) = w_{ji}(t) + \frac{\alpha_j \partial E(w)}{\partial w_{ji}} + \beta \Delta w_{ji}(t)$$

where,

$w_{ji}(t+1)$ is the update weight from node *j* to node *i* at time *t,*

$w_{ji}(t)$ is the weight from node *j* to node *i* at time *t*,

$$\frac{\partial E(w)}{\partial w_{ji}} = \sum \delta_j o_j ,$$

$\delta_j = f'(net_j)(t_j - o_j)$ is the error signal for the output node,

$o_j = f'(net_j)\sum w_{ji}\delta_{ji}$ is the output layer output,

$\Delta w_{ji}(t) = w_{ji}(t) - w_{ji}(t-1)$ is the weight change at time *t*, β is the momentum term,

α_k, ($k = 1, 2, \ldots, n$) is the learning rate and depend on $\left|\frac{\partial E(w)}{\partial w_{ji}}\right|$.

Step 6: Calculate the error term for the output layer as follows:

$$\delta_k = o_k(1-o_k)(t_k - o_k)$$

where o_k is the actual output and t_k is the target output unit *k*.

Calculate the error term for the hidden layer as follows:

$$\delta_k = o_j(1-o_j)\sum_k \delta_k w_{kj}$$

where δ_j is the error at unit *k* from hidden unit *j*.

Step 7: Compute the error function using the Mean Squared Error (MSE) formula:

$$E = \frac{1}{2}\sum_k (t_{kj} - o_{kj})^2$$

where *E* is the error function
t_{kj} is the target output from node *k* to node *j,*
o_{kj} is the network output from node *k* to node *j.*
Test if *E* < error tolerance specified then stop the training process
else repeat steps 4 to 7.

The improved BP is produced, by modifying an error function of [4] to increase the convergence rates of the BP training as:

$$mm = \sum \rho_k$$

with

$$\rho_k = \frac{E_k^2}{2a_k(1-a_k^2)}$$

where

$$E_k = t_k - a_k$$

and

E_k error at output unit *k,*

t_k target value of output unit *k*,

a_k an activation of unit *k.*

By taking derivatives for the updating weight using chain rule, the improved BP of the output layer is produced as:

$$\delta_k = \frac{2(E + \rho(1-3a_k^2))}{1+a_k}$$

and an error signal for the improved BP of the hidden layer is the same as the standard BP,

$$\delta_j = \sum \delta_k w_j \sigma'(a_j)$$

where

w_j weight on connection between unit,

$\sigma'(a_j)$ a sigmoid function of $\frac{1}{1+e^{-2x}}$.

1.2 Sequent Symmetry SE30 Architecture

Computer technology is becoming more and more sophisticated. Many scientific and mathematical applications require large volume of data. As the volume of data is becoming larger, the computing power of conventional sequential computers is being improved. Technology has produced faster VLSI sequential processors, which are combined together in multiprocessor computers to increase the computing power. Parallel processors are actually sequential processors, which work together in the same computer system [5]. The Sequent Symmetry SE30, which is used in this research, is an example of such multiprocessor system. The Sequent Symmetry SE30 is a type of Multiple Instruction Stream, Multiple Data Stream (MIMD) shared memory multiprocessor or tightly coupled machines, shown in Fig. 2. MIMD processors have its own control unit, local memory and arithmetic and logic unit. All the processors execute different programs while solving different sub-problems of a single problem. This means that the processors operate asynchronously. The processors access the shared memory through a common bus structure.

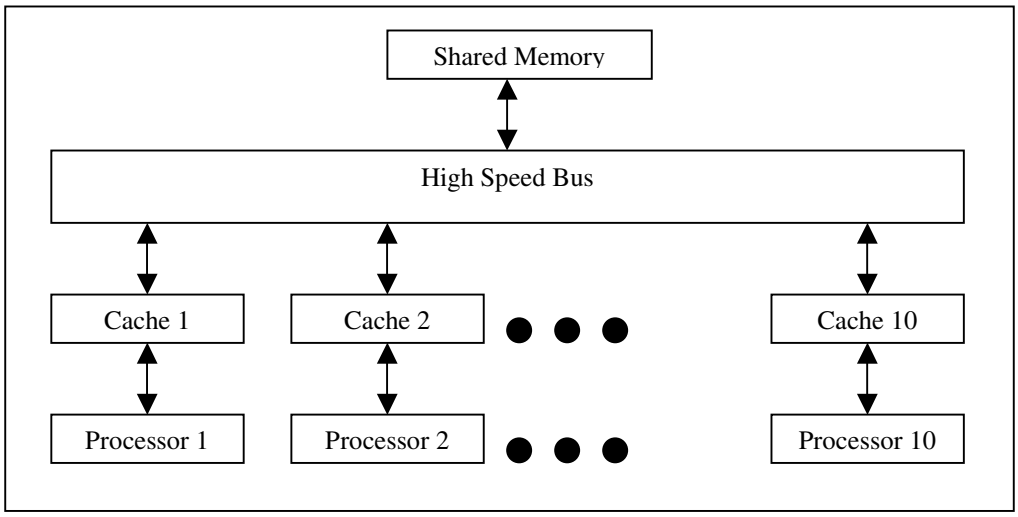

Fig. 2: Sequent Shared Memory Architecture

The hardware configurations of the Sequent Symmetry SE30 parallel machine are as follows [6]:

- 10 processors each with 100 Mhz P54C Intel Pentium microprocessors.
- 1 high speed bus connected between cache memory and shared memory with peak data rates of 240 MB per seconds.
- Support up to 4 VMEbus subsystem.
- 10 secondary cache memory, each with 2 MB.
- Support up to 441 SCSI devices.
- 64 MB RAM expandable to 3.5 GB.
- 2.1 GB Disk Storage expandable to 940 GB.

The following are the software that is supported by Sequent Symmetry parallel machine [6]:

- DYNIX/ptx® operating system version 4.4.5 that supports utilities, library and system calls.
- UNIX system V Release 4.0 i384.
- C programming language and Sequent parallel library: microtask.h and parallel.h.

The main purpose of parallel processing is to perform computations faster than can be done with a single processor by using a number of processors concurrently [7].

2.0 DATA REPRESENTATION

There are two modes of operation in the NN simulation: training and test mode. The trained data use the training mode and the test mode. Training data consists of complete input pattern. Untrained data use the test mode. In this research, untrained data consists of patterns, which are 10% corrupted. The trained and untrained data are represented in binary 0's and 1's.

The following is an example of a trained data sample of the letter 'A':

```
0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0
0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
```

The following is an example of an untrained data sample of the letter 'A':

```
0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 1 1 0 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 1 1 0 1 0 1 1 0 1 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 0 0 0 0
0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1
1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1
0 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 0
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
```

Each of the twenty-six uppercase letters tested is represented by a target output. The following is an example of an output representation of the letter 'A':

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
```

3.0 METHODOLOGY

This research combines the technique of improving the improved BP algorithm and implementing the algorithm on parallel processors to produce a faster speedup.

An important characteristic, which affects the design of parallel NN algorithm, is the weight updating strategy. Basically, there are three parallel NN strategies used, which is the pattern (on-line), batch and block mode strategy described as follows [8]:

i. Pattern mode strategy: weight updating is done after the presentation of each training pattern.
ii. Batch mode strategy: weight updating is performed after the presentation of all the training patterns.
iii. Block mode strategy: weight updating is performed after the presentation of a subset of the training patterns.

In this research, the batch strategy is adopted in the standard BP and improved BP. The batch mode is used because it is well suited for parallelisation due to weight errors that can be calculated independently.

Training set parallelism is applied for this BP multilayer NN paradigm. In training set parallelism, the training set is partitioned into several subsets, while the whole NN is duplicated on each processor. Each processor works with a subset of the training set as shown in Fig. 3.

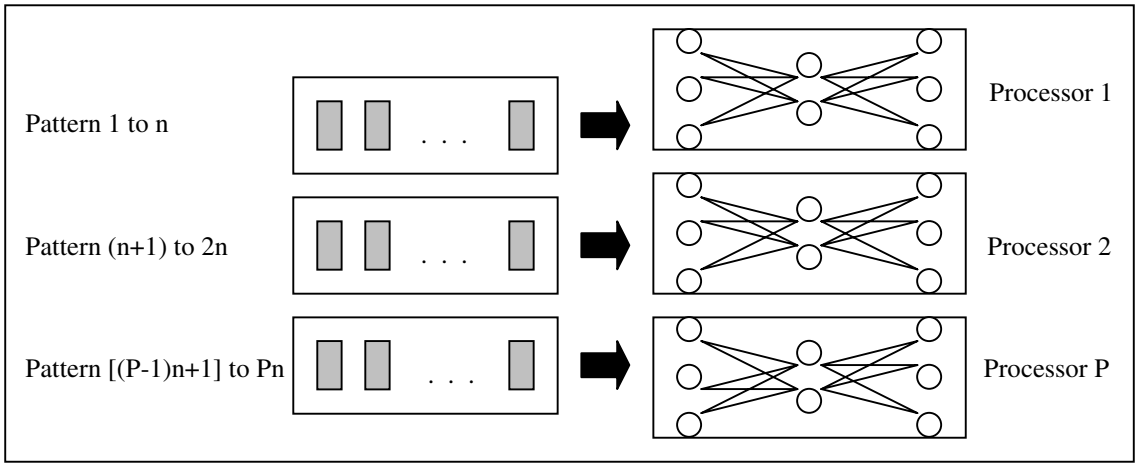

Fig. 3: Training Set Parallelism

4.0 IMPLEMENTATION AND RESULTS

The improved BP algorithm is parallelised using the columnwise block striped partitioning and coded using C programming language. Comparison is made between the performance of standard BP and improved BP using different numbers of hidden units. Comparison is also made in terms of number of cycles, which is fixed and also complete cycles.

4.1 Parallel Algorithm

Parallel routines are implemented in the BP program. The m_set_procs routine is used to set a number of child processes that will be used to execute a subprogram in parallel. The usclk_init is the Sequent system clock. Before a subprogram can be executed in parallel new processes must be created first. The m_fork routine is used to create new processes. The time is captured from the beginning of the BP training to the end of the BP training for each pattern. After all the processes have completed the parallel execution, the m_kill_procs is used to kill the child processes.

When m_fork routine is called, it initialises the shared memory and ensures that variables declared with keywords such as shared and private in C are mapped to the appropriate memory area. In Fig. 4, the BP_Training subprogram is forked and will be executed in parallel. Fig. 5 presents the portion of BP_Training subprogram. This figure illustrates the columnwise block striped partitioning, the forward propagation and BP of error using either MSE or improved BP formula, which is calculated in parallel.

Fig. 6 illustrates the declaration statements of the shared variables. The variables are declared as shared variables so that all the processes executed in parallel can access the same memory location for that variable.

```
/* BP Main Program */
main()
{
.
.
.
printf("Enter Number of Processors: ");
scanf("%d",&nprocs);
printf("\n");
m_set_procs(nprocs); /* set number of processes */

for (i = 0; errorfunc > ErrorFunc ;)  /* execute until */
                                      /* < error tolerance */
  {
   for(j = 0; j < MaxPatternNo ; j++) /* process the patterns */
     {
              usclk_init();
          timer=getusclk(); /* starting time */
               m_fork(BP_Training,j); /* execute  a subprogram */
                                      /* in parallel */
              timer=getusclk() - timer; /* ending time */
       total_time += (timer*0.000001); /* convert time to seconds */
      }
.
.
.
   }
m_kill_procs();
.
.
.
}
```

Fig 4: BP Main Program

```
/* BP Training Procedure */

void BP_Training(int p)
{
.
.
.
  nprocs=m_get_numprocs(); /* get number of processes */

/* columnwise block striped partitioning */

  bs1 = HiddenUnitNo/nprocs; /* block size for hidden units */
  if (HiddenUnitNo%nprocs != 0)
    bs1 += 1;
  bb1 = m_get_myid() * bs1; /* block begin for hidden units */
  be1 = bb1 + bs1 - 1; /* block end for hidden units */

  bs2 = OutputUnitNo/nprocs; /* block size for output units */
  if (OutputUnitNo%nprocs != 0)
    bs2 += 1;
  bb2 = m_get_myid() * bs2; /* block begin for output units */
  be2 = bb2 + bs2 - 1; /* block end for output units */

  bs3 = InputUnitNo/nprocs; /* block size for input units */
  if (InputUnitNo%nprocs != 0)
     bs3 += 1;
  bb3 = m_get_myid() * bs3; /* block begin for input units */
  be3 = bb3 + bs3 - 1; /* block end for input units */
```

Fig. 5: BP_Training Subprogram

```
/* forward propagation */

for (i=bb1; (i<=be1&&i<HiddenUnitNo); i++)
   {
    o2[i] = calculate_in_output(p,i,w21,o1,bias2);
   }

m_sync();

for (i=bb2; (i<=be2&&i<OutputUnitNo); i++)
   {
    o3[i] = calculate_mid_output(i,w32,o2,bias3);
   }

m_sync();

/* Improved BP error */

for (i=bb2; (i<=be2&&i<OutputUnitNo); i++)
d3[i]=((t[p][i] - o3[i]) + (pow(t[p][i]-o3[i],2.0)/((2.0 * f(o3[i])) *
(1.0-pow(f(o3[i]),2.0)))) * (1.0-(3.0 *
pow(f(o3[i]),2.0))))/(1+f(o3[i]));

/* d3[i]=(t[p][i]-o3[i]) * o3[i] * (1-o3[i]); MSE */

m_sync();

for(i=bb1; (i<=be1&&i<HiddenUnitNo); i++)
   {
    d2[i]=calculate_sum(i,dw32,w32,d3,o2);
   }

m_sync();

for (i=bb2; (i<=be2&&i<OutputUnitNo); i++)
   {
    dbias2[i] = Eta * d3[i] + Alpha * dbias3[i];
    bias3[i] += dbias3[i];
   }

m_sync();

for(i=bb3; (i<=be3&&i<InputUnitNo); i++)
    for(j=0; j<HiddenUnitNo; j++)
       {
        dw21[j][i] = Eta * d2[j] * o1[p][i] + Alpha * dw21[j][i];
        w21[j][i] += dw21[j][i];
       }

m_sync();

for(i=bb1; (i<=be1&&i<HiddenUnitNo); i++)
   {
    dbias2[i] = Eta * d2[i] + Alpha * dbias2[i];
    bias2[i] += dbias2[i];
    }
m_sync();
}
```

Fig. 5 (Cont.): BP_Training Subprogram

```
/* Output, weight and bias declaration */
shared double  **o1;    /* Output for Input layer - i */
shared double  *o2;     /* Output for Hidden layer - j */
shared double  *o3;     /* Output for Output layer - k */
shared double  **t;     /* Target Output */
shared double  **w21;  * Weights from layer i to layer j  */
shared double  **dw21; /* Change of above weights */
shared double  **w32;  /* Weights from layer j to layer k */
shared double  **dw32; /* Change of above weights */
shared double  *bias2;   /* Bias for layer j node */
shared double  *dbias2; /* Change of above bias */
shared double  *bias3;   /* Bias for layer layer k node */
shared double  *dbias3; /* Change of above bias */
shared double  *d2;    /* error signal for the hidden layer. */
shared double  *d3;   /* Change of above error signal */
```

Fig. 6: Shared Variables

The values of the parameters used in standard BP and improved BP are as follows:

$\alpha = 0.01,\ \beta = 0.9,\ e = 0.05,$

where α is the learning rate,
β is the momentum,
e is the maximum error.

The network is considered well trained, where a 100% accuracy is produced, if the maximum error function is $e <= 0.05$.

The combination value of $\alpha = 0.01$ and $\beta = 0.9$ is used because by experiments, these values produce a globally optimum weights to a desired accuracy resulting in a decreasing error. Theoretically, in NN the objective is to achieve the combination of α and β values that on average yield convergence to the network configuration that has the best generalisation over the entire input space, with the least number of epochs [9].

The experiments are done, by using different number of hidden units. The following fixed values are used for input and output units:

number of input units = 400,
number of output units = 26

The input patterns are represented in binary 0's and 1's explained earlier, consisting of 400 units. The target values for the patterns have 26 units, which represents the twenty-six uppercase letters.

4.2 Results of Execution Time and Speedup

Table 1 illustrates the execution times of standard BP and improved BP running sequentially and in parallel with different number of processors based on 10 cycles using 100 hidden units. The execution time of standard BP and improved BP running sequentially are 37.79 seconds and 19.26 seconds. The result indicate that the execution time of the improved BP is faster than the standard BP. When the same programs are applied using 8 processors, the execution time is improved much better which results in 7.54 seconds for standard BP and 6.43 seconds for improved BP. The standard BP speedup is 1.73 whereas the improved BP speedup is 1.52 using 2 processors. Applying 8 processors results in a speedup of 5.42 for standard BP and 3.79 for improved BP. The execution times of standard BP and improved BP versus the number of processors are plotted in Fig. 7. The speedup values versus the number of processors are plotted in Fig. 8.

Table 2 illustrates the execution times of standard BP and improved BP based on 10 cycles and 200 hidden units, running sequentially and in parallel. The execution time of standard BP and improved BP both running sequentially are 86.27 seconds and 51.44 seconds. Using 8 processors reduces the execution time: 14.10 seconds for standard BP and 12.51 seconds for improved BP. There is an increase of the execution time as the number of hidden units is increased from 100 hidden units to 200 hidden units. The speedup of standard BP and improved BP using 2 processors are 1.95 and 1.68. The speedup is improved by applying 8 processors: 6.84 for standard BP and 4.35 for

improved BP. The execution time values are plotted in Fig. 9. The speedup values for standard BP and improved BP are plotted in Fig. 10.

Table 1: Execution Times (in seconds) and Speedup Values for Standard BP and Improved BP based on 10 Cycles using 100 Hidden Units

# of Processors	Standard BP Execution Time	Improved BP Execution Time	Standard BP Speedup	Improved BP Speedup
Sequential	37.79	19.26		
1	40.91	24.39	1.00	1.00
2	23.56	15.96	1.73	1.52
3	16.81	11.76	2.43	2.07
4	13.18	9.39	3.10	2.59
5	10.58	8.17	3.86	2.98
6	9.09	7.25	4.49	3.36
7	8.33	6.97	4.90	3.49
8	7.54	6.43	5.42	3.79

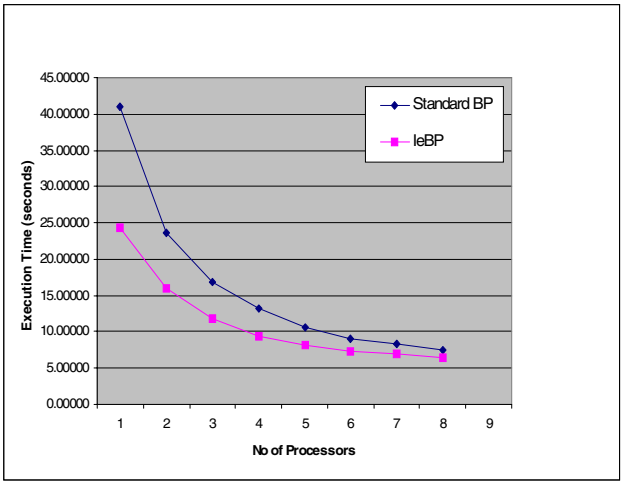

Fig. 7: Standard BP and Improved BP Execution Times (in seconds) with 10 Cycles using 100 Hidden Units

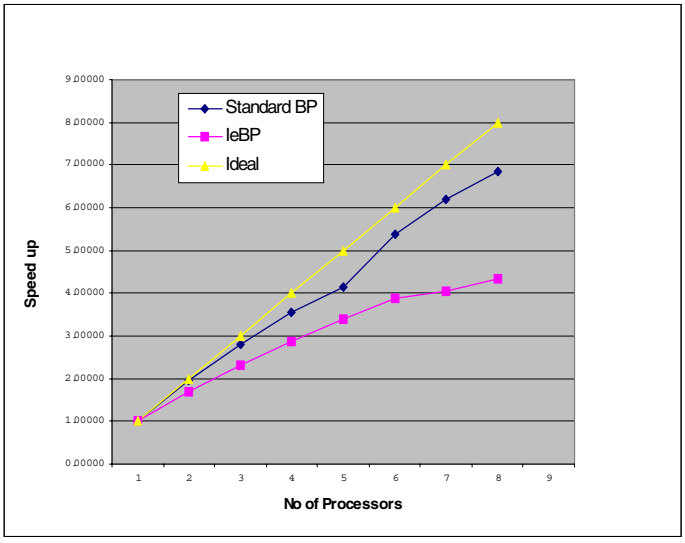

Fig. 8: Standard BP and Improved BP Speedup values with 10 Cycles using 100 Hidden Units

Table 2: Execution Times (in seconds) and Speedup Values for Standard BP and Improved BP based on 10 Cycles using 200 Hidden Units

# of Processors	Standard BP Execution Time	Improved BP Execution Time	Standard BP Speedup	Improved BP Speedup
Sequential	86.27	51.44		
1	96.49	54.48	1.00	1.00
2	49.46	32.31	1.95	1.68
3	34.60	23.47	2.78	2.32
4	27.12	19.09	3.55	2.85
5	23.29	16.02	4.14	3.40
6	17.92	14.02	5.38	3.88
7	15.54	13.46	6.20	4.04
8	14.10	12.51	6.84	4.35

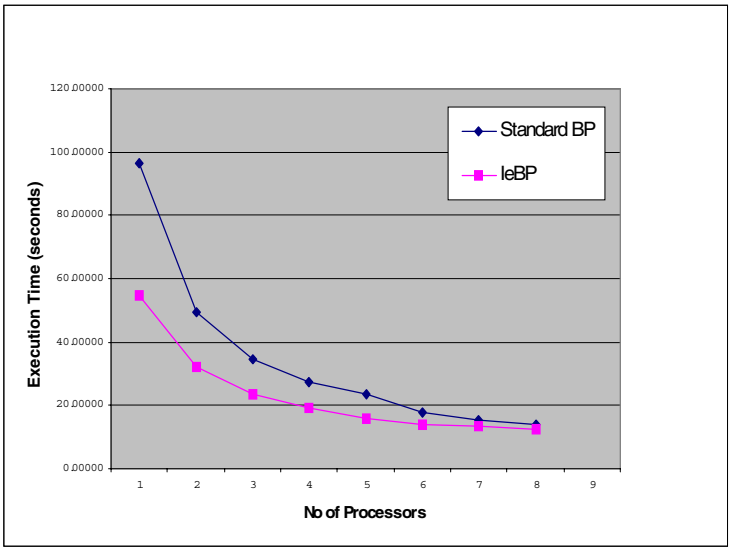

Fig. 9: Standard BP and Improved BP Execution Times (in seconds) with 10 Cycles using 200 Hidden Units

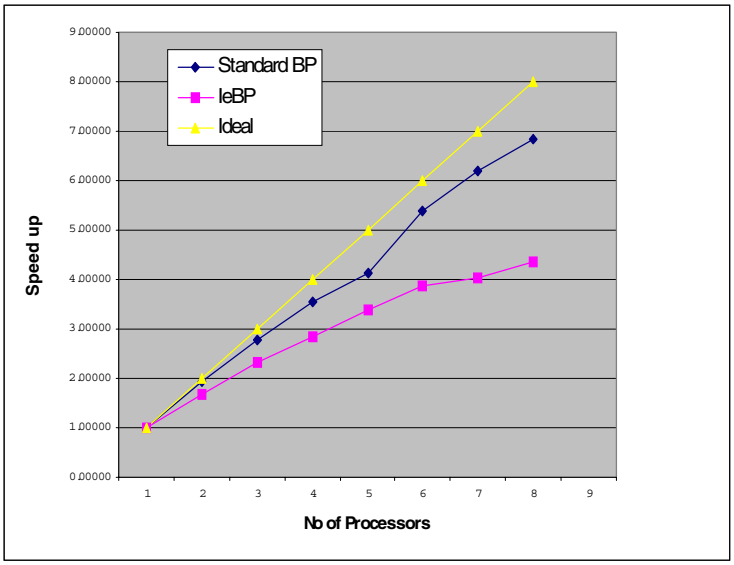

Fig. 10: Standard BP and Improved BP Speedup Values with 10 Cycles using 200 Hidden Units

Table 3 represents the execution times of standard BP and improved BP based on 10 cycles and tested using 300 hidden units. As the number of hidden units are increased, the execution time of standard BP and improved BP running sequentially are also increased: 147.57 seconds for standard BP and 90.39 seconds for improved BP. The execution time, are improved by applying 8 processors: 21.54 seconds for standard BP and 19.45 seconds for improved BP. The speedup for standard BP using 2 processors is 1.90 and is improved by using 8 processors, which results in 6.87. The improved BP running on 2 processors is 1.75 and running on 8 processors is 4.71. The execution time values are plotted in Fig. 11. The speedup values for the standard BP and improved BP are plotted in Fig. 12.

Table 3: Execution Times (in seconds) and Speedup Values for Standard BP and Improved BP based on 10 Cycles using 300 Hidden Units

# of Processors	Standard BP Execution Time	Improved BP Execution Time	Standard BP Speedup	Improved BP Speedup
Sequential	147.57	90.39		
1	148.19	91.73	1.00	1.00
2	77.89	52.19	1.90	1.75
3	51.41	36.32	2.88	2.52
4	39.15	29.19	3.78	3.14
5	32.17	25.97	4.60	3.53
6	26.89	22.86	5.51	4.01
7	23.31	20.14	6.35	4.55
8	21.54	19.45	6.87	4.71

Note: Speedup, $S_p = \frac{T_1}{T_p}$, where T_1 is the execution time for one processor,

T_p is the execution time for p processors.

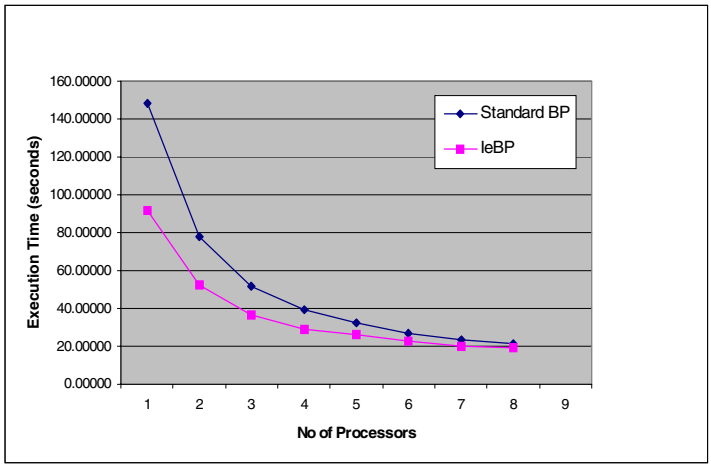

Fig. 11: Standard BP and Improved BP Execution Times (in seconds) with 10 Cycles using 300 Hidden Units

Table 4 and Table 5 illustrates the execution times and speedup values of improved BP based on network convergence using 100 and 300 hidden units. The complete cycle of the BP network to converge for 100 hidden units produced only 152 cycles. Whereas, by increasing the number of hidden units, the BP network converges after completing 276 cycles. The execution time and speedup values are plotted in Fig. 13 and Fig. 14.

Table 6 compares the number of cycles and execution times of improved BP and standard BP running sequentially. Improved BP only produced 152 cycles compared to standard BP, which is 301 cycles. As the number of hidden units are increased to 300, the cycles and execution time for improved BP and standard BP are also increased.

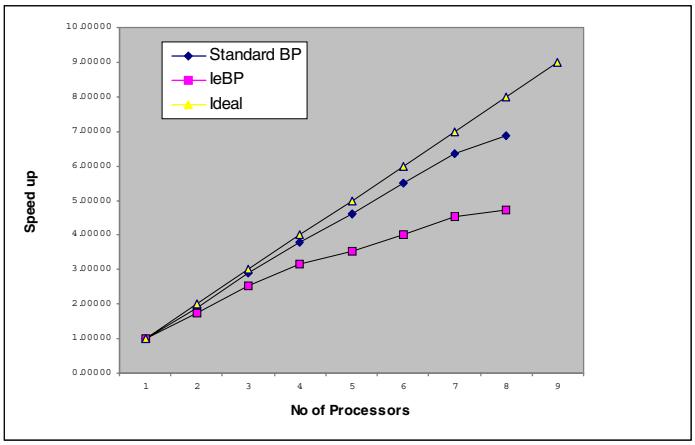

Fig. 12: Standard BP and Improved BP Speedup values with 10 Cycles using 300 Hidden Units

Table 4: Execution Times (seconds) and Speedup Values for Improved BP Based on Network Convergence with cycles = 152 and error = 0.05 using 100 Hidden Units

# of processors	Execution Time (seconds)	Speedup
Sequential	89.28	
1	92.76	1.00
2	56.56	1.64
3	41.98	2.20
4	33.94	2.73
5	28.39	3.26
6	24.98	3.71
7	23.11	4.01
8	22.22	4.17

Table 5: Execution Times (seconds) and Speedup Values for Improved BP Based on Network Convergence with cycles = 276 and error = 0.05 using 300 Hidden Units

# of processors	Execution Time (seconds)	Speedup
Sequential	184.85	
1	189.38	1.00
2	99.83	1.89
3	71.63	2.64
4	56.12	3.37
5	44.21	4.28
6	39.43	4.80
7	38.81	4.87
8	35.28	5.36

From the experimental results, it was found that the speedup values for, the standard BP increases almost ideally as the number of processors, are increased. The speedup for improved BP also increases almost ideally but less compared to the standard BP (refer to Fig. 8, Fig. 10 and Fig. 12 for the standard BP and improved BP speedup, which are based on ten cycles). This is because improved BP converges faster than the standard BP. The error function derived by Shamsuddin [1] causes the improved BP to converge fast. However, the execution time for improved BP is less than the execution time for standard BP (refer to Table 1, Table 2 and Table 3). The performance of improved BP in Table 4 and Table 5, which are based on complete convergence shows a good speedup as the number of processors are increased. The efficiency and effectiveness of the improved BP is illustrated in Table 6, where the number of cycles and execution time is less than the standard BP. As the number of hidden units is increased, the execution time also increases.

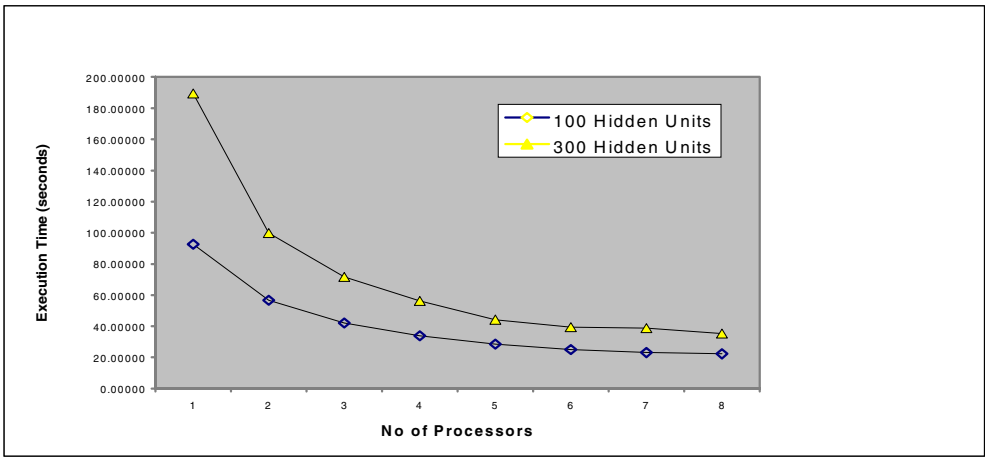

Fig. 13: Improved BP Execution Times (in seconds) based on Network Convergence using 100 and 300 Hidden Units

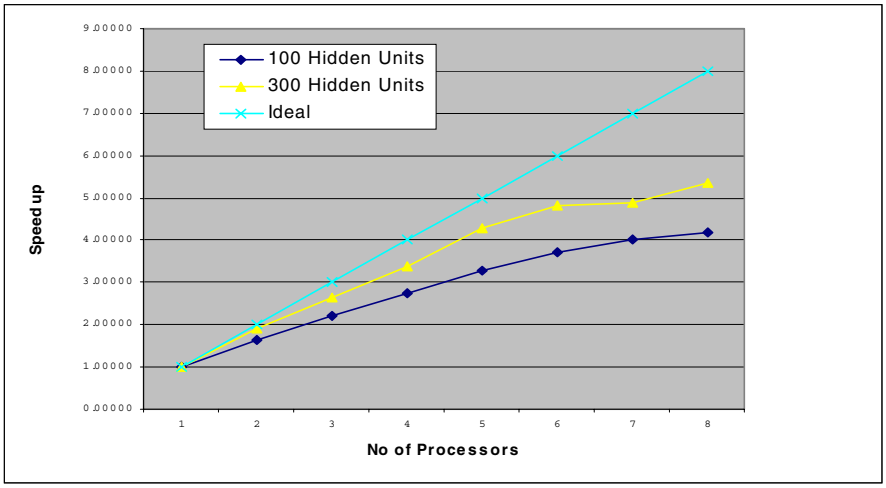

Fig. 14: Improved BP Speedup Values based on Network Convergence using 100 and 300 Hidden Units

Table 6: Comparison of Improved BP and Standard BP in terms of number of Cycles and Execution Time

Hidden Units	Improved BP		Standard BP	
	Cycles	Execution Time	Cycles	Execution Time
100	152	89.28	301	177.85
300	276	184.85	497	358.09

4.3 Results of Recognition Rates

The effectiveness of the BP network is measured in terms of percent of recognition rates for the trained and untrained data. From the experimental results done on the Sequent Symmetry machine, recognition rates for the standard BP and improved BP both produce 100% accuracy for the trained and untrained data using 33 hidden units and the same parameter values as explained earlier. The recognition rate is measured in terms of the output produced by the BP network. The output is in the range between, 0 to 1. The output, which is produced from the calculation of the weights and biases in the BP training, has a strong strength if the output is greater or equal to 0.5 [10]. The output results of trained data for the letter 'A', is shown in Fig. 15.

```
o3[0] = 0.00
o3[1] = 0.00
o3[2] = 0.00
o3[3] = 0.00
o3[4] = 0.00
o3[5] = 0.00
o3[6] = 0.00
o3[7] = 0.00
o3[8] = 0.00
o3[9] = 0.00
o3[10] = 0.00
o3[11] = 0.00
o3[12] = 0.00
o3[13] = 0.00
o3[14] = 0.00
o3[15] = 0.00
o3[16] = 0.00
o3[17] = 0.00
o3[18] = 0.00
o3[19] = 0.00
o3[20] = 0.00
o3[21] = 0.00
o3[22] = 0.00
o3[23] = 0.00
o3[24] = 0.00
o3[25] = 0.99
```

Fig. 15: Letter 'A' Output Results

5.0 DISCUSSION

From the experimental results carried out, the speedup of standard BP and improved BP increase almost ideally. The standard BP based on ten cycles produces a slightly better speedup because the speedup line is nearer the ideal line compared to improved BP. This is due to the improved BP that converges faster. However the execution time of improved BP is less than standard BP. The number of complete cycles produced by improved BP is much more, less compared to standard BP. Most important, improved BP running in parallel produces better speedup as the number of processors, are increased. This proves the efficiency of parallel computing and leads to the contribution of the research work. Besides the speedup improvement, the recognition rates of the trained and untrained pattern also produces 100% accuracy. Speedup and accuracy are two important factors in developing a NN application, that has achieved in this research work.

6.0 CONCLUSION

The combination to improve the BP algorithm and implementing it using parallel processors produces good results in two ways. First, by improving the BP algorithm running sequentially, the cycle, are reduced, converges faster and therefore produces less execution time. Second, by implementing the BP algorithm on parallel processors, the execution time is reduced much more and therefore produces a much better speedup compared to sequential processing.

REFERENCES

[1] S. M. Shamsuddin, M. N. Sulaiman and M. Darus, "Higher Order Centralised Scale-Invariance for Unsconstrained Isolated Handwritten Digits". *PhD Thesis*. Universiti Putra Malaysia (2000).

[2] O. L. Mangasarian and M. V. Solodov, "Serial and Parallel Backpropagation Convergence Via Nonmonotone Pertubed Minimization, Optimization". *Optimization Methods and Software*. Vol. 4, No. 2, (1994), pp. 103-116.

[3] S. M. Shamsuddin, M. N. Sulaiman, and M. Darus, "An Improved Error Signal for the Backpropagation Model for Classification Problems". *International Journal of Computer Mathematics*. Vol. 76, 2001, pp. 297–305.

[4] B. L. Kalman and S. C Kwasny. "A Superior Error Function for Training Neural Network". *International Joint Conference of Neural Network*. Vol. 2, 1991, pp. 42-52.

[5] B. P. Lester, *The Art of Parallel Programming.* Eaglewood Cliffs, New Jersey: Prentice-Hall International Edition (1993).

[6] Sequent Computer Systems, *Sequent Multiprocessor Architecture Overview.* USA: Sequent Computer Systems, Inc. (1994).

[7] J. Jaja, *An Introduction to Parallel Algorithms.* Redwood City, California: Addison-Wesley Publishing Company (1992).

[8] H. H. Ammar and Z. Miao, "Parallel Training Algorithms for a Neural Network Based Automated Fingerprint Image Comparison System". *International Journal of Modelling and Simulation*. Vol. 14, No. 1, 2000, pp. 3-25.

[9] S. Haykin, *Neural Networks: A Comprehensive Faundation.* New Jersey: Prentice Hall Inc. (1999).

[10] Z. Luo, "On the Convergence of the LMS Algorithm with Adaptive Learning Rate for Linear Feedforward Networks". *Neural Computation*. Vol. 3, (1991), pp. 226-245.

BIOGRAPHY

Teh Noranis Mohd Aris is a tutor at Department of Computer Science, Universiti Putra Malaysia (UPM). She received her MSc. from the same university in 2001. Her research interest includes artificial intelligence and parallel processing.

Md. Yazid Mohd Saman is currently a lecturer at Kolej Universiti Terengganu, UPM. He obtained his Ph.D. from Loughborough University UK. His research interest includes programming, parallel processing and distributed systems.

Md. Nasir Sulaiman is a lecturer in Computer Science Department, UPM. He was awarded his Ph.D. in Neural Network Simulation from Loughborough University UK in 1994. Research interest includes neural networks, parallel processing and information systems.

Mohamed Othman obtained his Ph.D. from Universiti Kebangsaan Malaysia. He is currently a lecturer and Head Department of Communication Technology and Networking, UPM. His research interest includes artificial intelligence, parallel processing, scientific computing, multigrid, expert system and logic programming.