A VERSATILE ALGORITHM FOR LINEAR SYSTEM PROBLEMS USING ADAPTIVE PIVOTING

Javed I. Khan, W. Lin & D. Y. Y. Yun

Department of Electrical Engineering
University of Hawaii at Manoa
javed@wiliki.eng.hawaii.edu
2540 Dole Street, Honolulu, HI-96822

Published in the Proceedings of the
5th ISMM International Conference on
Parallel and Distributed Computing and Systems
Pittsburgh, October 1992, pp313-315

Abstract
In this paper we present a new parallel method for the solution of three important linear system problems namely i) matrix inversion ii) solution of linear system and iii) computation of linear combination of the system variables, using the same framework on a MIMD array multi-processor with ***torus*** *interconnection. This algorithm requires* ***n steps*** *on* $\boldsymbol{n^2}$ ***processors*** *with 2 flops step-width. This method also uses a new distributed pivoting scheme, called* **Adaptive Pivoting** *to replaces the costly row interchange for stability. Due to* ***adaptive pivoting*** *and dual wavefront communication pattern, the resulting activity on the torus resembles the* ***ripples on a pond formed by the rain drops****. This paper presents the design, analysis and performance comparison of this matrix algorithm including the* ***simulation results on a 32 Processor Meiko Transputer****.*

1 Introduction

With the advent of parallel processing a considerable effort has been spent in solving linear systems in parallel on a wide variety of architectures [7,5,2]. Most of these algorithms basically parallelize the triangular factorization. However, it is now widely experienced that the triangular methods perform poorly in parallel computation [8] because of the subsequent forward and backward substitutions. In many cases even classical methods like Gauss-Jordan or Gauss-Seidal outperform them [4].

Besides efficiency, next important concern in linear systems is the problem of **computational stability** [9] due to finite precision arithmetic. Most of the existing parallel algorithms suggest the use of conventional row interchange. However, such interchange is prohibitively expensive [4] in parallel processing. Very few alternate proposals exist to circumvent this problem. Recently [3] has mentioned about the idea of using some kind of threshold. However, there is no satisfactory technique which can select appropriate threshold without incurring substantial communication cost.

In this paper we have proposed a new method for solving a set of linear system problems based on Faddeeva's [1] non-triangular method for computing matrix determinant. This new method is faster than most of the known parallel methods including the much used classical methods. We have also used a new technique called **adaptive pivoting** which no longer requires the costly row interchanges for improving stability. It can work with non predefined pivot set, can dynamically trace the consequences, and still can compute the exact result. Interestingly, it does so without any extra cost. Previously, we have used similar techniques to solve matrix inversion only [6]. This paper reports two important improvements on our earlier research. First, we have extended our method and mapping and applied it in solving two other linear system problems from the same framework, and secondly, we have a new communication structure which has improved the algorithm efficiency significantly, specially with this new **adaptive pivoting**.

2 The Computational Model

Let AX=B is a linear system where X is the vector defining the n system variables, A is the coefficient matrix, and let CX be any linear combination of the system variables. Given A, B and C, we want to compute (i). A^{-1} (ii). $A^{-1}B$ and (iii). $CA^{-1}B$.

(i) Inverse Computation: The formal derivation of the procedure from Faddeeva's [1] method can be found in [6]. In short, our scheme for the computation of inverse can be viewed as equivalent to the n-step elimination performed on the following augmented matrix of size 2nx2n . An elimination step refers to the computation of $a_{ij.k+1}=a_{ij.k}-a_{ik.k}*a_{kj.k}/a_{kk}$ where n<i,j,k<n+1.

$$\begin{bmatrix} \mathbf{A} & \mathbf{I} \\ \mathbf{-I} & \mathbf{0} \end{bmatrix}$$

This is equivalent to the introduction of extra $(1,0,...)^T$, $(0,1,...)^T$ ....$(0,0,...,1)^T$ columns and extra (-1,0,....), (0,-1,...) .... (0,0...,-1) rows respectively at the end of each phase.

(ii) Linear System Solution: AX=B can be solved by applying the same computation on the form:

$$\begin{bmatrix} \mathbf{A} & \mathbf{IB} \\ \mathbf{-I} & \mathbf{0} \end{bmatrix}$$

This is equivalent to the method of computing inverse except the introduced columns at the end of each iteration is $(b_1,0,...)^T$, $(0,b_2,...)^T$ ....$(0,0,\ b_n)^T$. At the end of this computation the set of sums of the column elements will provide the solution.

(iii) Linear Combination of The System Variables: A similar extension to the augmented matrix of the following form can calculate the linear combination of the system variables:

$$\begin{bmatrix} \mathbf{A} & \mathbf{IB} \\ \mathbf{-CI} & \mathbf{0} \end{bmatrix}$$

This procedure is equivalent to the procedure (ii) except now the introduced columns are $(c_1,0,..0)$, $(0,c_2,..0)...(0,0\ \ c_n)$. The sum of all the elements after performing the n-step computation will provide the result.

3 Mapping of The Model

The derivation of section 2 to solve all three forms of the problem is summarized below:

1.If (i) Set B=C=U, if (ii) Set C=U where U is a vector with all unit elements.

2. Enter the first phase k=1. Take the original matrix $[a_{ij}]$ and create an extended matrix E^1 with $e_{ij}=a_{ij}$ for i,j=1,2..n.

3. Add a new row and a new column with all zero elements except, $e_{(n+1),1.k}=-c_k$ and $e_{1,(n+1).k}=b_k$.

4. Calculate $e_{i,j.1}=e_{i,j.1}-e_{i,1.1}*e_{1,j.1}/e_{1,1.1}$ for all i,j= 2,..(n+1).

5.Consider the elements $e_{ij.k}$ of E^k with i,j=2...(n+1) as the elements $e_{i,j.k+1}$ of the new extended matrix E^{k+1}.

6. Repeat steps 2,3 & 4 until k=n.

7. If (ii) Calculate column sum or, if (iii) Calculate global sum.

Here the dotted subscript **.k** refers to the $\mathbf{k}^{th}$ phase. Steps 2 to 5 constitute the *core* part of our algorithm. As we shall see later the 7th step, when properly overlapped, will add just one more phase to the n phase *core*.

We will now map the *core*. Although the size of the augmented matrix is 2nx2n, but the size of the active portion of the matrix at any elimination phase is (n+1)x(n+1). Fig-1(a) shows the computations inside the active portion at any phase. In Fig-1(a) capital letters denote the stored value and small letters denote the updated value of the local elements stored in the processors. Fig-1(b) shows a communication pattern and the message packet contents which can satisfy the data dependency of Fig-1(a). In unidirectional flow, the last pivot suffers from long idle time because it becomes the last one in a row to receive data in the new phase. We can correct the situation if, instead of unidirectional flow, data is sent along all the 4 directions, then the network diameter reduces byhalf. Fig-1(c) shows the new communication pattern.

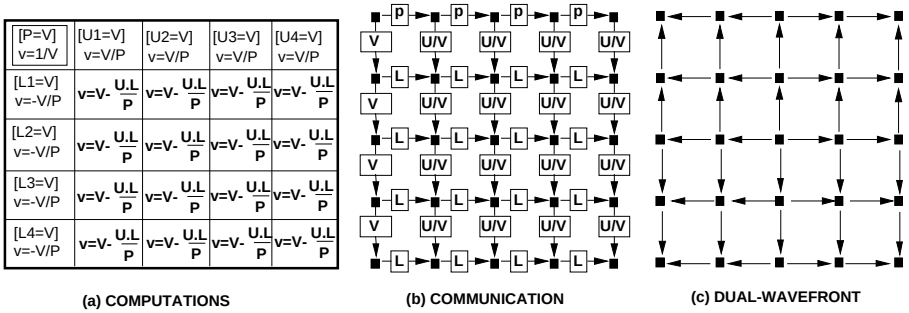

Fig-1

It is evident from the computational model that the above model directly maps onto a mesh of size (n+1)x(n+1). A little more watchful observation reveals that at the end of each phase, the data of some old nodes are no longer being used (the pivot row and pivot column) while, on the other hand, some new nodes have to be added (in the bottom). We have decided to map these new elements directly on the emptied nodes. The consequences are, i). the diagonal transfer of the entire matrix is no longer required, ii). the logical mesh containing the active matrix is wrapped and iii) we no longer need $(n+1)^{th}$ row or column.

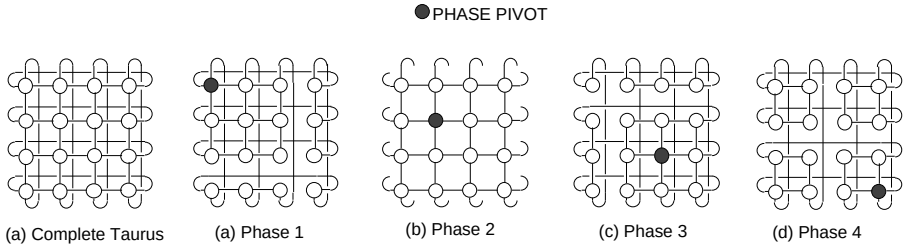

Fig-2

Torus can gracefully accommodate all these wrapped meshes. Fig-3 shows how; for a 4x4 structure. The recursive application of above computation and communication on these logical meshes creates a dual wavefront activity pattern on the torus with overlapped phases.

The sum-phase (7^{th} step) is required by problem forms (ii). and (iii). The same communication pattern (Fig 1(c)) is sufficient to support the sum phases of both the types. Therefore, the net effect of sum-phase on the over all scheme is just the addition of one extra phase.

4 Adaptive Pivoting

Most pivoting schemes travel along the principal forward diagonal (PFD). If there is a zero (or small) pivot, the row is usuallly interchanged with another row with non-zero (or a large) pivot. However, unlike the sequential algorithms, parallel algorithms have to incur prohibitively expensive communication cost to perform such interchange (consider the situation if the old and new rows are far away from each other). We therefore, suggest skipping the pivot position adaptively, rather than exchanging rows. Our computational model can indeed support such flexibility, and more interestingly, without any additional cost. This flexibility is based on the following two theorems:

Theorem 1*: If* $\boldsymbol{B}^T$ *is the resultant matrix generated from* $\boldsymbol{A}$ *after performing n eliminations steps and P is the set of n pivots used, then the sequence of selecting the pivots within this set in successive phases does not effect the final mapping* $\boldsymbol{B} \Rightarrow \boldsymbol{T}$ *where P is as follows:*

$$P = \{P_{x_1y_1}, P_{x_2y_2}, \ldots P_{x_ny_n} \ni x_1 \neq x_2 \neq .. \neq x_n, 1 < x_j < n, y_1 \neq y_2 \neq .. \neq y_n, 1 < y_j < n\}$$

Theorem 2*: If initially the matrix* $\boldsymbol{A}$ *is mapped onto torus* $\boldsymbol{T}$ *in such a way that* a_{ij} *is mapped on* t_{ij} *for i,j=1,2,3....n, and if* $\boldsymbol{B}$ *is the resultant matrix generated from matrix* $\boldsymbol{A}$ *after performing the Faddeeva eliminations (as derived in section 2) using principal diagonal pivots, and if* P_{lm} *and* P_{xy} *are two members of the applied pivot set P then at the end of the* n^{Th} *phase,* T_{mx} *contains* b_{ly} *and* T_{yl} *contains* b_{xm}.

The formal proof of these are in [6]. Property 1 allows our algorithm to be flexible in selecting pivot positions and to avoid costly row column interchange. Property 2 provides us with a means to retrace the final mapping based on the local information (thus at no extra communication cost).

5 The Algorithm

The algorithm generates a series of successive wavefronts. Each originates at a phase pivot and then propagates in all four directions. As shown in Fig-2, within each phase, each of the nodes transmits message packets containing **U/V** and **L** to its neighbors and performs one *elimination*. Like the ripples in a pond, the computation propagates from the pivot center. However, these centers are not static but drift with the phases. If there is any zero pivot, it is skipped and the algorithm starts with a psudo-arbitrary (so far the new one is from a new row and new column) position. To reduce numerical error, if full or column sort is used, then the pivot center jumps to the maximum valued pivot position. In either case, no row interchange is necessary. The following algorithm based on the theorem 2, computes all the elements of the result matrix and their co-ordinates, using only local information, without any additional cost. The successive dual-wavefronts, propagating from the

apparently random pivot positions in the torus, generate the interesting activity pattern which resembles the ripples on a pond caused by the rain drops.

Below we present a pseudo code version of the *core*. The program iterates for n phases over all the nodes of the torus. At the beginning of each phase, the **set_orientation()** routine finds the vertical and horizontal source and destination neighbors for each node depending on their own relative position w.r.t. the current pivot. If it is at the end, a null is returned so that **send()** routine performs a null operation. Since, several pivot selection strategies can be used for adaptive pivoting, we have assumed **xphase[]** and **yphase[]** arrays are supplying the pivot positions. For solving problems (ii) and (iii) a sum phase should be added at the end. The final identity of the elements are in (xx,yy).

```
int k=0, s=n, p_count=n;
int xphase[],yphase[],xx,yy;
getpid(i,j);

/* Loop for n phases*/
while(p_count) {
    px= xphase[k];
    py= yphase[k++];

    set_orientation(i, px,
      &vert_dst, &vert_src);
    set_orientation(j, py,
      &horz_dst, &horz_src);

/* If Phase pivot*/
   if(px=i and py=j) {
     if(v <= thres) {
       v= ABORT;
       xphase[s]=px;
       yphase[s++]=yx;
       p_count++;
     }
    send(v:south);
     send(v:east);
     send(v:north);
     send(v:west);
    v=1/v;
   }

/* If Pivot Row Elements*/
    elseif(py==j) {
      recvb(p:horz_src);
      if(p==ABORT) {
         xphase[s]=px;
         yphase[s++]=yx;
         v=ABORT;
         p_count++;
      }
       else v=v*b[k]/p;
       send(v:east);
       send(v:west);
       send(p:horz_dst);
          xx=k;
      }

/* If Left Column Elements*/
     elseif(px==i) {
      send(v:south);
      send(v:north);
      recvb(p:vert_src);
      if(p==ABORT) {
         xphase[s]=px;
         yphase[s++]=yx;
         p_count++;
      }
      send(p:vert_dst);
      v=-v*c[k]/p;
      yy=k;
     }

/* For Other Elements*/
     else {
      recvb(u:horz_src);
      if(u==ABORT) {
         xphase[s]=px;
         yphase[s++]=yx;
         p_count++;
      }
      send(u:horz_dst);
      recvb(l:vert_src);
      send(l:vert_dst);
      v=v-u*l;
     }
p_count--;
}
```

6 Complexity Analysis

6.1 Performance and Scalability of the Algorithm

The performance of the wave-front algorithms depend on the time between the initiation of two successive phases. For n phase computation

(n=n+1 for problem ii. & iii.) There are n-1 phase gaps. The last phase propagates to the logical end in time 2*(n/2) communication steps and n computations. Thus, the parallel execution time is:

$$T_{par}=(4t_{comm}+t_{sub}+t_{mul}+t_{div})*(n-1)+n*t_{comm} + .5n*t_{comp}$$

If the torus size is nxn and matrix size is mxm, using block decomposition the **scalability** measure is given by following execution time:

$$T_{par}=(4t_{comm}+(m/n)(t_{sub}+t_{mul}+t_{div}))*(m-1)+n*t_{comm} + m*t_{comp}$$

The performance improvement by dual-wavefront communication over mono-wavefront communication is evident in the 2nd and third terms of these two expressions.

6.2 Performance Comparison with Other Methods

Here we will provide a brief comparison between the three frequently used linear system algorithms namely LU decomposition, Gauss-Jordan and Gauss-Seidel. We will assume all use n^2 processors to process nxn matrix. LU has the worst parallel complexity (but best sequential). In addition to n steps for factorization it requires another 2n steps for forward and backward substitutions. Gauss-Jordan needs n steps for elimination and another n steps for substitution. The parallel performance of Gauss-Seidel is better, however it has a wider phase (4 Flops) because some (all for inversion) of the processors have to house two elements. If the communication cost is predominant, then this disadvantage becomes insignificant. For the problems (ii) and (iii) our method requires 1 additional step for the sum phase. Clearly our method has the least parallel complexity despite its high computation count. This advantage is most visible for matrix inversion which is often considered the most complex among the three problems.

7 Simulation Results

We have implemented the dual wave front algorithm with adaptive flexible (any general) order pivoting and tested its performance on a 32 node Meiko Transputer. Fig-3 shows the relative speedup with the variation of the matrix size for four different torus sizes. We varied the matrix size from 10x10 to 80x80 and used block decomposition. Fig-4 shows the improvement in speedup from mono to dual wavefront configuration.

Fig-5 shows the effect of *adaptive pivoting* on execution cost. In this experiment, we have used a principal forward diagonal pivoting with 4 different sequences and observed the computation time for both mono and dual wavefront versions. The dual wavefront communication preformed better than the mono wavefront communication.

DUAL WAVEFRONT SPEEDUP

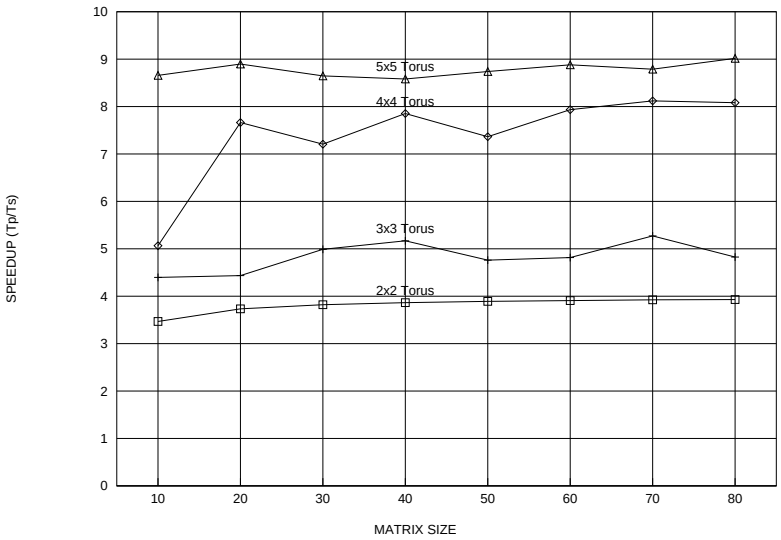

Fig-3

DUAL vs. MONO WAVEFRONT SPEEDUP

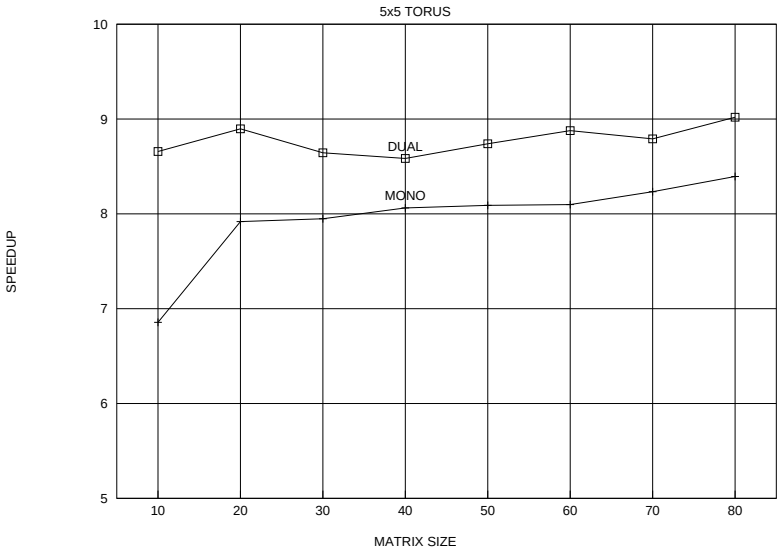

Fig-4

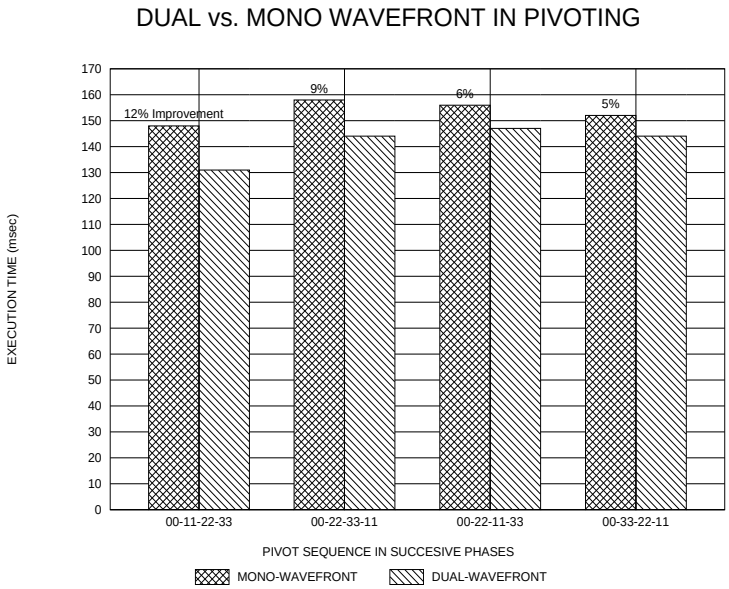

Fig-5

8 Conclusion

We have presented a parallel algorithm which can solve three important linear system problems from the same computational framework. It uses an nxn torus connected MIMD parallel processor to invert an nxn matrix in O(n) time. This algorithm is one of the most efficient and compact among the existing parallel methods.

On the issues of **numerical stability**, we have shown that our new *adaptive pivoting* can replace the traditional row interchange methods for improving numerical stability against zero or small pivots. However, to improve stability against ill condition matrix, our method suffers similar disadvantages [3] like the other elimination based algorithms. This new pivoting scheme takes the advantage of the fact, that the traditional *principal forward diagonal pivot selection* order is not an intrinsic part of the problem, but merely an arbitrary choice to suit the way we represent matrix. The uniform torus space (where no node is special) and the adaptive pivoting together transcend above this artificial restriction and exploit the full advantage of the flexibility inherent in the computation structure. This new adaptive pivoting has the potential of widespread application in vast area of parallel matrix computations.

References

[1] Faddeeva, V. N., Computational Methods of Linear Algebra, Chapter 2, Translated by C. D. Benster, Dover Pub., New York 1959.

[2] Geist, G. A., & M. T. Heath, Matrix Factorization on a Hypercube Multiprocessor, SIAM Proc. 1st Conference on Hypercube Multiprocessor, p-161-180, Aug 1985.

[3] Gill, P. E., W. Murray & M. H. Wright, Numerical Linear Algebra and Optimization, v. 1, Chapter 3 & 4, Addison-Wesley Publishing Company, California 1991.

[4] Heller, D., A Survey of Parallel Algorithms In Numerical Linear Algebra, SIAM Review, Vol 20, no 4, October 1978.

[5] Kant, R. M. & T. Kimura, Decentralized Parallel Algorithms for Matrix Computation, Proc. 5th Annual Symposium of Com. Arch., P-96-100, 1978.

[6] Khan, J. I., W. Lin & D. Y. Y. Yun, A Parallel Matrix Inversion Algorithm on Torus with Adaptive Pivoting, Proc. 21st International Conference on Parallel Processing, Chicaga, (to be published), August 1992.

[7] Kung, H. T. and C. E. Leiserson, Systolic Array (for VLSI), Technical Report CS79-103, Carnegie-Mellon University, April 1979.

[8] Modi, J.J., Parallel Algorithms and Matrix Computation, Oxford University Press, Oxford, 1988.

[9] Wilkinson, J. H., Error Analysis of Direct Methods of Matrix Inversion, J. Assoc. Comp. Mach. 8, p281-330, 1961.