Finite Element Emulation-based Solver for Electromagnetic Computations

M. Tarek Ibn Ziad*, Mohamed Hossam*, Mohamad A. Masoud*, Mohamed Nagy*, Hesham A. Adel*,
Yousra Alkabani*, M. Watheq El-Kharashi*, Khaled Salah†, and Mohamed AbdelSalam†
Email: {mohamed.tarek,mohamed.hossam}@eng.asu.edu.eg
{mohamad.masoud93,eng.mohamednagyelewa,eng.hesham.eldeeb}@gmail.com
{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) computations are the cornerstone in the design process of several real-world applications, such as radar systems, satellites, and cell-phones. Unfortunately, these computations are mainly based on numerical techniques that require solving millions of linear equations simultaneously. Software-based solvers do not scale well as the number of equations-to-solve increases. FPGA solver implementations were used to speed up the process. However, using emulation technology is more appealing as emulators overcome the FPGA memory and area constraints. In this paper, we present a scalable design to accelerate the finite element solver of an EM simulator on a hardware emulation platform. Experimental results show that our optimized solver achieves 101.05x speed-up over the same pure software implementation on MATLAB and 35.29x over the best iterative software solver from ALGLIB C++ package in case of solving 2,002,000 equations.**

Keywords- electromagnetic computations, emulation, finite element method, Jacobi iterative method

I. Introduction

Electromagnetic (EM) computations aim to model the interaction of EM fields with physical objects and the surrounding environment. They are considered very useful tools for the design and modeling of antenna, radar, satellite, and other communication systems. They use computationally efficient numerical approximations to Maxwell's equations based on the given boundary conditions and a set of initial conditions. One of these numerical approximations techniques is the finite element method (FEM) [1], which is widely used in modeling EM problems with complex geometries. It has received considerable attention from scientists and researchers around the world after the latest technological advancements and computer revolution in the twentieth century [2]. Unfortunately, it results in large sparse systems of linear equations that consume great part of the solution run-time in solving them. Thus, there is a need to accelerate the solver part of the FEM.

As software-based solvers are often too slow, the usage of ASICs arose as a suitable alternative. However, this approach requires a long development period and is inflexible and costly. FPGAs overcame this problem as they provide sufficient speed and logic density to implement highly complex systems with low cost and high reconfigurability. However, area constraints still represent a major problem that faces FPGA-based designs. So, we aim to use a physical hardware emulation platform instead [3].

Generally, hardware emulation platforms allow large SoC designs to be modeled in hardware. They could be described as huge arrays of connected FPGAs, which allow the design to run at clock rates of several KHz with advanced debug visibility and run-time control [4]. So, they are often used in system-level verification. In this work, we extend the emulator usage by introducing it as an efficient hardware accelerator for FEM solvers involved in EM computations.

The key contributions of this paper include: **(1)** proposing an efficient emulation technique, based on a physical hardware emulation platform, to accelerate the solver part of an EM simulator using FEM, **(2)** introducing an optimized hardware-based architecture, based on Jacobi iterative method, to solve the sparse system of linear equations arising in FEM formulations, **(3)** using a time-domain problem of solving Maxwell's equations in metamaterials using FEM as a case study to show the efficiency of our solution, **(4)** showing the logic utilization of implementing the proposed architecture and comparing the obtained timing results with three software-based implementations.

The rest of this paper is organized as follows. Section II summarizes some of prior work related to FEM hardware acceleration and solving systems of linear equations on hardware-based devices. Section III provides a brief overview of FEM and the Jacobi iterative method. Section IV details the hardware-based implementation of our Jacobi iterative solver. Experimental environment is described in Section V. The obtained results and comparisons between software and hardware solutions are introduced in Section VI. Section VII concludes the work.

II. Related Work

There are many work that targeted accelerating the numerical techniques used in the EM computations. For example, Durbano and Ortiz accelerated the finite difference time-domain (FDTD) method [5] by using custom-designed accelerator board, which supports up to 16 GB of DDR SDRAM, 36 MB of DDR SRAM, with Xilinx Virtex-II 8000 FPGA. El-Kurdi *et al.* developed a deeply pipelined FPGA design for efficient sparse matrix-vector multiplication (SMVM) [6] as this operation represents the kernel of many iterative numerical methods to solve sparse linear systems, resulting from using the FEM. We overcame this point by implementing a simpler algorithm based on the Jacobi iterative method and splitting the main SMVM into small independent ones.

On the other hand, many authors focused solely on implementing efficient solutions for solving systems of linear equations on FPGAs. For iterative solvers, Morris and Prasanna implemented a double precision Conjugate Gradient (CG) and a Jacobi iterative solver for sparse matrices on FPGA [7]. For the Jacobi iterative solver, the whole algorithm was implemented on the FPGA. For the CG solver, only the matrix-vector multiplication was implemented on the FPGA while the remainder of the algorithm was executed in software from the SPARSKIT library [8], which has routines for sparse matrix computation. Due to the limited on-chip memory, the matrix size is limited to 2,048 for the Jacobi iterative solver and 4,096 for the CG solver. Moreover, Morris and Abed implemented a sparse matrix Jacobi iterative solver that targeted a contemporary high-performance heterogeneous computer (HPHC) [9]. Instead of utilizing hardware description language (HDL) for their FPGA-based kernel designs, they used a high-level language (HLL)-based design. However, their solver could handle matrices up to order $n = 8K$ only.

For direct solvers, Zhuo and Prasanna implemented a direct solver using LU factorization method [10]. They utilized a circular linear array of processing elements (PEs) in double precision to perform the calculations on a Xilinx Virtex-II Pro XC2VP100. Johnson *et al.* presented the design and prototype implementation of sparse direct LU decomposition on FPGA [11]. They compared their performance to a general purpose processor based platform. In [12], Greisen *et al.* investigated several solver techniques, discussed hardware trade-offs, and introduced FPGA architectures of Cholesky direct solver and BiCGSTAB iterative solver. Although the authors outperformed software implementations, their iterative solver design was memory-bandwidth limited because vectors of the full problem size need to be accessed in each iteration. We handled this issue in our design as we divided the main matrix into independent clusters, as shown in Section IV.

Finally, we conclude that as the computation complexity order of direct solvers is higher than iterative solvers for the same matrix, the performance of direct solvers was limited by computation and not by memory bandwidth. So, iterative solvers were considered more suitable for solving sparse matrices like the ones resulting from the FEM. Thus, the main focus of the current work is to develop a scalable hardware architecture that can solve large systems of linear equations of any size, up to the massive capacity of the emulator resources as previous work failed to deal with very large sparse systems containing millions of equations that result from FEM discretization.

III. Background

In this section, we present a very brief overview of the FEM. Then, we describe the mathematical background of the Jacobi iterative method.

A. Finite Element Method (FEM)

FEM is a numerical method, that is used to solve boundary-value problems defined by a partial differential equation (PDE) and a set of boundary conditions [2]. The first step of using FEM to solve a PDE is discretizing the computational domain into finite elements. Then, the PDE is rewritten in a weak formulation. After that, proper finite element spaces are chosen and the finite element scheme is formed from the weak formulation. The next step is calculating those element matrices on each element and assembling the element matrices to form a global linear system. Then, the boundary conditions are applied, the sparse linear system is solved, and finally, post-processing of the numerical solution is done.

B. Jacobi Iterative Method

For a sparse linear system represented by $Ax = b$, where A is the coefficients matrix, x is the unknowns vector, and b is the right-hand-side (RHS) vector, the i^{th} equation can be represented as $\sum_{j=1}^{n} a_{i,j}x_j = b_i$, where i and j are row and column numbers, respectively. In order to solve for x_i iteratively, the previous equation can be rearranged to have a relation between ${x_i}^{(t+1)}$ and ${x_i}^{(t)}$, where t is the iteration number.

$$ {x_i}^{(t+1)} = \frac{b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{i,j}{x_j}^{(t)}}{a_{i,i}} \qquad (1) $$

Briefly, the Jacobi method starts with an initial guess of vector x and solves each unknown x using (1). The obtained ${x_i}^{(t+1)}$ is then used as the current solution and the process iterates many times until convergence, which is guaranteed with the Jacobi method if the coefficient matrix is symmetric, positive definite, and diagonally dominant. These conditions are satisfied in case of FEM [13]. So, from (1), each x_i can be solved independently and thus the Jacobi method has inherent parallelism.

IV. Hardware Implementation

In this section, we introduce the hardware implementation of our optimized architecture, which models the Jacobi iterative method.

Fig. 1 represents the block diagram of our proposed Jacobi hardware design. It consists of a control unit, main memory, diagonal memory, RHS memory, non-diagonal memory, and main ALU unit which consists of the result convergence check module and multiple ALUs. The design splits the coefficient matrix into clusters, where each cluster contains a number of rows. Each cluster is independent of the others, so all of them can be operated on in parallel.

The main memory contains the required solutions. Initially, it is loaded with an assumption of the result, which is chosen to be zero. Each memory row contains a whole cluster

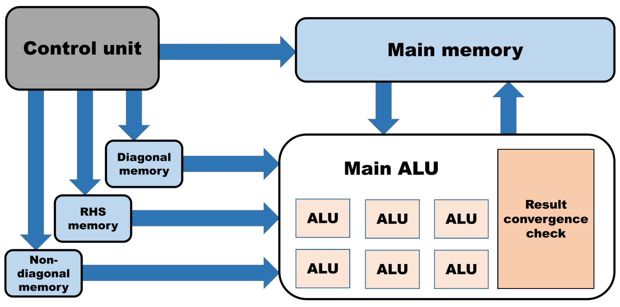

Fig. 1. Block diagram of the proposed Jacobi hardware design.

of data, not just one row of the matrix. This compensates memory constraints and allows memories to serve more ALUs simultaneously. During each iteration, the main memory loads one row of data to the ALUs, then stores the results of the calculation at the end of iteration. This stored data is then operated on during the next iteration until convergence is reached. Diagonal memory, RHS memory, and non-diagonal memory are all read-only memories. The first two memories contain the diagonal $a_{i,i}$ and RHS b_i, and the non-diagonal elements of each independent cluster of the coefficient matrix, respectively. Since the coefficient is sparse, only the non-zero elements of each row are stored in these memories. Memory depth of all memories equals the matrix dimension divided by the number of rows in each cluster.

Each ALU performs the arithmetic operations shown in (1), except that the final division operation is turned into a multiplication operation with the inverse of the diagonal element in order to minimize the critical path due to the considerable difference in latency and required hardware resources between multiplication and division floating-point modules. So, each ALU contains three 32-bit floating-point multipliers, one 32-bit floating-point adder, and one 32-bit floating-point subtractor. The result convergence check module is used to determine the termination criterion. It stops working on data based on a pre-defined value representing the error tolerance. Whenever the current error between any two consecutive iterations is below that pre-defined tolerance value, this module generates a halt signal indicating end of operations.

The control unit is responsible for synchronizing all memories with each ALU, controlling the result convergence check module, 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 main memory at the end of iteration. It also has a counter that represents the current iteration.

This design has many advantages. It is a fully pipelined design. During every clock cycle new data is fetched from memories and is operated on by one of the ALU operations that represents the pipeline stages. Furthermore, it could be configured with a pre-defined tolerance to control the accuracy of the final solution. Moreover, it is a very flexible multi-core system that can be configured for a broad range of matrix sizes with a broad range of ALU cores for increased productivity. Thus, it makes full use of the huge capacity of emulator logic resources and memory.

V. Experimental Setup

Our architecture was modeled using Verilog. Xilinx ISE Design suite 14.6 was used to check its functional correctness. The design was then compiled and run on a physical hardware emulation platform with 8 advanced verification boards (AVBs) [3]. That platform provides a total capacity of 128 crystal chips with 4GB of memory. Crystal chips are equivalent to FPGAs, but use different technology. Each crystal chip has around 500K of logic gates. That clearly solves the area and memory constraints of FPGAs as even the largest available FPGA cannot provide this massive capacity. Nowadays, emulation technology allows designs to use up to 64 AVBs, which means an equivalent logic resources of 1024 connected FPGAs.

The main procedures in the emulation design process are analyzing Verilog input files and performing syntax checking. Then, register transfer level compilation (RTLC) generates structural Verilog netlist of emulator primitives; LUT, flip-flop, latch, and memory. After that, the synthesizer performs partitioning and ASIC netlists for each crystal chip, which represents the programmable logic for emulation. The chip compiler step does placement and routing. Finally, the global scheduler performs final timing analysis and generates timing information for resources access, memories, emulator events, and clocks [4].

VI. Experimental Results

This section evaluates the performance of our Jacobi design, described in Section IV, in terms of resource utilization and speed-up over three software solutions. All test cases used are generated from the pre-processing part of the EM solver discussed below.

A. *Electromagnetic Solver*

To demonstrate the efficiency of our proposed solution, we implement a two-dimensional (2D) edge element code for solving Maxwell's equations for metamaterials using FEM. The code consists of three parts; pre-processing, solver, and post-processing. Pre-processing and post-processing calculations were performed on MATLAB [14]. Code inputs are the x and y co-ordinates of the lower and higher edges in the 2D element, number of meshes to define the needed resolution, and material properties. The outputs are the numerical Electric and Magnetic field graphs. Increasing the number of meshes increases the total number of equations-to-solve, and as a result, better accuracy is obtained. The solver part, which consumes most of time, is accelerated using the hardware emulation platform described in Section V. More information about the programming process including mesh generation, FEM calculations, and post-processing of numerical solutions is introduced in [15].

B. *Resource Utilization*

Table I shows the logic utilization, memory capacity, and operating frequency of our Jacobi design using single floating-point precision for different test cases. Number of ALUs represents the number of equations per clusters in the design, which is determined by the number of meshes in the x and y directions of the 2D element. The number of iterations needed until reaching the termination condition is also given. As illustrated before, termination happens based on a pre-defined tolerance, which is set to 10^{-6} in our test cases.

TABLE I. Hardware resource utilization for our single floating-point Jacobi design using different test cases.

	Our Jacobi design test cases				
Number of equations	420	11,100	44,700	179,400	2,002,000
Number of ALUs	14	74	149	299	1,000
Frequency (KHz)	1666.7	1754.4	1587.3	1754.4	1333.3
Number of iterations	21	22	22	22	22
Number of flip-flops	18,752	85,490	168,896	335,702	1,115,220
Number of LUTs	106,401	535,566	1,071,976	2,144,796	7,158,357
Memory bytes	8,704	376,832	1,521,664	6,115,328	40,943,616
Number of FPGAs	2	8	15	29	98

The obtained results show that the used frequency is almost stable and equals the maximum possible frequency for the used emulator. This guarantees linear run-time, as it will depend only on the number of clock cycles. The main source that increases the hardware resources consumption is the number of parallel ALUs.

C. Speed-up

The speed-up of our Jacobi design is evaluated against three different software-based solvers on a 2.00 GHz Core i7-2630QM CPU. The first solver is a standard Jacobi iterative implementation on MATLAB [16]. The second is mldivide (\), the MATLAB special operator for solving systems of linear equations that employs the best suitable solver after examining the coefficient matrix. Finally, the third solver is an iterative solver from ALGLIB [17], an open-source numerical analysis library that supports several programming languages, including C++. It was chosen as a software benchmark due to its ease of implementation and ability to be compiled across multiple platforms.

Table II gives the obtained results from comparing our emulator-based implementation against software solvers. Speed-ups were computed by dividing the software run-time by our proposed Jacobi design run-time. Fig. 2 gives a graphical representation of the speed-up results in Table II to highlight our performance improvement. As the speed-up increases with the number of equations-to-solve, more speed-up can be obtained at larger numbers of equations.

TABLE II. TIMING COMPARISONS BETWEEN OUR SINGLE FLOATING-POINT JACOBI DESIGN AND VARIOUS SOFTWARE SOLVERS.

Equations	Our Jacobi	MATLAB Jacobi		MATLAB mldivide		C++ ALGLIB	
	Time (Sec)	Time (Sec)	Speed-up	Time (Sec)	Speed-up	Time (Sec)	Speed-up
0,000,420	0.0003	0.0009	03.00	0.0002	00.66	0.0004	01.33
0,001,200	0.0007	0.0018	02.57	0.0005	00.71	0.0010	01.43
0,004,900	0.0011	0.0054	04.90	0.0017	01.55	0.0020	01.82
0,011,100	0.0017	0.0113	06.65	0.0041	02.41	0.0030	01.76
0,044,700	0.0038	0.0440	11.58	0.0253	06.66	0.0120	03.16
0,179,400	0.0062	0.2157	34.79	0.1608	25.94	0.0570	09.19
2,002,000	0.0240	2.4252	101.05	1.9138	79.74	0.8470	35.29

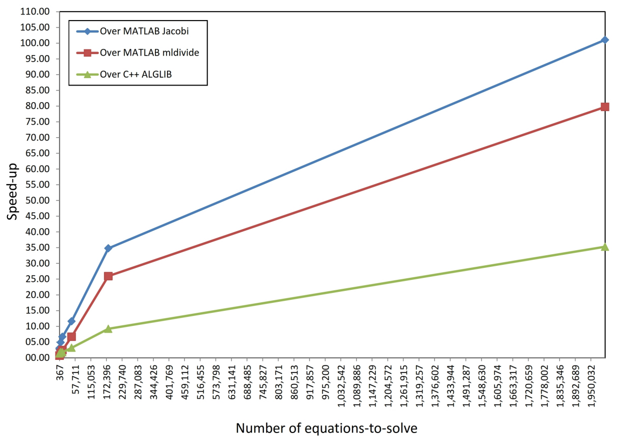

Fig. 2. Speed-up of our single floating-point Jacobi design over various software solvers.

VII. Conclusion

Our optimized single floating-point Jacobi design has achieved a remarkable improvement in matrix calculations run-time over the software-based solvers for a simple EM problem of solving Maxwell's equations in a 2D metamaterial edge element using FEM. The execution time of our design on the physical emulator with total capacity of 8 AVBs and 4GB of memory achieved a speed-up of 101.05x over a standard Jacobi MATLAB implementation executed on a 2.00 GHz Core i7-2630QM CPU for solving 2,002,000 equations. The same hardware configuration achieved a speed-up of 79.74x over the MATLAB special operator for solving linear equations and a speed-up of 35.29x over ALGLIB, a numerical analysis and data processing library that uses iterative solvers. Higher speed-up could be obtained upon using more emulator area as the design is fully parallelized.

References

[1] G. Strang and G. Fix. *An Analysis of the Finite Element Method: Englewood Cliffs*. Prentice Hall, New Jersey, USA, 1973.

[2] A. C. Polycarpou. *Introduction to the Finite Element Method in Electromagnetics*. Morgan & Claypool, 2006.

[3] Mentor Graphics Corporation. Veloce Emulation Platform. http://www.mentor.com/products/fv/emulation-systems/.

[4] Mentor Graphics Corporation. Veloce User's Guide Software Version 2.1, 2012.

[5] 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, April 2004.

[6] Y. El-Kurdi, D. Giannacopoulos, and W. J. Gross. Hardware Acceleration for Finite-Element Electromagnetics: Efficient Sparse Matrix Floating-Point Computations With FPGAs. *IEEE Transactions on Magnetics*, 43(4):1525–1528, April 2007.

[7] G. R. Morris and V. K. Prasanna. *Sparse Matrix Computations on Reconfigurable Hardware*. Computer, 40(3), March, 2007.

[8] Y. Saad. SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations - Version 2, 1994.

[9] 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.

[10] L. Zhuo and V. K. Prasanna. High-Performance and Parameterized Matrix Factorization on FPGAs. In *International Conference on Field Programmable Logic and Applications, FPL '06*, Madrid, August 2006.

[11] J. Johnson, T. Chagnon, P. Vachranukunkiet, P. Nagvajara, and C. Nwankpa. Sparse LU Decomposition Using FPGA. In *Proc. Int. Workshop State-of-the-Art Scientific Parallel Comput. (PARA)*, 2008.

[12] 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. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(8):1402–1407, August 2013.

[13] M. N. O. Sadiku. *Numerical Techniques in Electromagnetics*. Taylor & Francis, Second edition, 2000.

[14] MATLAB. *version 7.12.0.635 (R2011a)*. The MathWorks Inc., Natick, Massachusetts, 2011.

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

[16] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1994.

[17] S. Bochkanov. ALGLIB: A Cross-platform Numerical Analysis and Data Processing Library. http://www.alglib.net.