OPTIMIZED ARCHITECTURE FOR SPARSE LU DECOMPOSITION ON MATRICES WITH RANDOM SPARSITY PATTERNS

Dinesh Kumar Murthy, Semih Aslan

Electrical Engineering, Ingram School of Engineering
Texas State University
San Marcos, Texas, 78666, USA

THE SIXTEENTH INTERNATIONAL CONFERENCE ON DIGITAL TELECOMMUNICATIONS

ICDT 2021

Agenda

- Problem Statement
- Research Objective
- Sparse Matrices
- Why FPGS's?
- Sparse LU Decomposition
- Hardware Design
- Core Optimization
- Synthesis
- Implementation
- Error Analysis
- Goal Measurement Metrics
- Conclusion
- Future Work

Problem Statement

Indirect Addressing

- Indirect addresses must address the non-zero entries of a sparse matrix in its index array leading to random accesses that require more memory transactions and lower cache hit rate.

Memory Allocation

- The distribution of zero and non-zero entries are not known in advance. Pre-allocating memory blocks of a specific size may waste memory when the intersection of nodes is large.

Low Arithmetic Intensity

- This is caused by the lack of temporal locality in the access to sparse matrices. If the matrix is not structured or blocked, most of the entries in cache line fetched to get an element remain unused causing high memory overhead for sparse matrix operation.

Research Objective

To determine an algorithm for LU Decomposition by minimizing gate count, area, computational time, latency, number of multiplication & addition hardware and to improve throughput.

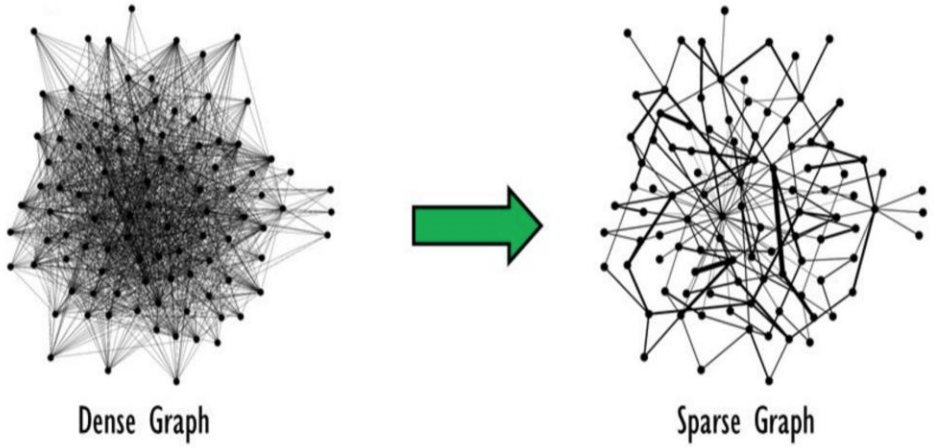
The developed algorithm must be capable of handling matrices of various sizes and should be simple to implement and highly scalable.

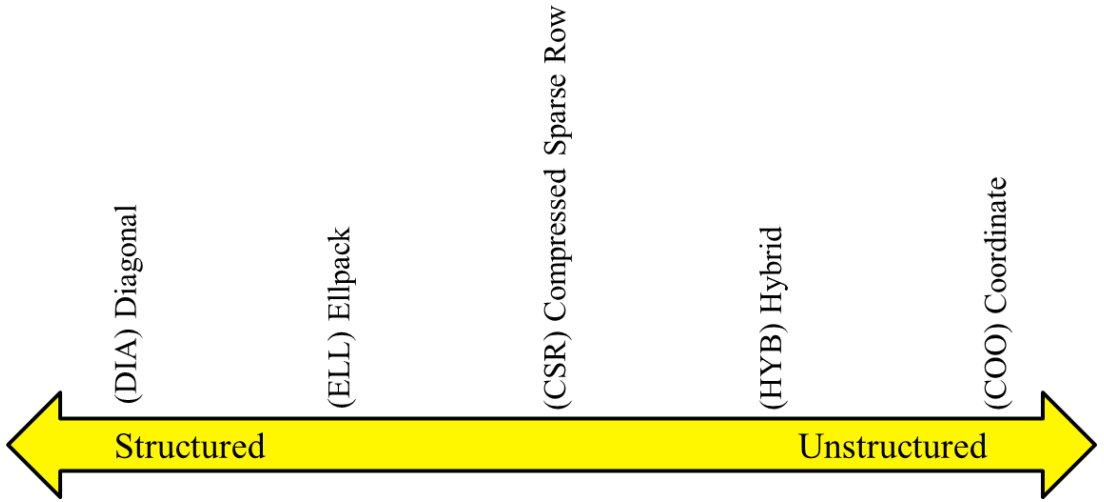
The primary goal of the thesis is to improve efficiency and reduce the resources used for the operation.

Comparison of the results and investigate the possible solutions and approaches for scaling up the design for larger matrix more efficiently.

Sparse Matrices

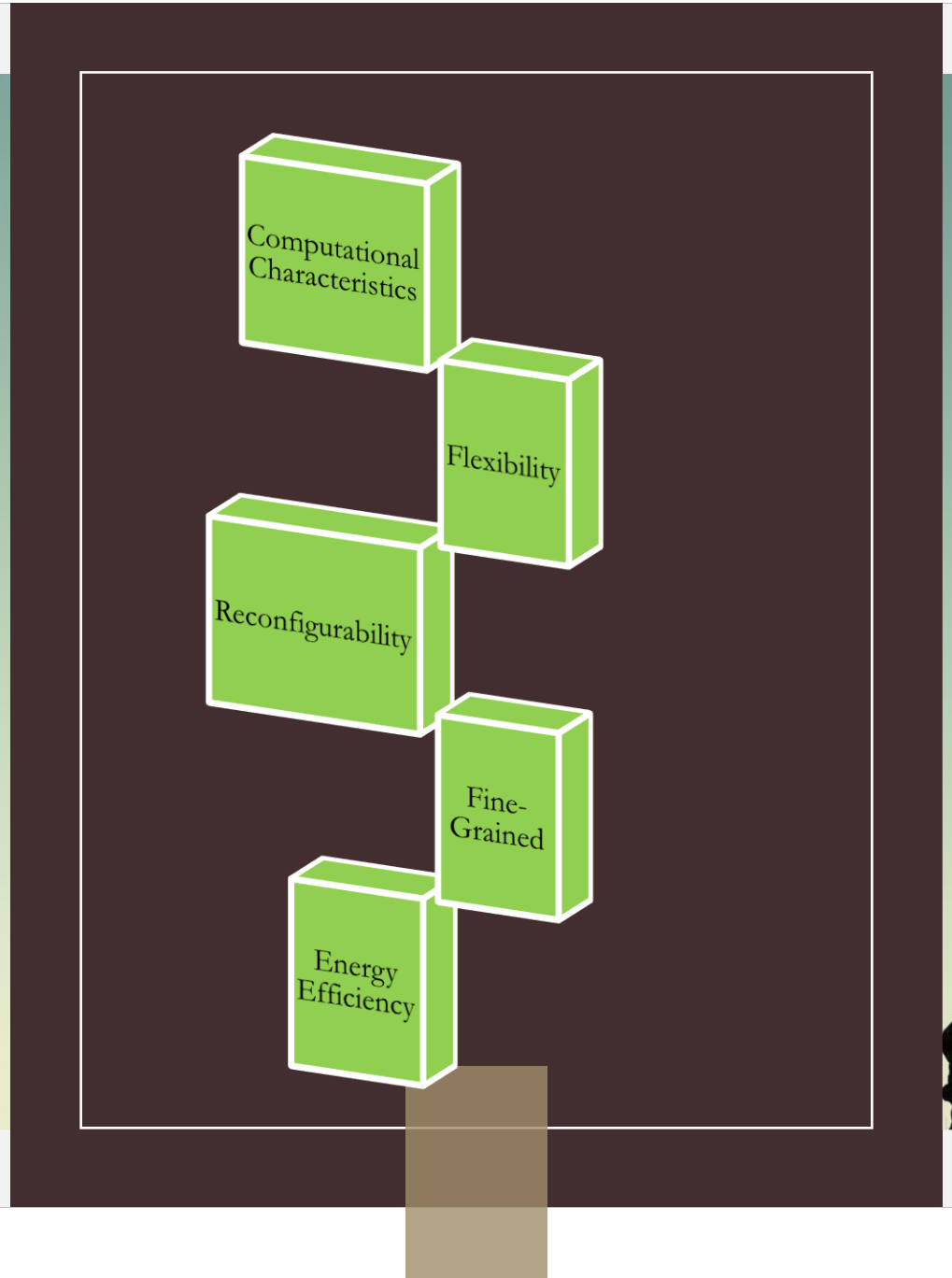
- In numerical analysis, a **sparse matrix** is a matrix in which most of the elements are zero.
- Large sparse matrices often appear in scientific or engineering applications when solving partial differential equations.
- When storing and manipulating sparse matrices on a computer, it is beneficial and often necessary to use specialized algorithms and data structures that take advantage of the sparse structure of the matrix.

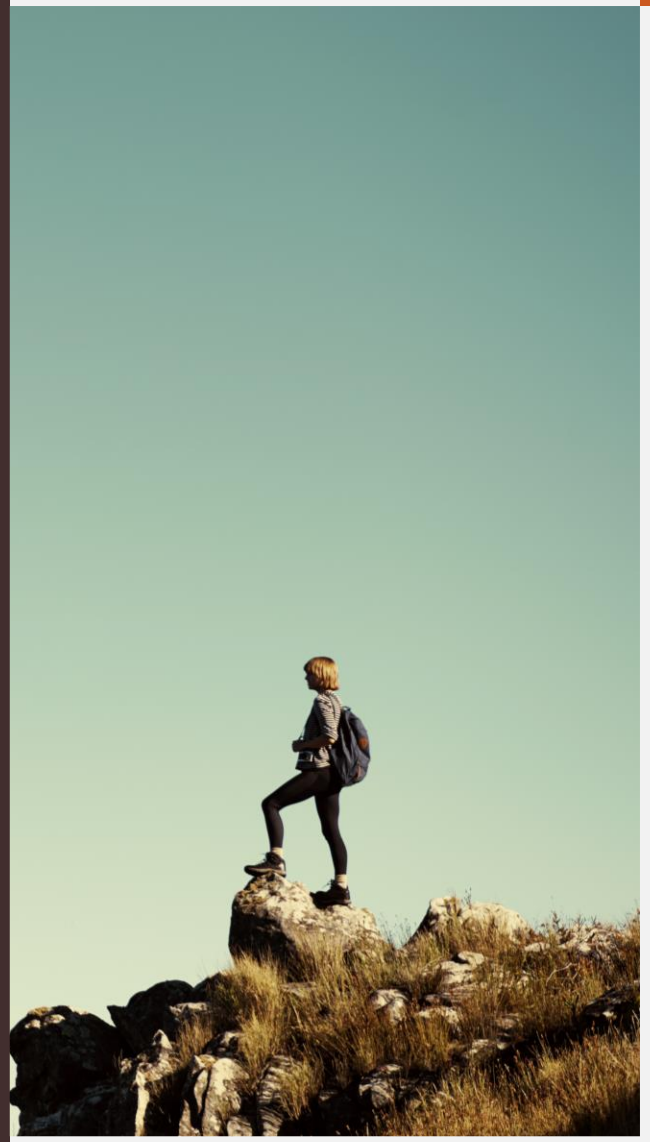




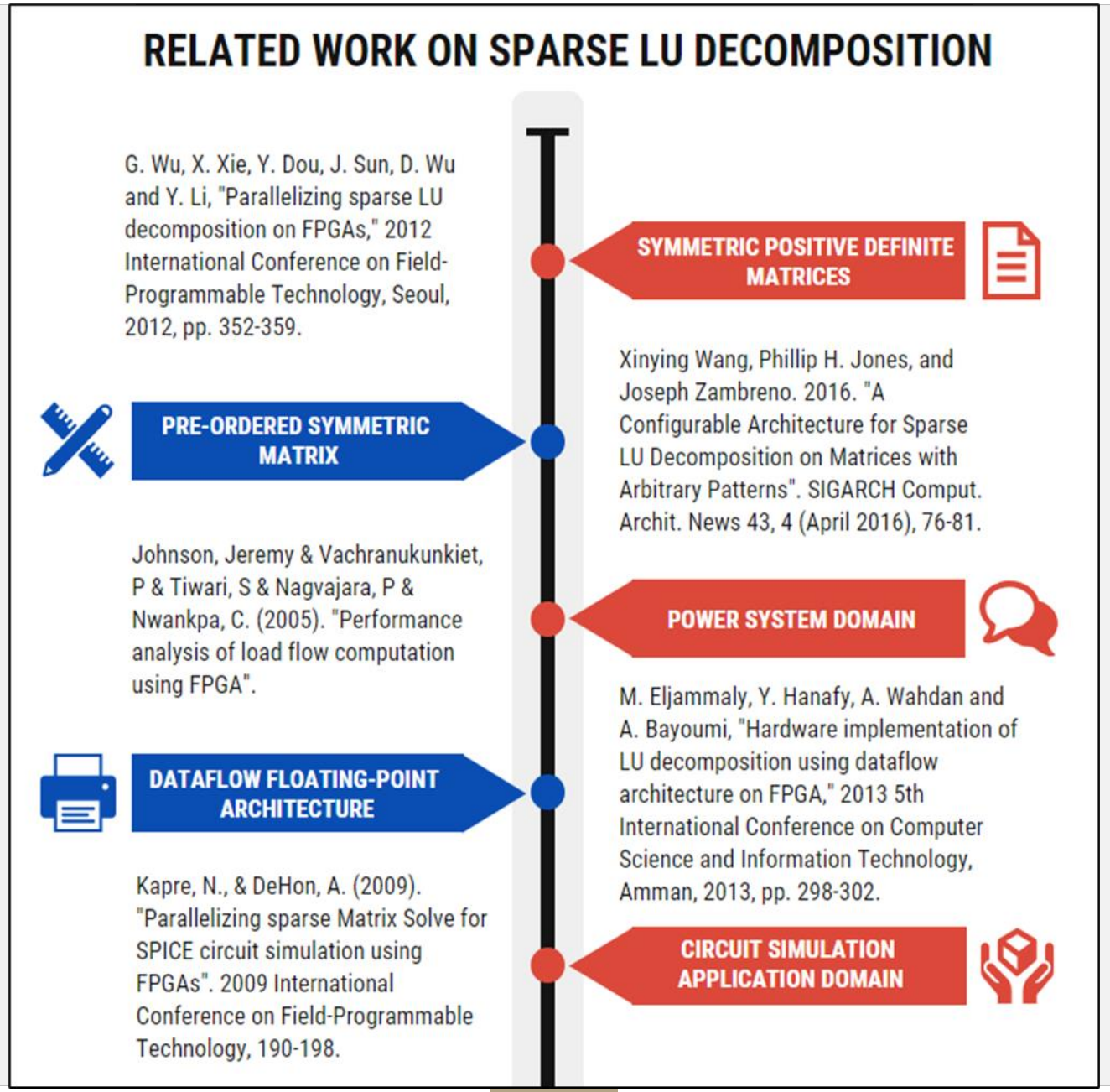
NEED FOR SPARSE FORMAT

Why FPGA's?


Computational Characteristics
Flexibility
Reconfigurability
Fine-Grained
Energy Efficiency



RELATED WORK ON SPARSE LU DECOMPOSITION



Sparse LU Decomposition

- Sparse LU decomposition is widely used in numerical analysis and engineering science.
- It factors a matrix as a product of lower triangular matrix (L) whose diagonal elements are equal to 1, and all the elements above 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 or permutations into two factors.

Common methods include

- ❑ Left-Looking Algorithm
- ❑ Right-Looking Algorithm
- ❑ Crout Algorithm

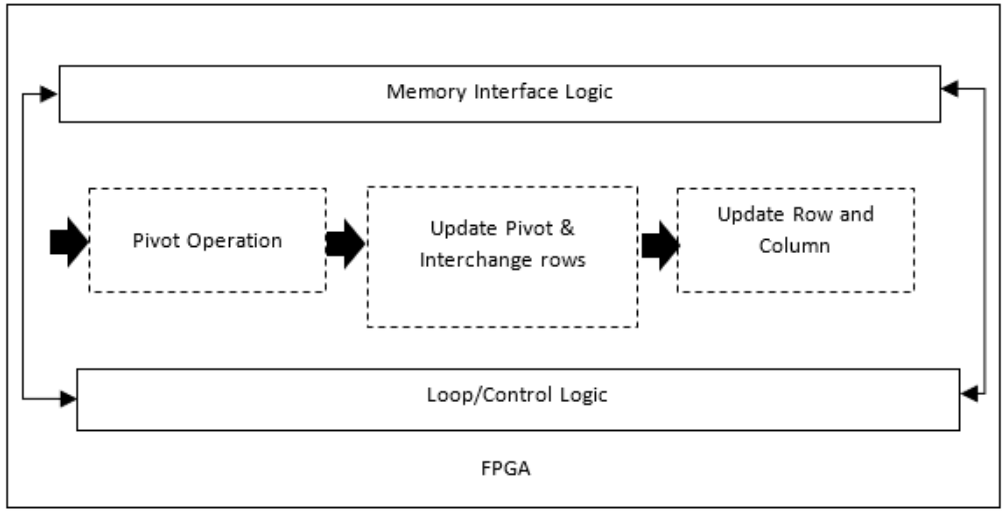
Applications

- ❖ Circuit Simulation
- ❖ Power System Modeling
- ❖ Image Processing

$$A = LU$$

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

Hardware Design



Block Diagram of Sparse LU Decomposition

- For maximizing the performance, LU hardware is designed to focus on maintaining regular computation and memory access pattern.
- The control and memory access handles the operations performed for decomposing the matrix.
- The design ensures the memory will have enough space to store the values.
- The performance of LU decomposition of the sparse matrix depends heavily on the quality of the placement tool.
- The initial design algorithm is inspired from Doolittle and the right-looking algorithm for sparse LU decomposition.

Core Optimization

Pivot Operation

- Conducts pivot search for each matrix elimination step.
- Index pointers are created for each pivoting to store row and column address.
- These value are sequentially checked for absolute maximum with index.

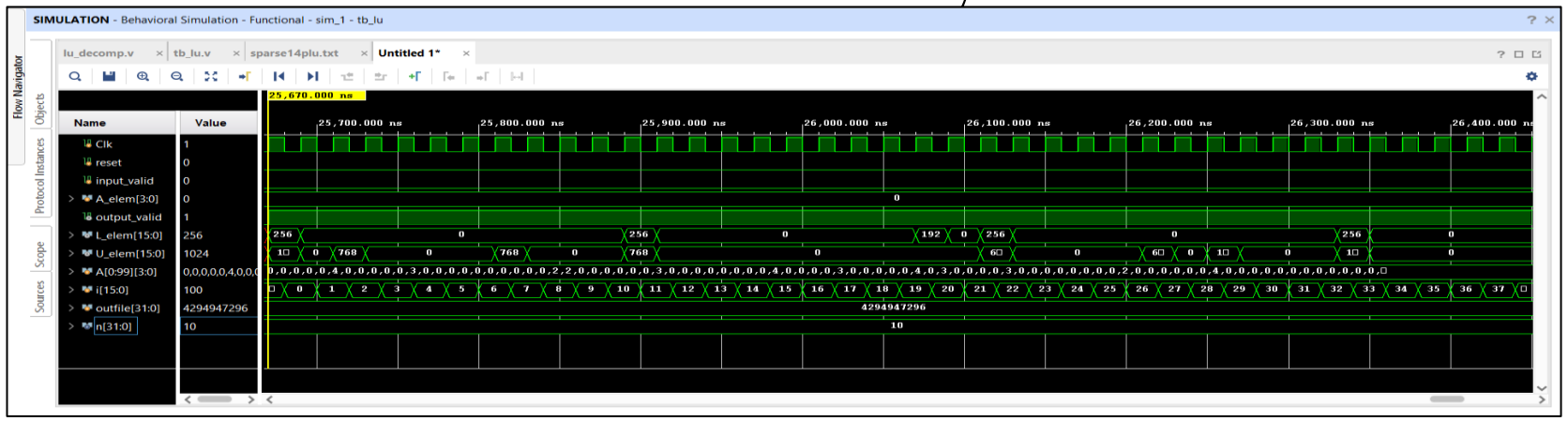
Update Pivot and interchange rows

- The update state will be responsible for computing the core computations of right looking algorithm.
- The logic performs normalization before elimination of the pivot values.

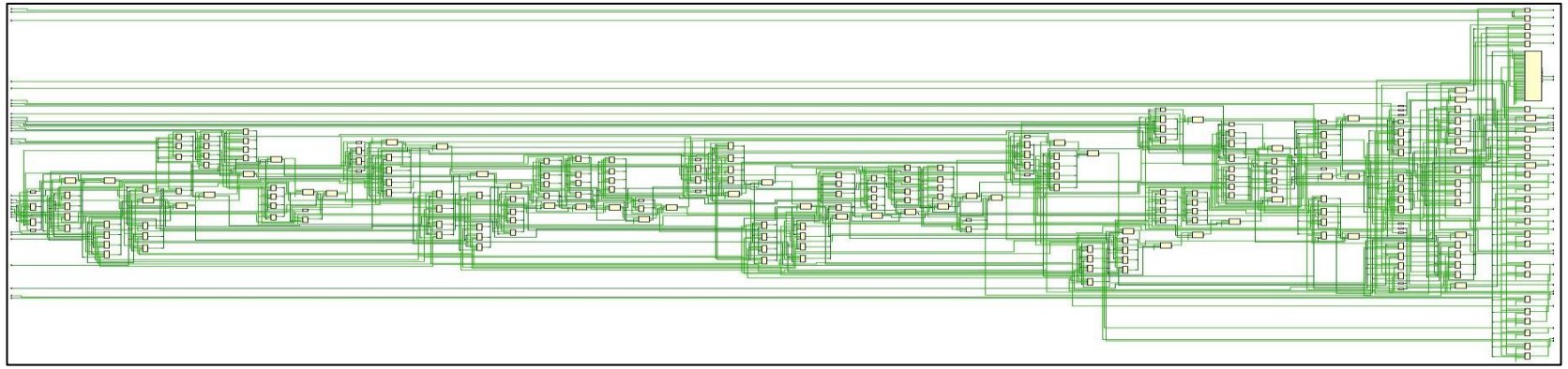
Update row and column

- First, it indicates if the row or column is to be updated from previous state memory.
- It manages the addresses of the nonzero to be stored.
- This unit operates in parallel for maximizing the utilization of all logic units.

Simulation and Synthesis



Design Simulation: LU Decomposition



Implemented Design Engine: LU Decomposition

Error Analysis

The simulated results are compared with MATLAB results for error analysis and is denoted as;

$$e_i = y_i - \bar{y}_i$$

Where y_i is the actual result from Vivado Design Suite and $\bar{y}_i$ is the result obtained using MATLAB tool.

$$L_{diff} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.001 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.001 & 0.003 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.007 & -0.001 & -0.002 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.002 & -0.001 & 0.002 & 0.003 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.005 & -0.001 & 0.006 & 0.001 & -0.001 & 0 & 0 & 0 \\ 0 & 0 & -0.001 & -0.002 & 0.0036 & 0.001 & -0.003 & 0.009 & 0 & 0 \\ 0 & 0 & 0 & -0.002 & 0.0059 & -0.004 & 0.003 & -0.01 & 0.006 & 0 \end{bmatrix}$$

PRECISION LOSS

L = -0.0871 to 0.0357

U = -0.0108 to 0.0057

$$U_{diff} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.004 & -0.002 & -0.002 & 0.001 & 0.003 \\ 0 & 0 & 0 & 0 & 0 & -0.005 & -0.01 & -0.029 & -0.026 & 0.011 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.00 & -0.013 & -0.017 & -0.003 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.017 & 0.014 & 0.012 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.013 & -0.009 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.003 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Goal Measurement Metrics

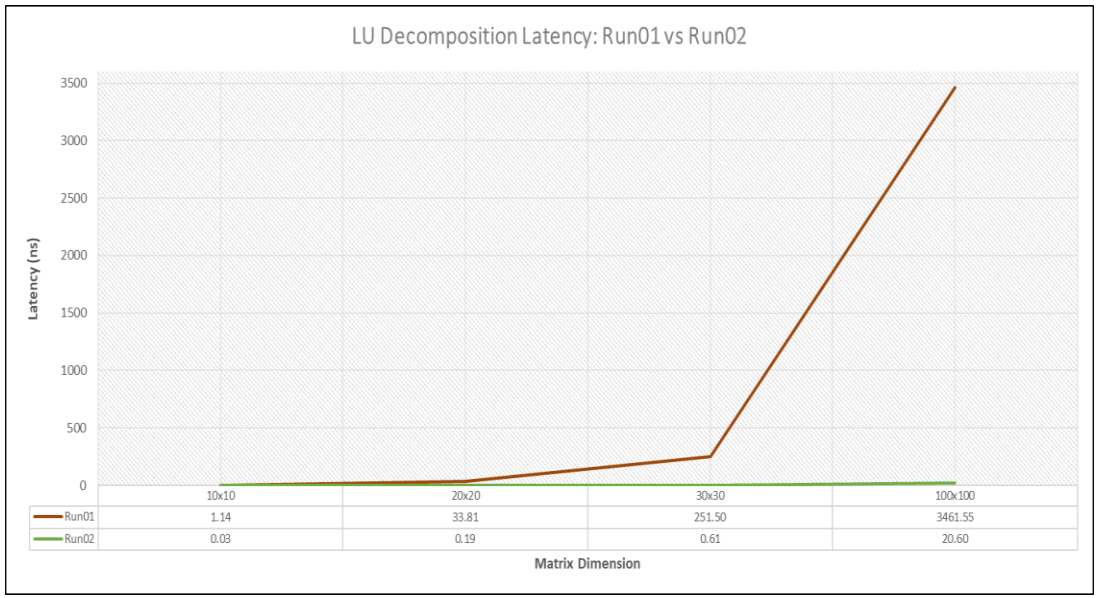
$$\boldsymbol{Latency}_n(ns) = \sum_{in}^{out} clock_{cycles} \times T_{min}(ns)$$

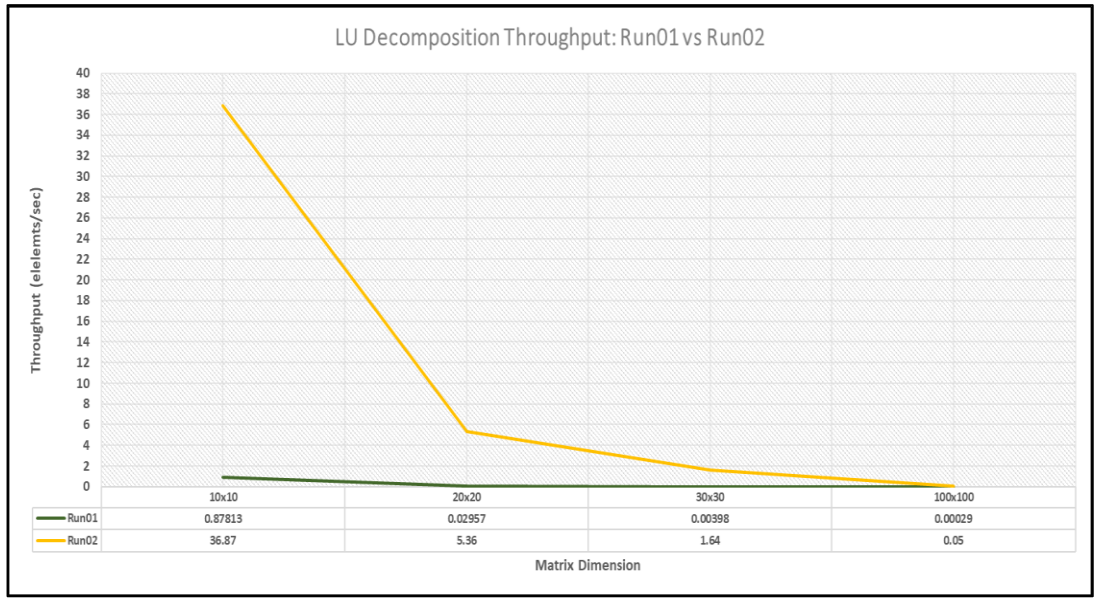
where n – *matrix size, with n=10,20,30, …100;*

T_{min} – *Minimum Clock Period of the design*

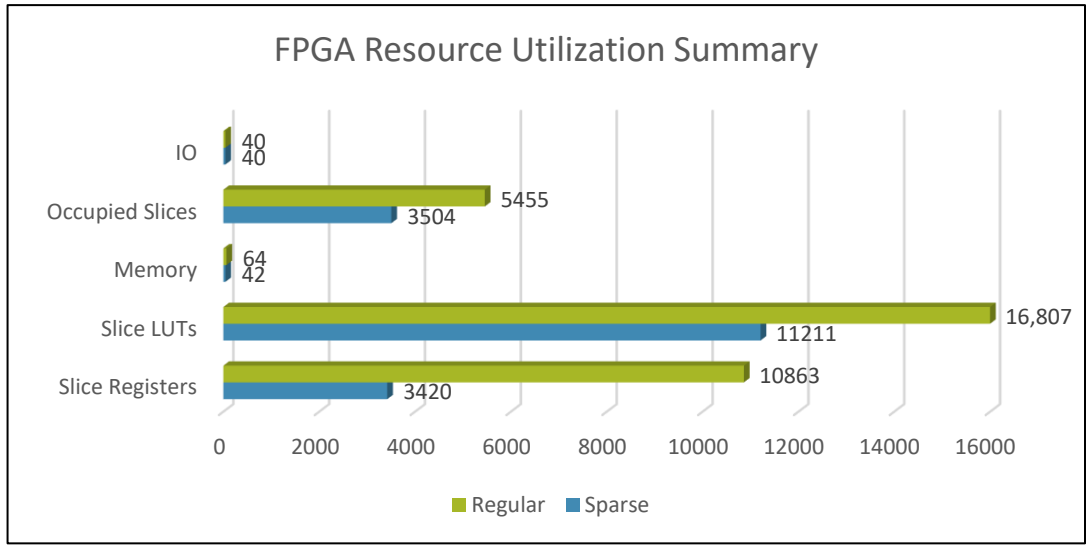
$$Throughput_n(elements/sec) = \frac{1\,sec \times 1\,bit}{latency_n}$$

where n – *matrix size, with n = 10,20,30, … 100*





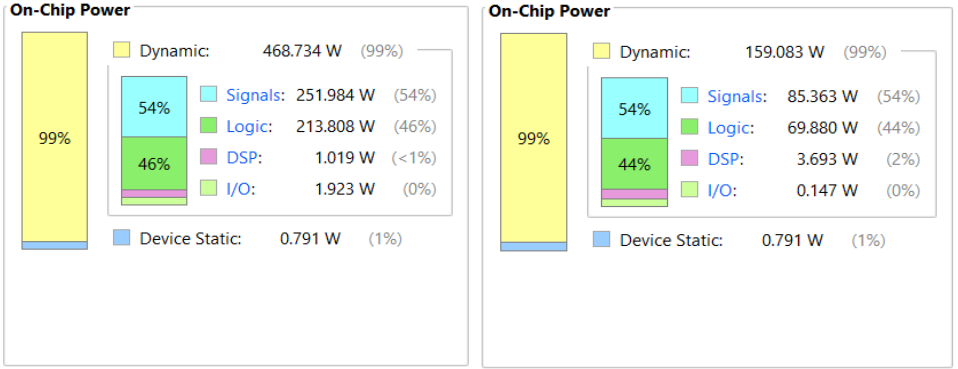
Performance Comparison



FPGA Resource Utilization: Regular vs Sparse for Sparse LU Decomposition

Hardware Implementation: Best- and worst-case Delays

	Input (ns)	Output (ns)
Best Case Delay (min)	7.41	6.54
Worst Case Delay (max)	19.68	6.71



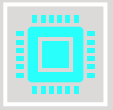
Regular Method Proposed Method

Power Analysis: Sparse LU Decomposition

Conclusion

The overall design was successful as the results were demonstrated with data from the implementation of LU Decomposition operation.

When comparing the performance to the regular algorithms and implementation, a significant achievement was made in performance and improved upon.



The design has simple and scalable implementation that consists of a small number of input and output parameters.

We have explored the optimizations not only for a specific application domain, but to make a generic architecture to be implemented irrespective of the application domain.

Future Scope

- Opportunities for future work includes increasing the simulations for a variety of benchmark matrices from all application domains and exploring further optimization.
- Research improvement in this area is needed for increase in logic resources by comparable increase in I/O bandwidth and on-chip memory capacity, especially when the matrix sparsity is unstructured and randomly distributed.
- It would be interesting to seek further optimization to obtain efficient hybrid algorithms for different arbitrary matrices.

Thank You