Report Number: WUCS-95-27

1995-01-01

Time Variability While Training a Parallel Neural Net Network

Tina L. Seawell and Barry L. Kalman

The algorithmic analysis, data collection, and statistical analysis required to isolate the cause of time variability observed while an Elman style recurrent neural network is trained in parallel on a twenty processor SPARCcenter 2000 is described in detail. Correlations of system metrics indicate the operating system scheduler or an interaction of kernel processes is the most probable explanation for the variability.

Time Variability While Training a Parallel Neural Net Network

Tina I. Seawell and Barry L. Kalman

WUCS-95-27

August 1995

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

Time Variability While Training a Parallel Neural Net Network

Tina I. Seawell and Barry L. Kalman

Abstract

The algorithmic analysis, data collection, and statistical analysis required to isolate the cause of time variability observed while an Elman style recurrent neural network is trained in parallel on a twenty processor SPARCcenter 2000 is described in detail. Correlations of system metrics indicate the operating system scheduler or an interaction of kernel processes is the most probable explanation for the variability.

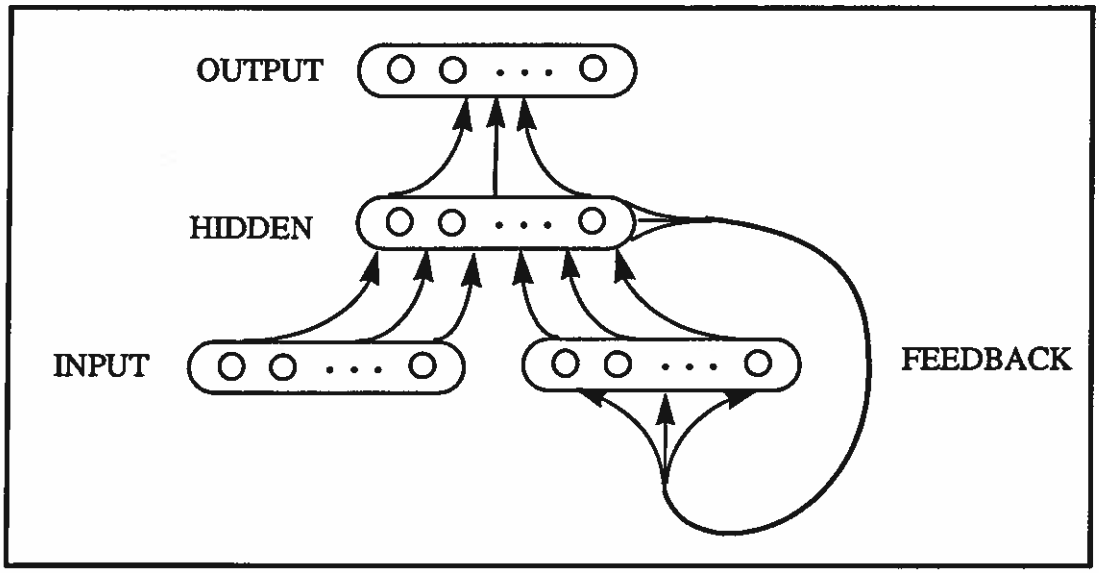
Introduction

Neural networks are a promising approach to nonlinear problems. One of the major drawbacks to practical neural networks is the time required to train the network. Parallelization of neural network training algorithms can decrease training time from months to weeks. Isolation of algorithm and system problems becomes more difficult when parallelism is used. Determining the source of a problem often becomes an exercise in statistical analysis.

A serial neural network training system, Trainrec, was developed by Barry Kalman and Stan Kwasny [KK93]. The training system combines an Elman simple recurrent network [EL88] architecture, with a self-scaling error function to minimize the error, and the conjugate gradient

method with linesearch to minimize the error. Elman's simple recurrent network adds a set of feedback units (pseudo input units) to a feedforward neural network (see Figure 1). This allows the neural network to have a memory of previous states. The training is epoch based, thus node weights are not changed after each pattern is input, but after an entire set of patterns has been input to the network. After a set of patterns is propagated forward, the error derivative or linesearch direction is calculated and changes to the node weights are propagated backward.

Figure 1: Elman Simple Recurrent Neural Network



A parallel implementation of Trainrec, ParalTrainrec, was developed by Pete McCann and Barry Kalman [MK94]. Original time experiments of the parallel algorithm did not exhibit time variability. After training for approximately six months without difficulties a problem of significantly larger size was attempted and the time required to execute a derivative epoch began to vary greatly. A derivative epoch requires error derivatives to be computed for all hidden and output units of the network. The time required to complete about eighty percent of the derivative epochs

varied by less than one second. For the remaining derivative epochs, the time varied from two to as much as thirty times the normal completion time.

ParalTrainrec was designed to run on a Sun SPARCcenter 2000 multiprocessor, running the Solaris2.4 operating system. The SPARCcenter 2000 is a shared-memory machine with a pair of packet-switched XDBuses providing approximately 500 MB/sec of aggregate bandwidth. Our SPARCcenter 2000 multiprocessor has 640 MB of main memory and twenty 40 MHz Super-SPARC processors each with 36 KB of internal cache and 1 MB of external cache.

Parallelization

ParalTrainrec uses coarse grained parallelism and is designed to train a neural net on sequences of patterns [MK94]. The patterns used for testing the time variability of ParalTrainrec are from Sejnowski and Rosenberg's NETtalk experiment [SR87]. Sequences are divided as evenly as possible among the available threads. A copy of the neural network is created for each thread (this includes copying both the weights and the configuration of the neural network).

When ParalTrainrec is invoked the user specifies the number of derivative processors and the number of linesearch processors that will be used. The number of linesearch processors must be less than or equal to the number of derivative processors. Two semaphores, sema-one and sema-two, are initialized for each thread that is to be created. Both semaphores are set to zero and are only used to synchronize threads within ParalTrainrec (they do not effect other processes that may be running). A thread is created for each derivative processor requested and is bound to a lightweight process. A lightweight process is an execution resource or kernel thread, for more

information see the SunOS5.2 Guide to Multi-Thread Programming.

When a thread begins execution, it is immediately bound to an available processor and placed in a wait for sema-one. After the main process has posted to sema-one, threads begin an epoch oriented forward propagation on sequences of patterns. Upon completion of the forward propagation, threads increment sema-two and wait for the main thread to gather all error derivatives, complete backpropagation on the weights, and increment sema-one. The first semaphore controls thread execution and forward propagation, while the second semaphore controls backward propagation, main process execution, and synchronization.

Data Collection

To isolate the cause of the observed fluctuations in training times, ParalTrainrec was executed approximately 500 times. Each execution was limited to two derivative epochs and the linesearch epochs necessary to calculate the gradient descent after each derivative epoch. All program runs were initiated with the same initial weights and the same neural network configuration. The configuration used was 105 input units, 42 hidden units, 27 output units, and 5 feedback units. The input data contained 8517 patterns in 1016 sequences. For the initial weights and configuration there were ten linesearch epochs for each derivative epoch. Some preliminary runs using different numbers of processors were made to determine if the number of processors used effected the time variability. The number of processors varied the lower bound of the time required to process a derivative epoch. All further runs were made using 12 processors for a

derivative propagation and 4 processors for a linesearch. Of the approximately 500 executions of ParalTrainrec, 434 were suitable for analysis.

Seven sets of tests were made. Test's contained from 41 to 83 executions of ParalTrainrec and varied an aspect of either the system environment or program thread control. The tests ran with and without interference, with nice and without nice, with the threads bound and unbound, and using a different set of processors (avoiding processor 0). Two of the seven sets of executions ran with interference. To create a known interference two copies of ParalTrainrec were executed simultaneously, one as described above (short executions of two derivative epochs), while the other ran continuously. This insured interference by a compute bound process using the same processors as the test runs. No other user jobs were being processed during these tests, the only competition was from system processes.

Data gathered included snapshots of all processes running on the Sparc 2000, the time required for each derivative epoch and linesearch epoch, and the value of system metrics for the duration of each execution. A snapshot of all processes running on the system was taken before and after each execution of the program. The Unix command, ps, was used with the -el switch. This listed all user and system processes running, their priorities, nice value, image size in main memory, and the state of the process (running, sleeping, in the run queue, etc.). System metrics were collected using the Unix command, timex, when invoking each execution. Timex was used with the -s option. Seventy-six of the operating system counters assessed by the timex command were placed into SAS data sets for statistical analysis. (SAS, Statistical Analysis System, is a statistical tool developed by SAS Institute Inc. of Cary, NC.) Statements within ParalTrainrec cap-

tured the time spent on derivative and linesearch epochs, indicating the part of the ParalTrainrec code involved in the time variability.

Problem Isolation

Initial examination of the raw data revealed that the real time (clock time) varied in proportion to the real time required to complete a derivative epoch. A linesearch ran for 7 to 10 seconds of real time, while derivative epochs ranged from 3 minutes to 80 minutes. For complete executions (2 derivative epochs and 20 linesearch epochs) Table 1 lists the time statistics with respect to interference. Notice that while the real and system time varied greatly the user time remained comparatively constant. This pattern of time variability remains the same regardless of interference. This and the large standard deviation of the system time was the first indication that the variability might be the result of system processes.

Table 1: Time variability with and without interference

Variable	Minimum	Maximum	Mean	STD
331 runs of ParalTrainrec without interference				
real time	639	5419	1741	1144
user time	5866	7623	6185	330
system time	6	19408	1985	3876
103 runs of ParalTrainrec with interference				
real time	1380	5919	2968	1197
user time	5974	8204	6800	546
system time	20	14670	4249	3945
time is in seconds				

Seventy-six system metrics were extracted from the data collection files using LISP and inserted into SAS data sets as variables. Simple statistics (mean, range, STD, skewness, and kurtosis) were calculated for each of these variables. Some variables such as the size available for kernel memory allocation were the same for all executions and were eliminated. Percentages that could be calculated from other variables were removed from the data set. Other metrics showed no activity and these were also eliminated. From the results, fourteen variables were selected for further analysis. Table 2 lists the fourteen variables, their range, mean, and standard deviation.

Table 2: System Metrics Selected for Analysis

434 runs of ParalTrainrec (each 2 derivative epochs long)				
Variable	Minimum	Maximum	Mean	STD
real time in seconds	639	5919	2032	1268
user time in seconds	5866	8204	6331	471
system time in seconds	6	19408	2522	4006
percentage idle time	20	91	63	15
system calls per second	27	3156	116	184
system reads per second	4	260	23	20
system writes per second	0	494	3.5	25
characters transferred by system reads per second	1598	131318	8667	8939
characters transferred by system writes per second	76	73805	1000	4735
process switches per second	34	25652	5030	6692

Table 2: System Metrics Selected for Analysis

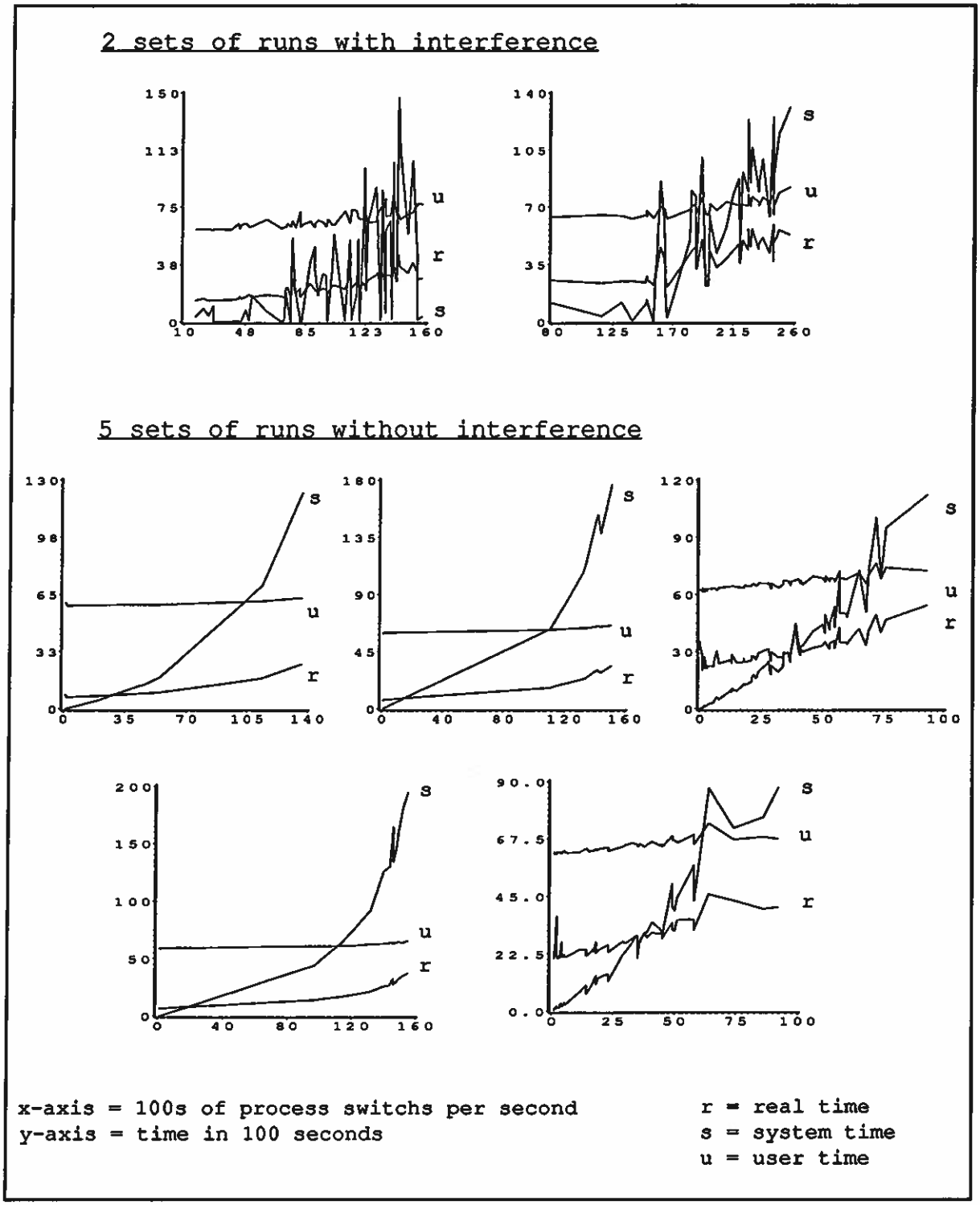
434 runs of ParalTrainrec (each 2 derivative epochs long)				
Variable	Minimum	Maximum	Mean	STD
average run queue length of processes in memory	1	8.5	1.67	1.65
% of process in run queue that are runnable	0	100	21	33
page faults per second (from protection errors)	0.22	9.5	1	1.12
address translation page faults per second (valid page not in memory)	5	4944	893	1136

To determine the independent variable causing the fluctuation of the system time, Pearson correlation coefficients were calculated for the fourteen variables listed in Table 2. When considering all 434 executions, two sets of variables had more than ninety percent correlation. The first set was the percentage of runnable processes in the run queue and the average run queue length. This correlation remained high for interference, but dropped drastically for noninterference. The maximum run queue length for non-interference was two. The second set was the system reads per second and the number of characters transferred by system reads per second. The correlation for this set remained high for non-interference, but dropped into the 80% range with interference. When only interference executions were considered, there was over a 90% correlation between address translation faults, process switches and real time. Without interference, the system time, process switches and percentage of runnable processes in the run queue were over 90% corre-

lated.

The set of data with the most correlation were 62 executions ran with threads bound to the processors, using processors 0 through 11, with the nice value set to 19, and without interference. There were 41 correlations greater than 90% (forty-eight percent of the variables). The real, user, and system time were all correlated with the system calls and system reads per second, the number of characters transferred by system reads per second, the process switches, percentage of runnable processes, and the address translation page faults per second. Graphing the real, user, and system times against process switches for the seven different types of runs produced graphs with essentially the same curves (see Figure 2). Process switching is the independent variable causing the system time variability. The process switches are dependent upon the number of runnable processes in the operating system run queue.

Figure 2: Real, User, and System Time against Process Switches



Source of process switches

From the graphs in Figure 2 it is obvious that the variability of the system time is related to the process switches per second. The correlation between process switches and system time occurs even when there is no interference from other user processes. Ron Marz of IBM [MR94] suggested that process priority is often related to the system time. A script file was created to take snapshots of all processes (system and user) every ninety seconds. At each snapshot, time information of the current derivative epoch and linesearch epochs were also gathered. While the script file collected data, ParalTrainrec ran continuously.

The amount of data to be analyzed was reduced by removing all information about processes that were sleeping and all linesearch times. The time variability was not affected by the script file, so all data pertaining to the script file was also removed. The remaining data showed only ParalTrainrec, a few system processes that periodically wake up and several instances of other user processes. During the few derivative epochs when other processes were running the basic pattern of priority changes exhibited by ParalTrainrec was not affected. Figure 3 lists a small portion of the process data collected. The list is typical of the repeat pattern revealed by the process snapshot. In Figure 3, between each "Derivative Time:" listing, the same source code is executed.

Figure 3: Priority Fluctuations of ParalTrainrec

```
  F S   UID   PID  PPID  C PRI NI     ADDR     SZ    WCHAN TTY      TIME COMD

  8 O   800  4408  4404 80  69 20 f7a2ace0   6957          pts/0   83042:17 yPARALTR
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83062:53 yPARALTR
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83089:57 yPARALTR
 Derivative Time: 206.848147 sec
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83104:52 yPARALTR
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83125:26 yPARALTR
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83152:31 yPARALTR
 Derivative Time: 206.847586 sec
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83167:25 yPARALTR
  8 O   800  4408  4404 80  45 20 f7a2ace0   6957          pts/0   83180:30 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83198:29 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83216:22 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83234:24 yPARALTR
 19 O     0     3     0 80   0 SY f739b998      0             ?     860:00 fsflush
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83252:19 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83270:12 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83288:08 yPARALTR
  8 R   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83306:07 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83324:10 yPARALTR
  8 O   800  4408  4404 80  65 20 f7a2ace0   6957          pts/0   83342:09 yPARALTR
  8 O   800  4408  4404 80  66 20 f7a2ace0   6957          pts/0   83360:08 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83378:09 yPARALTR
  8 O   800  4408  4404 80  67 20 f7a2ace0   6957          pts/0   83396:06 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83414:06 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83431:57 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83449:57 yPARALTR
  8 O   800  4408  4404 80  46 20 f7a2ace0   6957          pts/0   83467:55 yPARALTR
 19 O     0     3     0 80   0 SY f739b998      0             ?     861:38 fsflush
  8 O   800  4408  4404 80  68 20 f7a2ace0   6957          pts/0   83485:56 yPARALTR
  8 R   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83503:59 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83521:58 yPARALTR
  8 R   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83539:47 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83557:40 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83575:38 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83593:33 yPARALTR
  8 O   800  4408  4404 80  56 20 f7a2ace0   6957          pts/0   83611:37 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83629:16 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83646:01 yPARALTR
  8 O   800  4408  4404 80  46 20 f7a2ace0   6957          pts/0   83662:00 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83677:56 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83693:43 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83708:31 yPARALTR
 19 O     0     3     0 80   0 SY f739b998      0             ?     863:15 fsflush
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83722:30 yPARALTR
  8 O   800  4408  4404 80  68 20 f7a2ace0   6957          pts/0   83736:14 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83748:28 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83759:55 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83770:37 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83780:52 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83791:01 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957          pts/0   83801:12 yPARALTR
  8 O   800  4408  4404 80  39 20 f7a2ace0   6957 e09230a0 pts/0   83811:27 yPARALTR
  8 O   800  4408  4404 80  78 20 f7a2ace0   6957          pts/0   83829:35 yPARALTR
  8 O   800  4408  4404 80  59 20 f7a2ace0   6957          pts/0   83836:33 yPARALTR
 Derivative Time: 3912.954029 sec
  8 O   800  4408  4404 80  89 20 f7a2ace0   6957          pts/0   83842:31 yPARALTR
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83857:34 yPARALTR
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83884:39 yPARALTR
 Derivative Time: 206.845107 sec
  8 O   800  4408  4404 80  79 20 f7a2ace0   6957          pts/0   83905:05 yPARALTR
 19 O     0     3     0 80   0 SY f739b998      0             ?     864:22 fsflush
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83918:23 yPARALTR
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83945:27 yPARALTR
 Derivative Time: 206.845004 sec
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83967:37 yPARALTR
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   83977:09 yPARALTR
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   84004:13 yPARALTR
  8 O   800  4408  4404 80  99 20 f7a2ace0   6957          pts/0   84030:13 yPARALTR
 Derivative Time: 206.861655 sec
```

During the majority of derivative epochs, which took about 3.5 minutes, ParalTrainrec had a priority value of 99, the lowest priority. Infrequently this value would briefly drop, increasing the priority, without affecting the time required to complete a derivative epoch. In approximately one-fifth of the derivative epochs, the priority value would continue decreasing, giving ParalTrainrec a higher and higher priority. In most of these cases the priority bounces around, several times reaching a high priority of 39. This high priority is often maintained for 4 to 10 minutes. When the priority becomes low enough, the derivative completes. In these epochs, ParalTrainrec is put to sleep and eventually placed in the run queue. The process in the run queue that is causing the process switches is ParalTrainrec itself.

Conclusion

The number of patterns processed (the amount of input data) and the priorities given to ParalTrainrec by the system scheduler affect the time required to execute a derivative epoch. The time variability did not occur in the initial time trials of the parallel algorithm. Only after increasing the number of patterns processed was a noticeable increase in the real time of a derivative epoch observed. The data collected from numerous executions of ParalTrainrec show that the increase in real time is directly related to an increase in system time and the increase in system time is correlated to the number of process switches. The process switches result from the number of runnable processes waiting in the run queue. Snapshots show that the priority of ParalTrainrec changes dramatically during the longest execution times and that the only process in the

run queue during those episodes is the ParalTrainrec algorithm. This indicates that the scheduler or interaction of the scheduler and other kernel processes is the most likely source of the observed time variability.

Acknowledgments

Thanks to Dr. Ken Wong for his suggestions and guidance. Also, thanks to the Washington University Computer and Communications Research Center for uninterrupted use of the SPARC-center 2000 multiprocessor.

References

[EL88] J. L. Elmann, *Finding Structure in Time*, CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.

[KK93] Barry L. Kalman and Stan C. Kwasny, *TRAINREC: A System for Training Feedforward & Simple Recurrent Networks Efficiently and Correctly*, Technical Report WUCS-93-26, Washington University, May 1993

[MK94] Peter J. McCann and Barry L. Kalman, *Strategies for the Parallel Training of Simple Recurrent Neural Networks*, Technical Report WUCS-94-15, Washington University, June 1994.

[MR94] Ronald Mraz, *Reducing the Variance of Point-to-Point Transfers for Parallel Real-Time Programs*, IEEE Parallel & Distributed Technology, 1063-6552/94, 20-31, 1994.

[SR87] T. J. Sejnowski and C.R. Rosenberg, *Parallel Networks that Learn to Pronounce English Text.* Complex Systems 1, 145-168, 1987.