XILINX OPEN HARDWARE COMPETITION 2022: Acceleration of LU Decomposition on FPGAs

Yichen Zhang

16th June 2022

Team number: xohw22-006
Supervisor: Dr Danial Chitnis

Abstract

A brief document to illustrate how to use LaTeX.

1 Introduction

1.1 Sparse Linear System

Solving a linear system $Ax = b$ is relatively a basic algorithm problem and is closely related to the applications of almost every field, including engineering, physics, chemistry, etc. It is at the heart of various algorithms and applications, including computational geometry, circuit simulation and data science. A variety of approaches have been developed to solve the linear system, which can mainly be classified into two types: direct methods and iterative methods.

The direct methods solve the sparse linear system mainly by computing the determinant, inversion or factorisation of a matrix. For example, Cramer's rule calculates the solution in terms of the determinants of the coefficient matrix and the right-hand-side vector. Gaussian elimination finds the solution by calculating the inverse of the system. LU decomposition solves the system by factorising the system into a lower triangular matrix L and an upper triangular matrix U.

On the other hand, the iterative method is more modern. It attempts to solve the linear system based on an initial condition and successive approximates to the final solution, such as spectral sparsificaiton of graphs [1] and division-free inversion [2]. However, the behaviour and its stability of iterative methods is heavily based on the convergence[1] of the input matrices. On the contrary, the operation cost of behaviour methods is closely related to the cost of the matrix multiplication, making it preferred to the randomised and iterative methods in real applications as the directed methods are more robust and predictable.

LU decomposition is a direct method that can solve a large sparse linear systems multiple times, with various applications in circuit simulation, structure analysis, power networks, etc. A variety of parallel sparse linear solvers have adopted LU decomposition and have been running on massively parallel supercomputers. For example, Cray XE6 [3], which is a type of distributed-memory machines, has implemented SuperLU_DIST [4] solver for LU decomposition.

[1]For an $n \times n$ matrix A, A is convergent if:

$$\forall i, j \in [1, n], i, j \in \mathbb{N} \Rightarrow \lim_{k\to\infty} \left(A^k\right)_{ij} = 0$$

With the continuous development of IC industry, the size of FPGAs has grown to the extent that intense and heavy floating-point operations can be now accommodated. After a decade of research, it has been proved that the accelerating algorithms on FPGAs is a promising research avenue. Modern FPGAs have made it possible to fit a large computation kernel and establish parallel computation machines. In this paper, a sparse linear solver is build based on a CPU-FPGA architecture. While the pre-processing is made on CPU, the FPGA performs the numeric factorisation and solving of the matrices.

1.2 Project Aim

1.3 Contribution

- The contribution of the team is to design a sparse linear solver based on a CPU-FPGA architecture.
-

Our responses to the Xilinx Open Hardware 2022 are given below:

- **Technical Complexity:**
- **Implementation:**
- **Marketability/Innovation:**
- **Re-usability:**

2 Foundation of LU Decomposition

Suppose $A \in \mathbb{R}^{n\times n}$. An LU decomposition of A refers to the factorisation of A into a lower triangular matrix L and an upper triangular matrix U, with proper row and/or column permutations. Here, permutation is to change the order of the row or column of a matrix according to a row permutation matrix P and a column permutation matrix Q, and each row and column of the permutation matrix contains a single 1 with 0s everywhere else.

A sparse matrix refers to a matrix that has few nonzeros in it. Although the quantification of "few" is not defined, typically $\mathcal{O}(n)$ elements are in a sparse matrix, where n is the order of the matrix. Sparse matrices are omnipresent in scientific computation when modelling systems with numbers of elements with restricted couplings.

2.1 BTF

2.2 Fill-reducing Ordering

2.3 Left-looking Algorithm

2.4 Forward and Backward Substitution

One of the largest advantages of LU decomposition is that once the decomposition result L and U is found, only the triangular systems are needed to solve to get the unknown vector x, by applying forward and backward substitution.

For a linear system and its LU decomposition

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

$$A = LU \tag{2}$$

$$LU\boldsymbol{x} = \boldsymbol{b} \tag{3}$$

we can first substitute $\boldsymbol{Ux}$ with a vector $\boldsymbol{y}$

$$\boldsymbol{Ly} = \boldsymbol{b} \tag{4}$$

Here, L is the lower triangular matrix. Hence the forward substitution can be applied to solve this system for $\boldsymbol{y}$. Once $\boldsymbol{y}$ is obtained, the backward substitution can be implemented to solve for $\boldsymbol{x}$.

$$\boldsymbol{Ux} = \boldsymbol{y} \tag{5}$$

Then, if the right-hand side $\boldsymbol{b}$ changes, only one lower triangular system and one upper triangular system are required to compute to obtain the new solution. Unlike Gaussian elimination, there is no need to compute the main LU decomposition again.

To further illustrate, for the lower triangular system given in Equation 4, we can write them as

$$\begin{cases} l_{11}y_1 & & & = b_1 \\ l_{21}y_1 + l_{22}y_2 & & & = b_2 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1}y_1 + l_{n2}y_2 + \cdots + l_{nn}y_n & & & = b_n \end{cases} \tag{6}$$

To solve this lower triangular system, Equation 7 can be used, which solves $\boldsymbol{y}$ from y_1 to y_n. Here, L is the unit lower triangular, i.e., $l_{ii} = 1$, so it requires no division. Similarly, for backward substitution, Equation 8 is used, which solves in the reverse order of forward substitution from y_n to y_1. However, backward substitution requires the division operation on the diagonal element.

$$\begin{cases} y_1 = b_1 \\ y_i = b_i - \displaystyle\sum_{j=1}^{i-1} l_{ij}y_j \end{cases} \tag{7}$$

$$\begin{cases} x_n = \dfrac{y_n}{u_{nn}} \\ x_i = \dfrac{y_i - \sum_{j=i+1}^{n} u_{ij}x_j}{u_{ii}} \end{cases} \tag{8}$$

For the time complexity, both forward and backward substitution require n divisions, $\frac{n^2-n}{2}$ summations and $\frac{n^2-n}{2}$ multiplications. Therefore, the total number of operations is $2n^2 - n$ and the time complexity is $\mathcal{O}(n^2)$.

3 Implementation

The factorisation phase of the LU decomposition is implemented on the CPU and the solving phase is implemented in the FPGA. In this work,

3.1 HLS Configuration

- *# pragma HLS ARRAY_PARTITION variable = Xwork type = block factor = 32 dim = 2*
  An array that is implemented in the BRAM may subject to the limit port access, which prevent the parallel access to the same array. A dual-port RAM can also allow two simultaneous access in one

clock cycle.Therefore, arrays which are implemented as memory or memory ports can frequently create performance bottlenecks.

With array partition, it is easier to implement the vectorisation optimisation, which typically requires parallel access to the same array.

- *# pragma HLS UNROLL factor = 32*
  Loop unrolling can create multiple separate operations instead of a single set of operations. The *UNROLL* pragma changes loops by duplicating the loop body in the RTL design, allowing part of or entire loop iterations to run in parallel. By default, loops in C/C++ functions are kept rolled. Synthesis builds the logic for one iteration of the loop when loops are rolled, and the RTL design executes this logic in order for each iteration of the loop. The loop induction variable specifies the number of iterations for which the loop should be run. Logic inside the loop body, such as break conditions or changes to a loop exit variable, can also affect the number of iterations. The *UNROLL* pragma can be used to increase data access and throughput.

4 Results

5 Conclusion and Future Work

5.1 Conclusion

In this project, a parallel CPU+FPGA based architecture is proposed for the acceleration of LU decomposition for SPICE engine. While the preprocessing analysis, i.e., the permutation is implemented on CPU, most of the factorisation step and solving phase are implemented on FPGAs.

For preprocessing, BTF method is implemented to permute the matrix into several block triangular matrices in the diagonal, which help reduce the total factorisation steps. AMD method is deployed to help reduce the fill-ins.

On FPGA side,

5.2 Future Work

References

[1] D. A. Spielman and S. Teng, “Spectral Sparsification of Graphs,” *CoRR*, vol. abs/0808.4134, 2008, arXiv: 0808.4134, [Online]. Available: http://arxiv.org/abs/0808.4134.

[2] I. Koutis, G. L. Miller and R. Peng, “A Fast Solver for a Class of Linear Systems,” *Commun. ACM*, vol. 55, no. 10, pp. 99–107, Oct. 2012, ISSN: 0001-0782, DOI: 10.1145/2347736.2347759.

[3] J. Kwack, G. Bauer and S. Koric, “Performance test of parallel linear equation solvers on Blue Waters–Cray XE6/XK7 system,” English (US), 2016.

[4] X. S. Li and J. W. Demmel, “SuperLU_DIST: A Scalable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems,” *ACM Trans. Math. Softw.*, vol. 29, no. 2, pp. 110–140, Jun. 2003, ISSN: 0098-3500, DOI: 10.1145/779359.779361.