Improving the Performance of Parallel Triangularization of a Sparse Matrix Using a Reconfigurable Multicomputer*

José L. Sánchez[1], José M. García[2], Joaquin Fernández[1]

[1] Universidad de Castilla-La Mancha, Escuela Politécnica
Campus Universitario, 02071 Albacete, Spain
{jsanchez,joaquin}@info-ab.uclm.es
[2] Universidad de Murcia, Facultad de Informática
Campus de Espinardo, 30071 Murcia, Spain
jmgarcia@dif.um.es

Abstract. Many applications require the solution of a least-squares (LS) problem from a coefficient matrix and a measurement vector. In some cases, the solution must be obtained within a short period of time, requiring great computation power, as is the case of state estimation in electric power systems. In some cases, the coefficient matrix is a large and sparse one, requiring special techniques to reduce the computation-time and storage requirements. In these situations, fast Givens rotations are very well suited for parallel computers because they exhibit a great potential parallelism.
In this paper we improve the performance of the fast Givens rotations algorithm for sparse matrices by means of using a reconfigurable multicomputer. A reconfigurable multicomputer is a message-passing multiprocessor in which the network topology can change during the execution of the algorithm. In this way, the interconnection network can match the communication requirements of a given algorithm.
In this paper we show the improvement for applying this novel technique to improve the performance of the parallel fast Givens rotations, and we present general concepts related with it. This technique consists basically in placing the different processors in those positions in the network which, at each computational moment and according to the existing communication pattern among them, are more adequated for the development of such computation.

1 Introduction

Many numeric calculation problems need to be solved in the shortest possible time, therefore requiring the availability of computers with high calculation power. In this sense, massively parallel computers have become the best alternative to achieve this objective. Their high processing speed is based on parallel execution of different processes which, properly combined, will produce the solution required.

* This work was supported in part by CICYT under Grant TIC94-0510-C02-02

Parallel algorithm implementation is not immediate, being in many cases necessary a great programming effort and specially when these problems are specifically of sequential type. Another problem related with the parallel implementation is the election of the best parallel algorithm. Sometimes, the sequential and parallel behaviour is not the same. So, it is necessary to choose the algorithm with the best parallel execution.

In this paper, we solve in parallel the least-squares problem by means of numerical methods based on orthogonal transformations. This problem is very usual in many scientific applications, as lineal system resolution or eigenvalue problems. Among QR descomposition, Householder transformations are usually preferred when programming a serial computer, due to their higher speed. However, Givens rotations are very well suited for parallel computers, because they exhibit a great potencial parallelism. Moreover, there is an improved version of Givens rotations, known as fast Givens rotations. Finally, in some cases the coefficient matrix is a large and sparse one.

In this paper, we present a novel technique to improve the performance of parallel triangularization of a sparse matrix using fast Givens rotations. This parallel algorithm is executed in a multicomputer. Among all computer architectures developed to this moment, multicomputers present the highest performance for resolution of sparse matrices problems. In this class of machines, the communication between processors relies on an interconnection network, generally with a point-to-point topology. The main problem presented by multicomputers is precisely the saturation of the interconnection network. When a process requires one datum which is being calculated by other processes executed in different processor, communication is established between both. Another possible cause which can produce communication between two processes is when they must be synchronized.

In order to reduce the negative effect due the communication, a novel technique used to improve the performance in multicomputers is the dynamic reconfiguration of the interconnection network. This technique consists basically in placing the different processors in those positions in the network which are more adecuated for the development of the computation and communication.

In this paper we show the performance improvement in fast Givens rotations using a reconfigurable network multicomputer. We are encouraged by than to continue our researches in depth. The rest of the paper is organized as follows. In the next section we briefly introduce Givens rotations and the parallel algorithm. In section 3 we present the dynamic reconfiguration of the interconnection network, and in section 4 we show and analyse the evaluation results. Finally, in section 5 we give some conclusions and ways for future work.

2 Fast Givens Rotations

A Givens rotation can be defined by a transformation matrix $J(i,k,\theta)$, where θ is the rotation angle. The definition of $J(i,k,\theta)$ is well known and can be found

in [8]. The application of an mxn transformation matrix $J(i,k,\theta)$ to an mxn matrix A annihilates the element A_{ki}, choosing the appropiate value of θ.

The transformation of A into an upper triangular matrix can be achieved by calculating and applying to A a sequence of Givens rotations, which annihilate the elements below the diagonal. In several problems, including LS, the rotations are also applied to an $mx1$ coefficient vector b. The result of the transformation is a triangular equation system, which can be solved by backward substitution.

The classical algorithm for matrix triangularization on serial computers nullifies all the elements below the diagonal sequentially. A pair of rows is selected in each iteration, calculating a rotation and applying it to all the elements of the selected rows. This algorithm has a complexity $O(mn^2)$.

To obtain a parallel implementation of the algorithm, we must take into account that the rotations of rows of A are totally independent and can be applied in any order. The only requirement for applying rotation to a pair of rows is that the first nonzero elements of both rows occupy the same column position. So, any pair of rows of the same type can be processed in parallel with any other pair. Processes do not need to communicate during the rotation process. However, after each rotation one of the rows must be transferred to another process in most cases.

The algorithm to triangularizate a matrix A and its associated coefficient vector b, based on fast Givens rotations, the theoretical proof of the algorithm and more information about Givens rotations can be found in [8]. We have followed the studies developed by Duato [4] in order to apply fast Givens rotations in multicomputers.

2.1 Parallel Algorithm

The parallel implementation of the algorithm requires as many processes as columns the sparse matrix has. If we define the type of a row as the column position occupied by its leftmost non-zero element, then it is well known that only rows of the same type can be rotated together. Then we distribute the rows among processes in such a way that each process stores all the rows of the same type. After a pair of rows has been rotated, one of them increases its type, being sent to the corresponding process to be rotated again. Processes are mapped to processors depending on a predefined cost function, in order to minimize the communication cost. In this paper, we have ussually taken into account a round robin or cyclic distribution.

Empty rows are discarded and the algorithm finishes when there is at most a single row in each process. As the rotation of a pair of rows cannot produce a row of a lower type, a token is passed through all the processes to determine when the triangularization program has finished.

Our machine model allows that the communication is carried out asynchronously and in parallel with the processing.

The basic algorithm executed by a given process, is the following:

```
repeat
     if received()
        then begin
               receive rows from other processes
               insert rows in local matrix
             end
     if rows_counter>1
        then begin
               extract two rows from local matrix
               calculate and apply a rotation
               insert the first row in the local matrix
               calculate destination process of second row
               send second row to destination process
             end
until finished
```

It is interesting to note here that the algorithm has not a regular communication pattern. The communication pattern is not fixed and is varying along the time. Moreover, there is no locality of communications. So, it is very difficult to choose a good topology that can adjust this communication pattern. A deep study of this algorithm can be found in [9].

3 A Reconfigurable Multicomputer

Multicomputers rely on an efficient interconnection network. The network is a critical component because performance is very sensitive to network latency and throughput. In our work, we consider a multicomputer with a control-flow mechanism called wormhole routing [3]. Interconnection networks with wormhole routing mechanism are insensitive to the communication distance, but they are fairly sensitive to conflicts on the same link, that is, the congestion problem. Several techniques have been proposed to reduce or avoid congestion, such as virtual channels, random routing or message combining. The technique here presented is related to the reconfiguration capacity of the interconnection network topology.

The dynamic reconfiguration of the interconnection network is a solution adopted in order to reduce the cost of the communication. Basically, consists in placing the different processors in those positions in the network which, at each computational moment and according to the existing communication pattern among them, are more adequated for the development of such computation.

The basic idea is the following: when messages arriving by a given channel to their destination nodes have supported an important delay, the reconfiguration algorithm will try to put the destination node close to the site that is producing those delays, by exchanging its position with its neighbour more closed to the conflict zone.

A reconfigurable network has the following advantages:

- Programming a parallel application becomes more independent of the target architecture because the architecture adapts to the application.

- This feature provides the flexibility required for an efficient execution of various applications. Moreover, in this way it is easy to exploit the locality in communications.
- Finally, this feature is very well suited for parallel applications where communications pattern varies over time.

A reconfigurable network is controlled by a reconfiguration algorithm. There are two types of reconfiguration: static or dynamic. In this paper, we focus on dynamic reconfiguration, that is, the topology can change almost arbitrarily at run-time. We present the results we have obtained with this reconfiguration algorithm [6] for a numerical problem: the fast Givens rotations.

The algorithm we have developed for the dynamic network reconfiguration has the following properties: *uses global reconfiguration* (several changes can be carried out in one step), *preserves the topology* (after reconfiguration, the network has the same topology), *is based on contention network* (a node can reconfigurate the network taking into account information about the contention in the network), *produces a small alteration* (a node can only make an exchange with one of its neighbour nodes) and *uses three thresholds for network reconfiguration*.

Figure 1 shows the effects of this technique for a very elemental situation. In (a) and (b) messages sent by processors must in some situations go through the same channels until they reach their destination, originating logical delays in communication. Once situation (c) is reached, this problem disappears, thus accelerating emission and reception of these messages. The processor which receives the messages manages to place itself in its best position in the network. This is achieved by changing the situation of this processor by means of small alterations in the network, that is, exchanging its position with a neighbour processor.

4 Performance Evaluation

In this section we are going to evaluate and analyse the results obtained after we have applied the algorithm for the dynamic reconfiguration of the interconnection network at the parallel triangularization of a sparse matrix using fast Givens rotations.

Fast Givens rotations algorithm produces communication among the different processors of the multicomputer through its interconnection network which is mainly due to the movement of rows in the matrix from one process to another. It seems therefore obvious to try to reduce the negative effect introduced by that communication in the total time of execution of the algorithm.

This algorithm has been chosen because we cannot know a priori the communication pattern between nodes, because it depends on the structure of the sparse matrix, and therefore a suitable topology cannot be selected. Moreover, the communication pattern will vary over time.

For dense matrices, the rotation of two rows of type t produces two rows of types t and $t+1$ respectively. Then, this algorithm performs very well on a ring.

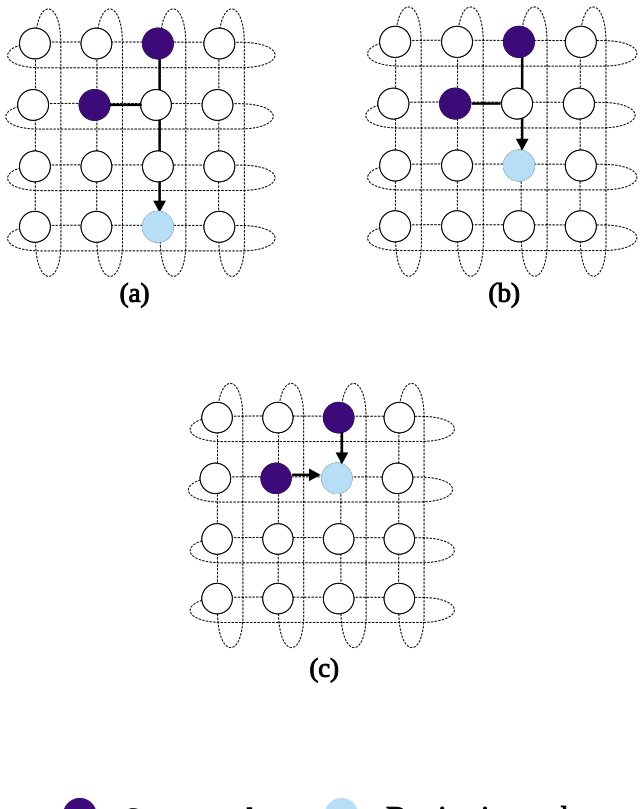

Fig. 1. An example of the reconfiguration algorithm effects

For sparse matrices, on the other hand, we cannot know a priori the communication pattern. As processing advances, matrix fill-in increases, approaching the behaviour of denses matrices.

Simulation has been used because it is difficult to analyse theoretically the arithmetic and communication complexities when sparse matrices are processed. This is specially true when the matrix structure changes dynamically during the processing, as is the case of Givens rotations. In this case, experimental results are needed to evaluate the performance of the algorithm.

The evaluation methodology we have used is summarized in the following sections.

4.1 Programming and Simulation Environment

The results we present have been obtained with Pepe, a Programming Environment for Parallel Execution [7]. Pepe takes a parallel program as input and generates intermediate code for an execution on a multicomputer. The most important parameters of this multicomputer can be varied by the user. Pepe generates performance estimates and quality measures for the interconnection network.

Pepe provides a user-friendly visual interface for all phases of parallel development. This graphical interface has been designed with the aim of keeping it really immediate and comfortable to the user, following the styles adopted nowadays by most of the human-oriented interfaces. In our environment, the user gets tools for easy experimentation both different parallelization possibilities and different network parameters. With this methodology the programmer can change very quickly parallelization strategies and evaluate this parallelization with analysis tools.

Pepe has two main phases and several modules within it. The first phase is language-oriented, and it allows us to code, simulate and optimize a parallel program. In this phase, interactive tools for specification, coding, compiling, debugging and testing were developed. This phase is architectural independent and it is the front-end of our environment. The second phase has several tools for mapping and evaluating the reconfigurable architecture. We can vary several parameters as different interconnection topologies or routing algorithms. The link between two phases is an intermediate code that is generated as possible result of the first phase. This allows that the user can handle our environment as whole or each phase singly. For example, we can execute only the first phase for testing the parallel behaviour of an algorithm on an ideal multicomputer. The other possibility is to obtain an intermediate code from a key parallel algorithm. Then, we can execute several times the second phase with different network parameters to evaluate them for this key algorithm.

4.2 Characteristic of the Testing Matrices

The testing matrices have been generated at random by making a homogeneous distribution of their non-zero elements. Large size matrices have been selected to produce a large message traffic to better appreciate in this way the advantages of the reconfiguration algorithm. For a specific number of columns in the matrix, tests with different rectangularity factors have been carried out for a minimum value of two, since smaller factors can hardly produce traffic in intermediate nodes of the network. An average number of two non-zero elements per row has been taken in order to ensure we are dealing with sparse matrices. Most of the results included in this work refer to matrices of 2400 rows and 1200 columns.

In order to increase the efficiency of the parallel algorithm, a column rearrangement in the matrix is usually made before applying Givens rotations. This is directed to attain that the first columns in the matrix are those with the lowest number of non-zero elements.

4.3 Performance Measures

The most important performance measures are delay and throughput. Delay is the additional latency required to transfer a message with respect to an idle network. It is measured in clock cycles. The message latency lasts since the message is introduced in the network until the last flit is received at the destination node. An idle network means a network without message traffic and, thus without channel multiplexing. Throughput is usually defined as the maximum amount of information delivered per time unit. It is measured in flits per clock cycle.

4.4 Results

In this section, we show some evaluation results. These results have been obtained making use of a hipercube topology. We choose this topology because is that we have obtained the best results for the static case.

Figure 2 shows the throughput as a function of network size. It can be seen that the algorithm achieves a higher throughput for the whole range of network sizes which have been analysed. In similar way we have obtained other graphs showing the average message delay as a function of network size. All of them clearly show that the dynamic reconfiguration of the network scales well with network size. Figure 3 shows the average message delay versus message length.

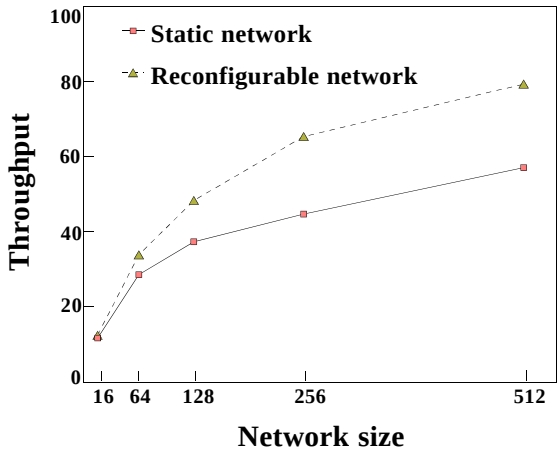

Fig. 2. Throughput as a function of network size

The curves correspond to different values for the number of virtual channels on a 256-node network: Static algorithm (Static 1vc), dynamic algorithm (Dynamic 1vc), static algorithm with two virtual channels (Static 2vc) and dynamic algorithm with two virtual channels (Dynamic 2vc).

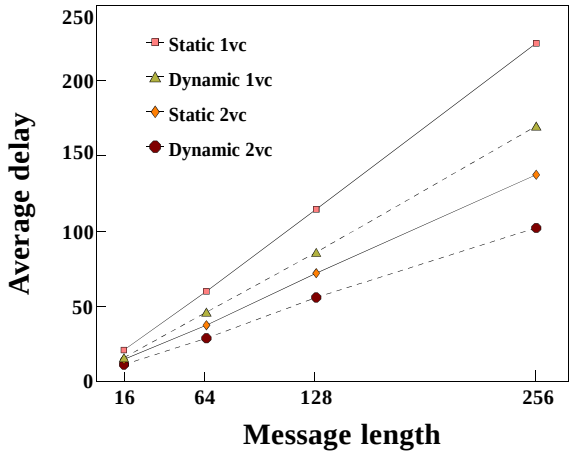

Fig. 3. Average message delay as a function of message length

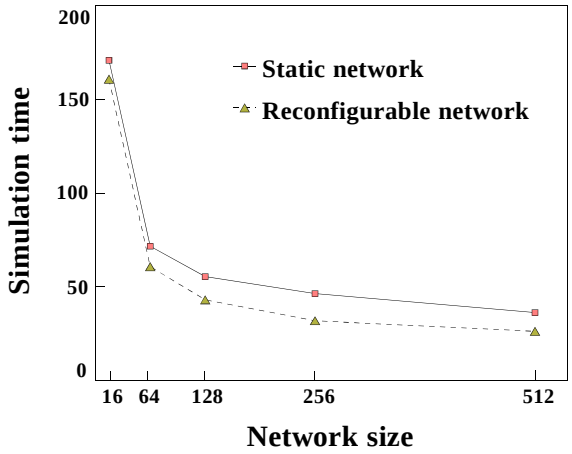

Fig. 4. Total simulation time as a function of network size

The results show a significant reduction in message delay, especially for long messages.

Finally, figure 4 shows the simulation time as a function of the network size. It is noticed that the reduction of the total time of the computation in the triangularization process is kept for the different sizes of the network.

5 Conclusions and Future Work

In this paper we show the advantages of using a reconfigurable multicomputer for solving a great variety of problems. In our case, the triangularization of sparse matrices by means of fast Givens rotations has been selected and we present the results here.

The evaluation of these results shows a reduction in message delay with respect to the static case. The improvement is more noticeable when the messages have a larger length, and the algorithm scales very well with network size.

As regards future work we want to evaluate the reconfiguration algorithm for larger matrices and larger networks and also other parallel programs.

References

1. Adamo, J., Bonello, C.: Tenor++: A dynamic configurer for Supernode machines. Lecture Notes in Computer Science. No. 457, pp. 640–651, Springer Verlag (1990)
2. Bauch, A., Braam, R., Maehle, E.: DAMP: A dynamic reconfigurable multiprocessor system with a distributed switching network. 2nd European Distributed Memory Computing Conference, Munich, April, 1991
3. Dally, W.J., Seitz, C.L.: Deadlock-free message routing in multiprocessor interconnection networks. IEEE Trans. on Computers, Vol. C–36, No. 5, pp. 547–551, May, 1987
4. Duato, J.: Parallel triangularization of a sparse matrix on a distributed–memory multiprocessor using fast Givens rotations. Linear Algebra and its Applications, 121:582–592, 1989
5. Fraboul, Ch., Rousselot, J.Y., Siron, P.: Software tools for developing programs on a reconfigurable parallel architecture. in D. Grassilloud and J.C. Grossetie (Eds.), Computing with Parallel Architectures: T. Node, pp. 101–110, Kluwer Academic Publishers, 1991
6. García, J.M., Duato, J.: Dynamic reconfiguration of multicomputer networks: Limitations and tradeoffs. in P. Milligan and A. Nunez (Eds.), Euromicro Workshop on Parallel and Distributed Proces., IEEE Computer Society Press, pp. 317–323, 1993
7. García, J.M., Sánchez, J.L., Duato, J., Fernández, J.: Pepe: A trace-driven simulator to evaluate reconfigurable multicomputer architectures. Technical Report DIS TR 4–95, University of Murcia, March, 1995
8. Golub, G.H., Van Loan, C.F.: Matrix computations, North Oxford Academic, 1983
9. Sánchez, J.L., García, J.M.: Estudio de la reconfiguración dinámica de la red. Evaluación de nuevas propuestas para mejorar su eficiencia. Technical Report DIS TR 13–94, University of Murcia, October, 1994