# Accelerating Electromagnetic Simulations: A Hardware Emulation Approach

M. Tarek Ibn Ziad*, Yousra Alkabani*, M. Watheq El-Kharashi*, Khaled Salah†, and Mohamed AbdelSalam†
{mohamed.tarek,yousra.alkabani,watheq.elkharashi}@eng.asu.edu.eg, {khaled_mohamed,mohamed_abdelsalam}@mentor.com
*Computer and Systems Engineering Department, Ain Shams University, Cairo 11517, Egypt
†Mentor Graphics Egypt, Cairo 11361, Egypt

***Abstract*—Electromagnetic (EM) simulation is an important tool for modeling and studying high frequency systems in modern industry. However, the solver part in EM simulations represents a serious bottleneck because its execution time rapidly increases as number of equations-to-solve increases. Although several existing research has proposed parallel hardware solvers, there still exists a room to improve the speed and scalability of these solvers. In this paper, we present a scalable architecture that can efficiently accelerate the solver core of an EM simulator. The architecture is implemented on a physical hardware emulation platform and is compared to the state-of-the-art solvers. Experimental results show that the proposed solver is capable of 522x speed-up over the same pure software implementation on Matlab, 184x speed-up over the best iterative software solver from the ALGLIB C++ library, and 5x speed-up over another emulation-based hardware implementation from the literature, solving 2 million equations.**

**Keywords- electromagnetic simulations, finite element method, hardware emulation, Jacobi over-relaxation method, sparse linear systems**

## I. Introduction

Electromagnetic (EM) simulation is necessary for understanding high-speed digital electronics, optical computer networking, and achieving electromagnetic compatibility (EMC) of products. It is known that any product has some EMC requirements, whether for the product to function properly in the immediate environment or to ensure compliance with government standards [1]. Using EM simulators is generally safer and more realistic than conducting experiments with a prototype of the final product. However, EM simulations often consume large amount of time on traditional CPUs, as they use numerical techniques, such as the finite element method (FEM) [2], to solve Maxwell's equations. The most time-consuming part in the simulation process is the solver part, which is responsible for solving large sparse systems of linear equations that arise from the discretization step of the numerical methods [3].

The availability of hardware solvers based on Field Programmable Gate Arrays (FPGAs) has made it possible to obtain accurate final solutions of these large systems within a reasonable time. However, using emulation technology is more appealing as it overcomes the FPGA memory and area constraints. The largest contemporary FPGA, Virtex UltraScale, provides up to 4,433 K Logic Cells and up to 132 MB Block RAM, whereas the emulation platforms can offer up to 2 Billion gates with 256 GB of memory. A hardware emulation platform could be seen as a multi-FPGA system that allows large SoC designs to be modeled in hardware. Although the emulators are mainly optimized for debugging and verification [4], we managed to extend their usage to be efficient solutions for accelerating EM solvers by leveraging their memory and logic resources, in spite of their low operating frequency.

In this paper, we extend the work in [5], [6], where a hardware architecture based on Jacobi iterative method was introduced to accelerate the FEM solver of an EM simulator. The key contributions of this paper include: **(1)** proposing an efficient architecture for solving the sparse linear systems (SLS) arising in FEM formulations based on the Jacobi over-relaxation (JOR) method, **(2)** optimizing the design by making use of the properties of the FEM coefficients matrix, **(3)** showing the logic utilization of implementing the proposed architecture on a commercial hardware emulation platform [4], and **(4)** comparing the obtained timing results with two software solvers and a hardware one for various test cases.

The remainder of this paper is organized as follows. Section II provides an overview of FEM and JOR method. Section III summarizes some of prior work related to the acceleration of numerical methods for EM simulations and SLS solving. Section IV describes the architecture of our hardware solver. Used environment is described in Section V. The obtained results and comparisons against software and hardware solvers are introduced in Section VI. Finally, Section VII concludes the work.

## II. Background

In this section, we provide a brief overview of the FEM. Then, we describe the details of the JOR method that is used to solve the SLS, resulting from using the FEM.

### A. Finite Element Method (FEM)

The FEM is a numerical technique, which is widely used in engineering to solve boundary-value problems, characterized by a partial differential equation (PDE) and a set of boundary conditions [2]. The basic procedures of using FEM are: **(1)** discretizing the computational domain into finite elements, **(2)** rewriting the PDE in a weak formulation, **(3)** choosing proper finite element spaces and forming the finite element scheme from the weak formulation, **(4)** calculating those element matrices on each element and assembling the element matrices to form a global linear system, **(5)** applying the boundary conditions, solving the SLS, and finally, **(6)** post-processing the numerical solution. In linear FEM analysis, the cost of solving the SLS rapidly overwhelms other computational phases. Thus, it is the targeted part in this work.

### B. Jacobi over-relaxation (JOR) Method

There exists many algorithms for solving SLS. Selecting the best algorithm not only depends on the matrix structure but also depends on different trade-offs, such as memory bottlenecks, computational complexity, and convergence properties. Among other iterative solvers, the JOR method, also called the simultaneous over-relaxation method [7], is selected as it

978-1-5090-0246-7/15/$31.00 ©2015 IEEE

combines the parallel nature of Jacobi iterative method used in [5] with higher convergence rate.

Algorithm 1 describes the JOR method, where $A$ is the coefficients matrix, $x$ is the unknowns vector, $b$ is the right-hand-side (RHS) vector, $t$ is the iteration number, and $\sigma$ is a summation variable to hold the result of multiplying a coefficients matrix row by the vector $x$. The relaxation parameter, $\omega$, is a floating point (FP) number that accelerates the convergence of the method. If $\omega = 1$, we have the regular Jacobi iterative method. For fast convergence, $\omega$ should be between 0 and 1. We followed the guidelines, provided by Young [7], to find the optimal value for $\omega$ by using parametric sweep.

**Algorithm 1** The Jacobi over-relaxation method.

1: **procedure** JOR($A, b, \omega$)
2: $t = 1$
3: **while** solution is not converged **do**
4: **for** $i = 1$ **to** $i = n$ **do**
5: $\sigma = 0$
6: **for** $j = 1$ **to** $j = n$ **do**
7: **if** $j \neq i$ **then**
8: $\sigma = \sigma + a_{i,j}{x_j}^{(t)}$
9: **end if**
10: **end for**
11: ${x_j}^{(t+1)} = (1 - \omega){x_j}^{(t)} + \frac{\omega}{a_{i,i}}(b_i - \sigma)$
12: **end for**
13: $t = t + 1$
14: **end while**
15: **return** $x$
16: **end procedure**

Convergence check is done by comparing the different between the solutions at any two successive iterations to a pre-determined accuracy. Convergence is guaranteed if $A$ is banded, symmetric, and positive definite [8]. Banded matrices are sparse matrices with all of the nonzero elements lie within a specified bandwidth of the main diagonal, whereas symmetric matrices are square matrices equal to their own transpose. These conditions are satisfied in case of the FEM matrices [9].

## III. Related Work

In this section, we survey some of prior work related to the acceleration of numerical methods for EM simulations, hardware designs for solving SLS, and application-specific multi-FPGAs systems.

There exist many work that aimed to accelerate the computations included in the EM simulations. For instance, Taylor *et al.* proposed a scheme, called SPAR, for performing computations on sparse matrices that arose in large FEM applications [3]. However, their architecture was not actually built. They only constructed a register-transfer level simulator and executed the sparse matrix computations on it. Durbano and Ortiz presented hardware-based implementations of the Finite Difference Time Domain method [10]. They utilized a custom accelerator board that supports up to 16 GB of DDR SDRAM, 36 MB of DDR SRAM with Xilinx Virtex-II FPGA. Taking advantage of this high-speed memory hierarchy, their solution achieves 3x speed-up over a 30-node PC cluster.

For large FEM problems, the most computationally intensive step is the solution of SLS. Iterative methods are more suitable for solving SLS because they maintain matrix sparsity, unlike direct methods that can introduce substantial fill-in, which limits their utility and increases storage requirements [3], [11]. Unfortunately, existing approaches generally only have been capable of either solving large matrices with limited improvement over software solvers or achieving high performance for relatively small matrices. Morris and Abed implemented a sparse matrix Jacobi iterative solver that targeted a contemporary high-performance heterogeneous computer (HPHC) [12]. Instead of utilizing hardware description language (HDL) for their FPGA-based kernel designs, they used a high-level language (HLL)-based design. Their solver could handle matrices with sizes up to 8000 equations only.

Recently, there has been a global trend towards using multi-FPGA systems to accommodate larger applications. For example, the Berkeley Emulation Engine 3 was mainly designed for computer architecture research [13]. It is composed of modules with four tightly coupled Virtex-5 FPGAs connected by ring interconnection. Also, there exists the Novo-G [14], which is composed of 24 compute nodes, each of which is a Linux server with an Intel quad-core Xeon processor and boards of ALTERA Stratix IV FPGAs. It was mainly built for various research projects on scalable reconfigurable computing. Ibn Ziad *et al.* implemented a Jacobi iterative solver on a physical hardware emulation platform to accelerate the finite element solver of an EM simulator [5]. They demonstrated the efficiency of their solution via implementing a 2D edge element code for solving Maxwell's equations for metamaterials using FEM. Their design achieved 101x speed-up over the same pure software implementation on MATLAB and 35x over the best iterative software solver from ALGLIB C++ library [15] in case of solving 2 million equations.

## IV. Hardware Implementation

In this section, we introduce the details of our JOR hardware design with all implemented optimizations to enhance the design performance.

The architecture shown in Fig. 1 represents the main four components used in our JOR design; the memory unit, the main ALU, the convergence check unit, and the control unit. In order to obtain maximum performance, the characteristics of the coefficients matrix, $A$, which results from the FEM formulation are investigated. It was found that all diagonal elements have non-zero values and the maximum number of non-zero elements in one row is three [16]. So, $A$ is divided into independent clusters of rows. Each cluster contains a number of $A$ rows that are independent of themselves and each cluster is independent of other clusters. Thus, they can be processed simultaneously without affecting each other.

The memory unit consists of four separate memories; the diagonal memory that contains the $A$'s diagonal elements $a_{i,i}$, the non-diagonal memory that contains the rest of $A$'s elements, the RHS memory that holds the values of vector $b$, and the result memory that is a read/write memory used to save the solution vector and is initially loaded with zeros. Each memory row contains a whole cluster of data, not the elements of a single $A$ row. Thus memory depth in all the previous memories equals $c$, where $c$ represents the number of clusters in matrix $A$. During each iteration, data is loaded

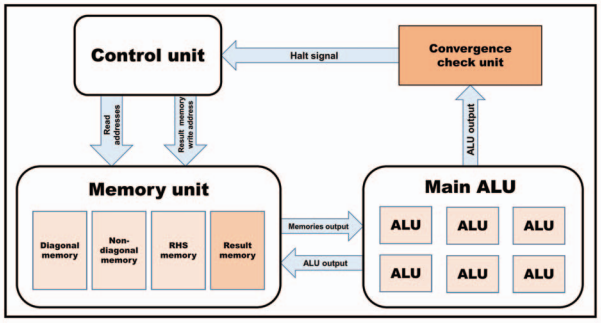


Fig. 1. Block diagram of main components for our proposed JOR design.

from these memories to the main ALU to be operated on. The ALU output is written back to the result memory at the end of iteration. Since the FEM matrices are guaranteed to be symmetric [3], only one half of the elements is stored in the non-diagonal memory.

The main ALU contains a number of independent ALUs, equaling the number of clusters, $c$. Each ALU is responsible for all arithmetic operations performed on data. ALUs utilize single-precision FP components generated using FloPoCo [17] that uses operator properties as an input and outputs synthesizable VHDL code. Based on the JOR steps, shown in Algorithm 1, each ALU starts operating on memories data using multiplication and addition FP modules to generate $\sigma$. Then, the subtraction module is used to calculate $b_i - \sigma$. It is worth mentioning that no division module is used in our architecture due to its high latency, which badly affects the design critical path. We perform a multiplication operation by the inverse of the diagonal element instead of a division operation by the diagonal elements themselves. Finally, the results are multiplied by the relaxation parameter, $\omega$, and added to the multiplication of the previous iteration results with $1-\omega$. These steps are repeated until the results converge.

Convergence here is performed by the result convergence check unit, which decides when the flow should terminate. The termination criterion is determined based on a pre-defined FP value, $\epsilon$, representing the accepted error tolerance. Whenever the current error, between any two consecutive iterations, is less than $\epsilon$, the convergence check unit generates a halt signal indicating end of operations.

The control unit is responsible for synchronizing all memories with each ALU, controlling the convergence check unit, and deciding when each iteration has been finished. It contains counters to indicate which row to read from the different memories and which row to write into the result memory at the end of iteration. It also has a counter that represents the current iteration.

Our JOR design is pipelined. Registers are inserted between ALU operations to allow for fetching new data from memories at every clock cycle. Furthermore, it is a parameterized design that could be reconfigured easily by just flipping a few parameters, such as the FP precision, the number of equations, $n$, the number of clusters, $c$, and the error tolerance, $\epsilon$. These parameters give the design a high flexibility to solve any number of equations and to make full use of the massive capacity of emulator logic resources and memory.

## V. Experimental Setup

Our architecture was modeled using Verilog. Then, it was compiled and run on a physical hardware emulation platform. The used platform consists of 16 advanced verification boards (AVBs). Each AVB has 16 Crystal chips with 4 GB of memory, where each Crystal chip has a 207 K Configurable Programmable Block and can emulate up to 1 M gates. So, this platform provides a total capacity of 256 M of logic gates with 64 GB of system memory and 512 MB of on-chip memory [4]. That clearly solves the area and memory constraints of individual FPGAs.

The main steps in the emulation design process are analyzing Verilog input files and performing syntax checking. Then, generating the structural Verilog netlist of emulator primitives using register transfer level compilation (RTLC). After that, the synthesizer performs partitioning and ASIC netlists for each Crystal chip. Then, placement and routing are done using the chip compiler. Finally, performing final timing analysis and generating timing information for resources access, memories and IO access, emulator events, and clocks are done by the global scheduler [4].

## VI. Experimental Results

This section evaluates the performance of our JOR design in terms of resource utilization and timing performance against the state-of-the-art software and hardware solvers. All the used test cases are generated from the pre-processing part of an EM simulator introduced by Li and Huang in [16]. The selected simulator targets solving Maxwell's equations in metamaterials using Time-Domain FEM. It takes the element dimensions and number of meshes as an input and outputs the numerical electric and magnetic field graphs. Profiling shows that the most time-consuming part in the simulation process is the SLS solver, especially when the number of equations-to-solve increases as the number of meshes increases.

### A. Resource Utilization

Table I shows the operating frequency, resource utilization, and memory capacity of our JOR design with single-precision FP accuracy for different test cases. Here, number of ALUs is the number of equations per cluster in the design. The number of iterations needed until reaching the termination condition is also given. As illustrated before, termination happens based on a pre-defined tolerance $\epsilon$, which is set to $10^{-6}$.

The lower frequency can be explained as emulators are mainly designed for debugging, not for application processing. However, the selected algorithm, along with our optimizations, eliminate the need for high operating frequency and make full use of the emulator massive capacity.

TABLE I. Resource utilization and maximum frequency for our JOR design using various test cases.

| | Proposed JOR design test cases | | | |
|---|---|---|---|---|
| Number of equations | 420 | 11,100 | 44,700 | 2,002,000 |
| Number of ALUs | 14 | 74 | 149 | 1,000 |
| Frequency (MHz) | 2.00 | 2.00 | 2.00 | 1.75 |
| Number of iterations | 5 | 5 | 5 | 5 |
| Number of LUTs | 164,449 | 770,432 | 1,527,869 | 10,122,146 |
| Number of flip-flops | 40,833 | 167,751 | 326,382 | 2,126,259 |
| Memory usage (KB) | 152.2 | 512 | 1,630 | 40,128 |

### B. Timing performance

The timing performance of our JOR design is evaluated by comparing the needed time to solve a given number of equations using our proposed design against two software solvers; *MatJOR* and *ALGLIB* on a 2.00 GHz Core i7-2630QM CPU. The *MatJOR* is a MATLAB implementation of the JOR design. MATLAB is selected as it is the tool used in the targeted EM simulator in [16]. *ALGLIB* is an open-source numerical analysis library that includes several direct and iterative solvers. The best iterative solver is selected to be our second benchmark.

Table II gives the results obtained from comparing our emulator implementation against the two benchmarks. Speed-up columns were calculated by dividing the benchmark execution time by our design execution time. In order to get the near accurate processing time, the software benchmarks were run for many iterations so that most of the data resides in the local cache and thus reduces any disk access overhead. Results show that the gained speed-up over the software solvers increases as the number of equations-to-solve increases.

TABLE II. TIMING COMPARISONS BETWEEN OUR PROPOSED JOR DESIGN AND TWO SOFTWARE BENCHMARKS.

| Number of equations | Proposed JOR design | MatJOR | | ALGLIB | |
|---|---|---|---|---|---|
| | Time (msec) | Time (msec) | Speed-up | Time (msec) | Speed-up |
| 420 | 0.104 | 0.345 | 3.32 | 0.350 | 3.37 |
| 11,100 | 0.464 | 9.292 | 20.03 | 3.001 | 6.47 |
| 44,700 | 0.914 | 39.581 | 43.31 | 12.00 | 13.13 |
| 2,002,000 | 4.580 | 2393.085 | 522.51 | 847.049 | 184.95 |

### C. Comparison

For the sake of completeness, we investigate the superiority of our design by making an experimental comparison with the Jacobi design in [5]. To give a fair comparison, both designs were run on the same hardware emulation platform with the configuration described in Section V. Table III compares the two designs in terms of the execution time and the resources. Speed-up is calculated as before. Area overhead is calculated based on the summation of LUTs and FFs using (1).

$$Area\ Overhead(\%) = \frac{JOR - Jacobi}{Jacobi} \times 100. \qquad (1)$$

Comparisons show that there is an almost fixed area overhead of 50%, as our basic ALU contains three more FP modules compared to the one in [5]. Speed-ups up to 5.25x were gained due to the higher convergence rate of the JOR method. It is worth mentioning that direct comparisons with other hardware solvers, mentioned in Section III, are not applicable here as the generated test cases used there are small, limited, and do not fit within the same specifications of our SLS generated from the FEM simulation process.

TABLE III. TIMING AND RESOURCES COMPARISON BETWEEN OUR PROPOSED JOR DESIGN AND THE JACOBI DESIGN IN [5].

| Number of equations | Proposed JOR design | | Jacobi design in [5] | | Speed-up | Area overhead (%) |
|---|---|---|---|---|---|---|
| | Time (msec) | LUTs/FFs (×1000) | Time (msec) | LUTs/FFs (×1000) | | |
| 420 | 0.104 | 164/40 | 0.319 | 106/18 | 3.07 | 65% |
| 11,100 | 0.464 | 770/167 | 1.721 | 535/85 | 3.71 | 51% |
| 44,700 | 0.914 | 1,527/326 | 3.793 | 1,071/168 | 4.15 | 49% |
| 2,002,000 | 4.580 | 10,122/2,126 | 24.040 | 7,158/1,115 | 5.25 | 48% |

## VII. CONCLUSION

Over an EM problem of solving Maxwell's equations in a 2D metamaterial edge element using FEM, our JOR hardware solver demonstrated a remarkable improvement in the execution time of solving SLS over various software and hardware solvers. The execution of our JOR design on a physical hardware emulation platform achieved a speed-up of 522x over the same implementation on MATLAB and a speed-up of 5x over a Jacobi hardware implemented on the same platform for solving 2 million equations. No previous work has achieved as high number of equations, within a reasonable execution time, as ours. Future work include evaluating the efficiency of our hardware solver against GPU solvers.

## REFERENCES


[1] R. Schmitt. *Electromagnetics Explained: A Handbook for Wireless/RF, EMC, and High-speed Electronics*. Newnes, Boston, MS, USA, 2002.

[2] A. C. Polycarpou. *Introduction to the Finite Element Method in Electromagnetics*. Morgan & Claypool Publishers, San Rafael, CA, USA, first edition, 2006.

[3] V. E. Taylor, A. Ranade, and D. G. Messerschmitt. SPAR: A new architecture for large finite element computations. *IEEE Transactions on Computers*, 44(4):531–545, April 1995.

[4] Mentor Graphics Corporation. Veloce user's guide software version 3.0.1, May 2015.

[5] M. Tarek Ibn Ziad, M. Hossam, M. A. Masoud, M. Nagy, H. A. Adel, Y. Alkabani, M. Watheq El-Kharashi, K. Salah, and M. AbdelSalam. Finite element emulation-based solver for electromagnetic computations. In *Proceedings of the 2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1434–1437, Lisbon, Portugal, May 2015.

[6] M. Tarek Ibn Ziad, M. Hossam, M. A. Masoud, M. Nagy, H. A. Adel, Y. Alkabani, M. W. El-Kharashi, K. Salah, and M. AbdelSalam. On kernel acceleration of electromagnetic solvers via hardware emulation. *Computers and Electrical Engineering*, 47:96–113, October 2015.

[7] D. M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, Inc., London, UK, 1971.

[8] R. Bagnara. A unified proof for the convergence of Jacobi and Gauss-Seidel methods. *SIAM Review*, 37(1):93–97, March 1995.

[9] M. N. O. Sadiku. *Numerical Techniques in Electromagnetics*. CRC Press, Boca Raton, FL, USA, second edition, 2000.

[10] J. P. Durbano and F. E. Ortiz. FPGA-based acceleration of the 3D finite-difference time-domain method. In *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 156–163, Napa, CA, USA, April 2004.

[11] M. Tarek Ibn Ziad, Y. Alkabani, and M. W. El-Kharashi. On hardware solution of dense linear systems via Gauss-Jordan elimination. In *IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing (PacRim)*, pages 364–369, Victoria, Canada, August 2015.

[12] G. R. Morris and K. H. Abed. Mapping a Jacobi iterative solver onto a high-performance heterogeneous computer. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):85–91, January 2013.

[13] J. D. Davis, C. P. Thacker, and C. Chang. BEE3: Revitalizing computer architecture research. Technical report, Microsoft Research Redmond, WA, USA, 2009.

[14] A. George, H. Lam, and G. Stitt. Novo-G: At the forefront of scalable reconfigurable supercomputing. *Computing in Science Engineering*, 13(1):82–86, January 2011.

[15] S. Bochkanov. ALGLIB: A cross-platform numerical analysis and data processing library. http://www.alglib.net.

[16] J. Li and Y. Huang. *Time-Domain Finite Element Methods for Maxwell's Equations in Metamaterials*. Springer Series in Computational Mathematics, 2013.

[17] F. D. Dinechin and B. Pasca. Designing Custom Arithmetic Data Paths with FloPoCo. *IEEE Design & Test of Computers*, 28(4):18–27, July 2011.