The parallel solution of sparse linear equations

Iain S. Duff
Computer Science and Systems Division
AERE Harwell
Didcot, Oxon
England, OX11 0RA

Abstract

We discuss the solution of large sparse systems using Gaussian elimination on both local and shared memory parallel computers.

There is a natural parallelism to Gaussian elimination that has been frequently exploited. We can take advantage of this parallelism in addition to that provided by the sparsity itself. We discuss this latter parallelism in some detail.

We discuss an approach that exploits the parallelism due to the sparsity and that can automatically benefit also from the parallelism of Gaussian elimination. This approach, which is applicable to quite general systems, is based on a multifrontal technique.

We look at the implementation of the multifrontal approach on shared memory machines and discuss its implementation on a hypercube.

1 Introduction

A system of linear equations is called sparse if most of the entries of its coefficient matrix are zero. The solution of large sparse systems of linear equations is one of the major calculations in large-scale computing in science and engineering. It is thus of great importance to study the performance of existing and new solution techniques on the range of parallel processors which are currently or soon to be marketed.

In this paper, we consider direct methods for the solution of sparse systems. We see that these techniques, based on Gaussian elimination, exhibit a very natural small granularity parallelism even when used in the solution of full systems (that is systems where all entries of the coefficient matrix are treated as nonzero). Further parallelism can be obtained from the sparsity, and it is primarily the exploitation of this that we will discuss. We examine Gaussian elimination parallelism in Section 2 and illustrate the phenomenon of parallelism due to sparsity in Section 3. We discuss a general technique for automatically using both forms of parallelism in Section 4 and consider its implementation on both shared and local memory machines in Sections 5 and 6 respectively. We draw some conclusions in Section 7.

In this presentation, we have assumed that our readership has a background in computer science but not necessarily in numerical analysis. We have therefore defined the principal numerical analysis terms when they are first used.
2 Parallelism in Gaussian elimination

Gaussian elimination involves factorizing a matrix $A$ into the product $A = LU$ of a lower triangular matrix $L$ and an upper triangular matrix $U$ so that the solution to the set of linear equations $Ax = b$ can be easily effected by solving the triangular system $Ly = b$ followed by the triangular system $Ux = y$. In the case of full matrices $A$, the work for the factorization is $O(n^3)$ for a matrix of order $n$, while the solution of the triangular factors only involves $O(n^2)$ work. Although the factorization of a sparse matrix has a much lower operation count and is dependent on the number and distribution of nonzeros (the sparsity pattern), it is still true that the factorization step is much more costly than the solution phase. We therefore concentrate on the factorization step in this paper.

Assume that the coefficient matrix $A$ has entries $a_{ij}$, $1 \leq i, j \leq n$. Gaussian elimination on a system of order $n$ consists of $n-1$ major steps. Major step $k$, $1 \leq k \leq n-1$, involves the selection of a pivot $a_{kk}$. This process of selection, called pivoting, is made to ensure good numerical performance and is followed by the elimination of the entries $a_{k+1,k}, \ldots, a_{nk}$ by adding appropriate multiples of row $k$ to rows $k+1, \ldots, n$ producing the updating operations

$$a_{ij} = a_{ij} - a_{ik}[a_{kk}]^{-1}a_{kj} \quad k < i, j \leq n \quad (2.1)$$

on rows and columns $k+1$ to $n$ of the matrix (the reduced matrix). The important feature is that each update to an entry $a_{ij}$ is independent so that they can all be performed in parallel.

If each single update operation is considered separately then the amount of parallelism is large ($(n-1)^2$ processes at the first stage) but the communication costs are high since each process must have the entries necessary for forming its triple product $(2.1)$. Indeed, the biggest problem in Gaussian elimination on a local memory machine is that the pivot row $(a_{k*})$ and the pivot column $(a_{*k})$ or the multipliers $(a_{kk}[a_{kk}]^{-1})$ must be communicated to all other rows or columns of the matrix. The most common way to overcome this is to treat all the operations updating a submatrix or a column or row as a single unit, thus increasing the granularity although decreasing the number of parallel processes. Many authors have suggested doing this for the Cholesky factorization (Gaussian elimination on a symmetric $A$ to produce the factorization $A = LL^T$) on a hypercube both by subdiving $A$ into submatrices and assigning a submatrix to each processor (Saad 1986) or by splitting the matrix into columns and assigning columns to processors either in a wrap around fashion or by contiguous blocks (Geist and Heath 1985). It is fairly easy to incorporate pivoting with this approach, advantage being taken of its lower order of complexity relative to the $(2.1)$ update (Geist 1985). It is also possible to design a sparse version of the Cholesky factorization although present experience with it is disappointing (George et al. 1986a, 1986b). In Section 4 we consider a more general technique for sparse systems.

For surveys of parallel algorithms for full systems we refer the reader to Heller (1978) or Sameh (1983).

3 Parallelism due to sparsity

In Gaussian elimination on full systems, the major steps must be done sequentially. Although it is not necessary to wait until the completion of the first step before commencing the second, the second pivot must be updated before it can be used. Similarly, the third pivot must have been updated by both first and second steps before it can be used. The wavefront approach to Cholesky decomposition (Kung et al. 1981, O'Leary and Stewart 1985) uses pivots as soon as they are updated but must still process the pivots in order. However, if sparsity is present, this constraint of sequential pivoting does not necessarily hold.

The easiest way to see the influence of sparsity in removing the sequential pivot restriction is to consider the simple example of a tridiagonal matrix, arguably the most common sparse matrix.