Performance Comparison of FPGAs and GPUs: Solving Sparse Matrices Case-Study

Khaled Salah[1], Mohamed AbdelSalam[1]
[1]Mentor, A Siemens Business
Cairo, Egypt
khaled_mohamed@mentor.com, mohamed_abdelsalam@mentor.com

Abstract: - In this paper, performance comparison of FPGAs and GPUs are introduced. Numerical methods to solve sparse matrices are evaluated as the main case-study. The experimental results showed that GPUs show superior performance over FPGAs/HW Emulation in terms of run time for small #equations. For large number of equations "in order of ten millions", the FPGAs/HW emulation outperforms GPUs as the parallelism rate of the emulation becomes higher in that case.

Key-Words: - Numerical Method, FPGA, GPU, Sparse Matrices, Matrices.

1 Introduction

Electromagnetic (EM) simulation is currently a highly-needed planning tool in high frequency systems. It enables the enhancement and the optimization of the performance of systems while they are still in the design phase.

The precise modeling of the relevant physical interactions in the simulation environment helps significantly in the optimization process and ensures that the system is optimally adapted to blend into its environment.

The main objective of EM simulation is to find an approximate solution for Maxwell's equations that satisfies the given boundary conditions and a set of initial conditions.

Several real-world electromagnetic problems are not analytically calculable, for the multitude of irregular geometries found in actual devices. Computational numerical techniques can overcome the inability to derive closed form solutions of Maxwell's equations under various constitutive relations of media, and boundary conditions.

Many numerical techniques are used for solving Maxwell's equations, e.g., the finite difference method (FDM) [1], finite element method (FEM) [2], finite volume method (FVM) [3], and the boundary element method (BEM) [4]. Computations involved in FEM consume too much time, which affects the final time-to-market value. Profiling shows that the most time-consuming part in the simulation process is the solver part, which is responsible for solving the resultant system of linear equations generated from the FEM.

The total number of equations may reach thousands or millions of linear equations. Thus, software-based EM solvers are often too slow.

Several alternatives are introduced to accelerate the solver part including Application-specific Integrated Circuits (ASICs) for their high speed, Graphics Processing Units (GPUs) for their parallelization power, Field Programmable Gate Arrays (FPGAs) for their high speeds and relatively low cost and finally multi-FPGA systems for their abundance of logic resources. Although all these several solutions [5]-[6], there is still a room to improve the speed and scalability of these solvers.

Many publications targeted accelerating the numerical techniques used in the EM computations using different technologies. For example; Huan-Ting Meng and Bao-Lin Nie accelerated the Finite Element Computation for Electromagnetic Analysis [7]-[15] as they focused on two bottlenecks, the first is related to the calculation of the elemental matrices and the second is related to the assembly of the elemental matrices into the global finite element matrix. There was also remarkable approaches to accelerate the Method of Moments [15]-[16] and the FDTD [17]-[18] using GPUs.

For direct solvers, Maxim Naumov implemented Incomplete-LU and Cholesky Preconditioned Iterative Methods [18] using CUSPARSE [19] and CUBLAS [20], the work focused on the Bi-Conjugate Gradient Stabilized and Conjugate Gradient iterative methods, which can be used to solve large sparse non-symmetric and symmetric positive definite linear systems. G. Ortega and E.M. Garzón accelerated Bi-Conjugate gradient method [21] the purpose was to compare the performance of sparse matrix vector product using CUSPARSE and ELLR-T routine.

The ELLR-T has proven to outperform the other sparse matrix kernels as demonstrated by F. Vázquez and J. J. Fernández [22]. Zhang has proposed a CUDA-Based implementation for the Jacobi Iterative Method [23], the paper focused on dense linear equations.

For Sparse Matrix vector product, Nathan Bell and Michael Garland implemented kernels for the popular sparse matrix representation [24] kernels were implemented for Diagonal, ELLPACK, Coordinate, CSR, Hybrid and Packet formats.

In this work, we propose hardware architecture for the conjugate gradient method to be used in solving linear equation. This algorithm is implemented on both FPGA and GPU to evaluate the performance improvement between them. For the best of our knowledge, this is the first work to address such a comparison for linear equation solvers.

GPU technology was introduced as solution to accelerate those different solvers for its parallelization power.

This paper is organized as follows: In Section II, problem formulation and contribution are discussed. In Section III, the background for FEM and conjugate gradient (CG) is presented. In Section IV, the proposed methodology is introduced. In Section V, Results are discussed. In Section VI, Conclusions are given.

2 Problem Statement and Contribution

Computations involved in FEM consume too much time, which affects the final time-to-market value. Profiling shows that the most time-consuming part in the simulation process is the solver part which is responsible for solving the resultant system of linear equations generated from the FEM. The total number of equations may reach thousands or millions of linear equations.

Thus, software-based EM solvers are often too slow. Thus the main contribution of this work is as follows:

1. Improve runtime performance of the solver part of the EM simulator on the emulator by using new techniques/designs.

2. Do extensive comparisons between our proposed solvers and state-of-the-art solvers on GPUs/FPGAs.

3. Implement a GUI for plotting EM computations using the hardware emulator co-model channel. This channel implies using SystemVerilog Direct Programming Interface (DPI) to provide the link between hardware running on emulator and software running on a co-model server connected to the emulator.

3 Background

In this section, we provide a brief overview of the electromagnetics simulations and FEM. Then, we describe the details of the conjugate gradient method used to solve the matrices resulting from using the FEM. Moreover, we give a quick introduction about GPU and CUDA.

3.1 EM Simulations

The Key steps for electromagnetic simulation are:

- Creation of Physical Model: Drawing/Importing Layout Geometry, assigning materials, etc...

- Setup of EM Simulation: Defining boundary conditions, ports, simulation settings, etc...

- Performance of EM Simulation: Discretizing the physical model into mesh cells and approximating the field/current using a local function (Expansion/Basis function) and adjusting the function coefficients until the boundary conditions.

- Post-Processing: Calculating S-parameters, TDR response, Antenna far field patterns, etc...

3.2 Finite Element Method

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. 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 system of linear systems (SLS), and finally.

(6) Post-processing the numerical solution.

3.3 Conjugate Gradient Method

Basic iterative methods such as Jacobi Method and Gauss Seidel method cannot solve all the linear systems. The Conjugate Gradient method is one of the Krylov subspace methods. The conjugate gradient method derives its name from the fact that it generates a sequence of conjugate (or orthogonal) vectors. These vectors are the residuals of the iterates. They are also the gradients of a quadratic functional, the minimization of which is equivalent to solving the linear system.

Conjugate Gradient is an Iterative Method applicable to sparse systems that are too large to be handled by a direct implementation. It has the advantage that it reaches the required tolerance after a relatively small number of iterations compared to Jacobi and gauss methods.

The conjugate gradient method is used to solve equations where the matrix is symmetric. Conjugate gradient method can reach the required tolerance after a relatively small number of iterations compared to Jacobi AND Gauss Method [7]-[10]. The conjugate gradient algorithm used in this work is shown in Listing 1.

Listing 1: Conjugate gradient algorithm.

Conjugate gradient algorithm : The algorithm is detailed below for solving **Ax** = **b.**

Input: Number of linear equations.

Variables:

A is a real, symmetric, positive-definite matrix.

The input vector x_0 can be an approximate initial solution or 0.

Output: Solution.

```
$r_0 = b - Ax_0$
$p_0 = r_0$
$k = 0$
Repeat
$\alpha_n = \frac{r_n^T r_n}{p_n^T A p_n}$

$x_{n+1} = x_n + \alpha_n p_n$
$r_{n+1} = r_n - \alpha_n A p_n$
If $r_{n+1}$ is very small then exit the loop else
$p_{n+1} = r_{n+1} + \beta_n p_n$
$n = n + 1$
End repeat
The solution is $x_{n+1}$
```

3.4 GPU and CUDA

GPUs had been mostly used for computation of computer graphics, they have only lately been explored for their parallel computation power that can be used to accelerate algorithms that can be parallelized. Speed-ups resulting from using GPU-parallelized algorithms can reach 100x compared to algorithms running on Central Processing Units (CPUs) [11]–[13].

CUDA is a parallel computing platform and programming model. The CUDA platform is designed to work with programming languages such as C, C++ and FORTRAN. It allows software developers to use GPUs for general purpose processing – an approach known as GPGPU. Memory Bandwidth for different GPUs are shown in Fig. 1.

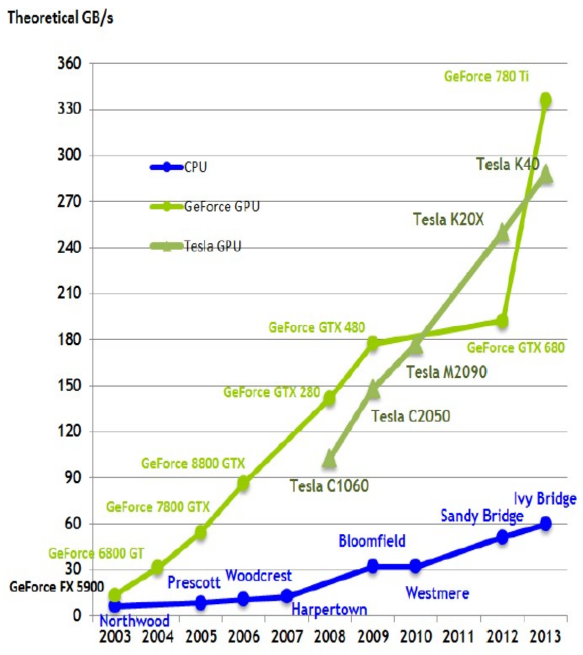

Fig. 1 Memory Bandwidth for different GPUs

The design philosophy of the GPUs is shaped by the fast growing video game industry, which exerts tremendous economic pressure for the ability to perform a massive number of floating-point calculations per video frame in advanced games. This demand motivates the GPU vendors to look for ways to maximize the chip area and power budget dedicated to floating point calculations.

The prevailing solution to date is to optimize for the execution throughput of massive numbers of threads. The hardware takes advantage of a large number of execution threads to find work to do when some of them are waiting for long-latency memory accesses, thus minimizing the control logic required for each execution thread. Small cache memories are provided to help control the bandwidth requirements of these applications so multiple threads that access the same memory data do not need to all go to the DRAM.

As a result, much more chip area is dedicated to the floating-point calculations. In contrast, there is the design philosophy of the CPU. The design of a CPU is optimized for sequential code performance. It makes use of sophisticated control logic to allow instructions from a single thread of execution to execute in parallel or even out of their sequential order while maintaining the appearance of sequential execution.

More importantly, large cache memories are provided to reduce the instruction and data access latencies of large complex applications. Neither control logic nor cache memories contribute to the peak calculation speed

3.5 Hardware Emulation Platform

In general, Veloce emulators are multi-FPGA systems, they are fast, hardware-assisted verification systems, delivering comprehensive best-in-class emulation [14] and acceleration platforms for SoC and embedded system verification.

Each Veloce emulator provides a significant increase in productivity for system-level verification because of their fast compiles, accurate modeling, productive de-bugging, and time-to-visibility features. Furthermore, the Veloce emulator family provides high-performance transaction-based acceleration, which delivers targetless acceleration with faster performance than other software solutions.

Instead of using emulation in design verification only, in this paper, we extend the emulator usage to be an efficient hardware accelerator for EM solvers calculations.

That platform provides a total capacity of 32 Crystal chips with 1GB of memory, Crystal chips are equivalent to FPGAs but use different technology.

A Crystal chip has around 500K of logic gates. The Veloce2 family of emulators includes the Quattro, Maximus, and Double Maximus.

- Quattro: 256 million gates, 16 logic boards.
- Maximus: 1 billion gates, 64 logic boards.
- Double Maximus: 2 billion gates, 128 logic boards.

4 The Proposed Architecture

In order to gain a high performance on the hardware for solving system of linear equations, we need to solve them in parallel. We could not solve linear equation in parallel as long as there are high dependencies between them, so we need a way to reduce those dependencies between equations.

There is a property in the output matrix of the finite element method that can be used to reduce those dependencies named clustering.

This property appears in the output matrix as the finite element method divides the test element into square meshes and calculations takes place on the edges of the mesh. Vertical edges in one row and horizontal edges in one column each contribute in the main output matrix with equations with high dependencies between them and low dependencies with other equations in the matrix we named each group of those equations a cluster (Fig. 2).

The proposed architecture for this method is shown in Fig. 3, where the architecture consists of the following modules:

- **Top module**

This module encapsulates all of the design modules and connects them together.

- **Control unit**

The control unit is responsible for handling the interaction between the ALU and the memories in the design.

- **Memories**

The algorithm makes use of various kinds of matrices and vectors, which we need to store in our design implementation in memories. Those memories are:

- **memA**: This memory is used to store the square coefficient matrix A.
- **memR**: This memory is used to store the dense vector (r) used in the algorithm. Initially vector (r) is loaded with the right hand vector (b), then it is updated each iteration as the algorithm requires.
- **memP**: This memory is used to store the dense vector (p). Just like vector (r), it is initially loaded with the right hand vector (b), then it is updated each iteration as the algorithm requires.
- **memX**: This is the solution vector.

- **ALU**

The ALU is responsible for all arithmetic operations performed on data. It has the following modules:

- **Dot product module**:
  - The algorithm required a number of dot product operations.
- **Matrix by vector module**:
  - The algorithm requires to perform one matrix by vector operation A*P.
- **Mult_add module**:
  - This module is responsible for performing operations of the kind ($X = Y \pm c*Z$).

The most important part in conjugate gradient method is the dot product as depicted in Fig.4. The most critical part in the dot product is the summation at the end stage. So, we propose to implement it as a tree adder (Fig. 5).

Moreover, the summation in the Inner Product is implemented using parallel reduction as the process is done as if a binary tree is being constructed. This reduces the complexity of the summation from O(N) to O(logN) as illustrated by Fig. 6. We proposed many modifications to the architecture as inserting a pipeline register in the middle of the ALU. This enabled two clustered to be worked on at the same time.

Moreover, instead of one pipeline register we added two, which is the maximum number of pipeline registers allowed by the algorithm. This allowed three clusters to be processed at the same time in different parts of the pipeline.

Tolerance calculation is not done by a distinct specialized module, as it is an implicit part in the algorithm itself and the tolerance value must be calculated every iteration anyway. When the calculated value gets smaller than the specified tolerance, a halt signal is raised and the solution vector is outputted to the output file.

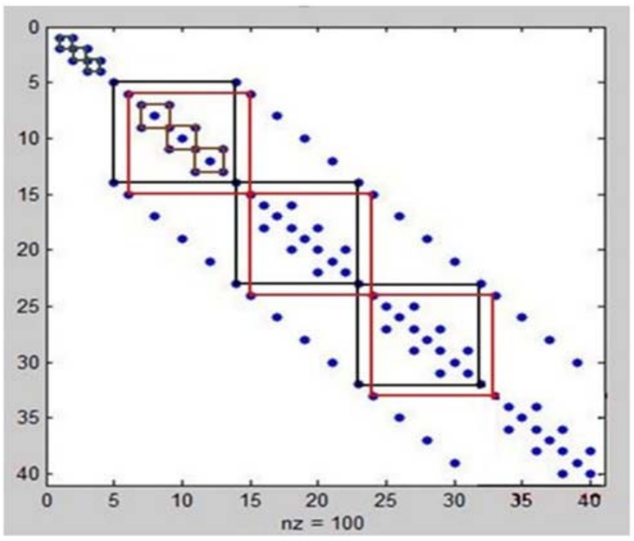

Fig. 2 An example for clustering.

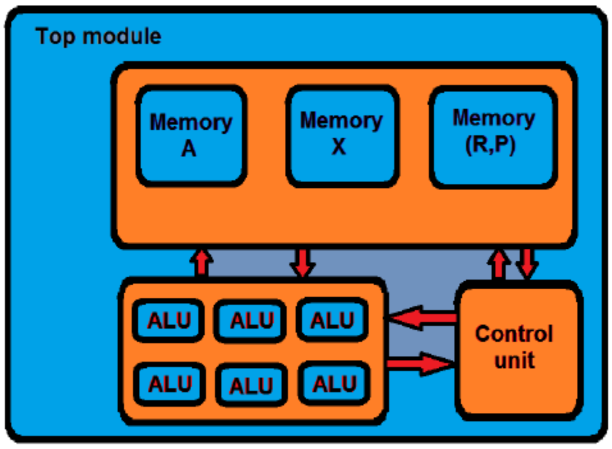

Fig. 3 The proposed architecture for Conjugate Gradient method, where A, X, R, and P represent the coefficient matrix, solution, residue and conjugate vectors respectively.

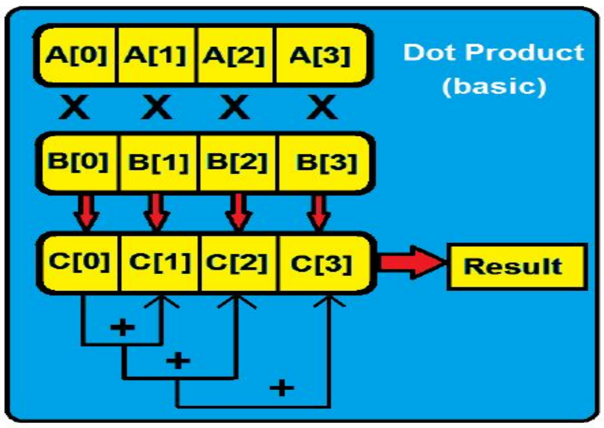

Fig. 4 Basic dot product.

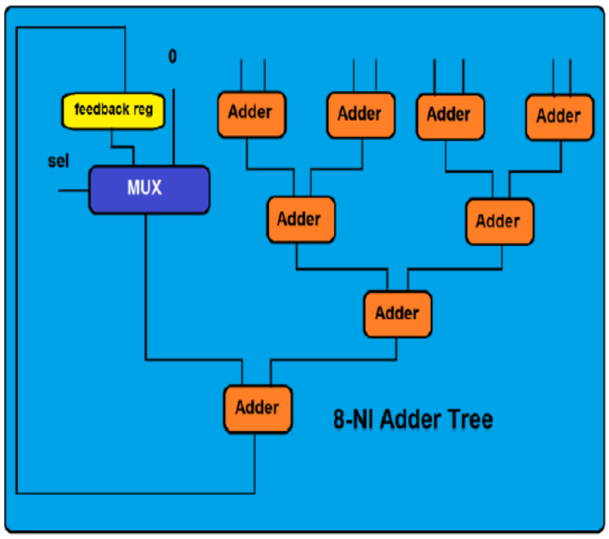

Fig. 5 Adder tree.

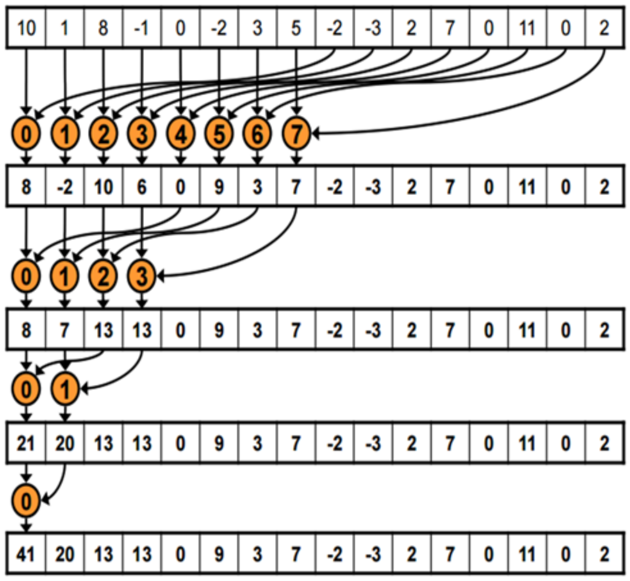

Fig. 6 Parallel reduction algorithm.

Data is sent between the software and hardware directly without the need to write any files. The user does not bother him with the details of how the program should work, all he has to do is enter the required data for the problem and then communication between software and hardware are done automatically without any intervention from the user. The design uses system Verilog DPI functions to grab the data needed for the hardware solver from software side. Hardware side: Solver implemented in Verilog. Software side: written in C language.

It works as follows:

1. User opens the program and enters the properties of the simulation system.
2. The program then save the properties of the system and runs a Makefile to run the hardware part.
3. As soon as the hardware start working it first imports a function from the C side that will perform preprocessing on the data gathered from the user.
4. When preprocessing finishes it returns to the hardware again where the solver will start working.
5. First the memory modules in the solver will call functions from the C side which will return the values of the matrices needed to be solved, data is collected from the arrays not from text files.
6. When the memories finish collecting data, the solver will continue and solve the system of equations.
7. After the solver finishes the test bench will call the post processing function which takes the values produced from the solver and produce the simulation graph.
8. When the user finished examining the simulation results the control is given back to the software side where the program will ask him whether he wants to simulate another system.

5 Results and Discussions

In order to gain a high performance on the hardware for solving system of linear equations, we need to solve them in parallel. We could not solve linear equation in parallel as long as there are high dependencies between them, so we need a way to reduce those dependencies between equations. There is a property in the output matrix of the finite element method that can be used to reduce those dependencies named Clustering.

This property appears in the output matrix as the finite element method divides the test element into square meshes and calculations takes place on the edges of the mesh.

Vertical edges in one row and horizontal edges in one column each contribute in the main output matrix with equations with high dependencies between them and low dependencies with other equations in the matrix we named each group of those equations a cluster.

GPU and FPGA are formally compatible but the results are very sensitive to the particular calculation task, the architecture of DSP blocks in particular FPGA, and the architecture of the designed core for particular calculation in FPGA.

The proposed architecture is implemented on hardware FPGA using Verilog and on GPUs using CUDA. For GPU, our results are verified using Intel core i5-4510U and GeForce 920m graphics card. GeForce 920m is based on NVIDIA’s Maxwell architecture; it has a global memory of size 2048 MBs and clock rate 900 MHz, 384 CUDA cores, a shared memory per block of size 49152 Bytes and number of threads that can reach 1024 threads per block. For FPGA, our results are verified using VERTEX-7 from Xilinx.

As a proof of concept, we applied our proposed full solution on a complete case study by using time-domain finite element methods for solving metamaterial model equations. We choose Metamaterials because they attracted great attention of researchers since their successful construction in 2000.

There is currently an enormous effort in the electrical engineering, material science, physics, and optics communities to come up with various ways of constructing efficient metamaterials and using them for potentially revolutionary applications in antenna and radar design, subwavelength imaging, and invisibility cloak design. Hence, simulation of electromagnetic phenomena in metamaterials becomes a very important issue [25]-[26].

In this section, we evaluate and compare the performance improvement of both approaches. GPU-based implementation of Jacobi versus conjugate gradient results is shown in Fig. 7. Comparison between GPUs and Emulators/FPGAs in terms of run time are shown in TABLE I, where GPUs show superior performance.

For FPGA implementation, the resources used by the algorithm including the total number of lookup tables (LUTs), slice registers, and digital signal processing blocks (DSPs), as well as the throughput obtained, are shown in TABLE II. We used VERTEX-7 from Xilinx (~244,300 equations), but for beyond we used HW Emulation.

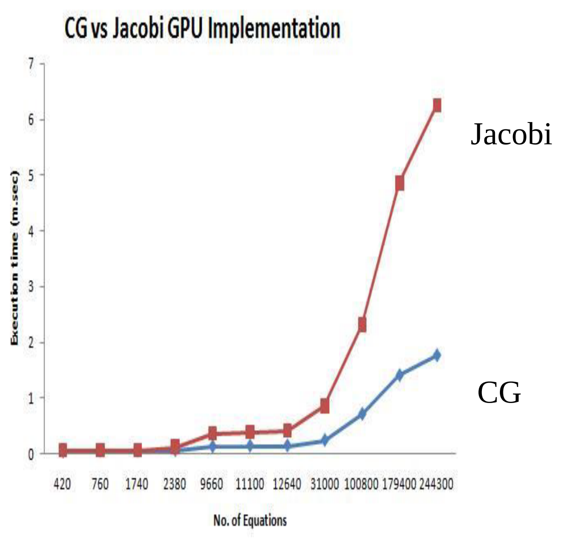

Fig. 7 Jacobi clustered Vs. CG clustered.

TABLE I
COMPARISON BETWEEN GPUS AND FPGAs

# Equations	Run-Time in ms	
	FPGAs	GPUs
420	2.6	0.02
760	3.5	0.042
1740	5.2	0.043
2380	6.1	0.05
9660	12.1	0.12
11100	13	0.13
12640	13.8	0.14
31000	21.6	0.2
100800	38.8	0.7
179400	51.7	1.7
244300	60.3	1.8

TABLE II
AREA, DELAY, AND POWEROVERHEAD FOR OUR PROPOSED ENCRYPTION RESOURCE UTILIZATION

Platform	Virtex 7
LUTs	27,000
Registers	29,000
Total DSPs	64
Freq (MHz)	300
Slices	16000

6 Conclusions

In this paper, two different approaches for solving systems of linear equations are introduced. The first approach is based on using GPU. The second approach is based on using FPGA. The experimental results show that GPUs show superior performance over FPGAs in terms of run time for small #equations. GPUs play a major role in accelerating many classes of applications, improving their performance and energy efficiency.

References

[1] SmithGD. Numerical Solution of Partial Differential Equations: Finite Difference Methods. Oxford, UK: Oxford University Press, 1978.

[2] FixG. StrangG, An Analysis of the Finite Element Method. Englewood Cliffs NJ,USA: Prentice-Hall, 1973.

[3] LeVequeR, Finite Volume Methods for Hyperbolic Problems. Cambridge, UK: Cambridge University Press, 2002.

[4] K. Banerjep,"Boundary Element Methods in Engineering". New York, NY, USA: McGraw-Hill, 1994.

[5] R. F. Carvalho, C. A. P. S. Martins, R. M. S. Batalha, and A. F. P. Camargos, '3D parallel conjugate gradient solver optimized for GPUs', in Digests of the 2010 14th Biennial IEEE Conference on Electromagnetic Field Computation, 2010, pp. 1–1.

[6] G. Wu, X. Xie, Y. Dou, and M. Wang, 'High-Performance Architecture for the Conjugate Gradient Solver on FPGAs', IEEE Trans. Circuits Syst. II Express Briefs, vol. 60, no. 11, pp. 791–795, Nov. 2013.

[7] Kendall A. Atkinson, an Introduction to Numerical Analysis (2^{nd} ed.). New York: John Wiley & Sons, 1989.

[8] Mordecai Avriel, Nonlinear Programming: Analysis and Methods. Dover Publishing, 2003.

[9] Gene H. Golub and Charles F Van Loan, "Chapter 10". Matrix computations (3^{rd} ed.). Johns Hopkins University Press, 2011.

[10] Y. Saad, "Iterative methods for sparse linear systems" (2nd ed.).SIA, 2005.

[11] http://www.nvidia.com/object/cuda_home_new.html

[12] David B. Kirk and Wen-mei W. Hwu, Programming Massively Parallel Processors - A Hands-On Approach.: Morgan Kaufmann, 2012.

[13] K. Salah. "IP Cores Design from Specifications to Production: Modeling, Verification, Optimization, and Protection." IP Cores Design from Specifications to Production. Springer International Publishing, 2016.

[14] Mentor Graphics. Veloce Emulator. [Online]. http://www.mentor.com/products/fv/emulation.html.

[15] B.-L. Nie, S. Wong, C. Macon, and J.-M. Jin H.-T. Meng, "GPU accelerated finite-element computation for electromagnetic analysis," IEEE Antennas Propag. Mag., vol. 56, no. 2, pp. 39-62, Apr. 2014.

[16] Z. Peng and Z. Nie, "Acceleration of the method of moments calculations by using graphics processing units," IEEE Transactions on Antennas and Propagation, pp. 2130-2133, July 2008.

[17] A. Karwowski, and A. Noga T. Topa, "Using GPU with CUDA to accelerate MoM-based electromagnetic simulation of wire-grid models," EEE Antennas and Wireless Propagation Letters, pp. 342-345, april 2011.

[18] A. Esposito, G. Monti, and L. Tarricone D. De Donno, "Parallel efficient method of moments exploiting graphics processing units," Microwave and Optical Technology Letters, Nov. 2010.

[19] B. Livshitz, and V. Lomakin S. Li, "Fast evaluation of Helmholtz potential on graphics processing units (GPUs)," Journal of Computational Physics, Nov. 2010

[20] .E. Lezar and D. B. Davidson, "GPU-accelerated method of moments by example: Monostatic scattering," IEEE Antennas and Propagation Magazine, Dec. 2010.

[21] A. Dziekonski, and M. Mrozowski P. Sypek, "How to render FDTD computations more effective using a graphics accelerator," IEEE Transactions on Magnetics, March 2009.

[22] V. Demir, "A stacking scheme to improve the efficiency of finite-difference time-domain solutions on graphics processing units," Applied Computational Electromagnetics Society Journal, Apr. 2010.

[23] V. Demir and A. Z. Elsherbeni, "Compute unified device architecture (CUDA) based finite-difference time-domain (FDTD)

implementation," Applied Computational Electromagnetics Society Journal, Apr. 2010.

[24] Naumov M, "Incomplete-LU and Cholesky preconditioned iterative methods using CUSPARSE and CUBLAS," Technical report and white paper 2011.

[25] N. Bell and M. Garland., "Efficient sparse matrix-vector multiplication on CUDA," NVIDIA Corporation, NVIDIA Technical Report NVR-2008-004 2008.

[26] Jichun Li and Yunqing Huang, Time-Domain Finite Element Methods for Maxwell's Equations in Metamaterials.: Springer Series in Computational Mathematics, 2013.