Pipelining the Givens Linear Solver on Distributed Memory Machines

J. Papay, M.J. Zemerly and G.R. Nudd

Parallel Systems Group, Department of Computer Science,
University of Warwick, Coventry CV4 7AL, UK
Email: {yuri, jamal, grn}@dcs.warwick.ac.uk
Tel: +44 1203 523 523 ext 2863
Fax: +44 1203 525 714

Abstract

Pipelining is normally associated with shared memory and vector computers and rarely used as an algorithm design technique for distributed memory architectures. In this paper we show how pipelining enables communication and computation to be overlapped on a distributed memory parallel computer (128 T800 Parsytec SuperCluster) yielding a significant speedup. A linear solver based on Givens rotations is selected and parallelised using two different techniques. A non-overlapping algorithm using collective communication, such as optimised broadcast and collection, is compared with a pipelined (overlapping) algorithm using only simple point-to-point communications between neighbouring processors. Both algorithms use the same computational modules which have been identified and extracted from the sequential code.

1. Introduction

The aim of parallelisation is to reduce the execution time of sequential programs. The first step in parallelisation process is the selection of the parallelisation technique. As a result of this step the data space and the computation are partitioned. Although the partition reduces the execution time it also introduces communication and synchronisation overheads which are not present in sequential programs. The computation can be optimised by selecting an appropriate parallelisation strategy which yields good load balancing. Reduction of the communication time can be achieved by using techniques such as locality of access, avoidance of routing hotspots in the network and overlap the computation with communication. One efficient technique for parallelisation is functional pipelining which partitions the control flow into stages and share the same data flow. This is most effective if all the stages perform a similar amount of computation such that all stages of the pipeline are busy [6]. Data parallelism is a commonly used technique which assumes that the same or similar operations are performed on a large amount of data [4]. The data decomposition is an important factor which influences the load balance, the amount and pattern of communications and the degree of parallelism.

Linear solvers using pipeline on a distributed memory machine has not been frequently addressed. Typically pipelining is associated with shared memory and vector computers and has rarely been used as an algorithm design technique for distributed memory architectures [5]. Chu et al.[3] described a Gaussian elimination with partial pivoting and row-oriented data decomposition. This data decomposition makes pivoting difficult because the rows must be swapped between processors which generates extra communication overhead. A good processor utilisation by using systolic arrays for solving the Gauss-Jordan algorithm with partial pivoting has been reported by Wyrzykowky et al.[9].

The parallel solution of a linear solver based on Givens rotations has not received a substantial attention in the literature. In this paper we present two different algorithms for this problem implemented on a mesh connected distributed memory parallel computer (128 T800 Parsytec SuperCluster). The first is a non-overlapping solution and uses broadcast and collection communication routines. The second is a pipelined solution and uses only simple point-to-point communications and enables the computation and communication to be overlapped. This yields a significant speedup as will be demonstrated in this paper. Although the parallel work is organised in different ways, both algorithms are built from the same set of computational modules. We also introduce here a "Processor Activity Graph" (PAG) which provides a graphical representation of the parallel system runtime behaviour. This representation helps in understanding the parallel algorithm behaviour, shows the overlap between computation and communication, level of parallelism, load balance and enables to predict the execution time but this is beyond the scope of this paper.

2. Linear solver based on Givens rotations

Linear solvers are important for many applications such as computational chemistry, engineering, simulation and design. Linear solvers belong to the computational and communication intensive class of algorithms which can be represented in the following form.

$$P\,X = Q \tag{1}$$

where P is a non-singular square matrix, Q is the right hand side vector and X is a vector of unknowns. There are several solution methods for the system of linear equations which can be classified as direct and iterative methods. In the case of direct methods the amount of computation can be specified in advance, whereas for the iterative methods the number of computation steps depends, on the value of the initial solution vector and the required precision. Typical examples of direct methods are Gaussian elimination with back-substitution, Gauss-Jordan elimination and LU factorisation [1,2,7,8]. All these methods require pivoting which is difficult to parallelise [7]. Givens rotation is particularly interesting since it does not require pivoting and is known to be numerically more stable and inherently more accurate than the other methods. Givens transformation is defined by a 2×2 rotation matrix:

$$G = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \tag{2}$$

where $c^2 + s^2 = 1$

A Givens rotation is used to eliminate the elements of a vector or a matrix as follows:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix} \tag{3}$$

where

$$c = \frac{a}{\sqrt{a^2 + b^2}} \quad \text{and} \quad s = \frac{b}{\sqrt{a^2 + b^2}}$$

The Givens method requires more computation but it is easier to parallelise than Gaussian elimination which is difficult because of pivoting. Both these algorithms produce an upper triangular matrix which can be solved by back-substitution. The complexity of back-substitution is $O(N^2)$ compared with the triangulation phase which is $O(N^3)$.

3. Parallelisation of the Givens linear solver

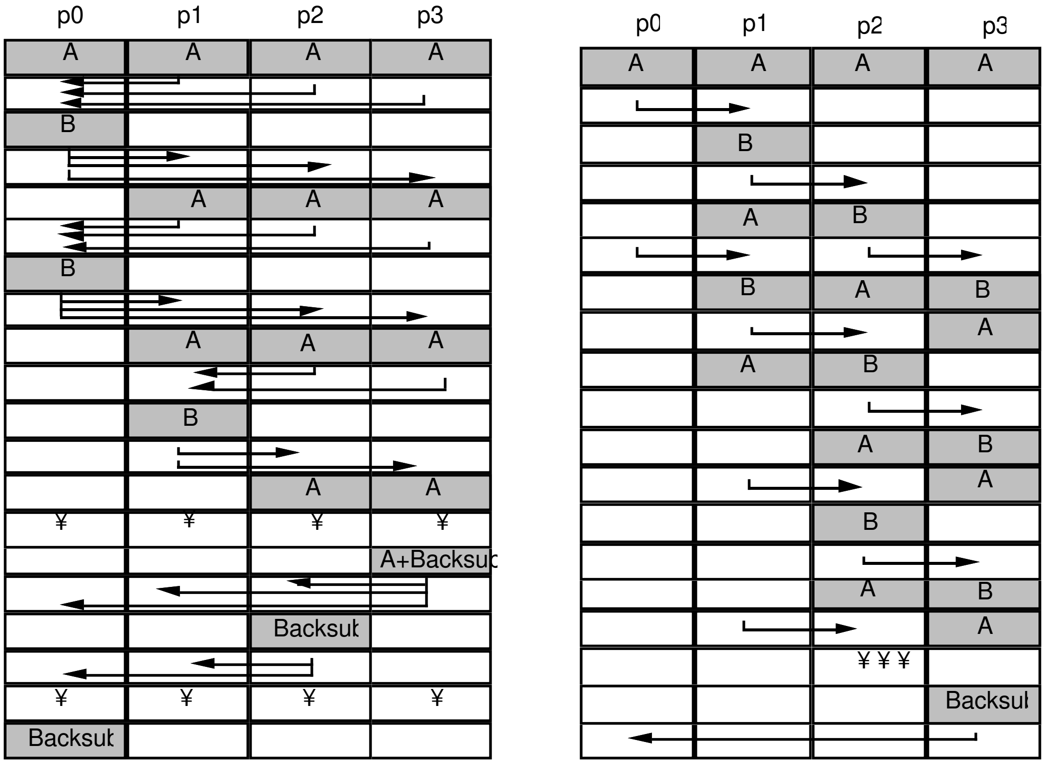
In the parallel Givens algorithm several orthogonal rotations can be applied simultaneously. The basic question is how to choose the sequence of rotations in order to minimise the amount of computations and the communication overhead. For the parallelisation a block-row data decomposition is considered which partitions the original matrix horizontally and assigns adjacent blocks of rows to neighbouring processors. As a result of this decomposition each processor obtains a matrix A(N/p,N+1). Matrix B is an auxiliary matrix used in the elimination process. The size of matrix B for the non overlapping algorithm is (p,N+1) and for the pipelined algorithm is (2,N+1) respectively. The parallel algorithms for non-overlapping and pipelined versions are given in Figure 1.

```
* Non-overlapping Givens solver *
colum = 0
for(master=0;master<p-1;master++){
  for(i=0;i<=N/p-1;i++, colum++){
    if(my_id == master){
      eliminate_current_column_of_matrix_A
      collect_rows_from_processors
      put_rows_into_matrix_B
      eliminate_current_column_of_matrix_B
      scatter_rows_to_the_processors}
    if(my_id > master) {
      eliminate_current_column_of_matrix_A
      send_row_to_master
      receive_updated_row_from_master }}}
if(my_id==p-1)
  eliminate_remaining_columns_of_A
for(i=p-1;i>0;i--){
  if(my_id == i){
     backsubstitute
     broadcast}
  if(my_id < i)
      receive}
 if(my_id == 0)
    backsubstitute
```

```
* Pipelined Givens solver *
colum = 0
for(master=0;master<p-1;master++){
   for(i=0;i<=N/p-1;i++, colum++){
      if(my_id == master){
          eliminate_current_column_of_matrix_A
          send_row_to_processor(my_id+1)}
      else {
              if(my_id > master ){
                 receive_row_from_processor(my_id-
                 eliminate_colomn_matrix_B
                 send_to_processor(my_id+1)}}}}
if(my_id ==0)
    receive_solution_vector
if(my_id ==p-1){
    eliminate_remaining_column_of_matrix_A
    backsubstitute
    send_solution_vector_to_processor(0)}
```

Figure 1 - Non-overlapping and pipelined Givens algorithms

The Processor Activity Graphs for non-overlapping (a) and pipeline (b) algorithms are given in Figure 2, where the hashed boxes represent computation, the empty idle processors and the arrows communication.

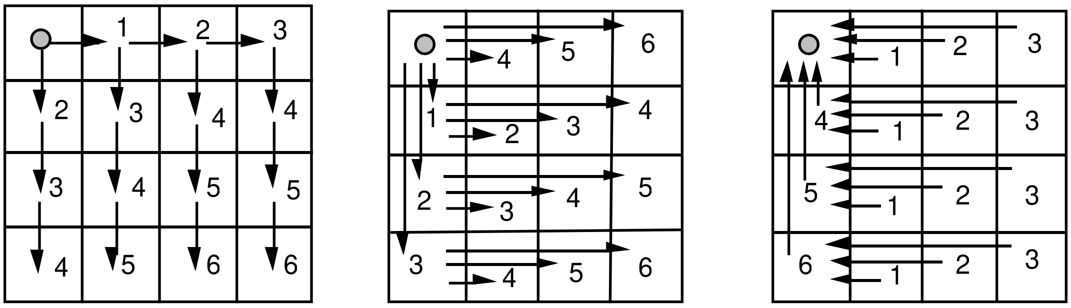


a)

b)

Figure 2 - Processor activity graph for non-overlapped (a) and pipelined (b) Givens linear solvers

The back-substitution for the pipelined algorithm is performed on a single processor (the las processor) and the solution vector is sent to processor p_0 It is worth noting that the non overlapping algorithm uses optimised collective communication modules. The scheduling of communication steps for optimised broadcasting, scattering and colection on a 4x4 processor gri is given in Figure 3, where the sequence of communication steps is marked by numbers.



(a)

(b)

c)

Figure 3 - Sequence of communication steps for optimised broadcast (a), scattering and collection (c)

4. Results

The Givens linear solver algorithm has been implemented on the Parsytec SuperCluster and the execution times have been measured for various matrix sizes (128, 256 and 512) and number of processors . Figure 4 shows the measurements in a logarithmic scale and Figure 5 the speedup for the non-overlapped (n) and pipeline (p) linear solvers.

N\p	1	2	4	8	16	32	64	128
non_overlapped(128)	13.65	9.5	5.79	4.32	4.92	9	18.81	38.54
non_overlapped(256)	106.93	74.66	42.26	26.84	24.22	37.41	73.55	151.99
non_overlapped(512)	846.59	586.68	322.72	185.13	135.35	167.23	297.83	590.1
pipeline(128)	13.65	9.61	6.29	3.02	1.79	1.3	1.16	1.38
pipeline(256)	106.93	74.45	48.85	22.14	12	7.33	5.26	4.74
pipeline(512)	846.59	585.88	385.2	169.65	87.72	48.3	29.62	21.47

Table 1 - Execution time measurements for the Givens linear solver

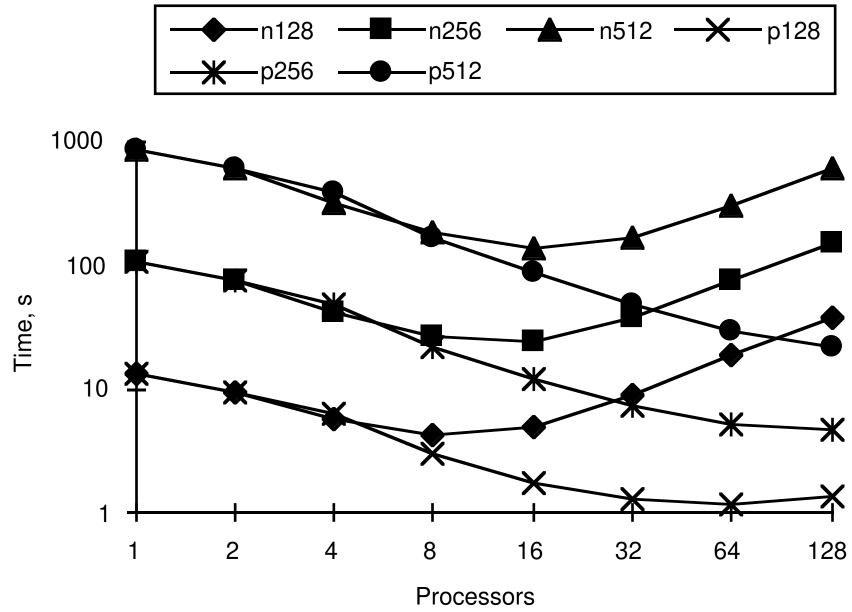


Figure 4 - Execution time for non-overlapped (n) and pipelined (p) Givens linear solver

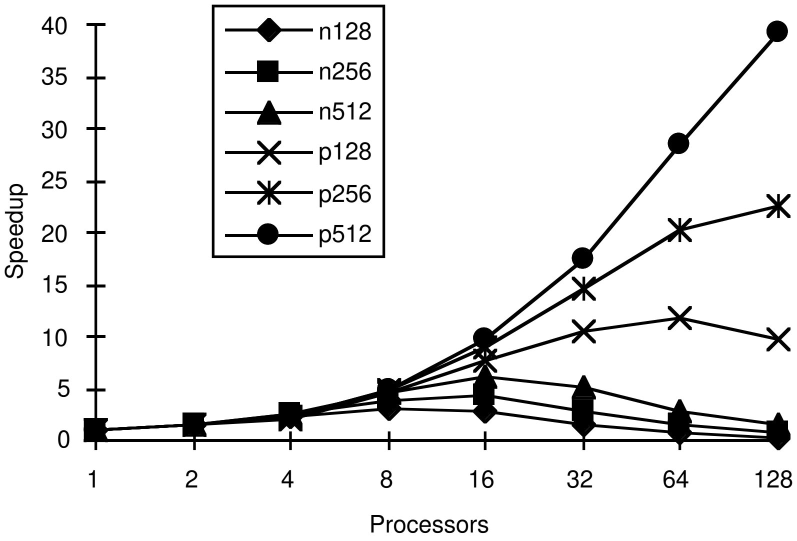


Figure 5 - Speedup for non-overlapped (n) and pipelined (p) Givens linear solver

Figure 4 and 5 show that the pipeline method is superior to the non-overlapping algorithm because it enables to overlap computation with communication and also reduces the idle time. Figure 4 shows that the execution time for the non-overlapping algorithm increases after 8 or 16 processors depending on the matrix size due to the communication overhead while the pipeline algorithm provides a good scalability for larger number of processors. Figure 4 also shows that the non-overlapped method performs better for 4 processors since its communication overhead is less than that of the pipeline. The pipeline method for 512x512 matrix on 128 processors gives a maximum speedup 39.43 in contrast with the non-overlapping method where the speedup is about 6.25.

5. Summary

This paper presented a linear solver based on Givens rotations which has been parallelised using a non-overlapping and pipelined algorithms. A Processor Activity Graph (PAG) has also been introduced which is a useful graphical representation of the parallel algorithm runtime behaviour. PAG can be used for algorithmic analysis and performance predictions at the early stages of program design. The comparison of two algorithms showed that the pipelining technique is superior to the non-overlapping because it enables the communication with computation to be overlapped.

Acknowledgement

This work is funded by the ESPRIT project 6942 (PEPS) - Performance Evaluation of Parallel Systems. The authors wish to thank Mr. Trevor Chambers (National Physical Laboratory) for the valuable discussions of this work.

Bibliography

[1] S. Barnett Matrices Methods and Applications, Clarendon Press, Oxford 1990.

[2] E. R. Champion. Numerical Methods for Engineering Applications, Marcel Dekker, Inc, 1993.

[3] E. Chu, A. George. Gaussian Eliminination with Partial Pivoting and Load Balancing on A Multiprocessor. Parallel Computing 5(1987), 65-74, North-Holland.

[4] G. C. Fox et. al. Solving Problems on Concurrent Processors, Prentice Hall, 1988.

[5] R. W. Hockney, C.R. Jesshop: Parallel Computers 2, Architectue, Programming and Algorithms., Second edition, Institute of Physics Publishing, Bristol, 1988.

[6] Z. Hussain Digital Image Processing: Practical Applications of Parallel Processing Techniques, Ellis Horwood Ltd., UK, 1991.

[7] J. J. Modi. Parallel Algorithms and Matrix Computation, Clarendon press, 1988.

[8] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press, 1988.

[9] R. Wyrzykowski, J.S. Kanevski, H. Piech. One-Dimensional Processor Arrays for Linear Algebraic Problems. IEE Proc. Computers, Digital Techniques, Vol. 142, No. 1, January 1995, pp 1-4.