Short Communications / Kurze Mitteilungen

Parallel Methods for Tridiagonal Equations

D. Kaur and Y. Wallach, Detroit

Received December 3, 1987

Abstract — Zusammenfassung

Parallel Methods for Tridiagonal Equations. Three methods of solving tridiagonal linear sets are adapted for execution on an Alternating Sequential-Parallel system. The first two are similar to “marching”, the third is based on “red-black” ordering. Examples and results show the last method to be the best.


The tridiagonal set of linear equations

\[
\begin{bmatrix}
 a_1 & b_1 \\
 c_2 & a_2 & b_2 \\
 & c_3 & a_3 & b_3 \\
 & & \ddots & \ddots & \ddots \\
 & & & \cdots & c_k & a_k & b_k \\
 & & & & \ddots & \ddots & \ddots \\
 & & & & & \cdots & c_{n-1} & a_{n-1} & b_{n-1} \\
 & & & & & & \cdots & c_n & a_n
\end{bmatrix}
\begin{bmatrix}
 x_1 \\
 x_2 \\
 x_3 \\
 \vdots \\
 x_{n-1} \\
 x_n
\end{bmatrix} =
\begin{bmatrix}
 d_1 \\
 d_2 \\
 d_3 \\
 \vdots \\
 d_{n-1} \\
 d_n
\end{bmatrix}
\]

may be solved iteratively, using the Gauss-Seidel method. In every iteration “i”, an \( x_k \) is recomputed through

\[
x_k^{(i)} := d_k - c_k * x_{k-1}^{(i)} - b_k * x_{k+1}^{(i-1)}; \quad k = 1, \ldots, n.
\]

Here, we have assumed that the set was “scaled” i.e. equation “k” was divided by \( a_k \) making \( a_k = 1 \). The set is considered solved if for an assumed accuracy \( \varepsilon \)

\[
\sum (x_k^0 - x_k^{(i-1)})^2 < \varepsilon.
\]
Since the process may be rather time-consuming, parallelization is called for. In this paper, the Alternating Sequential-Parallel, or ASP-System will be used. It was described in [1,2] and is assumed to be known. It has a "master" and p slaves numbered q = 1, ..., p.

To calculate the speedup of the algorithms, we count the operations and assume that a single multiplication, division, addition or subtraction requires the time of \( \omega \). Eqs. (2, 3) yield for a single iteration and a single processor, the "time" (actually, the operations count)

\[
t_1 = 7 \times n \times \omega. \tag{4}
\]

Suppose that there are \( n = 90 \) equations and an ASP with \( p = 3 \). An algorithm for parallel solution could proceed as follows: In iteration \( i = 1 \), slave \( q = 1 \) solves Eq. (2) for \( k = 1 \) to 30.

\[
x_{30} := d_{30} - c_{30} \times x_{29} - b_{30} \times x_{31} \tag{5}
\]

the value of \( x_{31}^{(0)} \) must be retrieved from slave \( q = 2 \). In iteration \( i = 2 \), slave \( q = 1 \) repeats Eq. (2) for \( k = 1 \) to 30, while slave \( q = 2 \) computes \( x_k \) for \( k = 31 \) to 60, starting from

\[
x_{31} := d_{31} - c_{31} \times x_{30}^{(1)} - b_{31} \times x_{32}^{(0)} \tag{6}
\]

for which \( x_{30}^{(1)} \) has to be transferred. In the same way, slave \( q = 3 \) waits for \( x_{60} \) to calculate Eq. (2) for \( k = 61 \) to 90.

The timing picture of the first few iterations is shown in Fig. 1a. Starting from iteration \( i = 4 \), all three slaves work. Note though that if slave \( q = 1 \) works on iteration \( i \), slaves 2 and 3 work on iterations \( i-1 \) and \( i-2 \) respectively. If many iterations are required — and this is usually the case — then this time-shift is insignificant.

At the start of iteration \( i = 2 \), slave \( q = 1 \) could calculate the \( x \)'s going "upwards", i.e. \( x_{29}, x_{28}, ..., x_2, \) and \( x_1 \). In iteration \( i = 3 \), slave \( q = 2 \) will iterate "up". From that iteration on, all slaves are moving either up or down (Fig. 1 b). The algorithm is therefore called "do-up", while that of Fig. 1a was called "do-do".

For the particular case of a tridiagonal matrix \( A \), it can be shown that for two iterations and \( p = 3 \)

\[
t_3 = 10 \times n \times \omega. \tag{7}
\]