Parallel Solution of Linear Recurrence Systems

The paper describes some recently proposed divide-and-conquer parallel algorithms for solving linear recurrence systems. Such systems arise in many computational problems. A special case - solving linear systems with constant coefficients - is also discussed. As an example of an application of such linear systems, parallel algorithms for finding trigonometric sums are presented.

1. Introduction

Let us consider a linear recurrence system of order m for n equations:

\[
x_k = \begin{cases} 
0 & \text{if } k \leq 0, \\
 f_k + \sum_{j=k-m}^{k-1} a_{kj} x_j & \text{if } 1 \leq k \leq n.
\end{cases}
\]  

(1)

Many parallel algorithms for solving this problem have been proposed [1-9]. In [14] two efficient medium-grain divide-and-conquer parallel algorithms were introduced and later implemented on a Sequent [11]. These algorithms have very good numerical properties [15]. Recently, a modified algorithm has been proposed to handle the special case of the system (1) - a system with constant coefficients [12]:

\[
x_k = \begin{cases} 
0 & \text{for } k \leq 0, \\
 f_k + \sum_{j=1}^{m} a_{kj} x_{k-j} & \text{for } 1 \leq k \leq n.
\end{cases}
\]  

(2)

2. Parallel Algorithms

First, let us consider the general linear recurrence system (1). Computation of values \( x_k \) can be represented as a solution of a system of linear equations:

\[
(I - A)x = f \quad \text{where } I, A \in \mathbb{R}^{n \times n} \text{ and } x, f \in \mathbb{R}^n,
\]  

(3)

where \( x = (x_1, \ldots, x_n)^T \), \( f = (f_1, \ldots, f_n)^T \) and \( A = (a_{ik}) \) with \( a_{ik} = 0 \) for \( i \leq k \) or \( i - k > m \).

Without loss of generality we can assume that \( n \) is divisible by \( p \) (where \( p \) is the number of processors) and \( q = n/p > m \). The system (3) can be written in the following block form:

\[
\begin{pmatrix}
L_1 & U_2 & L_3 & \cdots & U_p
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2 \\
\vdots \\
x_{p-1}
\end{pmatrix}
= \begin{pmatrix}
f_1 \\
f_2 \\
\vdots \\
f_{p-1}
\end{pmatrix},
\]  

(4)

where \( L_j \in \mathbb{R}^{q \times q}, x_j \in \mathbb{R}^q, f_j \in \mathbb{R}^q \) for \( j = 1, \ldots, p \) and \( U_j \in \mathbb{R}^{q \times q} \) for \( j = 2, \ldots, p \). Vectors \( x_j \) in (4) satisfy the following recurrence relation:

\[
\begin{cases} 
x_1 = L_1^{-1} f_1, \\
x_j = L_j^{-1} f_j - L_j^{-1} U_j x_{j-1} \quad \text{for } j = 2, \ldots, p.
\end{cases}
\]  

(5)
Let \( U_j^k \) denote the \( k \)th column of the matrix \( U_j \). Since \( U_j^k = 0 \) for \( j = 2, \ldots, p \) and \( k = 1, \ldots, q - m \) we obtain the following formula for Algorithm 1:

\[
\begin{align*}
  x_1 &= L_1^{-1} f_1, \\
  x_j &= z_j - \sum_{k=0}^{m-1} x_{(j-1)q - k} y_j^k \\
  &\quad \text{for } j = 2, \ldots, p,
\end{align*}
\]

where \( L_1 z_j = f_j \) and \( L_1 y_j^k = U_j^{q-k} \) for \( k = 0, \ldots, m - 1 \).

Let us now observe that each matrix \( U_j \) can be rewritten as:

\[
U_j = - \sum_{k=1}^{m} \sum_{i=k}^{m} \beta_{ij}^k e_k e_i^{T},
\]

where \( \beta_{ij}^k \) denotes the \( k \)th unit vector of \( \mathbb{R}^q \). Substituting into (6) we obtain the following formula for Algorithm 2 (for a more detailed description of both algorithms see [14]):

\[
\begin{align*}
  x_1 &= L_1^{-1} f_1, \\
  x_j &= z_j + \sum_{k=1}^{m} \alpha_{kj}^* y_j^k \\
  &\quad \text{for } j = 2, \ldots, p,
\end{align*}
\]

where \( \alpha_{kj}^* = \sum_{i=k}^{m} \beta_{ij}^k e_i^{T} \). Substituting into (6) we obtain the following formula for Algorithm 2 (for a more detailed description of both algorithms see [14]):

\[
\begin{align*}
  x_1 &= L_1^{-1} f_1, \\
  x_j &= z_j - \sum_{i=k}^{m} \alpha_{kj}^* y_j^k \\
  &\quad \text{for } j = 2, \ldots, p,
\end{align*}
\]

In the constant coefficient system (2) observe that blocks \( L_j \) and \( U_j \) simplify to \( L \) and \( U \) respectively and the recurrence relation (5) can be thus rewritten as:

\[
\begin{align*}
  x_1 &= L^{-1} f_1, \\
  x_j &= L^{-1} f_j - L^{-1} U x_{j-1} \\
  &\quad \text{for } j = 2, \ldots, p.
\end{align*}
\]

Note also that to compute vectors \( y^k, k = 1, \ldots, m \), we need to find only the solution of the system

\[
Ly^k = e_k,
\]

where

\[
y^1 = (1, y_2, \ldots, y_q)^T,
\]

and

\[
y^k = (0, \ldots, 0, 1, y_2, \ldots, y_{q-k+1})^T.
\]

The method [8] can be now modified to define Algorithm 3:

\[
\begin{align*}
  x_1 &= z_1, \\
  x_j &= z_j + \sum_{k=1}^{m} \alpha_{kj}^* y_j^k \\
  &\quad \text{for } j = 2, \ldots, p,
\end{align*}
\]

where \( L z_j = f_j \) for \( j = 1, \ldots, p \), and \( y^k \) is given by (11) and

\[
\begin{align*}
\alpha_{j}^1 &= \alpha_m x_{(j-1)q - m + 1} + \alpha_{m-1} x_{(j-1)q - m + 2} + \ldots + \alpha_j x_{(j-1)q} \\
\alpha_{j}^2 &= \alpha_m x_{(j-1)q - m + 2} + \ldots + \alpha_2 x_{(j-1)q} \\
\vdots \quad & \quad \vdots \\
\alpha_{j}^m &= \alpha_m x_{(j-1)q}.
\end{align*}
\]

Methods (6), (8) and (12) are examples of divide-and-conquer algorithms. First, several linear recurrence systems of order \( m \) for \( q \) equations are solved separately (in parallel). Second, a set of values is calculated in each
block and communicated to the next block thus effectively decoupling the original system. This is the sequential part of the algorithms. Finally, the solution to the original problem is calculated independently in each block (in parallel). Let us also observe that when Algorithm 3 is applied to the constant coefficient problem only $p + 1$ subsystems of order $m$ for $q$ equations must be solved in parallel, so the number of subsystems does not depend on the order of the recurrence system.

For all these algorithms the optimal number of processors is $P_{OPT} = O(\sqrt{n})$ and the solution to the problem can be obtained in $O(\sqrt{n})$ steps. Algorithm 3 is much less sensitive to changes in the value of $m$.

3. Application of Algorithms

A slight modification of the proposed parallel algorithms can be easily applied to the problem of computing trigonometric sums

$$C(x) = \sum_{k=0}^{n} b_k \cos kx, \quad S(x) = \sum_{k=1}^{n} b_k \sin kx. \quad (14)$$

There are two well-known sequential algorithms for finding solutions of (14): Reinsch's algorithm [13], which works for any value of $x$, and Goertzel's algorithm [13], which can be applied for $|x|$ not too close to zero. These algorithms transform the original problem (14) to the solution of a linear recurrence system of order 2:

$$x_k = \begin{cases} 0 \\ f_k + a_k x_{k-1} + a_{k-1} x_{k-2} \end{cases} \quad \text{if } k \leq 0, \quad \text{if } 1 \leq k \leq N. \quad (15)$$

For Goertzel's algorithm we need to compute the following linear recurrence system with constant coefficients (thus we apply Algorithm 3):

$$S_k = \begin{cases} 0 \\ b_k + 2S_{k+1} \cos x - S_{k+2} \end{cases} \quad \text{for } k = n + 1, n + 2, \quad (16)$$

and then we compute $C(x) = b_0 + S_1 \cos x - S_2$ and $S(x) = S_1 \sin x$. In Reinsch's algorithm, we set $S_{n+2} = D_{n+1} = 0$ and if $\cos x > 0$ then we solve

$$\begin{cases} S_{k+1} = D_{k+1} + S_{k+2} \\ D_k = b_k + eS_{k+1} + D_{k+1}, \end{cases} \quad (17)$$

for $k = n, n - 1, \ldots, 0$, where $e = -4 \sin^2 \frac{x}{2}$. If $\cos x \leq 0$ then we solve

$$\begin{cases} S_{k+1} = D_{k+1} - S_{k+2} \\ D_k = b_k + eS_{k+1} - D_{k+1}, \end{cases} \quad (18)$$

where $e = 4 \cos^2 \frac{x}{2}$. Finally, we compute $C(x) = D_0 - \frac{e}{2} S_1$ and $S(x) = S_1 \sin x$.

Let $p$ be the number of available parallel processors. For the parallel version of Goertzel's algorithm, we have to solve $p$ linear recurrence systems with constant coefficients for $q$ equations (it is assumed that $q = n/p$ is an integer) in parallel, and compute the last two terms of the final solution using a recursive doubling scheme [9]. For the parallel version of Reinsch's algorithm we have to solve $p + 2$ linear recurrence systems for $q$ equations (where $2n = pq$) in parallel, and then find the last two terms of the solution of system (17) or (18) using a similar recursive doubling scheme.

Although the parallel version of Goertzel's algorithm reaches the solution faster than the parallel version of Reinsch's algorithm, Goertzel's algorithm has a smaller speedup. This is a typical example of Amdahl's Effect as Reinsch's algorithm solves $2n$ equations whereas Goertzel's algorithm solves only $n$ equations. For both parallel algorithms the speedup increases as the size of the problem (n) increases [16]. The following table [16] presents the speedup computed for $n = 6390$.

<table>
<thead>
<tr>
<th># proc</th>
<th>2</th>
<th>3</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>14</th>
<th>15</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goertzel</td>
<td>0.38</td>
<td>1.31</td>
<td>2.13</td>
<td>2.54</td>
<td>2.93</td>
<td>3.53</td>
<td>3.79</td>
<td>4.17</td>
<td>4.89</td>
<td>5.34</td>
<td>5.75</td>
</tr>
<tr>
<td>Reinsch</td>
<td>1.01</td>
<td>1.51</td>
<td>2.47</td>
<td>2.92</td>
<td>3.37</td>
<td>4.30</td>
<td>4.70</td>
<td>5.45</td>
<td>6.28</td>
<td>6.48</td>
<td>7.38</td>
</tr>
</tbody>
</table>
It was also observed that both algorithms scale well for more than 100 equations per processor. Finally, numerical tests [17] show that the rounding errors of parallel algorithms are of the same order as the rounding errors of the corresponding sequential algorithms.

4. Concluding remarks

Two parallel algorithms for solving linear recurrence systems were presented and their modification for the case of recurrence systems with constant coefficients was discussed. The latter algorithm can be easily applied to the problem of finding trigonometric sums. The experiments performed so far suggest that the presented algorithms behave well on shared memory computers with limited number of processors. In the next step we plan to experiment with these algorithms on message passing computers.

5. References

3 CHEN, S.-C.: Speedup of iterative programs in multiprocessor systems, University of Illinois at Urbana 1975.

Addresses: Dr. MARCIN PAPRZYCKI, Department of Mathematics and Computer Science, University of Texas of the Permian Basin, Odessa, TX 79762
Dr. PRZEMYŚLAW STYCZYŃSKI, Institute of Mathematics, Numerical Analysis Department, Marie Curie-Skłodowska University, Pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin, POLAND