A Parallel Processing Architecture for Solving Large-Scale Linear Systems

Arun Nagari, Itamar Elhanany, Ben Thompson
Department of Electrical Engineering and Computer Science
The University of Tennessee
Knoxville, TN 37996-2100, USA
Phone: 865-974-3891 Fax: 865-974-5483
email: {itamar,anagari,bthomp13}@utk.edu

Abstract— **Solving linear systems with a large number of variables is at the core of many scientific problems. Parallel processing techniques for solving such systems have received much attention in recent years. A pivotal theme in the literature pertains to the application of LU decomposing which factorizes an $N \times N$ square matrix into two triangular matrices so that the resulting linear system can be more easily solved in $O(N^2)$ work. Inherently, the computational complexity of LU decomposition is $O(N^3)$. Moreover, it is a process that is challenging to parallelize. In this paper, we propose a highly-parallel methodology for solving large-scale dense linear systems by means of a novel application of Cramer's Rule. A numerically stable scheme is described, yielding an overall computational complexity of $O(N)$ with N^2 processing units.**

I. Introduction

Solving a system of linear equations is a fundamental problem in many scientific and engineering applications, including electric power networks analysis, circuit simulation and structural analysis [8]. LU decomposition is a common method used to solve a system of linear equations. In applications where matrices have thousands of elements, LU factorization demands exhaustive computation. When operations have to be carried out in real time, reduced execution time is of utmost importance. For this purpose, extensive research is directed toward the application of parallel processing techniques to LU factorization of linear systems [8]. Parallel supercomputers have accomplished a great deal of success in solving compution intensive problems, but involve some drawbacks such as high price/performance ratio, tedious design, programming complexity as well as their high maintanance costs[7].

On the other hand, with rapid evolution of Field Programmable Gate Arrays (FPGA) into multi-million gate System on a Programmable Chip(SOPC) computing platforms, it is now possible to integrate a large number of computational resourses on one silicon die, which consequently would highly contribute towards concurrently solving linear systems[7][8][4]. Such efforts pertain to both fine-grained as well as coarse parallelism.

In this paper we introduce a novel approach for solving large-scale linear systems, using Cramer's rule[6]. The latter yields a highly parallel design which can be directly realized in hardware. The computational complexity of the proposed method is $O(N)$, given N^2 processors. The rest of the paper is structured as follows. In Section II we present the variation on Cramer's Rule and the respective parallel computation of matrix determinants, which are at the core of the proposed method.

II. Concurrency in Solving Linear Systems

A. Revisiting Cramer's Rule

Cramer's Rule expresses the solution to a system of simultaneous linear equations in terms of the ratios of determinants. For a linear system of the form $Ax = b$, where $A = [a_{ij}]$ is an invertable $N \times N$ matrix with $|a_{11}| > 0$, Cramer's Rule states that

$$x_i = \frac{\det(A_i)}{\det(A)}, (i = 1, ...n) \tag{1}$$

where A_i is the matrix formed by replacing the i^{th} column of A by the column vector b[6]. Although Cramer's Rule provides an elegant way to solve a consistent system of equations, it is often viewed as a highly impractical method since it is computationally too expensive for large systems, particularly when compared to Gaussian elimination or iterative methods. Given that the most efficient method of calculating the determinant of an $N \times N$ matrix is $O(N^3)$, Cramer's Rule is generally perceived as an $O(N^4)$ method, limiting its scalability.

In this paper we challenge the generally-accepted notion stated above by introducing two relevant contributions. The first is a numerically-stable and efficient method for calculating determinants on a parallel processing system, while the second is a framework for concurrently obtaining $\det(A_i)$. We show that for a system with P parallel processing units, the overall computational complexity of the method is reduced to $O(N^3/P)$. Moreover, the inherent communication requirements render this method highly attractive for large-scale parallel processing platforms.

B. Proposed Architecture

1) Chio's Pivotal Condensation Process: The conventional approach to determinant computation of an order N square matrix is to express the elements of any one row or column, and the corresponding cofactors of the elements, as a linear combination of N determinants of order $N-1$. For higher order matrices, the process becomes prohibitively lengthy. An alternative to the conventional method of determinant evaluation is to use a condensation method [3][2]. In condensation methods, an initial matrix of order N is successively reduced one order at a time, until the basic order of 2×2 is reached. This method reduces the lengthy process of determinant computation when compared to the standard method.

Let $A = [a_{ij}]$ be an $N \times N$ matrix with $|a_{11}| > 0$. Letting D denote the matrix obtained by replacing each element a_{ij} in A by the term $\begin{vmatrix} a_{11} & a_{1j} \\ a_{i1} & a_{ij} \end{vmatrix}$, it can be shown that $|A| = |D|/(a_{11}^{N-2})$. That is,

$$|A| = (1/a_{11}^{N-2}) \times \begin{vmatrix} \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} & \cdots & \begin{vmatrix} a_{11} & a_{1N} \\ a_{21} & a_{2N} \end{vmatrix} \\ \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & \cdots & \begin{vmatrix} a_{11} & a_{1N} \\ a_{31} & a_{3N} \end{vmatrix} \\ \vdots & \vdots & \cdots & \vdots \\ \begin{vmatrix} a_{11} & a_{12} \\ a_{N1} & a_{N2} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{N1} & a_{NN} \end{vmatrix} & \cdots & \begin{vmatrix} a_{11} & a_{1N} \\ a_{N1} & a_{NN} \end{vmatrix} \end{vmatrix} \quad (2)$$

Let the application of the above equality for a matrix of order k to a matrix of order $k-1$ be be defined as *iteration k*. Applying $N-2$ iterations will reduce the determinant matrix to the order of a 2×2. The above process exhibits some attractive attributes in the context of parallel processing. First, it is clear that in each iteration, all 2×2 elements can be calculated independently. This suggests that given N^2 processing units, calculating the determinant is reduced to $O(N)$. Moreover, and probably more important from a distributed-processing standpoint, the communications involved is simply a broadcast from the first column to all other columns. This suggests a communications complexity of $O(N)$ with clear independence between the right-most $N-1$ columns.

2) Parallel Application of Cramer's Rule: Consider N simultanious linear equations, $Ax = a_c$, of the form

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1N} \\ a_{21} & a_{22} & \cdots & a_{2N} \\ a_{31} & a_{32} & \cdots & a_{3N} \\ \vdots & \vdots & \cdots & \vdots \\ a_{N1} & a_{N2} & \cdots & a_{NN} \end{bmatrix} \times \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ \vdots \\ X_N \end{bmatrix} = \begin{bmatrix} a_{c1} \\ a_{c2} \\ a_{c3} \\ \vdots \\ a_{cN} \end{bmatrix} \quad (3)$$

We address the solution of such a system by means of an efficient utilization of Cramer's Rule. The core calculation performed by each processing unit will be the determinant of a 2×2 matrix. This will remain the case throughout the process. Custom hardware can be designed to optimize this core calculation and further reduce the aggregate computation time.

Reduction of computation time is achieved by eliminating the need to independently calculate each of the different numerators involved in Cramer's Rule. Performing Cramer's Rule traditionally requires computing N numerators. Understanding this method hinges upon realizing that each numerator can be found using the original matrix and the constant column (a_c) matrix. Extending this notion, half of the numerators of Cramer's Rule can be found by replacing, in succession, half of the columns of the orignal matrix with the column matrix. The other half of the numerators can be found by replacing the other half of the original matrix. This idea is exploited in this new method by mirroring the original matrix into two matrices which each are used to attain half of the variables. Chio's Process is used to reduce those matrices until both are mirrored, creating four matrices. Thus, each matrix will be used to find one fourth of the variables. This continues until each matrix is finally reduced to contain only information pertaining to one or two variables. We next provide a more detailed of the 5 basic steps involved by the proposed method.

Step 1: Mirroring: The first step involves the creation of a new matrix by mirroring, horizontally, the original coefficient matrix. Recall that by interchanging two columns in a matrix, one must be negated for their determinants to be equal,

$$\begin{bmatrix} \mathrm{a}_{11} & \mathrm{a}_{12} & \cdots & \mathrm{a}_{1N} \\ \mathrm{a}_{21} & \mathrm{a}_{22} & \cdots & \mathrm{a}_{2N} \\ \mathrm{a}_{31} & \mathrm{a}_{32} & \cdots & \mathrm{a}_{3N} \\ \mathrm{a}_{41} & \mathrm{a}_{42} & \cdots & \mathrm{a}_{4N} \\ \vdots & \vdots & \cdots & \vdots \\ \mathrm{a}_{N1} & \mathrm{a}_{N2} & \cdots & \mathrm{a}_{NN} \end{bmatrix} \Longrightarrow \begin{bmatrix} \mathrm{a}_{1N} & \cdots & -\mathrm{a}_{12} & -\mathrm{a}_{11} \\ \mathrm{a}_{2N} & \cdots & -\mathrm{a}_{22} & -\mathrm{a}_{21} \\ \mathrm{a}_{3N} & \cdots & -\mathrm{a}_{32} & -\mathrm{a}_{31} \\ \mathrm{a}_{4N} & \cdots & -\mathrm{a}_{42} & -\mathrm{a}_{41} \\ \vdots & \cdots & \vdots & \vdots \\ \mathrm{a}_{NN} & \cdots & -\mathrm{a}_{N2} & -\mathrm{a}_{N1} \end{bmatrix}$$

Mirroring thus refers to multiple interchanges, resulting in one half of the new matrix being negated with respect to the orignal matrix. This is the integral step in parallelizing Cramer's Rule. The last half of the columns of the original matrix will be used to find the last half of the variables values. By the same token, the last half of the columns of the mirrored matrix, which are the first half of the columns of the original matrix, are used to obtain the first half of the variables. Notice that using Chio's process, every column is combined with the first column. Thus, the first column is the only one that is not combined with itself, resulting in its information being lost in the context of this architecture.

Step 2: Column Replacement: The second operation involved in the process is the replacement of the last column of the original matrix and the mirrored matrix with the constant column matrix, and storage of the replaced columns. Using the mirrored matrix as an example, we note that

$$\begin{bmatrix} \mathrm{a}_{1N} & \cdots & -\mathrm{a}_{12} & \mathrm{a}_{c1} \\ \mathrm{a}_{2N} & \cdots & -\mathrm{a}_{22} & \mathrm{a}_{c2} \\ \mathrm{a}_{3N} & \cdots & -\mathrm{a}_{32} & \mathrm{a}_{c3} \\ \mathrm{a}_{4N} & \cdots & -\mathrm{a}_{42} & \mathrm{a}_{c4} \\ \vdots & \cdots & \vdots & \vdots \\ \mathrm{a}_{NN} & \cdots & -\mathrm{a}_{N2} & \mathrm{a}_{cN} \end{bmatrix} \blacktriangleright \begin{bmatrix} -\mathrm{a}_{11} \\ -\mathrm{a}_{21} \\ -\mathrm{a}_{31} \\ -\mathrm{a}_{41} \\ \vdots \\ -\mathrm{a}_{N1} \end{bmatrix}$$

This set of matrices, a larger matrix and an accompanying column matrix, is the fundamental unit which is repeated throughout the architecture. Hereafter, this unit will be referred

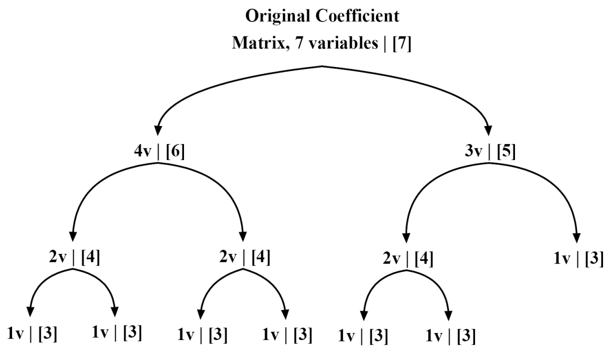


Fig. 1. Number of variables being solved for vs. matrix set size.

to as a *matrix set*. The original coefficent matrix and the constant column matrix of the system of equations can be thought of as the first matrix set. Note that after mirroring, two matrix sets will be present, not just the one depicted above.

Step 3: Variable Assignment: Reduction of matrix sets occurs until a certain size matrix is reached. The appropriate size is determined by the number of variables being solved for by each matrix set. Following mirroring, each matrix set solves for half of the variables. In the case of an odd number N, such as $N = 9$, the original matrix set would be used to solve for $\lceil \frac{N}{2} \rceil = 5$ and the mirrored matrix set for $\lfloor (\frac{N}{2}) \rfloor = 4$ variables. To compensate for the loss of information inherent to Chio's process, the matrices are only reduced to size $\left(\frac{N}{2} + 2\right)$, 7×7 and 6×6, respectively, in this case. Such calculation is utilized throughout the process to denote at which point reduction is terminated.

The tree diagram in figure 1 illustrates the relationship between the number of variables being solved for and the size of each matrix set, using an original 7×7 matrix as an example. The number followed by "v" is the number of variables being solved by that particular matrix set, and the number inside the brackets is the size of the matrix set. In this example, 5 total variables exist. The process continues until the final 3×3 matrix sets are reached. Notice that the final number of variables in the branches sum to 5, and only one variable is solved for by any matrix set. It is important to note that all individual matrix sets are independent of each other and can computed independently.

Step 4: Reduction: Next ,we discuss the reduction of the individual matrix sets using Chio's Process to the appropriate size. We observe that the reduction of the accompanying column matrix using Chio's Process, is equivalent to treating it as if it was an additional column of the larger matrix. However, we are required to store it separately. Once reduced to the appropriate size, steps (1), (3), and (4) are re[eated until all matrix sets are down to 3×3.

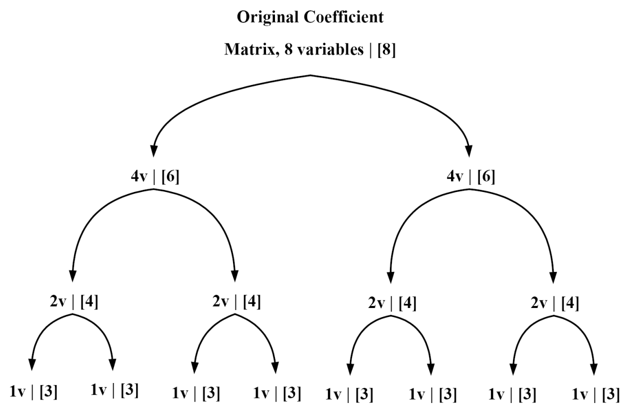
Recall that Chio's Process requires that the a_{11} element in the matrix being reduced should be one. Due to the number of multiplications involved in the process, individual elements tend to grow in magnitude which can eventually grow to very large values. To counteract this problem, after each iteration of matrix reduction to $N - 1$ order, the first row of both the large matrix and the first element of its accompanying column matrix are interchanged with the row which has the largest-magnitdue first value $|a_{i1}|$, and we negate one of the rows being interchanged. Negation ensures that the determinant remains the same after interchanging rows. This process prevents all of the elements in the matrix from growing in magnitude.

To make the first element of the matrix one, we divide the first column of the large matrix by a_{11}. This forces the multiplying factor $(1/a_{11}^{N-2}) = 1$ in Chio's process , since the division causes the new a_{11} to be one. A record of each divided factor is kept for each branch of matrices to multiply back at the final step. When the matrix set is reduced to its minimal dimensions, a list of such multiplicative factors $F_A = \{F_1,\ F_2, ..., F_i\}$ will have been formed. As each split occurs, the two new branches begin adding unique factors to their respective lists, although their first factors are common, F_A. Thus, all branches have lists that begin with common factors between sibling branches. However, each unique branch adds its own unique factors.

Step 5: Solving for Variables: Each column, from the original coefficient matrix to the final 3×3 matrices, can be traced through the reduction and mirroring process. This dictates which variable are solved for by each final matrix set. To solve for all but the first and last variables, we replace the middle columns of the final 3×3 matrices with the *negation of* the accompanying column matrix, forming a matrix Z. The original coefficient matrix is C, $\frac{\det(Z)}{\det(C)} \prod [F] = a_{ci}$. The first and last variables, a_{c1} and a_{cN}, are solved for without replacing columns in the final matrices. The branch that is always mirrored contains the information of the first variable. Let that branch's 3×3 matrix be denoted by M. $\frac{\det(M)}{\det(C)} \prod [F] = a_{c1}$. Similarly, a_{cN} can be found from the branch that is always the original matrix. This process yields $N+2$ variables; the two will be repeated and can be neglected. The following summarizes the process described:

- 1) Form a matrix which is the mirror image of the original matrix A.
  2) Replace the last column in each matrix with the constant matrix, and store the column which has been replaced.
  3) Determine the number of variables being solved for by each matrix, labeled V.
  4) Reduce each matrix to $(V + 2) \times (V + 2)$ using Chio's process. Store division factors in list F for each branch. Repeat steps 1, 3, and 4 until all matrices are reduced to order 3×3.
  5) Solve for the variables.

C. Computational Complexity



Tree diagram illustrating the derivation of computational complexity

An approximation of the number of 2×2 determinants calculated for a given $N \times N$ matrix yields an expression for computational complexity. For simplicity, references to determinants hereafter implies 2×2 determinants. To reduce a matrix using Chio's process from an $N \times N$ to an $(N-1) \times (N-1)$ matrix requires $(N-1)^2$ determinants to be calculated. Recall that an extra column matrix is carried in each matrix set. This requires an extra N determinants to reduce from an $N \times 1$ to an $(N-1) \times 1$ column matrix. As such, each reduction of one matrix set requires $(N-1)^2 + (N-1)$ determinants.

The number of determinants depends on N and on the number of mirror steps performed. The computational complexity expression below was derived assuming that N is a power of 2. This simplifies the calculation since $N/2$ will also be a power of 2, and the number of splits is exactly $\log_2(N)$. Using an original 8×8 matrix and figure 2 as an illustration, the computational complexity expression is next obtained. To reduce one 8×8 matrix to 7×7, $\left(7^2+7\right)$ determinants are required. Similarly, to reduce that matrix to 6×6, $\left(6^2+6\right)$ determinants are required. So, to reduce both of the 8×8 matrices to 6×6, $2\left(7^2+7+6^2+6\right)$ determinants are involved. At this point, both 6×6 matrices are mirrored, creating four matrices instead of two. This process continues until the final 3×3 size for each matrix is reached.

The number of determinants required for an 8×8 matrix, for example, is given by $2\left(7^2+7+6^2+6\right)+4(5^2+5+4^2+4)+8(3^2+3) = 492$.Letting $\log_2(N) = G$, this expression can be compacted and generalized using

$$\sum_{h=\frac{N}{2}+2}^{N-1}\left[2\left(h^2+h\right)\right]+\sum_{k=2}^{G}\left[2^k\sum_{m=\frac{N}{2^k}+2}^{\frac{N}{2^{k-1}}+1}\left[m^2+m\right]\right] \quad (5)$$

Expanding the second summation to include the $k = 1$ case and subtracting out the resulting factor, the expression becomes

$$\sum_{k=1}^{G}\left[2^k\sum_{m=\frac{N}{2^k}+2}^{\frac{N}{2^{k-1}}+1}\left[m^2+m\right]\right] - 2\Big(N^2+N+(N+1)^2+(N+1)\Big) \quad (1)$$

Rewriting the second summation using the identities $\sum_{i=1}^{t}\left[i^2\right] = \frac{t(t+1)(2t+1)}{6}$ and $\sum_{i=b}^{t}[i] = \frac{(t-b+1)(t+b)}{2}$, the expression becomes

$$\frac{1}{3}\sum_{k=1}^{G}\left[\frac{7N^3}{4^k}+\frac{18N^2}{2^k}+11N\right]-4N^2-8N-4, \quad (7)$$

which is equal to exchanging the limits of the summation and negating the k exponents, such that

$$\frac{1}{3}\sum_{k=-G}^{-1}\left[4^k 7N^3+2^k 18N^2+11N\right]-4N^2-8N-4 \quad (8)$$

This facilitates the use of the identity $\sum_{i=b}^{t}\left[i^x\right] = \frac{x^{t+1}-x^b}{x-1}$. Finally, the expression becomes

$$\frac{7}{9}N^3+2N^2-\frac{133}{9}N+\frac{11}{3}N\log_2(N)-4, \quad (9)$$

suggesting a complexity of $O(N^3)$ This result is precisely accurate for any N that is a power of 2. The error for other N is well below 0.01% for $N > 256$. Because the application of this algorithm is for $N \gg 256$, the computational complexity expression above is sufficiently accurate.

III. Comparison with LU Decomposition Method

A. Elimination of zero calculations for sparse matrices

In the proposed process, as well as in LU decomposition, some calculations yield a result of zero. In performing LU decomposition using systolic arrays, it is impossible to predict the zero results. However, in the proposed process, any 2×2 matrix with a column or row of zeros yields a determinant of zero. Therefore, that particular determinant computation does not need to be fully performed. Simply checking for a row or column of zeros will allow skipping many determinant computations. This will greatly increase the overall speed of the process. Given that prediction of zeros is not possible in LU decomposition using systolic arrays, the new process is significantly faster.

B. Number of Dividers

The proposed process includes a step where the first column of a given matrix is divided by its own first element. The largest matrix size is N, and therefore the most number of concurrent divisions is $2N$. Thus, only $2N$ dividers are required. On the contrary, LU decomposition using systolic arrays requires many dividers to parallelize the procedure. Because dividers are difficult and slow to implement, the difference in required dividers provides a significant speed gain in the new process as compared to LU decomposition.

C. Pivoting

Pivoting is a critical issue in parallel LU factorization [8] Pivoting is applied by arranging rows and/or columns of the matrix to choose the largest element as the pivot. This step maintains numerical stability during factorization. Pivoting in parallel LU decomposition has increased complexity because the arrangement of rows and columns requires greater communication and synchronization between processors[8]. It also exacerbates the problem of load imbalances. This load imbalance problem can become more prominent if matrices are stored within dynamic data structures.[1][8] Generally, the pivoting process in the LU decomposition algorithm requires us to maintain a separate matrix (comprising of 1's and 0's) which keeps a record of the various row and column interchanges that have occurred during the process of executing the algorithm. However, the method described in this paper does not require us to maintain a separate pivoting matrix, thereby reducing the memory requirement of the entire system.

D. Communication overhead

One of the crucial issues that a parallel implementation of LU factorization has is data dependences. If the matrix elements are stored among processors of a parallel processing platform, then each processor requires to communicate with other processors to access the matrix elements, which not only effects the efficiency of parallel algorithms but also increases the hardware complexity of the machines [8] This has been a persistent problem in many of the LU decomposition algorithms. Bounded broadcast is one of the ways to reduce the execution time in LU decomposition method [5]. In the proposed process, only the first column of elements in a given matrix is broadcasted to the right-most $(N-1)$ columns. The remaining calculations, namely the computations of each 2×2 determinant, are performance independetly and as such less time is required to simply pass the needed information between processing elements.

IV. Conclusions

This paper described a methodology for solving large-scale linear systems on a parallel processing platform using a variation on Cramer's rule. The proposed architecture has a comparable computational complexity to that of LU decomposition, while offers distinctive properties in terms of parallelism and storage efficiency. Thus, given P parallel processing units, the computational complexity of the method is $O(N^3/P)$. Moreover, the communication overhead, which is a major challenge in parallel LU factorization schemes, is overcome by the proposed algorithm. For large-scale systems, this architecture can be implemented on hardware platforms, such as FPGAs, to yeild a lower cost/performance ratio to that of supercomputers.

References

[1] O. Y. K. Chen, "A comparison of pivoting strategies for the direct lu factorization." [Online]. Available: citeseer.ist.psu.edu/293110.html

[2] E. Howard, *Elementary Matrix Theory*, 1966.

[3] H.Teimoori, A. M.Bayat, and E.Sarijloo, "A new parallel algorithm for evaluating the determinant of a matrix of order n." European Conference on Combinatorics, Graph Theory and applications.

[4] N. Johnson.J.R, Nagvajara.P, "High-performance linear algebra processor using fpga," 2004, drexel University Philadelphia.

[5] D. Kim and S. V. Rajopadhye, "An improved systolic architecture for lu decomposition," pp. 231–238, 2006.

[6] L.Mirsky, *An Introduction to Linear Algebra*. Dover Publications, 1982.

[7] X. Wang and S. Ziavras, "Parallel lu factorization of sparse matrices on fpga-based configurable computing engines," 2004. [Online]. Available: citeseer.ist.psu.edu/wang03parallel.html

[8] X. Wang and S. G.Ziavras, "Parallel direct solution of linear equations on fpga-based machines," in *Parallel and Distributed Processing Symposium, 2003*, 2003.