Parallel algorithms for large scale econometric models

Bogdan Oancea*, Tudorel Andrei, Ion Gh. Rosca, Andreea Iluzia Iacob

a Nicolae Titulescu University, 040051, Bucharest, Romania,
b The Bucharest Academy of Economic Studies, 010374, Bucharest, Romania

Abstract

In this paper we have developed algorithms to solve macroeconometric models with forward-looking variables based on Newton method for nonlinear systems of equations. The most difficult step for Newton methods represents the resolution of a large linear system for each iteration. Thus, we compare the performances resulted by solving this linear system using two iterative methods and the direct method. We’ve also described an implementation of the parallel versions of such algorithms using a software package. Our experiments confirm that the iterative methods have a low computational complexity and storage requirements, but the parallel versions of direct methods show a superior speedup.

Keywords: parallel algorithms; macroeconometric models; rational expectations models; linear algebra; Newton methods; Krylov techniques; direct methods;

1. Introduction

Advances in the computational power have a large influence on almost all fields of scientific computing. Although, during the last decade, microprocessors’ performance has significantly increased and new architectures like multi-core processors has appeared, there are still problems that cannot be solved on a single desktop computer [5]. One of the fields that need a special attention is macroeconometric modeling. Macroeconometric models with forward-looking variables are a special class of models which involve very large systems of equations. For such models it is necessary to develop high performance parallel algorithms that can be run in parallel execution environments like parallel computers, clusters of workstations or grid environments.

A special kind of macroeconometric models are the rational expectations models [9]. These models contain variables that forecast the economic system state for the future periods $t + 1, t + 2, \ldots, t + T$, where $T$ is the forecast time horizon. Depending on the size of the forecast time horizon, macroeconometric models with rational expectations could give raise to systems with tens or hundreds of thousands of equations.

Examples of such models are MULTIMOD [11], QPM (Quarterly Projection Model) [2], FRB/US [3] or FRB/GLOBAL.

One of the first methods used to solve such models was the extended path algorithm proposed by Fair and Taylor [8]. They use Gauss-Seidel iterations to solve the model, period after period, for a given time horizon. The convergence of this method depends on the order of the equations. The advantage of this method is its simplicity in implementation and the low storage requirements but this method has a main disadvantage: if the initial values for...
the endogenous variables are not “well” chosen, the convergence of the system is very poor or the system is not convergent at all. An alternative method to solve the model is to built a system of equations written for successive periods $t, t + 1, ..., t + T$, and to solve this system of $nT$ nonlinear equations by one of the existing methods for nonlinear systems. Due to the large scale of the system, this method has been avoided in the past. Due to the recent advances in the parallel algorithms field it is now possible to solve such large scale systems with efficiency.

We will analyze high performance iterative and direct methods used to solve large linear systems that result by applying the Newton method, then we will describe an implementation of the parallel versions of such algorithms that we’ve developed using a software package called PLSS (Parallel Linear System Solver).

2. Serial iterative and direct methods for the solving of linear systems

For very large linear systems, the most appropriate iterative methods are the so-called Krylov techniques [13]. Contrary to stationary iterative methods such as Jacobi or Gauss-Seidel, Krylov techniques use information that changes from iteration to iteration. For a linear system $Ax=b$, Krylov methods compute the $i$th iterate $x(i)$ as:

$$x(i)=x(i-1)+d(i) \quad i=1,2,...$$ (1)

Operations involved to find the $i$th update $d(i)$ are only inner products, saxpy and matrix-vector products that has the complexity of $\Theta(n^2)$, so that Krylov methods are computational attractive comparing to the direct methods for linear systems.

Perhaps the best known of the Krylov’ method is the conjugate gradient method. This method solves symmetric positive definite systems. This method can be implemented using only one matrix-vector multiplication per iteration. In exact arithmetic, the CG method gives the solution for at most $n$ iterations. The complete description of the CG method can be found in Golub [10].

Another Krylov method for general non symmetric systems is the Generalized Minimal Residuals (GMRES) introduced by Saad [13]. GMRES method uses a Gram-Schmidt orthogonalization process. GMRES requires the storage and computation of an increasing amount of information. To overcome these difficulties, the method can be restarted after a chosen number of iterations $m$. The current intermediate results are used as a new starting point.

Another Krylov method implemented by the authors is the BiConjugate Gradient method. BiCG uses a different approach based upon generating two mutually orthogonal sequences of residual vectors and $A$-orthogonal sequences of direction vectors. The updates for residuals and for the direction vectors are similar to those of the CG method, but are performed using $A$ and its transpose. The disadvantage of the BiCG method is an erratic behaviour of the norm of the residuals and potential breakdowns. We used a version of this method, BiCGSTAB that have the computational complexity of the method is $\Theta(n^2)$ like the other Krylov methods. The operation count per iteration cannot be used to directly compare the performance of BiCGSTAB with GMRES because GMRES converges in much less iterations than BiCGSTAB. We have implemented these iterative methods and run experiments to determine the possible advantages of them over the direct methods. The results of our experiments are presented in the next section.

The other alternative to solve a linear system $Ax=b$ is the direct method that consists in two steps: first, the matrix $A$ is factorized, $A=LU$ where $L$ is a lower triangular matrix with 1s on the main diagonal and $U$ is an upper triangular matrix; in the case of symmetric positive definite matrices, we have $A=LL^\top$ and second, we have to solve two linear systems with triangular matrices: $Ly=b$ and $Ux=y$.

The standard LU factorization algorithm with partial pivoting is given in Golub [10]. The computational complexity of this algorithm is $\Theta(2n^3/3)$. After we obtain the matrix factors $L$ and $U$ we have to solve two triangular systems: $Ly=b$ and $Ux=y$. These systems are solved using forward and backward substitution that have a computational complexity of $\Theta(n^2)$, so the most important computational step is the matrix factorization. That’s why we have to show a special attention to the algorithms for matrix factorization.

In practice, using actual computers with memory hierarchies, the above algorithm is not efficient because it uses only level 1 and level 2 BLAS operations [7]. As it is well-known, level 3 BLAS operations [6] have a better efficiency than level 1 or level 2 operations. The standard way to change a level 2 BLAS operations into a level 3 BLAS operation is delayed updating. In the case of the LU factorization algorithm we will replace $k$ rank-1 updates.
with a single rank-\(k\) update.

3. The implementation of parallel algorithms for linear systems

For very large matrices that result for the econometric models presented above, the serial algorithms may not be appropriate to solve the models. Thus, parallel versions of the above presented algorithms have to be developed and implemented. Software packages for solving linear systems have known a powerful evolution during the last 35 years. LINPACK was the first portable linear system solver package followed at the end of ‘80 by a new software package for linear algebra problems LAPACK [1] which was adapted for parallel computation resulting ScaLAPACK [4] library.

Although parallel algorithms for linear systems are studied and very well understood nowadays, the availability for general purpose, high performance parallel linear algebra libraries is limited by the complexity of implementation.

We have developed a library that implements parallel algorithms for linear systems solving - PLSS (Parallel Linear System Solver). The library was designed with an easy to use interface, which is almost identical with the serial algorithms’ interface. The parallelism is hidden from the user and the algorithms are almost identical with their serial versions. This goal was obtained by means of data encapsulation in opaque objects that hide the complexity of data distribution and communication operations. The PLSS library was developed in C and for the communication between processors we used MPI library [14]. It has a 4 level structure.

The first level contains the standard BLAS, MPI and C libraries. This level is architecture dependent. The second level provides the architecture independence, which implements the interface between the first level and the rest of the PLSS package. The next level implements the data distribution model – all details regarding distribution of vectors and matrices on processors are localized at this level. At this level, the data are encapsulated in objects that are opaque to users, hiding thus the complexity of communication operations. The top level of the PLSS library is, in fact, the application programming interface. PLSS API provides a number of routines that implements parallel BLAS operations and parallel linear system solving operations: direct methods based on LU and Cholesky matrix factorization and nonstationary iterative methods GMRES, BiCG, BiCGSTAB.

The current version of the PLSS library implements a subset of the parallel BLAS routines and for matrix factorization, PLSS library has two routines Cholesky(Object A) – computes the Cholesky factorization of a SPD matrix A and LU(Object A, Object pivots) – computes the LU factorization with partial pivoting. Finally, the PLSS package contains routines that implement the GMRES, BiCG and BICGSTAB iterative methods.

4. Experimental results

We have conducted performance experiments for both serial and parallel versions of the algorithms for two iterative methods – GMRES(35) and BiCGSTAB and for the direct method that consists in matrix factorization. For our experiments we have considered nonlinear systems containing between 2000 and 20000 variables. The tolerance for the solution was fixed at 10\(^{-4}\) for all methods. The serial versions of the algorithms are implemented using the C programming language under the Linux operating system. Both iterative methods behave relatively well for our problems but BiCGSTAB is slightly less expensive in number of floating point operations and memory requirements. Table 1 shows the number of floating point operations per iteration for each Newton variant to converge and the amount of memory needed.

These results show that the iterative methods can be a good alternative to direct methods for systems containing a higher number of equations mainly due to the low memory requirements.

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{Size} & \text{GMRES(35) MFLOP} & \text{GMRES(35) Memory (Mb)} & \text{BiCGSTAB MFLOP} & \text{BiCGSTAB Memory (Mb)} \\
\hline
2000 & 149 & 0.66 & 2000 & 135 & 0.38 \\
4000 & 261 & 1.51 & 4000 & 260 & 0.66 \\
\hline
\end{array}
\]
Parallel versions of the algorithms implemented using the PLSS package were executed on a cluster of workstations, connected through a 100Mb Ethernet local network. The PLSS package uses the MPICH implementation of the MPI library and, for the local BLAS operations, uses the ATLAS library [15] that provides a high performance for local operations. We have tested the PLSS package for both iterative and direct methods, for 1, 2, 4, 8, and 16 processors. The dimension of the matrix was 20000 rows and columns. Figure 1(a) shows the speedup of the parallel algorithms for the case when iterative methods are used to solve the model and figure 1(b) shows the speedup in the case of using direct methods.

![Speedup for iterative methods](image1)

![Speedup for the LU](image2)

As we can observe, the speedup for the GMRES is slightly better than for the BiCGSTAB method. Compared to the iterative methods, the direct method based on matrix factorization shows a better speedup, meaning that these algorithms are more scalable than the iterative ones. They are better suited for very large problems which run on parallel environments with a large number of processors.

5. Conclusions

In this paper we have developed algorithms to solve macroeconometric models with forward-looking variables based on the Newton method for nonlinear systems of equations. The most difficult step for Newton methods represents the resolution of a large linear system for each iteration. We also compared the performances resulted by solving this linear system using two iterative methods and the direct method. For serial algorithms, Krylov methods proved to be an interesting alternative to exact Newton method with LU factorization for large systems. The computational cost and the memory requirements are inferior in the case of Krylov methods compared with LU factorization due to a low computational complexity.

Regarding the parallel algorithms, we have developed a parallel library PLSS with an interface easy to use. All the complexity of the parallel algorithms is hidden from the users by encapsulating the matrices and vectors in opaque objects. The experiments using our library for the direct methods using LU factorization showed a better scalability compared to iterative methods because the iterative algorithms involve a global communication step at the end of each iteration. These results recommend the use of parallel algorithms for very large systems in parallel
environments with a large number of processors.

Acknowledgements

The work related to this paper was supported by the CNCSIS–UEFISCSU projects PNII – IDEI ID_1814 and ID_1793.

References