An FPGA based floating point Gauss-Seidel iterative solver

Ramakant Joshi, Aman Raghuvanshi, Yatin Gilhotra*,
Shivani Sharma, Suyash Sharma, Preyesh Dalmia, Neeta Pandeyˆ
Dept. of Electronics and Communication Engineering
Delhi Technological University
New Delhi, India
Email: yatingilhotra_2k14@dtu.ac.in*,
neetapandey@dce.ac.inˆ

***Abstract*—In this paper, an FPGA-based single precision floating point hybrid iterative architecture for solving a linear system of equations is proposed. The novel architecture implements Gauss-Seidel method using a Jacobi method based building block. The design takes advantage of the fast convergence of Gauss-Seidel Method conflated with parallel, pipelined architecture of Jacobi Iterative solver resulting in a much efficient architecture with acceptable hardware augmentation. The whole design has been implemented in Verilog HDL, having Virtex 7 XCV2000T as targeted device. Other design optimizations include using modified high-speed radix 4 multiplier and optimized high-speed 2's complementer. The efficacy of the design is tested and implemented in solving nonsingular, exactly determined and strictly diagonally dominant coefficient matrix based dense and sparse linear system of equations with different number of variables. The design results in reduced number of iterations and equivalent speedups are presented, which are calculated factoring in, the increased delay effects.**

***Index Terms*—Jacobi method; Gauss-Seidel method; diagonally dominant matrix; direct methods; iterative methods.**

I. Introduction

Partial Differential Equations (PDEs) possess radical importance in defining and modeling physical phenomenon. Complex problems from a multitude of domains such as electrical and mechanical engineering, materials science, fluid dynamics, quantum mechanics, etc, all converge down to solving different types of PDEs. They can be discretized and linearized into an approximate system of equations with a finite number of unknowns that can be computed and processed. This system of equation can be represented by the vector equation

$$Ax = B \tag{1}$$

where A and B are known coefficient matrices and x is an unknown vector.

Other than PDEs, many scientific models and problems such as network modeling, assignment problems[11], etc. can be disintegrated into a problem of solving linear system of equations. Though, mathematically trivial, computing this kind of system can prove to be severely complex. Hence, solving this crucial form of system has been a topic of research for the whole past century. The classical methods which can solve this linear system of equations can be broadly classified into two categories, namely direct and iterative methods. Direct methods such as Gaussian Elimination method are the traditional methods for solving sparse linear systems but these methods compromise with the accuracy of the solutions in the form of round-off errors[1], which propagate through a number of arithmetic operations to give inaccurate results and they use prohibitively large storage[12]. Hence, in order to solve systems with higher dimensions, iterative methods were introduced, which tend to be computationally less expensive because of their repeating nature.

Jacobi and Gauss-Seidel iterative methods are two of the most prominent methods used in solving linear systems of equations of the form $Ax = B$ and are used in many specific implementations[5],[11],[13],[15],[16]. Now, one of the major concerns with using iterative methods is their speed of finding accurate solutions which can be measured by their convergence rate. Convergence rate is a highly sensitive issue because these solvers are used in high-speed scientific computing and massively parallel computing architectures. Other than being stand-alone solvers, these classic iterative methods, these days, are also used as preconditioners, conflated with more sophisticated methods such as Krylov Subspace accelerator techniques[2]-[5] or Multigrid methods to create a multilevel solving system. Using multilevel solving systems may drastically increase the speed of finding the solution.

Preconditioning aims at bringing an implicit or explicit modification to the ill-conditioned, original linear system for sophisticated solvers creating a multilevel procedure. Preconditioners can be considered as subsidiary approximate solvers, whose outputs, when worked upon by outer iterative solvers, give enhanced accuracy of the solutions and facilitate the fast convergence.

Talking about these two classical iterative methods, Gauss-Seidel method usually converges much faster than the Jacobi method[14]. The reason lies in the fact that the Gauss-Seidel method uses the newly computed approximation of a solution component as soon as it is available. However, as a consequence, the Gauss-Seidel method is inherently sequential, and does not possess natural parallelism[13]. The architectural implementation of the Gauss-Seidel method designed to exhibit parallelism would take prohibitively large area and high computation cost. In this paper we explore a novel hardware implementation of the Gauss Seidel method, which uses Jacobi architecture[7] with some hardware augmentation and propose a new hybrid architecture, which conflates the advantages of the two, i.e. the parallelism of the Jacobi and the convergence rate of Gauss Seidel method.

The rest of the paper is organized as follows. The fundamentals and functioning of the two iterative methods are discussed in Section II. The proposed architecture is a consequence of a mathematical manipulation discussed under Section III. In this paper, the Gauss Seidel method is implemented for equation set with a maximum size of 5×5. The hardware design and various blocks are discussed in section IV. The critical path delay and the speed up of the proposed architecture with respect to Jacobi is discussed in Section V and the overall results are shown. Finally, the paper is concluded in Section VI and references are presented.

II. Method Analysis

The fundamental idea behind iterative methods is to produce a sequence of approximate solutions $x^{(\delta)}$ that refine themselves with each iteration to finally converge to the actual value.

$$x = \lim_{\delta \to \infty} x^{(\delta)} \tag{2}$$

This equation represents the concept of convergence, where x is the exact solution and $x^{(\delta)}$ is the approximate solution after δ number of iterations. Theoretically, it takes infinite iterations to reach the exact solution, but practically, the solution is obtained when it reaches a certain region of tolerance. The iterative process is terminated such that $\left\|x^{(\delta)} - x\right\| \leq \epsilon$, at the minimum value of δ , where $\|.\|$ is the vector norm and ϵ is the tolerance. Jacobi and Gauss-Seidel Iterative methods both have their own advantages. In this section, we discuss these two methods and study their convergence analysis. The analysis and proposed design is specifically presented for nonsingular, exactly determined and strictly diagonally dominant coefficient matrices.

A. Jacobi Method

Let us represent the equation (1) in the matrix form as,

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \tag{3}$$

Where A, the coefficient matrix, with coefficients represented as a_{ij}, B, the constant matrix with coefficients represented as b_i, x be the variable vector with unknowns represented as x_i. If $B = 0$, the system is called homogeneous system, else it is called heterogeneous system. Jacobi method considers the following assumptions for solving the system.

1) The coefficient matrix is a nonsingular matrix, i.e. $|A| \neq 0$.
2) The system has a unique solution.
3) The coefficient matrix has no zeros on its main diagonal i.e. $a_{ii} \neq 0$.

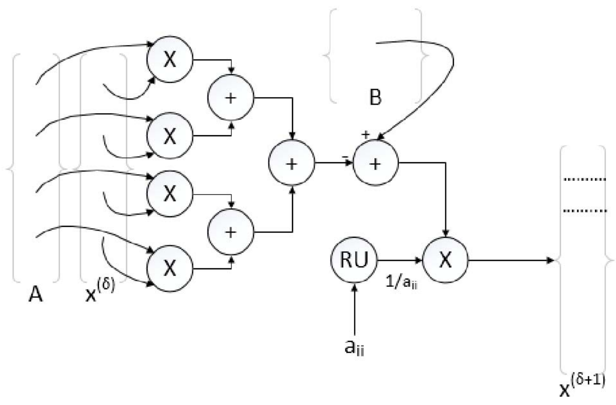


Fig. 1: Jacobi Iterative Architecture

Let A matrix presented in equation (3) be decomposed according to the equation,

$$A = L + U + D \tag{4}$$

where L is the strictly lower triangular matrix, U is the strictly upper triangular matrix and D is the diagonal matrix. Solving for $x^{(\delta+1)}$ using equations (3) and (4) yield,

$$x^{(\delta+1)} \Leftarrow D^{-1}[B - (L + U)x^{(\delta)}] \tag{5}$$

The same iteration in the point form can be expressed as presented in [7] and the architecture is shown in Fig. 1.

$$x_i^{(\delta+1)} \Leftarrow \frac{1}{a_{ii}} \left[b_i - \sum_{j=1, j \neq i}^{j=n} a_{ij} x_j^{(\delta)} \right] \forall \ i = 1, 2 \dots n. \tag{6}$$

Where, $x_i^{(\delta+1)}$ is the iteratively approximated value of x_i, after $\delta + 1$ iterations. In Jacobi method, the values $x_i^{(\delta)}$ remain unchanged until the $(\delta+1)th$ iteration occurs. In equation (5), let $Y = D^{-1}(L + U)$. The necessary and sufficient condition for the Jacobi iterative method to converge is $\rho(Y) < 1$, where $\rho(Y)$ represents the spectral radius of Y as discussed in [6].

B. Gauss Seidel

In Jacobi method, the $x_i^{(\delta)}$ values remain unchanged until the $(\delta + 1)th$ iteration occurs and the entire next iteration values get calculated, but in Gauss-Seidel method, the values of $x_i^{(\delta+1)}$ get updated and are used in the calculation of next variables as soon as they are calculated. The assumptions taken into account before are same as Jacobi method. The solution can be presented in vector form as,

$$(L + D)x^{(\delta+1)} \Leftarrow [B - Ux^{(\delta)}] \tag{7}$$

$$\Rightarrow (x)^{(\delta+1)} \Leftarrow (L+D)^{-1}[B - x^{(\delta)}] \tag{8}$$

The necessary and sufficient condition for the Gauss-Seidel method to be convergent is $\rho((D+L)^{-1}U) < 1$ [6]. This condition ensures that the coefficient matrix is convergent with the Gauss-Seidel method. The Gauss-Seidel iteration can be represented in the point form as:

$$x_i^{(\delta+1)} \Leftarrow \frac{1}{a_{ii}}\left[b_i - \sum_{j=i+1}^{j=n} a_{ij}x_j^{(\delta)} - \sum_{j=1}^{j=i-1} a_{ij}x_j^{(\delta+1)}\right] \quad \forall\, i = 1,2\ldots n. \tag{9}$$

III. Proposed Method

The proposed architecture is the consequence of a mathematical manipulation on equation (9). Here, we demonstrate the manipulation with an example on an equation set of 5 variables and then generalize the result. By specifying the equation (9) for x_2, we get,

$$x_2^{(\delta+1)} = \frac{1}{a_{22}}\big[b_2 - a_{21}x_1^{(\delta+1)} - a_{23}x_3^{(\delta)} - a_{24}x_4^{(\delta)} - a_{25}x_5^{(\delta)}\big] \tag{10}$$

A scaled term $a_{21}x_1^{(\delta)}$ is added and subtracted from the equation, shown in parenthesis, to give,

$$x_2^{(\delta+1)} = \frac{1}{a_{22}}\big[b_2 - a_{21}x_1^{(\delta+1)} - a_{23}x_3^{(\delta)} - a_{24}x_4^{(\delta)} - a_{25}x_5^{(\delta)} + (a_{21}x_1^{(\delta)} - a_{21}x_1^{(\delta)})\big] \tag{11}$$

which can be rearranged as:

$$x_2^{(\delta+1)} = \frac{1}{a_{22}}\big[(b_2 - a_{21}x_1^{(\delta)} - a_{23}x_3^{(\delta)} - a_{24}x_4^{(\delta)} - a_{25}x_5^{(\delta)})(\mathbf{I}) + (a_{21}x_1^{(\delta)} - a_{21}x_1^{(\delta+1)})(\mathbf{II})\big] \tag{12}$$

The first part **(I)**, of this equation is a manifestation of (6), which can be implemented by the Jacobi architecture as presented in Fig. 1. The second part **(II)** can be implemented using synchronized, combinational hardware additional to the Jacobi architecture to give the proposed architecture as shown in Fig. 2. The proposed design in Fig. 2, like the equations above, is an illustrious design for solving a linear system of equations with 5 variables. The general, point form of the proposed approach can be represented as:

$$x_i^{(\delta+1)} = \frac{1}{a_{ii}}\left(\left[b_i - \sum_{j=1, j\neq i}^{j=n} a_{ij}x_j^{(\delta)}\right](\mathbf{I}) - \left[\sum_{j=1}^{j=i-1} a_{ij}(x_j^{(\delta)} - x_j^{(\delta+1)})\right](\mathbf{II})\right) \forall\, i = 1,2\ldots n. \tag{13}$$

Adding the parallel, multiplier-adder tree architecture, the critical path delay increases a bit; by the delay of an adder in comparison to the Jacobi architecture. But the improved convergence rates of the Gauss-Seidel method massively overshadow this increased delay when implemented for systems of higher dimensions resulting in overall speed efficient circuit with acceptable hardware augmentation, discussed further in the Section V.

IV. Hardware Design

The designed hardware is a 32-bit floating point architecture implemented in Verilog using Xilinx ISE design suite, having Virtex 7 XCV2000T as the target device. The proposed architecture requires a synchronized augmented hardware, which uses a multiplexer as a selector circuit to select the coefficients required in the current cycle for computations. The coefficients are selected by the 5X1 multiplexer according to Table I and fed into the system.

TABLE I: Cycle number Vs Matrix Coefficient

Cycle Number	Coefficients		
	Select Lines (s_1, s_2, s_3)	A	C(c_1, c_2, c_3, c_4)
1	000	$a_{12}a_{13}a_{14}a_{15}$	0000
2	001	$a_{21}a_{23}a_{24}a_{25}$	$a_{21}000$
3	010	$a_{31}a_{32}a_{34}a_{35}$	$a_{31}a_{32}00$
4	011	$a_{41}a_{42}a_{43}a_{45}$	$a_{41}a_{42}a_{43}0$
5	100	$a_{51}a_{52}a_{53}a_{54}$	$a_{51}a_{52}a_{53}a_{54}$

A. Functioning of proposed architecture

Coefficients of the A matrix are stored in a lookup table to access in each cycle. The unknowns are initialized as a zero vector ($x^{(0)} = [0, 0, 0....0]$). For instance, in the first cycle, the value of $x_1^{(\delta+1)}$ is computed using the initialized vector. The coefficients $a_{12} - a_{15}$ are read from the lookup table and given as input to the first stage of the multiplier-adder tree to the left of the highlighted adder along with the unknowns evaluated in the previous iteration. Simultaneously, the element a_{11} is fed to the reciprocal unit. The elements in C are taken to be zero for the first cycle. These elements vary in each cycle as described in Table I and appropriate values are selected by the synchronized multiplexer. The newly computed value of $x_1^{(\delta+1)}$ is stored in a register. Next cycle involves the computation of $x_2^{(\delta+1)}$. Coefficients $a_{21} - a_{25}$ are read from the lookup table and fed to the multiplier-adder tree and a_{22} to the reciprocal unit. The coefficient c_1 of the C matrix in the augmented multiplier adder tree is selected to be a_{21} using the

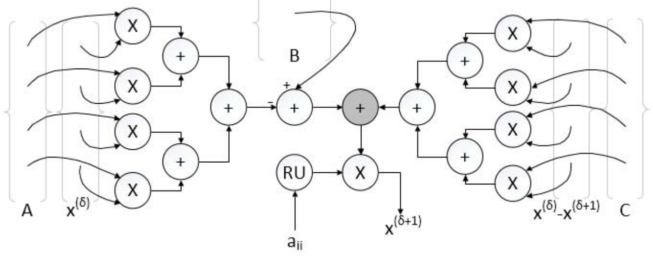


Fig. 2: Proposed Design

multiplexer while the other coefficients $c_2 - c_4$, are forced to 0. The stored value of $x_1^{(\delta)}$ along with the newly computed $x_1^{(\delta+1)}$ is also fed to this block while the values not required in the current cycle are forced to zero. This gives the newly computed value $x_2^{(\delta+1)}$ which is stored for use in the next cycle. These cycles are repeated to obtain the values of $x_3^{(\delta+1)}$, $x_4^{(\delta+1)}$ and $x_5^{(\delta+1)}$ with the required coefficients of A and C matrix as described in the proposed equations and given in Table I. The other newly computed values, $x_1^{(\delta+1)} - x_4^{(\delta+1)}$, are used in successive cycles as per (II) of equation (13). Computing all 5 unknowns once completes one iteration. These values are stored for use in the next iteration and this cycle goes on till the unknowns converge to an approximate solution with desired tolerance. The architecture can be broken down into building blocks which are designed and optimized for best performance, as discussed below.

B. Multiplier

Instead of a conventional multiplier, a 32-bit floating point multiplier which uses a 24-bit Radix-4 Booth Encoded multiplier to multiply mantissas of the numbers is designed. The study in [8] shows that Radix-4 Booth Encoded multiplier is the most speed efficient and area efficient among other higher radix Booth Encoded multipliers. The partial products required for computation are significantly reduced compared to the traditional multipliers. These partial products are added using a Dadda tree structure and the final summation is carried out using a Koggestone adder. The Dadda tree structure is used over Wallace because it is relatively speed efficient and possesses less complexity as shown in [9]. The Koggestone adder was chosen among Carry-Save adder, Carry-Select adder etc. after simulation and testing for speed to get as low combinational delay as possible. The proposed design is presented in the Fig. 3.

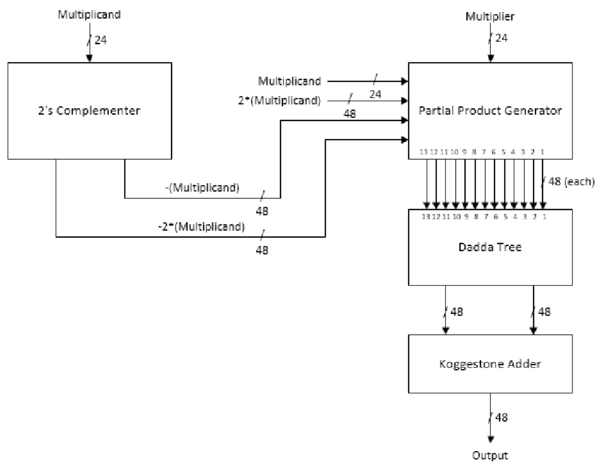


Fig. 3: Proposed Radix 4 Multiplier

C. 2's Complementer

An improved 2's complementer unit is designed for generating the partial products of the radix 4 multiplier. Unlike the conventional method of complement and add we propose a parallel complementer unit whose logic delay remains the same for an arbitrary input size. The proposed design as shown in Fig. 4 works as following:

1) Starts from the right of the number and forms a quadruple of bits. Repeats this process and forms non overlapping quadruples. Adds additional 0's towards the end if no more bits are left.
2) Logical OR these sets of 4 bits to generate signals $o_0^{'}, o_1^{'}, o_2^{'}......o_n^{'}$ where $o_0^{'}$, is the signal from the right-most 4 bits.
3) Uses the above signals to generate $o_0 = o_0^{'}, o_1 = (o_0^{'} + o_2^{'}), o_2 = (o_0^{'} + o_1^{'} + o_2^{'})$ and so on.
4) Passes the 4 bits: unchanged, complemented or in 2's complement form based on the combination $o_i o_{i-1}$, take $o_{-1} = 0$, as given in Table II.
5) The output thus obtained is the 2's complement of the original number.

TABLE II: Multiplexer Output

o_i, o_{i-1}	4 Bit Output
00	Pass Bits Unchanged
01	0000∗(this case never occurs)
10	2's Complement Form
11	Complemented Form

D. Reciprocal Unit(RU)

Instead of using a conventional full divider architecture, the reciprocal unit is based on a compact iterative architecture using Newton-Raphson method as described in [10]. The architecture proposed in [10] is designed to compute the reciprocal $1/x$ for $x \in (0.5, 1]$, but our design is optimized to compute the reciprocal for $x \in (0, \infty]$. Hence, reciprocal of any diagonal coefficient $a_{ii} \in R$(Set of Real numbers) can be computed by normalizing the coefficient by 2^p to first bring the coefficient into the working range of the reciprocal unit i.e. $(0.5, 1]$, which actually corresponds to p number of arithmetic left or right shifts depending on whether the number lies in the

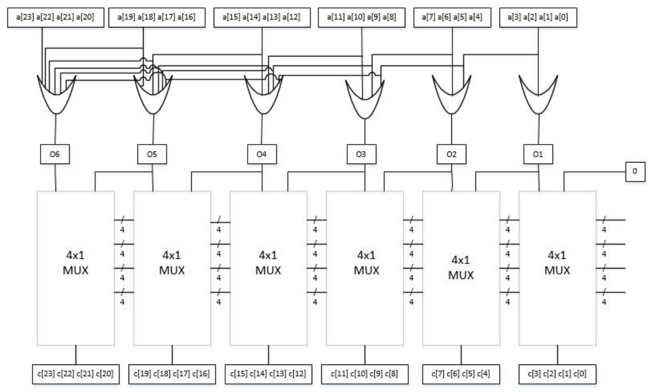


Fig. 4: Proposed 2's Complementer

range $(0, 0.5]$ or $(1, \infty)$ respectively. After the computations, p number of arithmetic shifts in the complementary direction yield the true result. The circuit approximates the reciprocal of a number in just 3 iterations and the architecture effectively uses same hardware to compute the initializing value and the reciprocal resulting in compact, fast implementation of the reciprocal unit.

V. Results

The proposed architecture has been synthesized, using Xilinx design tools. The combinational delay of each building block, the maximum operating frequency of the proposed hardware, the area and the percentage resources utilized of the target FPGA are presented in Table III.

TABLE III: Hardware Summary

Module	Combinational Delay(ns) (Δ)	f_{max} (MHz)	Area (Sliced Used)	%age Resources Utilized
adder_32_bit	20.64	-	444	0.03
adder_64_bit	33.56	-	1036	0.08
2's complementer	3.69	-	28	0
multiplier_64_bit	17.65		1205	0.09
reciprocal_64_bit	-	25.76	7344	0.06
Jacobi Block	127.15	25.98	18126	1.48
Proposed Design(GS)	162.17	25.18	29761	2.43

It can be manifested from the Table III that the proposed method has larger critical path in comparison to the Jacobi architecture. The increased value in delay corresponds to the 64-bit adder used to combine the results of the Jacobi architecture and the extended hardware as discussed in Section III. However, the fast convergence rate of the proposed method, which is by virtue of the Gauss-Seidel method, massively overshadows the increased critical path delay. This can be manifested from Table IV which presents the comparison of Jacobi method and the proposed architecture. The architectures are tested for equation sets, chosen randomly, of various dimensions of two types: Dense and Sparse coefficient matrix(A). In order to test the efficacy of the system, 10 trials were performed for each type of equation presented in Table IV and their averaged results are shown.

Here, speed up is a figure of merit, which demonstrates the relative speed of the proposed architecture to the Jacobi architecture. The Speed Up is introduced as:

$$S = \frac{N_J}{N_{GS}} * \frac{\Delta_J}{\Delta_{GS}} \tag{14}$$

Where N_J is the number of iterations required to converge by Jacobi method, N_{GS} is the number of iterations required to converge the Proposed Gauss-Seidel architecture, and Δ_J and Δ_{GS} are respectively the critical path delays of Jacobi architecture and the Gauss-Seidel architecture respectively, which are in this case 127.15ns and 162.17ns respectively.

TABLE IV: Gauss Seidel Vs Jacobi Iterative Solver

Type of matrix	N_J (Jacobi)	N_{GS} (Gauss-Seidel)	Speed Up(S)
Dense 5×5	19	10	1.48
Sparse 5×5	9	5	1.41
Dense 4×4	14	8	1.36
Sparse 4×4	8	5	1.24
Dense 3×3	10	6	1.31
Sparse 3×3	5	3	1.30
Dense 2×2	7	5	1.09
Sparse 2×2	2	2	0.78

Here we emphasize on the system of dimension 5×5, as it is the highest dimensional system tested in this paper. The mean error percentage of the computed, converged solutions and the theoretical solutions is presented in Table V, averaged over 10 different test equation sets, chosen randomly. It can be observed from Table IV that the speed up value increases as we go from a lower to a higher dimension linear system of equations. Following this trend, it is safe to conclude that system would perform even better at solving larger dimensional systems.

TABLE V: Mean Error Percentage

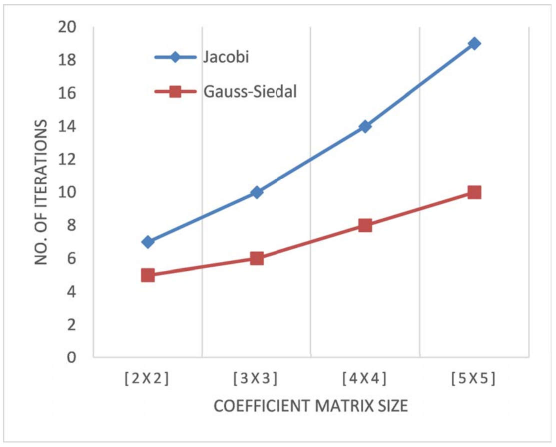
Type of Matrix	Error Percentage
Dense 5×5	0.00002097782
Sparse 5×5	0.00003614089

VI. Conclusion

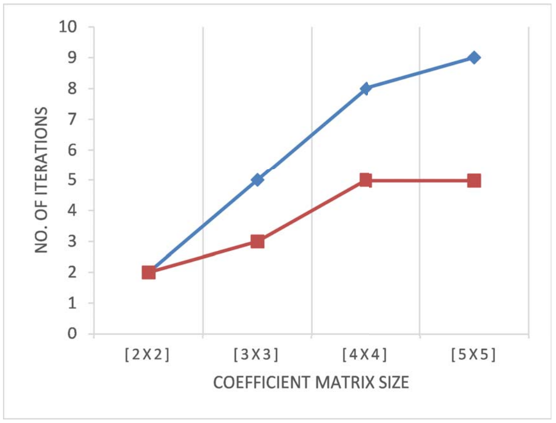
This paper introduces a novel implementation of an existing iterative method used to solve linear system of equations, viz. the Gauss-Seidel method. The proposed design uses the basic Jacobi architecture and optimized adders and multipliers to implement the Gauss-Seidel method which offers better convergence rates than the conventional Jacobi method to solve the linear system of equations especially with large dimensions. The proposed method can be employed to solve higher dimension system of equations and is expected to deliver larger speedup(S) as the system gets larger(following the trend in Fig. 5). This speed efficiency comes conflated with an acceptable increase in hardware with respect to the Jacobi architecture but reduced hardware requirement relative to the Gauss-Seidel implementation designed to exhibit parallelism. The data to compare our design with other hardware implementations of Gauss-Seidel is not readily available in literature to the best of our knowledge. Hence, our future aim is to test and compare this architecture with other existing Gauss-Seidel implementations used in specific applications [5],[11],[13],[15],[16] for speed, area, hardware cost, etc.

Acknowledgement

The authors would like to thank anonymous reviewers for their constructive comments and scrutiny.



(a) Dense Matrices.



(b) Sparse Matrices

Fig. 5: No. of iterations used by the Jacobi and Gauss-Seidel(Proposed) method in solving dense and sparse matrices

References

[1] Larson,Ron. Numerical methods in Elementary Linear Algebra, 6th edition, Brooks/Cole,Cengage Learning, 2009.

[2] Reusken, Arnold. ”Convergence analysis of the Gauss-Seidel preconditioner for discretized one dimensional Euler equations.” SIAM journal on numerical analysis 41.4 (2003): 1388-1405.

[3] Liu, Qingbing, Guoliang Chen, and Jing Cai. ”Convergence analysis of the preconditioned GaussSeidel method for H-matrices.” Computers & Mathematics with Applications 56.8 (2008): 2048-2053.

[4] Adams, Mark, et al. ”Parallel multigrid smoothing: polynomial versus GaussSeidel.” Journal of Computational Physics 188.2 (2003): 593-610.

[5] Dziekonski, Adam, Adam Lamecki, and Michal Mrozowski. ”Jacobi and Gauss-Seidel preconditioned complex conjugate gradient method with GPU acceleration for finite element method.” Microwave Conference (EuMC), 2010 European. IEEE, 2010.

[6] Liu, Hongxia, and Tianxiang Feng. ”Study on the Convergence of Solving Linear Equations by Gauss-Seidel and Jacobi Method.” Computational Intelligence and Security (CIS), 2015 11th International Conference on. IEEE, 2015.

[7] Morris, Gerald R., and Viktor K. Prasanna. ”An FPGA-based floating-point Jacobi iterative solver.” Parallel Architectures, Algorithms and Networks, 2005. ISPAN 2005. Proceedings. 8th International Symposium on. IEEE, 2005.

[8] Swee, Kelly Liew Suet, and Lo Hai Hiung. ”Performance comparison review of radix-based multiplier designs.” Intelligent and Advanced Systems (ICIAS), 2012 4th International Conference on. Vol. 2. IEEE, 2012.

[9] Townsend, Whitney J., Earl E. Swartzlander, and Jacob A. Abraham. ”A comparison of Dadda and Wallace multiplier delays.” Proc. SPIE, Advanced Signal Processing Algorithms, Architectures, and Implementations 13 (2003): 552-560.

[10] Habegger, Andreas, et al. ”An efficient hardware implementation for a reciprocal unit.” Electronic Design, Test and Application, 2010. DELTA'10. Fifth IEEE International Symposium on. IEEE, 2010.

[11] Zhu, Pengfei, et al. ”An FPGA-based acceleration platform for auction algorithm.” Circuits and Systems (ISCAS), 2012 IEEE International Symposium on. IEEE, 2012.

[12] Saad, Yousef, and Henk A. Van Der Vorst. ”Iterative solution of linear systems in the 20th century.” Journal of Computational and Applied Mathematics 123.1 (2000): 1-33.

[13] Bylina, Jaroslaw, and Beata Bylina. ”Merging Jacobi and Gauss-Seidel methods for solving Markov chains on computer clusters.” Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on. IEEE, 2008.

[14] Bahi, Jacques Mohcine, Sylvain Contassot-Vivier, and Raphael Couturier. Parallel iterative algorithms: from sequential to grid computing. CRC Press, 2007.

[15] Chassin, David P., et al. ”Gauss-Seidel accelerated: implementing flow solvers on field programmable gate arrays.” Power Engineering Society General Meeting, 2006. IEEE. IEEE, 2006.

[16] Wu, Zhizhen, et al. ”Efficient architecture for soft-output massive MIMO detection with Gauss-Seidel method.” Circuits and Systems (ISCAS), 2016 IEEE International Symposium on. IEEE, 2016.