Computational Challenges in Vector Functional Coefficient Autoregressive Models*

Ioana Banicescu[1,2], Ricolindo L. Cariño[2], Jane L. Harvill[3,2], and John Patrick Lestrade[4]

[1] Department of Computer Science and Engineering, PO Box 9637
[2] Center for Computational Sciences ERC, PO Box 9627
[3] Department of Mathematics and Statistics, PO Box MA
[4] Department of Physics and Astronomy, PO Box 5167
Mississippi State University, Mississippi State MS 39762, USA
(ioana@cse,rlc@erc,harvill@math,lestrade@ra).msstate.edu

Abstract. An important research area in statistical computing is found in the literature for vector functional coefficient autoregressive models, a special case of vector nonlinear time series. Methods used are computationally intensive. As a result, analyses and simulations can run into weeks, or even months. Statisticians have been known to base empirical results on a relatively small number of simulation replications, sacrificing precision, accuracy and reliability of results in the interest of time and productivity. The simulations are amenable for parallelization; however, parallel computing technology has not yet been widely used in this specific research area. This paper proposes an approach to the parallelization of statistical simulation codes to address the challenge of long running times, without resorting to extensive code revisions. This approach takes advantage of recent advances in dynamic loop scheduling on workstation clusters to achieve high performance, even with the presence of unpredictable load imbalance factors. Preliminary results of applying this approach in the simulation of normal white noise and threshold autoregressive model obtains efficiencies in the range 95–98% on 8–64 processors.

1 Introduction

A vector time series is a set of observations of multiple related phenomena across time. The mathematical underpinnings of the statistical analysis of time series incorporate the correlation across time and between series – properties that complicate statistical theory. This is especially true for nonlinear models, where mathematical theory may be extremely difficult, even intractable. Consequently, Monte Carlo simulation is often relied upon to give direction, to interpret, and

* This work was partially supported by the National Science Foundation Grants: CAREER #9984465, ITR/DMS #0313274, ITR/ACS #0081303, ITR/ACS #0085969, #0132618, and #0082979.

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3514, pp. 237–244, 2005.

to explain complex analytical results. Gentle [1] gives a thorough discussion of Monte Carlo simulation methods, their usefulness, and an idea of the underlying theory of their application to statistical computing. In general, it can be said that as the number of Monte Carlo replications increases, the result of the simulation approaches the "truth." The more complex the structure, the larger the number of replications needs to be.

On a single processor, to examine statistical properties of methods related to vector functional coefficient autoregressive models via simulations would require execution times of a few weeks or even months. Statisticians have been known to base empirical results on a relatively small number of simulation replications, sacrificing precision, accuracy, and possibly compromising the reliability of results in the interest of time. As a result, techniques which may require thousands of replications for accuracy and reliability often have at most, a few hundred. Fortunately, the replications are amenable for computation in parallel. Thus, parallel processing technology can be exploited to enable the extensive simulation of a variety of models within reasonable running time limits. This paper demonstrates an approach to the simulation of such models which takes advantage of recent advances in dynamic loop scheduling on clusters of workstations, in order to shorten the simulation time.

The remainder of this paper is organized as follows. Section 2 contains a brief overview of vector functional coefficient autoregressive models, along with the computational issues related to the statistical methods that make use of these models. Section 3 describes an approach to facing the challenges in implementing the procedures numerous times, as would be done in a typical Monte Carlo study. Section 4 gives preliminary results which demonstrate the high parallel performance achieved by the proposed approach on general-purpose clusters. Concluding remarks are in Section 5.

2 Vector Functional Coefficient Autoregressive Model

Although complicated in both presentation and theory, vector nonlinear time series are especially useful for describing complicated nonlinear dynamic structures that exists in many time-dependent multivariate series. Let $\boldsymbol{Y}_t = (Y_{1,t}, \ldots, Y_{k,t})'$ denote the vector time series at time $t = 1, 2, \ldots, T$. Then the vector functional coefficient autoregressive model of order p (VFCAR(p)) is defined as

$$\boldsymbol{Y}_t = \boldsymbol{f}^{(0)}(\boldsymbol{Z}_t) + \sum_{j=1}^{p} \boldsymbol{f}^{(j)}(\boldsymbol{Z}_t)\boldsymbol{Y}_{t-j} + \boldsymbol{\varepsilon}_t, \quad t = p+1, \ldots, T, \tag{1}$$

where $\boldsymbol{f}^{(j)}$, $j = 0, \ldots, p$ are $k \times k$ matrices whose elements are real-valued measurable functions that change as a function of the (possible vector-valued) $\boldsymbol{Z}_t$, and which have continuous second-order derivatives. The error terms $\boldsymbol{\varepsilon}_t$ in (1) are such that for each i, the series $\{\varepsilon_{i,t}\}_{t=1}^{T}$ is a white noise sequence, independent of $\{\boldsymbol{Y}_t\}_{t=1}^{T}$. However contemporaneous cross-correlation may exist

between $\{\varepsilon_{i,t}\}$ and $\{\varepsilon_{j,t}\}$, $i \neq j$. The primary motivation for studying this model is that specific choices for the elements of the $\boldsymbol{f}^{(j)}$ yield parametric models.

The VFCAR(p) model may be considered a hybrid of parametric and nonparametric models since the autoregressive structure is assumed, but there is little or no information about the form of the elements of the $\boldsymbol{f}^{(j)}$. As such, estimation of the parameters of the VFCAR(p) model is done nonparametrically via local regression. Simultaneous estimation of the elements of the $\boldsymbol{f}^{(j)}$ provides improved statistical efficiency when the error terms have positive cross-correlation. In the process of fitting the model (1), modified multifold cross-validation is used to determine an optimal bandwidth and value for p by finding the pair of values that minimize the accumulated prediction error. This multistage procedure requires an immense number of arithmetic operations on a univariate series. That number increases exponentially for multivariate series.

The VFCAR model in used in statistical tests of model misspecification. However, the null distribution is unknown, and so a procedure that fits the model to a large number of bootstrapped realizations of the series under the null model is used to obtain a numerical p value for the test. Specifics for fitting the model and the testing procedure are found in [2]. Forecasting can be accomplished using the VFCAR model either recursively or via bootstrap as in [3]. The mathematical complexity of the statistical theory of the estimators, testing procedures, and forecasts using the VFCAR model are highly complicated, necessitating the use of simulation to study their statistical properties.

3 Simulation

A high-level outline of the Fortran program for simulation to investigate statistical properties of methods using the VFCAR model is given by Figure 1. Executing the program on a desktop computer with a 1GHz Pentium 4 processor and 256MB RAM, for 1000 replications of a single bivariate model using a sample size 400 takes approximately nine (9) days. An investigation of various models using more error cross correlation values and larger sample sizes or number of replications will run into months on a single machine.

The bulk of the arithmetic (sample generation, parameter estimation, hypothesis testing) is performed by the iterations of the replication loop, in `Model testing`. These iterations are independent, hence they can be distributed among P processors to shorten the loop completion time. In general, equally distributing the replications among the processors leads to high performance if the replications have the same execution time and the processors are homogeneous. However, this static assignment may result in load imbalance due to heterogeneity of processors, irregular iteration execution times, or unpredictable systemic effects like operating system interference leading to varying effective processor speeds. Load imbalance is typically indicated by highly uneven processor finishing times, and is a major cause of performance degradation in parallel applications. Dynamic loop scheduling is therefore necessary in order to balance processor loads and minimize the loop completion time.

```
{Input model specifications; error cross correlations: no_corrs,
  corrs(1..no_corrs); sample sizes: no_ns, ns(1..no_ns); and
  no. of replications: no_reps}
emp_rr = 0
DO corr_no = 1, no_corrs   ! correlation loop
  {Correlation initializations}
  DO n_no = 1, no_ns       ! sample size loop
    n = ns(n_no)
    DO rep_no = 1, no_reps ! replication loop
      {Model testing; update emp_rr()}
    END DO
  END DO
END DO
emp_rr = emp_rr/no_reps
Output emp_rr
```

Fig. 1. High-level outline of the serial program

The parallelization of the simulation program was accomplished with minor alterations to the serial code. Essentially, three (3) lines of code are added before and after the original replication loop code, and the loop extents were changed, as illustrated by Figure 2. The routines `LS_Finalize`, `LS_Initialize`, etc., are contained in a module specifically designed for quick integration of dynamic loop scheduling into sequential programs containing parallel loops. This module evolved from previously developed code for dynamic loop scheduling in scientific applications that utilize the Message Passing Interface (MPI) library [4]. Prior versions of the module were used to improve the performance of applications such as the profiling of automatic quadrature routines [5, 6] and simulation of wave packets by the quantum trajectory method [7].

```
call LS_Initialize (foreman, 1, no_reps, method)
do while ( .not. LS_Terminated() )
  call LS_StartChunk (cStart, cSize)
  DO rep_no = cStart, cStart+cSize-1 ! replication loop
    {Model testing; update emp_rr()}
  END DO
  call LS_FinishChunk()
end do
call LS_Finalize (nIters, workTime)
```

Fig. 2. The modified replication loop for parallelization and dynamic scheduling

The routine `LS_Initialize` signals the start of dynamic scheduling of loop iterations: `foreman` specifies the rank of the process that will serve as the scheduler, (`1,no_reps`) is the loop extent, and `method` identifies the scheduling technique.

The logical function `LS_Terminated` tests for loop termination. `LS_Finalize` synchronizes the processes, and returns `nIters` and `workTime`, the count and total execution time, respectively, of `Model testing` computations performed by the calling process. These two quantities are useful in analyzing the performance of load balancing; ideally, the work times of the processors should be approximately equal.

The routine `LS_StartChunk` returns the start `cStart` and size `cSize` of a chunk of iterations to be executed by the calling processor. These quantities are determined according the the scheduling technique specified by `method`. After the chunk is finished, the routine `LS_FinishChunk` is invoked. Internally, the calling processor sends performance data pertaining to the recently executed chunk to the scheduler. The scheduler receives the performance data, and if there are remaining iterations, it computes and sends the next (`cStart`, `cSize`) to the processor; otherwise, the scheduler sends a termination message.

In addition to the changes in the replication loop, other modifications to enable parallelization pertain to the use of MPI. The usual calls to MPI_Init(), MPI_Comm_size() and MPI_Comm_rank() are added at the start of the program, and MPI_Finalize() added just before the program ends. Also, each processor computes only a fraction of the total number of replications, giving a partial result in the `emp_rr` array. The full set of results is obtained by performing the summation MPI_Allreduce(..., MPI_SUM, ...) on the `emp_rr` after the correlation loop.

4 Performance Measurements

Performance tests of the Fortran 90+MPI implementation of the model testing program were conducted on a general-purpose Solaris and Linux clusters at the Mississippi State University Engineering Research Center. The Solaris cluster consists of Myrinet-interconnected Sun Microsystems SMPs. Each SMP has four (4) UltraSPARC 400MHz processors, and the cluster has 16 nodes for a total of 64 processors. However, the cluster usage policy allows a job to utilize no more than 32 processors and no more than 48 hours execution time. The Linux cluster has a total of 1038 Pentium III (1.0 GHz and 1.266 GHz) processors, runs the Red Hat Linux operating system, and is connected via fast Ethernet switches with Gigabit Ethernet uplinks. On the Linux cluster, the queuing system (PBS) attempts to assign homogeneous compute nodes to a job, but this is not guaranteed. Other jobs were also running on the clusters along with the experiments, thus network traffic volume may have varied during the experiments, with unpredictable effects on the performance tests.

Table 1 summarizes the tests that were attempted. WNnnn and TARnnn denote a normal white noise and a threshold autoregressive model, respectively, with 'nnn' sample size. The last two columns give the job execution time (in hh:mm format) without loop scheduling (STAT) or with loop scheduling using the adaptive factoring technique (AF) [8, 9, 10]. The AF was chosen since it is the most sophisticated technique, as it addresses all sources of load imbalance. However, the AF has a higher overhead than other techniques, which may cause

Table 1. Performance tests

Cluster	Test Id	Sample size	Replications	P	STAT time	AF time
Sun	TAR30	30	10000	8	11:09	11:25
Solaris	TAR75	75	10000	16	27:47	29:05
	WN150	150	10000	32	31:15	23:28
Intel	WN500	500	10000	64	5:56	4:47
Linux	TAR500	500	10000	64	15:42	15:59

the AF to perform slightly lower than other techniques for problems that exhibit no load imbalance.

The job times (STAT, AF times) demonstrate the dramatic reduction in simulation time achieved by the simple code modification. For instance, running the original code for the TAR500 model on one of the processors of the Linux cluster would take approximately 64*15.7 hours, or 42 days.

Except for WN150 and WN500, the parallelization without loop scheduling (STAT) consumed slightly lesser time than with loop scheduling using the AF technique. This indicates that application-induced load imbalance was not a significant issue. However, the cases of the WN150 and WN500 provide clear evidence for the influence of system-induced load imbalance. Figure 3 for WN150 illustrates this influence. For STAT, where each processor executed (`no_reps` + $P - 1)/P$ replications (the gray bars), the plots of the processor work times (the gray triangles) show unusually longer times for processors 4, 12, 20 and 28. Most likely, these processors belonged to the same SMP node, and that this node had an extraneous process. With the AF technique, system-induced imbalance

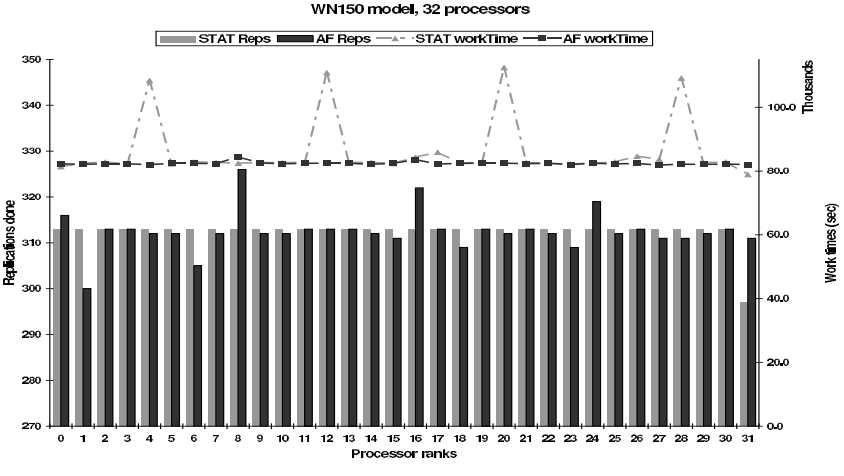

Fig. 3. Loop scheduling summary for WN150 on the Solaris cluster

was also present, as evidenced by the markedly unequal number of replications executed by the processors. The almost flat trend line of the processor work times for the AF technique (dark squares) indicates that the load imbalance was successfully addressed.

The cluster resources allocated to the simulation were very efficiently utilized, as indicated in Table 2 which shows that up to 98% parallel efficiency is achievable. The lowest is 77% for WN150 with STAT, which may be attributed to severe system-induced load imbalance. This load imbalance is successfully addressed by AF obtaining 98% efficiency for the same problem. The speedup (Spd) and efficiency (Eff) metrics were computed as follows. Denote by t_r the time spent by processor r on the modified replication loop, and w_r the `workTime`. A t_r is the difference of the times taken just before `LS_Initialize` and right after `LS_Finalize`. A w_r is the cumulative time spent in `Model testing`. The parallel time T_P for the replication loop is $T_P = \max t_r$, and an estimate of the serial time T_1 is given by $T_1 = \sum_{\forall r} w_r$. This estimate for T_1 is accurate if the processors are homogeneous. From P, T_P and T_1, Spd=T_1/T_P and Eff=$T_1/(P \times T_P)$.

Table 2. Speedup and efficiency

		No loop scheduling (STAT)				With loop scheduling (AF)			
Test Id	P	T_1	T_P	Spd	Eff	T_1	T_P	Spd	Eff
TAR30	8	315071	40118	7.9	0.98	313965	41121	7.6	0.95
TAR75	16	1589994	100030	15.9	0.99	1594960	104680	15.2	0.95
WN150	32	2757215	112490	24.5	0.77	2636511	84475	31.2	0.98
WN500	64	1066760	21359	49.9	0.78	1067011	17232	61.9	0.97
TAR500	64	3582159	56531	63.4	0.99	3581880	57527	62.3	0.97

5 Conclusion

This paper presents an approach to the parallelization of simulations to investigate statistical properties of methods using VFCAR models. The parallelization allows improved accuracy and precision of statistical results, and is based on novel techniques for improving performance of scientific computing in parallel and distributed environments. The approach is capable of successfully overcoming the challenge of long running times and requires very minimal revisions to an existing sequential code. The revisions generate parallel code which incorporates dynamic load balancing. The parallel code addresses factors that may give rise to load imbalance, such as those that may be inherent in the computations or induced by the computing environment. Tests on a homogeneous Sun Solaris cluster and a heterogeneous Linux cluster indicate that the parallel code achieves very high performance, even with unpredictable system-induced load imbalance, obtaining efficiencies in the range 95–98% for the normal white noise and threshold autoregressive model, with sample sizes up to 500 and 10000

replications, on up to 64 processors. This combination of sample size and replication count, to best knowledge, has not been previously attempted when using simulation to investigate properties of statistical methods using VFCAR models. Future work will include experiments with new models and more error correlation values. Furthermore, this interdisciplinary research work finds application in a variety of other statistical disciplines. The authors plan to to apply these state of-the-art techniques to an important problem in astrophysics, the classification of gamma-ray bursts.

References

1. Gentle, J.: Random Number Generation and Monte Carlo Methods. 2nd edn. Springer, New York (2003)
2. Harvill, J., Ray, B.: Functional coefficient autoregressive models for vector time series. Computational Statistics and Data Analysis (2005) (To appear)
3. Harvill, J., Ray, B.: A note on multi-step forecasting with functional coefficient autoregressive models. International Journal of Forecasting (2005) (To appear)
4. Cariño, R.L., Banicescu, I.: A load balancing tool for distributed parallel loops. In: Proc. Int. Workshop on Challenges of Large Applications in Distributed Environments, IEEE Computer Society Press (2003) 39–46
5. Cariño, R.L., Banicescu, I.: Dynamic scheduling parallel loops with variable iterate execution times. In: Proc. 16th IEEE Int. Parallel and Distributed Processing Symposium - PDSECA, IEEE Computer Society Press (2002) on CDROM
6. Cariño, R.L., Banicescu, I.: Load balancing parallel loops on message-passing systems. In Akl, S., Gonzales, T., eds.: Proc. 14th IASTED Int. Conf. on Parallel and Distributed Computing and Systems, ACTA Press (2002) 362–367
7. Cariño, R.L., Banicescu, I., Vadapalli, R.K., Weatherford, C.A., Zhu, J.: Message-passing parallel adaptive quantum trajectory method. In Yang, L.T., Pan, Y., eds.: High performance Scientific and Engineering Computing: Hardware/Software Support, Kluwer Academic Publishers (2004) 127–139
8. Banicescu, I., Liu, Z.: Adaptive factoring: A dynamic scheduling method tuned to the rate of weight changes. In: Proc. High Performance Computing Symposium. (2000) 122–129
9. Banicescu, I., Velusamy, V.: Load balancing highly irregular computations with the adaptive factoring. In: Proc. 16th IEEE Int. Parallel and Distributed Processing Symposium - HCW, IEEE Computer Society Press (2002) on CDROM
10. Banicescu, I., Velusamy, V., Devaprasad, J.: On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring. Cluster Computing: The Journal of Networks, Software Tools and Applications **6** (2003) 215–226