Optimized Architecture for Sparse LU Decomposition on Matrices with Random Sparsity Patterns

Dinesh Kumar Murthy
Ingram School of Engineering
Texas State University
San Marcos, TX, USA
d_m410 @txstate.edu

Semih Aslan
Ingram School of Engineering
Texas State University
San Marcos, TX, USA
aslan@txstate.edu

***Abstract* — This paper investigates a method for improving the performance of sparse Lower-Upper (LU) decomposition which is widely used to solve sparse linear systems of equations, appearing in many scientific and engineering application models. However, LU decomposition is considered a computationally expensive tool. When dealing with large sparse matrices, numerical decomposition takes more time using normal matrix LU implementation. The problem of interest here is the irregular sparsity pattern which limits performance gain. An efficient architecture for sparse LU decomposition is proposed for both symmetric and asymmetric matrices with random sparsity percentages and patterns. The algorithm spends time in simultaneous localization and mapping of the sparse matrix and then solving the linearized system. The performance of the algorithm with matrices of varying parameters is calculated and compared with a regular LU decomposition algorithm. In most cases, there are performance improvements in terms of speed, area, and power.**

Keywords – Pivoting; latency; linear systems; throughput; LU Decomposition; Field Programmable Gate Arrays (FPGAs).

I. INTRODUCTION

Numerical solutions of large linear systems are important for scientific and engineering applications like linear programming, circuit simulation, semiconductor device simulations, image processing, and power system modelling. Solving such systems of equations generally involves two methods: the direct method including Cholesky decomposition, LU decomposition, QR decomposition, and iterative methods. The Cholesky decomposition is a special form of LU decomposition which deals with symmetric positive definite matrices. Adapting these parallel architectures to solve large sparse linear system of equations is a main focus of research [1].

A number of software- and hardware-based approaches have been developed to obtain better solutions for LU decomposition. Software implementation includes a Super nodal approach which considers the matrix as sets of continuous columns with the same nonzero structure, and a Multifrontal approach organizing a large sparse matrix into a small dense matrix [2]. Field Programmable Gate Arrays (FPGAs) have unique advantages in solving these problems. Depending on the characteristics of the algorithm, an architecture is designed with reconfigurable computational resources and memory. The consumption of energy is reduced and is a platform for experimentation and verification. Though there are many FPGA-based architectures for dense matrices, only a few are proposed for sparse matrix decomposition [3][4]. The three main direct methods for sparse LU decomposition are left-looking, right-looking and count algorithms. The proposed FPGA-based architecture for sparse LU decomposition can efficiently decompose the sparse matrix with varying sparsity patterns. The architecture first factorizes the columns from the lower triangular part of the matrix in parallel with the rows from the upper triangular part of the matrix. The control structure performs pivoting operations while factorizing the rows and columns of the matrix.

The rest of the paper is organized as follows. Section II introduces the theoretical background of LU decomposition, Section III describes the architectural design with proposed algorithm, Section IV proves the simulation of the design using Xilinx Vivado Design suite with verification of MATLAB results, and Section V provides FPGA mapping of the design and discussion of performance results. This paper concludes with a brief conclusion in Section VI.

II. BACKGROUND

A. Sparse LU Decomposition

LU decomposition or factorization is a popular matrix decomposing method for many numerical analysis and engineering science problems. It decomposes the matrix as a product of the lower triangular matrix (**L**) whose diagonal elements are equal to 1 and all the elements above the diagonal are equal to 0, and an upper triangular matrix (**U**) whose elements below the diagonal are equal to 0. If **A** is a square matrix, LU decomposes **A** with proper row and/or column orderings into two factors, which is shown in Fig. 1.

$$\boldsymbol{A} = \boldsymbol{LU} \tag{1}$$

$$\begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ L_{21} & 1 & 0 \\ L_{31} & L_{32} & 1 \end{pmatrix} \times \begin{pmatrix} U_{11} & U_{12} & U_{13} \\ 0 & U_{22} & U_{23} \\ 0 & 0 & U_{33} \end{pmatrix} \tag{2}$$

LU decomposition is a direct method that can solve large systems of linear equations that arise from important applications such as circuit simulation, power networks, and structural analysis [5]. To ensure stability during LU decomposition, pivoting operations are performed to remove zero elements from the diagonal of matrix **A**. Without proper pivoting, the decomposition may fail to materialize. A proper permutation in rows or columns is sufficient for LU decomposition, which is also known as Partial Pivoting. This approach is suitable for a square matrix, and it is numerically stable in practice.

$$\boldsymbol{PA = LU} \tag{3}$$

On the other hand, Full Pivoting involves both row and column permutations.

$$\boldsymbol{PAQ = LU} \tag{4}$$

where **Q** is a permutation matrix which reorders the columns of **A**.

The forward reduction and backward substitution techniques are more stable compared to matrix inverses to solve systems of linear equations because every nonsingular matrix possesses an LU decomposition. When compared with regular matrices, sparse matrices can benefit from algorithms that reduce the number of operations which are required to calculate **L** and **U**. However, the disadvantage is that sparse methods will suffer from irregular computation patterns as they are dependent on the nonzero structure of the matrix.

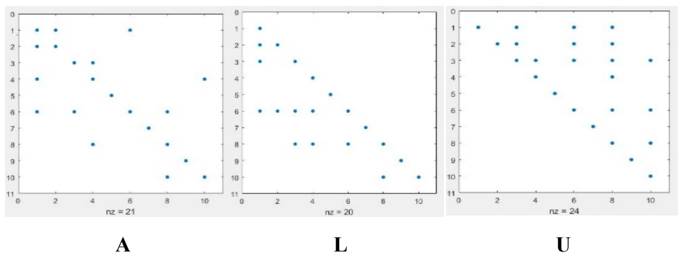


Figure 1. Example of a sparse matrix and its factors L and U

B. Related Work

There have been many architectures proposed for sparse LU decomposition which either target domain-specific sparsity patterns or require a pre-ordered symmetric matrix [6]. Blocking is a useful technique for gaining higher throughput for dense matrices. When decomposing in blocks using a Block Sparse Row (BSR) format for solving linear systems, it is limited to a matrix containing square blocks of a single dimension. When decomposition is executed in parallel, it often tries to avoid pivoting by using threshold pivoting or static pivoting beforehand. The architecture proposed in [7] implements a right looking algorithm and includes a hardware mechanism for pivoting. The performance of this is primarily I/O bandwidth limited.

Another implementation captures the static sparsity pattern and is exploited to distribute the data flow representation of computation for circuit simulation [8]. A more general hardware design is proposed parallelizing a left looking algorithm to support processing symmetric positive definite or diagonally dominant matrices. The factor limiting architecture efficiency is dynamically depending data dependencies. One more algorithm proposes choosing a pivoting strategy, where the matrix is decomposed block-wise. FPGAs have been shown to be effective in accelerating a wide range of matrix operations in recent years [9] [10].

The algorithm with row pivoting yields **LU=PA**, where the matrix overwrites **A** with **LU-I**, and **I** is an identity matrix. The first half of the algorithm will be triangular solving, leaving behind pivoting and scaling. In the case of sparse matrix, it will be inefficient for swapping rows. Due to having a single unreduced row or column, full pivoting is not easily achievable. The control system is implemented as a Finite State Machine (FSM), which tracks the progress of the units for synchronization. The algorithm for sparse matrix LU decomposition is in Fig. 2.

Algorithm

```
A → $n×n$ sparse matrix
P → $n×n$ identity matrix
[n, m] = size(A)
set reset high
U = A
L = P = $I_{n*n}$

[Perform pivoting operation]
function pivot (A, P, i)
        P = choose pivot ($A_i$: end, i)
        if ($P \neq k$) then
                SWAP ($A_i$, *, $A_p$, *)
                SWAP ($P_i$, *, $P_p$, *)
        end if
        return (A, P)
end function
[Interchanging rows in matrix]
If $m \neq j$
  U ([m, j], :) = U ([j, m], :)
  P ([m, j], :) = P ([j, m], :)
  If j<=2
    L ([m, j], 1: j-1) = L ([j, m], 1: j-1)
  end
end
[Update row and column entries]
for i = j+1 to n
  for j = 1 to n
    $L_{i,j} = U_{i,j} / U_{j,j}$
      for k = j+1 to n-1
          U (i, *) = U (i, *) - L (i, j) × U (j, *)
        end
  end
end
```

Figure 2. Pseudo code for Sparse LU Decomposition

III. Sparse LU Decomposition Architecture

The proposed approach for sparse LU decomposition consists of the following operations:

1. Pivoting strategy, when A has nonzero entries which are at fill-up locations.
2. Symbolic decomposition, which estimates the memory requirements for L and U factors.
3. Numerical calculation, which is computed using Gaussian elimination.

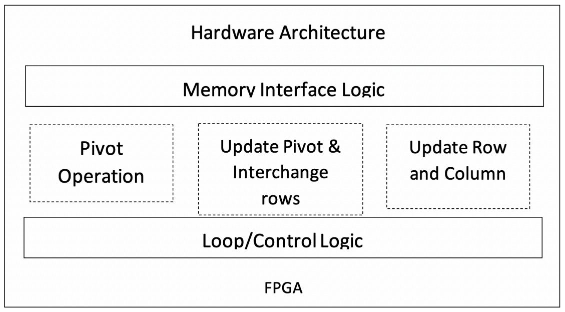


Figure 3. Proposed LU Decomposition Hardware Algorithm

To maximize performance, LU hardware is designed to focus on maintaining a regular computation and memory access pattern. Fig. 3 shows a block diagram of the proposed hardware algorithm. The control and memory access handle the operations performed for decomposing the matrix. The design ensures the memory will have enough space to store the values [11] [12].

A. Pivot Operation

In order to perform a pivoting operation, the design includes usage of lookup tables and memory pointers to keep track of memory mapping. It performs a pivot search for each step of matrix elimination. Index pointers are created for each pivot to store the row and column physical address, accordingly. These physical addresses are then used to fetch the values from memory. These values are sequentially checked as they arrive for the absolute maximum values with the index. Using a register, it is stored as a pivot element. The minimum amount of memory utilized is proportional to the size of the matrix. Once pivoting is complete, an update is sent back to the lookup tables.

B. Update Pivot and Interchange Rows

The “Update Pivot and Interchange Rows” logic block performs normalization prior to elimination for the pivot values of row and column requested from memory. The necessary data such as pivot index, values and column are inferred from the previous state. This process is executed one by one after each pivot value is fetched and read. The updated row and column values and the normalized row and column values are then stored in registers.

C. Update Row and Columns

The remaining computations required are performed during this transition state. First, it indicates if the given row or column should be updated. Second, it manages the addresses of nonzero entries that are to be stored. This unit contains the necessary floating-point multiplier and adder to perform the required arithmetic operations [13]. This unit is operational in parallel to maximize the utilization of all logic units. This will update the number of updated logics that fits in FPGA chip. There are enough resources available in the FPGA that can accommodate all of the units.

IV. Implementation and Verification

Various arbitrary matrices with different sparsity patterns are generated using MATLAB and are tested using the hardware architecture. A parameter n is included along with

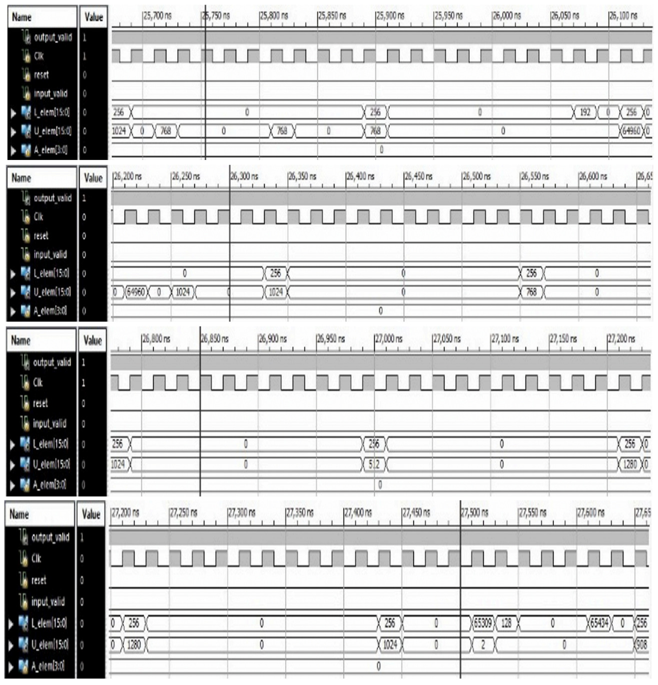


Figure 4. Simulation Waveform for LU Decomposition

The simulated results are stored in an external .txt file and are verified with the results from MATLAB for precision loss. For **L** matrix, the error ranges between -0.0872 to 0.0357 and for **U** matrix it ranges between -0.0108 to 0.0057 as shown in Fig. 5 below.

$$L_{diff} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.001 & 0.003 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 7.81e^{-04} & -0.001 & -0.002 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2.60e^{-04} & -0.001 & 0.002 & 0.003 & 0 & 0 & 0 & 0 \\ 0 & 0 & -5.20e^{-04} & -0.001 & 9.44e^{-04} & 0.001 & -0.001 & 0 & 0 & 0 \\ 0 & 0 & -0.001 & -0.002 & 0.0036 & 3.24e^{-04} & -0.003 & 0.009 & 0 & 0 \\ 0 & 0 & 0 & -0.002 & 0.0059 & -0.004 & 0.003 & -0.01 & 0.006 & 0 \end{bmatrix}$$

$$U_{diff} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.007 & 0.001 & 0 & 0.004 & -5.20e^{-04} & -0.002 & 0.001 & 0.003 \\ 0 & 0 & -0.007 & -0.023 & -0.02 & -0.005 & -0.01 & -0.029 & -0.026 & 0.011 \\ 0 & 0 & 0.003 & -0.01 & -0.003 & -0.005 & -0.00 & -0.013 & -0.017 & -0.003 \\ 0 & 0 & 0 & 4.44e^{-016} & -0.01 & 0.01 & 0.04 & 0.017 & 0.014 & 0.012 \\ 0 & 0 & 0 & 0.01 & 0.023 & 0.01 & -0.003 & 0.008 & 0.013 & -0.009 \\ 0 & 0 & -0.007 & 0.02 & 0.003 & 0.019 & -0.02 & 0.011 & 0.027 & 5.95e^{-04} \\ 0 & 0 & 0.007 & 0.01 & -0.003 & -0.023 & 0.04 & -0.019 & -0.011 & -0.00 \end{bmatrix}$$

Figure 5. MATLAB Calculated Errors Values

V. Performance Analysis

A comparison of LU decomposition of sparse matrix of size 10x10 and 100x100, with a different sparsity range of 10% to 50% is shown in Fig. 6 below. The proposed LU decomposition design was able to achieve lower latency than the regular algorithm LU decomposition. The results are also verified with the MATLAB LU decomposition outputs for precision loss.

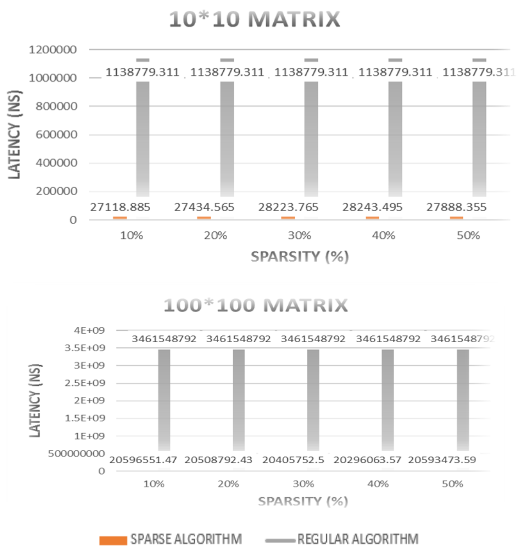


Figure 6. Latency Comparasion

A comparison of the throughput calculated from the sparse matrix algorithm and regular algorithm is plotted in the form of a graph and is represented in Fig. 7. As the throughput needs to be high for better performance, we are able to infer from the graph that high throughput was achieved.

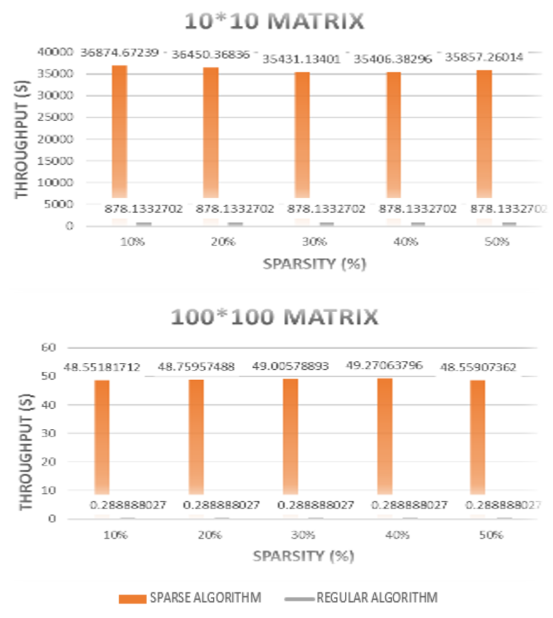


Figure 7. Block Diagram of the TX and RX Module

The data from Table I shows that the matrix storage format proposed in this research was able to achieve minimum resource utilization, as opposed to the traditional regular LU decomposition algorithm. The proposed design was implemented on a Xilinx Artix7 XC7A100T-1CG324C board comprising of 15,850 logic slices and a maximum of 4,860 Kbits fast block RAM. This is achieved with optimization through the implemented design for the LU decomposition. A difference in about one third of the total resources utilized was achieved, as seen in Fig. 8 and 9, respectively.

The performance of the design is based on the architecture and its parameters. As an FPGA has enough computational resources and the design is memory-bound, the performance is totally dependent on memory access time.

TABLE I. Resources Utilized for Proposed Algorithm

Device Utilization Summary			
		Proposed Sparse Algorithm	**Regular Algorithm**
Slice Logic Utilization	**Available**	**Used**	
Slice Registers	126,800	3,420	10,863
Slice LUTs	63,400	11,211	16,807
Memory	19,000	8	64
Occupied Slices	15,850	3,504	5,455
IOBs	210	40	40

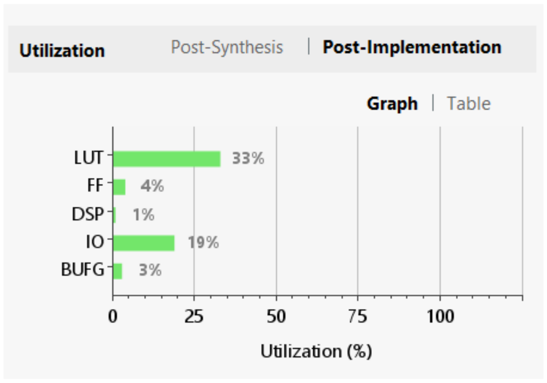
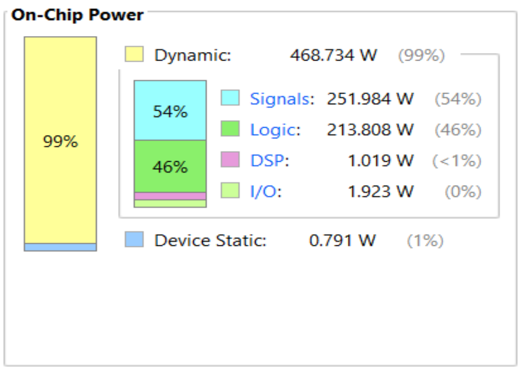


Figure 8. FPGA Design Utilization



Dynamic:	468.734 W	(99%)
Signals:	251.984 W	(54%)
Logic:	213.808 W	(46%)
DSP:	1.019 W	(<1%)
I/O:	1.923 W	(0%)
Device Static:	0.791 W	(1%)

Figure 9. Design Power Requirements

VI. Conclusion

Numerous engineering and machine learning applications rely largely on solving linear equations using LU decomposition, due to rapid developments in the field of mathematics and computation. Compared with a CPU and GPU, the FPGA does not have an instruction set. Instead, it possesses a number of reconfigurable logic blocks which could perform any digital logic function. In this paper, a computational implementation of the LU decomposition is proposed using an optimized algorithm. The proposed architecture can achieve further improvement by increasing the overall design clock.

References

[1] M. Wielgosz, G. Mazur, M. Makowski, E. Jamro, P. Russek, and K. Wiatr "Analysis of the Basic Implementation Aspects of Hardware-Accelerated Density Functional Calculations," OJS Computing and Informatics, vol. 29, no. February, pp. 989–1000, 2010.

[2] A.Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One Trillion Edges : Graph Processing at Facebook-Scale," , Proceedings of the VLDB Endownment, vol. 8, no. 12, pp. 1804-1815, 2015.

[3] A. Pinar and M. T. Heath, "Improving Performance of Sparse Matrix-Vector Multiplication," Proceedings of the 1999 ACM/IEEE Conference on Supercomputing, January 1999 Pages 30–39 doi:10.1145/331532.331562.

[4] T. Mattson et al., "Standards for graph algorithm primitives," IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2013, pp. 1-2, doi: 10.1109/HPEC.2013.6670338.

[5] S. Jain, N. Kumar, J. Singh, and M. Tiwari, "FPGA Implementation of Latency, Computational time Improvements in Matrix Multiplication," International Journal of Computer Applications, 2014, vol.86, no.8, doi:10.5120/15007-3261.

[6] S. Aslan and J. Saniie, "Matrix Operations Design Tool for FPGA and VLSI Systems," 2016, Circuits and Systems, vol. 7, no.2, pp. 43–50, doi: 10.4236/cs.2016.72005.

[7] P. Greisen, M. Runo, P. Guillet, S. Heinzle, A. Smolic, H. Kaeslin, and M. Gross, "Evaluation and FPGA Implementation of Sparse Linear Solvers for Video Processing Applications", in IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 8, pp. 1402-1407, Aug. 2013, doi: 10.1109/TCSVT.2013.2244797.

[8] W. Liu and B. Vinter, "An Efficient GPU General Sparse Matrix-Matrix Multiplication for Irregular Data", 2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, 2014, pp. 370-381, doi: 10.1109/IPDPS.2014.47.

[9] J. Johnson, T. Chagnon, P. Vachranukunkiet, P. Nagvajara, and C. Nwankpa, "Sparse LU Decomposition using FPGA", International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA), pp. 1-12, 2008.

[10] G. Wu, X. Xie, Y. Dou, J. Sun, D. Wu, Y. Li, and A. S. Matrix, "Parallelizing Sparse LU Decomposition on FPGAs", 2012 International Conference on Field-Programmable Technology, Seoul, Korea (South), 2012, pp. 352-359, doi: 10.1109/FPT.2012.6412160.

[11] L. Polok and P. Smrz, "Pivoting Strategy for Fast LU Decomposition of Sparse Block Matrices", HPC'17: Proceedings of the 25th High Performance Computing Symposium April 2017, no. 14, Pages 1–12.

[12] X. Wang and S. G. Ziavras, "Parallel LU Factorization of Sparse Matrices on FPGA-Based Configurable Computing Engines," Wiley Concurrency Computat.: Pract. Exper.,, vol. 16, no. April, pp. 319-343, 2004.

[13] Siddhartha and N. Kapre, "Breaking Sequential Dependencies in FPGA-Based Sparse LU Factorization," 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, Boston, MA, USA, 2014, pp. 60-63, doi: 10.1109/FCCM.2014.26.