A fast QIF algorithm implementation on a shared memory dewavefront machine

M.P. Bekakos
Department of Informatics, Athens University of Economics and Business, Athens, Greece

Abstract

The principal theme herein is the performance evaluation of a fast direct hardware implementation of an alternate matrix procedure for the solution of linear systems $Ax=b$, where $A$ is a compact dense $(nxn)$ matrix, on a simulated special-purpose shared memory machine utilizing the dewavefront concept (rf. Evans[5]).

The simulated DEWavefront Array Processor Machine (DEWAP) consists of a square mesh of processors on which, and in distinct stages, the factorization and the solution phases of the method are performed.

1 Introduction

The introduction of VLSI circuit technology has offered the means for processing very large scale data on multiprocessor arrays in a highly parallel manner. The efficient utilization, however, of such an advanced technology in large processor array structures has to deal with the following fundamental problems imposed by this technology:

a) Localized communication which is the most critical issue of the system design;
b) novel descriptive tools to assist in the visualization description and verification of the parallel algorithm implemented on very large computing network structures;
c) self-timed schemes to avoid the clock skew which incurs in signal distribution over the entire array of processors for very large computing structures;
d) the utilization of repetitive modular structures to minimize the design complexity and cost, and
e) programmable processor modules, instead of dedicated modules, which are more preferable due to cost effectiveness considerations.
28 High-Performance Computing in Engineering

The improved feature of the simulated DEWAP machine investigated herein is its computational notation which removes the need for global synchronization requirements; this is succeeded by adopting the notion of continuously advancing waves of data and computational activity, resembling a physical wave propagation phenomenon.

Herein, we investigate the performance of the quadrant interlocking factorization (QIF) algorithmic procedure (Evans[5]) for the solution of linear systems \( Ax=b \), where \( A \) is a compact dense \((nxn)\) matrix.

The algorithm can be expressed in a recursively regular manner for VLSI implementation to obtain the factorization and the forward/backward processes.

The paper concludes with hints for further enhancements for a possible, area efficient, combined utilization of both pipelined and wavefront conceptual computational tools, for a mixed pipelined/parallel solution of general linear systems.

2 The QIF method for solving linear systems

The problem considered here is the solution of system

\[
Ax=b, \tag{1}
\]

where \( A \) is a non-singular compact dense \((nxn)\) matrix, \( x \) is an unknown \((nx1)\) column matrix and \( b \) is a given \((nx1)\) column matrix.

According to the QIF method, the matrix \( A \) is factorized into two matrices \( W \) and \( Z \) (ref. Evans[5]), and the following relationship holds

\[
A=WZ \tag{2}
\]

The matrices possess an interlocking quadrant appearance with matrix structures similar to the capital letters \( W \) and \( Z \). The matrices \( W \) and \( Z \) can be written in the compact form

\[
W=[W_1,W_2,\ldots,W_n] \quad \text{and} \quad Z^T=[Z_1,Z_2,\ldots,Z_n], \tag{3}
\]

where \( W_i \) and \( Z_i \), for \( i=1(1)n \), are the column vectors of matrices \( W \) and \( Z^T \).

In order to solve the system (1) by the QIF algorithmic procedure it can be seen from eqn[2] that the system

\[
(WZ)x=b, \tag{4}
\]

can be solved instead of linear equations (1). Hence, we need to solve two related and simpler linear systems of the form

\[
Wy=b, \tag{5}
\]

and
Each computational stage of the factorization and the solution procedures of the QIF method can be generally distinguished into two sub-stages correspondingly performing

- the solution of \((2x2)\) linear system(s), and
- the modification of the appropriate matrix element(s).

More analytically, the QIF algorithm can be expressed in a recursively (single stage) regular manner for VLSI implementation to obtain the following homogenized steps:

(i) **Factorization Process**

The elements of the matrices \(W\) and \(Z\) can be evaluated in \(\frac{1}{2} (n - 1)\) distinct stages. At each computational stage \(i\) we have to solve \((n-2i)\) \((2x2)\) linear systems to evaluate the corresponding \(w_{ij}\)’s (moving at each step, one column inwards from each side of matrix \(A\)), and to perform the modification of the \((n+2i)^2\) elements \(a_{ij}\) of the inner square (at every stage) of the coefficient matrix \(A\); thus, we end up with a central peak matrix element if \(n\) is odd or a \((2x2)\) central submatrix in \(n\) is even. From the forms of matrices \(W\) and \(Z\) and eqn[2], it can be easily observed that the values of the elements of the first and last rows of matrix \(Z\) are as follows:

\[
z_{1i} = a_{ii} \quad \text{and} \quad z_{ni} = a_{ni}, \quad \text{for } i = 1(1)n
\]  

The elements of the first and last columns of matrix \(W\) are then evaluated by solving \((n-2)\) sets of \((2x2)\) linear systems given by

\[
\begin{align*}
z_{ii}w_{i1} + z_{ni}w_{in} &= a_{ii} \\
z_{in}w_{il} + z_{nn}w_{in} &= a_{in}
\end{align*}
\]

for \(i = 2(1)(n - 1)\) \hspace{1cm} (8)

This then completes the first stage of the factorization process, whereas, in preparation for the next stage, the elements of matrix \(A\) are modified according to the following formula

\[
a_{ij} = a_{ij} - w_{ii}z_{ij} - w_{in}z_{nj}, \quad \text{for } i, j = 2(1)(n - 1)
\]  

In general, at the \(i^{th}\) stage of the factorization process, we have the relationships

\[
\begin{align*}
z_{ij} &= a_{ij} \\
z_{n+i+l_j} &= a_{n+i+l_j}
\end{align*}
\]

for \(j = i(1)(n - i + 1)\), \hspace{1cm} (10)

and the solution of the \((2x2)\) linear systems
to give the unknown quantities \( w_i^, w_{i+1}^, \) or \( j = (i+1)(l)(n-i) \); and finally, the modified \( a_i^\prime \)’s are evaluated using the formula

\[
\begin{align*}
%_{ki} &= a_{ki} - w_{ki}z_{i+1}, & \text{for } k, l = (i+1)(l)(n-i) \tag{12}
\end{align*}
\]

(ii) Solution of System \( Wy = b \)

In this phase of the QIF algorithm we have the analogous modification of \( (n-2i) \) \( b_i \)’s (moving, at each computational step \( i \), one row inwards from the top and bottom of the r.h.s. vector \( b \)). The solution of the linear system (5) can be obtained in \( \frac{1}{2}(n-1) \) steps, with the evaluation procedure carried out in pairs from the top and bottom of the vector \( y \), i.e., \( y_1^, y_n^ \) are evaluated first, then \( y_2^, y_{n-1}^ \), and so on. To typify the computational process, in general, at the \( i^\text{th} \) stage \( (i=1, 2, \ldots, \frac{1}{2}(n-1)) \) we have

\[
\begin{align*}
 y_i &= b_i, & y_{n-i+1} &= b_{n-i+1}, \tag{13}
\end{align*}
\]

and

\[
\begin{align*}
 b_j &= b_j - w_{ji}y_i - w_{ji,n-i+1}y_{n-i+1}, & \text{for } j = (i+1)(l)(n-i) \tag{14}
\end{align*}
\]

and then we proceed to the next stage.

(iii) Solution of System \( Zx = y \)

For the final solution of \( x \) of the system (1), we have to solve at every stage a \( (2x2) \) linear system (with an exception for the case that \( n \) is odd), and to modify all the outer-positioned \( y_i \)’s (moving, at each computational step \( i \), one row outwards to the top and bottom of the r.h.s. vector \( y \)). For the solution of the linear system (6) we distinguish the cases that \( n \) is odd or even.

If \( n \) is odd, then we can find that

\[
\begin{align*}
 x_1 &= \frac{y_1}{z_1}, & \text{for } l = \frac{1}{2}(n+1) \tag{15}
\end{align*}
\]

and in preparation for the next stage we compute

\[
\begin{align*}
 y_j &= y_j - x_i z_{ji}, & \text{for } j = l(l)n \text{ and } j \neq \frac{1}{2}(n+1) \tag{16}
\end{align*}
\]

The remaining elements of the vector \( x \) can again be evaluated in pairs by solving \( \frac{1}{2}(n-1) \) sets of \( (2x2) \) linear systems in \( \frac{1}{2}(n-1) \) distinct stages. In general, at the \( i^\text{th} \) stage, we solve the following \( (2x2) \) linear systems
to compute $x_i$ and $x_{n-i+1}$. We then set

$$y_j = y_j - x_i z_{ji} - x_{n-i+1} z_{j,n-i+1}, \quad \text{for } i = (1 - 1)(-1)l,$$

$$j = 1(1)(i - 1) \text{ and } (n - i + 2)(1)n$$

and proceed to the next stage.

On the other hand, if $n$ is even, then all the components of the vector $x$ are found in pairs. To find all the pairs, the linear system (15) and the formula (16) are correspondingly executed, for $i = \frac{1}{2} n(- 1)l$.

### 3 Software implementation of the single stage computational ‘dewavefront’

Herein, a programming implementation of the ‘butterfly’ linear system solver (rf. Evans[2]) is presented, on a dewavefront mesh which is simulated on a shared memory architecture provided by the available software tool environment (rf . Lester[1]). Each one of the three distinct phases is simulated separately due to the restriction imposed by this environment.

At the beginning of each phase the appropriate mesh (consisting of $(n-2)^2$ processors) is initialized with the data required (rf . Evans[5])). A $(2x2)$ linear system solver based on Crammer’s rule is used to produce the W elements and input them to the available shared memory modules. A communication network, which interconnects the processors and it is responsible for the movement of the data, is implemented using the channels technique provided by the software environment.

In accordance with the dewavefront concept, instead of having a single wave originating from the top northwest cell and travelling downwards, we have an additional concurrent wave originating from exactly the opposite bottom southeast cell. Each of the opposite computational wavefronts will have the capability to activate the successor neighbouring processors in its own propagating direction.

The factorization phase, and the solution phases of systems $Wy=b$ and $Zx=y$ are described by the programs given in Figures 1-3.

### 4 Experimental results and time-complexity analysis

The timing performance evaluation results obtained from the experimentation with a compact dense $(5x5)$ matrix are given in Table 1. The number of the mesh processors, depending on the matrix size $n$ experimented with, is $(n-2)^2$. From the results it can be easily concluded that the speedup tends to an analogous increase along with the size of the matrix, and consequently the
size of the DEWAP. Note that, however, the size of the matrix experimented with is considerably small due to the restrictions imposed by the software environment.

The overall time-complexity estimated in Evans[5] is

\[ T_t \approx 3 \times T_{\text{init}} + T_{\text{fact}} + T_f + T_b, \] (19)

where \( T_{\text{init}} \) is the initialization time required for each computational phase and \( T_{\text{fact}} \) is the time of the factorization phase. The time-complexity of the factorization phase is estimated to be

\[ T_{\text{fact}} = \sum_{i=1}^{\frac{n}{2}} \left( \frac{n - 2i}{2} \right) + \sum_{i=1}^{\frac{n}{2}} \left( \frac{n - 2i}{2} \right) + \frac{3(n - 2i)}{2} \] (20)

The \( T_f \) and \( T_b \) are the time-complexities required for the forward and backward solution phase, respectively.

In fact, in the worst bound case, the phases of the forward and backward solution of the resulting subsystems, due to the similarity of the occurring stages, can be roughly considered as of the same time order to the factorization phase.

Hence, formula (19) can be roughly modified as

\[ T_t \approx 3 \times (T_{\text{init}} + T_{\text{fact}}/b) \] (21)

### Table 1. Timing Results

<table>
<thead>
<tr>
<th>Matrix Size</th>
<th>Phase</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>(5x5)</td>
<td>Factorization</td>
<td>2.17</td>
</tr>
<tr>
<td></td>
<td>Solution of ( Wy=b )</td>
<td>2.32</td>
</tr>
<tr>
<td></td>
<td>Solution of ( Zx=y )</td>
<td>2.36</td>
</tr>
<tr>
<td>(7x7)</td>
<td>Factorization</td>
<td>3.20</td>
</tr>
<tr>
<td></td>
<td>Solution of ( Wy=b )</td>
<td>4.09</td>
</tr>
<tr>
<td></td>
<td>Solution of ( Zx=y )</td>
<td>3.26</td>
</tr>
</tbody>
</table>

### 5 Discussion and conclusive remarks

Each PE in the orthogonal network, in order to simulate the phenomenon of the propagation of the waves, can be considered as capable of being in any of the three states: 'active', 'dormant', and 'disable'. The 'dormant' state implies that a PE in that state is 'waiting' to be activated, and thus changes to the 'activate' state, by an oncoming computational wavefront, since processing can be performed only by active PEs. The 'disabled' PE remains unaffected by the activity wavefronts, being in effect 'dead', until it is 'woken' out of that state by a special control wavefront. Note the significance of the last state due to the fact that some applications (like the QIF algorithm) need a 'shrinking' in the effective size of the array of PEs.
Albeit a PE may be active, it has to ‘wait’ for the data front to arrive before it starts the actual processing. For the implementation of such a wait state, the PEs must be provided with data transfer buffers. Therefore, a ‘fetching’ of data involves an inherent ‘waiting’ for the buffer to be filled by the adjacent PE. On the other hand, the latches on each entrance to the PEs will ensure that a processor cannot send new data to the buffer unless the old data have been used by the receiver neighbour. The waiting property for wavefronts of data and activation allows for globally asynchronous operation of the PEs in the network.

To conclude, for further investigation, the overall time complexity can be significantly reduced if we consider that each distinct stage of the algorithmic procedure is performed in a mixed pipelined/parallel manner on three different meshes. This way, the initialization time will be decreased, since each processor mesh will be initialized in a parallel fashion, while an overlapping will take place between the factorization and the forward solution phases.

References

1. Book:

2. Paper in journal:

3. Paper in conference proceedings:


4. Technical report:


Program QIF_Wavefront_A;
Declaration Section
Procedure Wait(var S Semaphore); Procedure Signal(var S Semaphore); Procedure Initialize; Procedure PrintR;
Procedure ModifyB(var b : typeP; str : char); Begin (* main program *) Initialize; n=0; m=n + 1; for stage=1 to n do begin p=p+1; q=q+1;forall i= (stage + 1) to (n - stage) do begin Wqi=0; Wqi=0; Zli=0; Zli=0; for r=1 to n do begin fork (®i) ModifyB(b,top); fork (®i) ModifyB(b, bottom); end; end;forall i= (stage + 1) to (n - stage) do begin j=j+1; Wqi=0; Wqi=0; Zli=0; Zli=0; q=q+1; for r=1 to n do begin fork (®i) ModifyB(b,top); fork (®i) ModifyB(b, bottom); end; end;

Figure 1: Factorization Phase

Program QIF_Wavefront_B;
Declaration Section
Procedure Wait(var S Semaphore); Procedure Signal(var S Semaphore); Procedure Initialize; Procedure PrintR;
Procedure ModifyB(var b : typeP; str : char); Begin (* main program *) Initialize; p=0; q=m+1;forall i= (stage + 1) to (n - stage) do begin j=j+1; Wqi=0; Wqi=0; Zli=0; Zli=0; q=q+1; for r=1 to n do begin fork (®i) ModifyB(b,top); fork (®i) ModifyB(b, bottom); end; end;forall i= (stage + 1) to (n - stage) do begin j=j+1; Wqi=0; Wqi=0; Zli=0; Zli=0; q=q+1; for r=1 to n do begin fork (®i) ModifyB(b,top); fork (®i) ModifyB(b, bottom); end; end;

Figure 2 cont'd...
for i:=1 to 18 do
  join;
init(b[2])= b[2];
init(b[3])= b[3];
init(b[5])= b[5];

end; (* for all stages *)
End (* QIF_Wavefront_B *)

Figure 2: Solution of system \( Wy=b \).

Program QIF_Wavefront_C;
Declaration;
Procedure InitialC;
Procedure Cramer (a,b,c,d, r1, r2 : real; var x,y : real);
Procedure PrintC;
Procedure ModifyC(var y : typeP; str : char);
Begin (* main program *)
InitialC;
i= (n+1) div 2;
j= i;
m2= (n+1) div 2;
for stage= 1 to (n+1) div 2 do
  begin
    if stage= 1 then
      x(i)= init(i)[Z1]J
    else
      begin
        i= i-1;
j= n+i;
    end
  end;
begin (* for stages 1 to (n+1) div 2 *)
  for k= 1 to n-2 do
    begin
      x(k)[k]= 0;
x(k)[k]= 0;
Z(k)= 0;
Z(k)= 0;
  end;
end;
Figure 3: Solution of system \( Zx=y \)