Solving RFIC Simulation Tasks Using GPU Computations

M.-I. Andrei[1], **S. Kula**[2]

[1] *Politehnica University of Bucharest, Electrical Engineering Faculty, Numerical Methods Laboratory, LMN*
Splaiul Independentei 313, 060042 Bucharest,Romania
E-mail: iulian@lmn.pub.ro

[2] *Kazimierz Wielki University, Institute of Mechanics and Applied Computer Science*
ul. Kopernika 1, 85-074 Bydgoszcz, Poland
E-mail: skula@ukw.edu.pl

Received: 12 November 2012; revised: 19 February 2013; accepted: 19 February 2013; published online: 14 March 2013

Abstract: New generation of General Purpose Graphic Processing Unit (GPGPU) cards with their large computation power allow to approach difficult tasks from Radio Frequency Integrated Circuits (RFICs) modeling area. Using different electromagnetic modeling methods, the Finite Element Method (FEM) and the Finite Integration Technique (FIT), to model Radio Frequency Integrated Circuit (RFIC) devices, large linear equations systems have to be solved. This paper presents the benefits of using Graphic Processing Unit (GPU) computations for solving such systems which are characterized by sparse complex matrices. CUSP is a GPU generic parallel algorithms library for sparse linear algebra and graph computations based on Compute Unified Device Architecture (CUDA). The code is calling iterative methods available in CUSP in order to solve those complex linear equation systems. The tests were performed on various Central Processing Units (CPU) and GPU hardware configurations. The results of these tests show that using GPU computations for solving the linear equations systems, the electromagnetic modeling process of RFIC devices can be accelerated and at the same time a high level of computation accuracy is maintained. Tests were carried out on matrices obtained for an integrated inductor designed for RFICs, and for Micro Stripe (MS) designed for Photonics Integrated Circuit (PIC).

Key words: GPGPU computing, iterative methods, Radio Frequency Integrated Circuits, Photonics Integrated Circuits

I. INTRODUCTION

Emergence of the CUDA technology and TESLA series devices in 2006 opened new perspectives for computational and engineering science. NVIDIA's Tesla architecture, first introduced in the GeForce 8800 GPU, unifies the vertex and pixel processors and extends them, enabling high-performance parallel computing applications written in the C language using CUDA [1]. TESLA is one of the first dedicated GPGPU. High computational power, multicore structure and relatively large Random Access Memory (RAM) make TESLA GPU cards an alternative for classical CPUs.

There are many applications that could not be run, because their requirements imply high execution time and high hardware resources like large quantity of memory, fast CPU or high speed interconnect.

Using the parallel computing provided by GPGPUs, along with distributed computing provided by clusters or supercomputers, very fast hybrid algorithms can be developed. The hybrid algorithm achieves an average speedup of 30% compared to a pure GPU algorithm thereby indicating the benefits of the hybrid approach [2]. This means that the development of classical computers into high performance computers, and sequential algorithms into parallel hybrid algorithms, gives a possibility to approach high complexity applications. A new concept is introduced as parallelism. The task of applying parallel or distributed computing refers to the parallelization of the sequential algorithm. In order to use parallel computing, the algorithm has to fulfill certain conditions. The main and the most important condition is that the operations must be independent. Therefore a preliminary analysis must be carried out in order to identify the step of the sequential algorithm that requires the use of the parallel computing.

Solutions based on the parallel computing are the answer for challenges facing the academic world, but also engineering problems from the industry area. Integrated circuits working in high frequency (HF) are used in many of today's products such as mobile computer networks, mobile devices, fiber networks, sensors, and wires computer networks. Applications of these devices are very broad, therefore the design process requires high specialization and special care. The operation frequencies, bandwidths and data capacities of communications systems are continuously increasing by employing advanced technologies and aggressive scaling of device dimensions [3]. However, the restrictions inherent in scaling make the design of RFICs a demanding task. Therefore the design, modeling or simulation process in the HF has to include electromagnetic (EM) field effects, but this leads to a more demanding computational power.

In this paper we present some techniques regarding the use of the GPU computing for acceleration of the electromagnetic modeling methods used in RFICs. noindent

II. ELECTROMAGNETIC MODELING METHODS (FIT and FEM)

The electromagnetic process presented in [4] has two important steps. The first one is the generation of the state space model, and the second one is the extraction of the reduced order model.

The presented method is using the Finite Integration Technique (FIT) which is the name of a class of numerical methods dedicated to find solutions of field problems in various physic areas based on spatial discretization. The name of the Finite Integration Technique (FIT) was proposed by Thomas Weiland from the Technical University of Darmstadt, Germany [5].

The process, in order to generate the state space model, consists of the continuous model description and continuous model discretization. Extraction of the continuous model is based on Maxwell equations for the electromagnetic elements. In order to discretize the continuous model the FIT method was used [5]. The discretized FIT model is assembled as a semi-state space model presented in Eqs.(1). Details on using the FIT in this concept are given in [6].

$$\begin{cases} C\dfrac{dx}{dt} + Gx(t) = Bu(t) \\ \qquad\qquad y(t) = Lx(t) \end{cases} \tag{1}$$

$$u = [v\ i]^T, \ \ x = [v_e\ v_m]^T$$

where $\boldsymbol{v}$ is the electric voltages, $\boldsymbol{i}$ the electric current, $\boldsymbol{v}_e$ the electric voltages defined on the electric grid and $\boldsymbol{v}_m$ is the the magnetic voltage (also called the magnetomotive force) defined on the magnetic grid [6]. The concept of magnetic voltage is applicable in quasi-magnetic fields when the rate of change of electric field normal to the subsystem energy boundary is negligible. Grids are necessary to solve the electromagnetic simulation problem, the structure of the device has to be spatially discretized by a grid division of the computational domain. The geometrical model mesh (grid) is based on a dual, orthogonal grid, one for the electric and one for the magnetic characteristics of the electromagnetic field. The matrices **C**, **G**, **B**, **L** are semi-state space matrices. The C and G are matrices with very sparse structures, and B and L are topological matrices with a sparse structure as well. The C matrix contains mainly different types of capacitances, and the G matrix – mainly different types of conductances, B is the matrix that relates the input vector u to the state x, and L is the matrix that links those inner states to the outputs y, more at [7]. The resulted model contains sparse, large, complex and unsymmetrical matrices (Fig. 5).

For a new test the state space model will be generated using the commercial software COMSOL (linked with MATLAB) based on the Finite Element Method (FEM). This model has a similar formulation with the previous one:

$$\begin{cases} M_C\dfrac{dx}{dt} = M_A x + M_B u \\ \qquad\quad y = Cx \end{cases} \tag{2}$$

where $\mathbf{M}_C$ is the mass matrix and $\mathbf{M}_A$ is the stiffness matrix [8]. The matrices contained by this model are still sparse and complex, but with improved conditioning due to the distribution of nonzero elements, which is diagonally dominant and close to a symmetrical matrix (Fig. 4).

The state space matrices obtained with the two methods are used as input data for the second step of the electromagnetic modeling process which is the extraction of the reduced order model. This step will be described in the next section.

III. APPLYING PARALLEL COMPUTING

The extraction of the reduced order model is done by using Adaptive Frequency Sampling by Vector Fitting (AFS-VF) [4,9]. This is an accurate method for modeling passive components of the high frequency integrated circuits.
The AFS-VF algorithm has steps of:

1. Generating the semi-state space model, in our case obtained with two different methods: by FIT and FEM.
2. Choosing an initial set S of sampling frequencies in the frequency range and flagging them as unmarked.
3. Choosing a set S' of test sampling frequencies, interleaved between the sampling frequencies and flagging them also as unmarked.

4. Computing transfer function $\mathbf{H}(S)$ and $\mathbf{H}(S')$ by solving the system

$$y/u = H(\omega) = L\,(G + j\omega C)^{-1}\,B \tag{3}$$

for all unmarked frequencies in the sampling set S and the test sampling set S'. Flag every entry in S and S' as marked when its associated system has been solved. If we introduce

$$\begin{aligned} A &= G + j\omega C \\ b &= B \end{aligned} \tag{4}$$

it means that the linear system of equations (LSE) has to be solved

$$A^{-1} \cdot b = x \tag{5}$$

Finally the transfer matrix will be computed as

$$H(\omega) = Lx \tag{6}$$

5. Computing the parameters using an interpolation answer frequency by impoving the rational approximation

$$H_{\mathrm{VF}} = \sum_{i=1}^{q} \frac{K_i}{j\omega - p_i} + K_\infty + j\omega K_0$$

called Vector Fitting (VF). The order q is successively increased until the fitting error described by the Frobenius norm $||\mathbf{H}(\omega_k) - \mathbf{H}_{VF}(\omega_k)||/||\mathbf{H}(\omega_k)||$ gets smaller than the imposed threshold ε_{VF}.

6. Computing $\mathbf{H}_{VF}(S')$using the parameters obtained in step 5 and the relative error ε_i for each test frequency $\omega_i \in S'$, where

$$\varepsilon_i = ||\mathbf{H}(\omega'_k) - \mathbf{H}_{VF}(\omega'_k)||/||\mathbf{H}(\omega'_k)||\,.$$

7. If $\varepsilon_i < \varepsilon_{\mathrm{AFS}}$ for every test frequency in S', then the algorithm will stop. Otherwise, move all the test frequencies from S' to S and interleave a new set of test frequencies in the new intervals thus created. Add the new test frequencies to S' and flag them as unmarked.

8. If no other stopping criteria are met, repeat from step 4. Otherwise stop.

The analysis revealed that the main bottleneck of this algorithm is Step 4 (Fig. 1) which involves the solving of several large, sparse and complex linear systems ($A \times x = b$). That is why parallel versions of this algorithm were proposed in [10].

Numerical results revealed another important bottleneck of those parallel versions. This bottleneck refers to the limitation of the solver used in step 4. In this case the UMFPack is used [11], a direct solver which requires a lot of memory for solving such large linear systems. That is why the iterative solvers have to be approached in order to test if it is possible to increase the size of the solving system which will be equivalent to a larger problem (finer discretization grid).

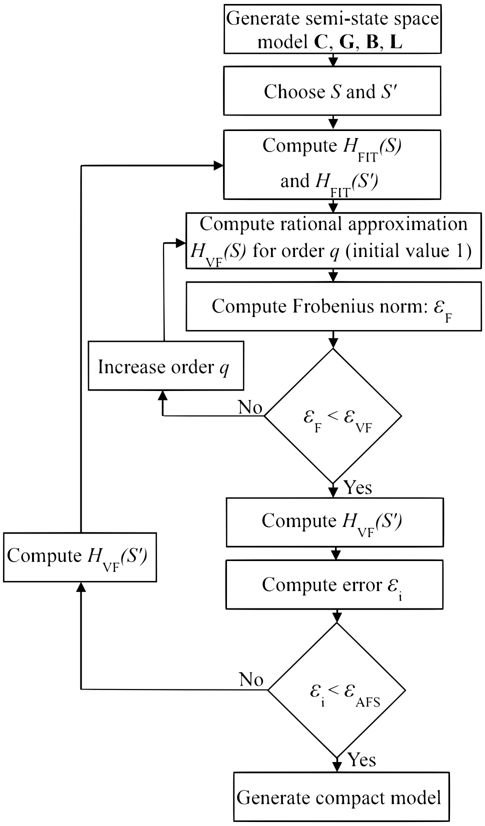

Generate semi-state space model **C**, **G**, **B**, **L**
Choose S and S'
Compute $H_{\mathrm{FIT}}(S)$ and $H_{\mathrm{FIT}}(S')$
Compute rational approximation $H_{\mathrm{VF}}(S)$ for order q (initial value 1)
Compute Frobenius norm: ε_{F}
Increase order q
No
$\varepsilon_{\mathrm{F}} < \varepsilon_{\mathrm{VF}}$
Yes
Compute $H_{\mathrm{VF}}(S')$
Compute $H_{\mathrm{VF}}(S')$
Compute error ε_{i}
No
$\varepsilon_{\mathrm{i}} < \varepsilon_{\mathrm{AFS}}$
Yes
Generate compact model

Fig. 1. AFS-VF algorithm

IV. ITERATIVE SOLVERS OVERVIEW

Iterative solvers can be slow, because they need a lot of iterations to find the solution. That is why the use of the parallel computing is reasonable to speed up the process of solution finding.

There are a lot of iterative methods, such as Conjugate-Gradient (CG), Biconjugate Gradient (BiCG), Biconjugate Gradient Stabilized (BiCGstab), Generalized Minimum Residual (GMRES) and others [12,13]. The above-mentioned algorithms of the iterative solvers revealed a lot of Basic Linear Algebra Subprograms Level 2 (BLAS 2) operations (matrix-vector operations). They in turn are involved for each iteration, and since the iterations are not independent (the main requirement of the parallel computing), the only solution to apply the parallel computing and to obtain a parallel iterative solver is to use a parallel version of BLAS 2 operation for sparse matrices.

One way to parallelize the matrix-vector operation is by using CUDA programming on the GPU devices [14]. Due to the fact that the A matrix of linear system ($A \times x = b$) is sparse and stored in different formats (COO, CSR, CSC, HYB [15]), the performance of the parallel solver depends on the efficiency of the algorithm used for matrix-vector operation, but also depends on the specifications of the GPU card (CUDA cores, Processor Clock, Memory Clock).

The aim of this work is to check the efficiency of using GPU computing along with iterative solvers. In order to achieve this goal, a CUDA library, called CUSP library [16], was used to access parallel iterative solvers based on the GPU computations.

If every node of a computer cluster is equipped with a GPU card, then the parallel approach of the AFS-VF algorithm [4] can be used with the proposed parallel iterative solvers. Unfortunately, our current cluster has only one GPU card so for further test only the performances of the iterative solvers will be presented.

V. CODE IMPLEMENTATION

Iterative solvers based on the GPU computing are available inside Matlab. Two mex files were written in order to be able to call real and complex solvers from the CUSP library and to exchange data between Matlab workspace and GPU card memory. The tree structure of the program is:

```
| -- cuda_programs

|   | -- iterative_solver.m

|   | -- matrix_conv

|   |    `-- csc2coo_matrix.m

|   `-- solvers

|    | -- solver_complex_mex.cu

|    `-- solver_real_mex.cu

| -- main_test_script.m

| -- make_install.m

`-- read_me_first.txt
```

- read_me_first.txt - file with install, uninstall and use instructions;
- make_install.m - the installation file;
- main_test_script.m – the starting file with settings of input data;
- solver_complex_mex.cu – the CUDA source file with complex solvers, compilable under Matlab;
- solver_real_mex.cu – the CUDA source file with real solvers, compilable under Matlab;
- csc2coo_matrix.m – a procedure that converts sparse matrices from CSC to the COO format, because CUSP solvers do not agree with the CSC format (only COO, CSR, HYB, etc);
- iterative_solver.m – a m-file which calls the convert procedure and selects the real or complex solver (depending the matrix that is received).

In the code four iterative methods provided in the CUSP-library were implemented. These four methods (GMRES, GMRES with preconditioner, BiCGstab, and CG) are placed in the code as functions, which can be called from the Matlab workspace. The only differences between these functions in the content of the code concern commands which are used to call the GMRES, preconditioned GMRES, CG or BiCGstab method. These commands in the CUDA programming language and CUSP-library are presented below:

```
// solve the system with the GMRES
cusp::krylov::gmres(A, x, b, restart,
   monitor);
// solve the system with the
   preconditioned GMRES
// set preconditioner (identity)
cusp::identity_operator<
   ValueType_complex, gpu_MemorySpace>
// solve
cusp::krylov::gmres(A, x, b,restart,
   monitor,M);
// solve the system with BiCGstab
cusp::krylov::bicgstab(A, x, b, monitor,
    M);
// solve the system with CG
cusp::krylov::cg(A, x, b, monitor, M);
```

Below is presented part of the code from the *solver_complex_mex.cu* file. This part is the function for the GMRES method with a preconditioner dedicated to solve the linear system with sparse and complex matrices (complex solver). The fragment of the code is the following:

```
 // GMRES with PRECONDITIONER complex
     solver
 /* solve system */

        // set stopping criteria:
        // iteration limit "no_it"
        // relative tolerance "rel_tol"
        // default_monitor - no
           information
        // verbose_monitor - display
           information

        cusp::default_monitor<
           ValueType_real> monitor(b,
           no_it, rel_tol);
        int restart = dimension;
        mexPrintf("restart␣=%
           dtextbackslash␣n",restart);
```

```
       // set preconditioner (identity)
       cusp::identity_operator<
          ValueType_complex,
          gpu_MemorySpace> M(A.
          num_rows, A.num_rows);

       // solve the linear system A*x=b
           with the GMRES
       cusp::krylov::gmres(A, x, b,
          restart, monitor,M);

       /* default_monitor */
       if (monitor.converged()){
        std::cout << "Solver␣converged␣
           to␣" << monitor.tolerance()
            << "␣tolerance";
        std::cout << "␣after␣" <<
           monitor.iteration_count()
           << "␣iterations";
        std::cout << "␣(" << monitor.
           residual_norm() << "␣final␣
           residual)" << std::endl;
       }
       else {
        std::cout << "Solver␣reached␣
           iteration␣limit␣" <<
           monitor.iteration_limit()
           << "␣before␣converging";
        std::cout << "␣to␣" << monitor.
           tolerance() << "␣tolerance␣
           ";
        std::cout << "␣(" << monitor.
           residual_norm() << "␣final␣
           residual)" << std::endl;
       }

/* save solution */
       //saving solution under *
          x_res_re and *x_res_im
          memory address
       for (i=0;i<dimension;i++){
        x_res_re[i] = x[i].real();
        x_res_im[i] = x[i].imag();
       }
```

VI. RESULTS

In order to validate the created code, which is dedicated to solve the linear systems of a state space model, the results of tests will be presented and discussed in this section. Two types of tests were carried out on ATLAS Cluster, serial computation (on CPU) and parallel computation (on GPU). ATLAS Cluster has the following hardware configuration:

- CPU – node Psi, 2 x Intel Xeon i7 CPUs running at 2.66 GHz with 8 MB of cache memory, for a total of 8 cores per node, and 24 GB of RAM memory;
- GPU – NVIDIA Tesla C1060, 240 cores, 1.296 GHz per core, 4GB of RAM memory.

Also serial and parallel computing tests were made on a classical PC containing a GPU card. PC has the following hardware configuration:

- CPU – Intel Core2 Duo E7600 @ 3.06GHz, 2GB of RAM memory.
- GPU – NVIDIA Tesla C2070, 448 cores, 1.150 GHz per core, 6GB of RAM memory.

The (FEM) Ushape inductor tests were executed on the ATLAS cluster and PIC (FIT) tests on a classical PC with a GPU card.

The state matrices used to test the parallel solvers are obtained with COMSOL (Finite Elements Method) for an inductor (Fig. 2) and with prototype software developed in LMN, called CHAMY (Finite Integration Technique) for MS PIC (Fig. 3) [17]. The inductor depicted in Fig. 2 is made from Al and is placed in the structure which has three layers: air, SiO2 and Si. The MS-type element from Fig. 3 has on the top metallization layer made from Au, and below there is a dielectric layer made from benzocyclobutene (BCB). Under BCB there is ground metalization made from Au, and below an InP substrate. Three tests (FEM1–Fig. 4a, FEM2–Fig. 4b, FEM3–Fig. 4c, PIC FIT1–Fig. 5a, PIC FIT2–Fig. 5b, PIC FIT3–Fig. 5c) were considered for each method.

Both tests (FEM and PIC FIT) consist of a solution of the LSE equation by using the created code. In the description of the AFS-VF algorithm it can be seen that the aim is to compute the frequency response, meaning that matrix A of system $(A \times x = b)$ is complex, so the tests will be executed only for the complex parallel solvers. Because the matrix A has the same sparsity for every frequency, Figs. 4/5 are not frequency dependent. After the execution of test $(A \times x = b)$, residual norm, $R = (\text{norm}(b - A \times x)/(\text{norm}(b)))$, and speedup, $S = T_{\text{sequential}}/T_{\text{parallel}}$, are computed.

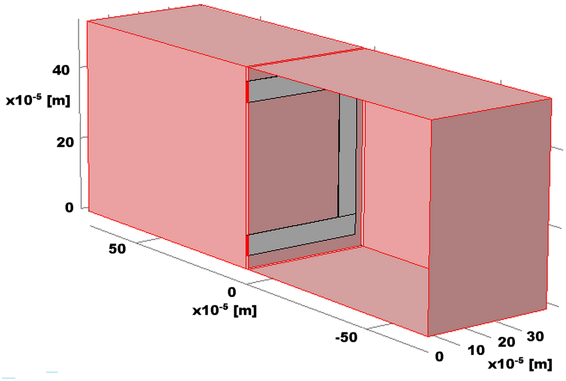


Fig. 2. Inductor problem, used for FEM tests

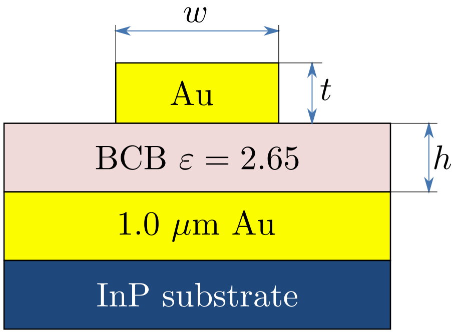
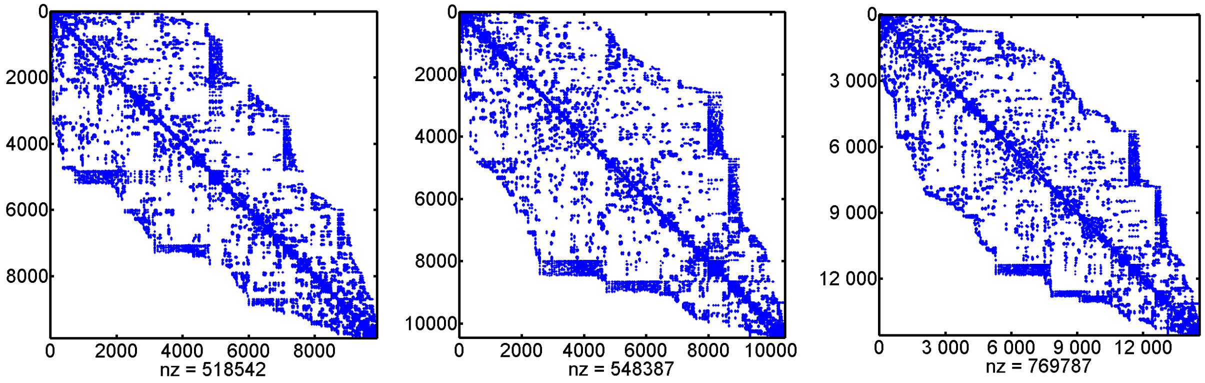


Fig. 3. Micro Stripe line PIC problem, used for tests, where w – width (of metallization), t – thickness (of metallization), h – height (of BCB dielectric) [18]

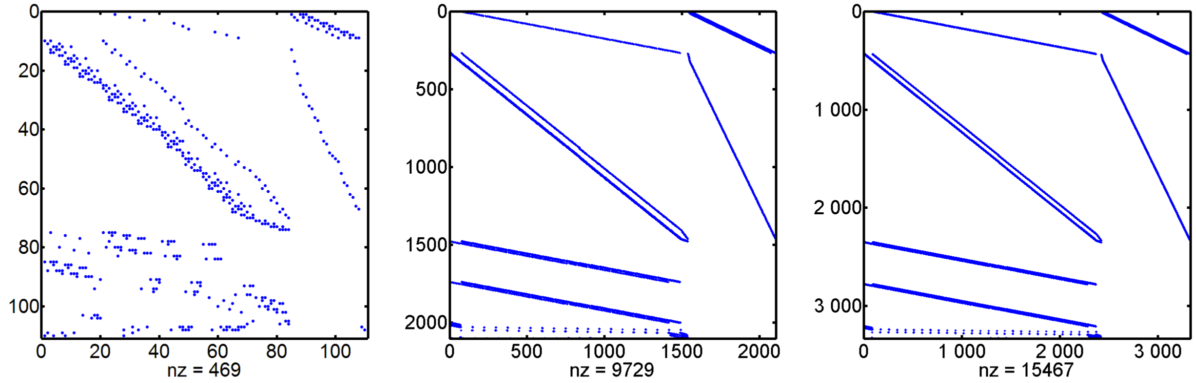
Regarding the FEM tests (Table 1), DoFs are Degrees of Freedom. In case of FIT discretization DoFs are physically measurable global quantities as voltages, fluxes, currents and charges on grid elements (edges, faces or cell volumes). In FEM tests, three meshes (grids) with 7196, 11 568 and 19 486 number of DoFs were used. The maximum number of iteration for parallel solvers was set hundreds of times bigger than the maximum number of iterations for sequential solvers. However, it can be seen that the parallel solvers obtained faster the solution for the linear systems. While executing the tests, it was also observed that the accuracy and performance of the parallel solvers from the CUSP library strongly depends on three factors: the required tolerance, the number of iterations and the restart value. Due to that fact, before using these solvers, some preliminary tests must be carried out in order to set the best values for the parameters mentioned above.

Another important issue for these tests is the quantity of memory needed, which was more than 4GB on CPU, and less than 4GB on GPU. Therefore, the parallel solvers and GPU card can be used also when the host computer does not have a big quantity of memory. Using an improved tolerance for parallel solvers, it is shown that a higher accuracy and a smaller execution time can be obtained in comparison with sequential solvers.



(a) FEM1 test (b) FEM2 test (c) FEM3 test

Fig. 4. Matrices A of FEM tests, (a) FEM1, (b) FEM2, (c) FEM3



(a) PIC FIT1 (b) PIC FIT2 (c) PIC FIT3

Fig. 5. Matrices A of PIC FIT tests, (a) PIC FIT1, (b) PIC FIT2, (c) PIC FIT3

Tab. 1. Numerical results for FEM tests

Problem		FEM1		FEM2		FEM3	
No. of DoFs		7196		11 568		19 486	
GMRES	Results	Time [s]	Norm	Time [s]	Norm	Time [s]	Norm
	Sequential restart $= 50$	5086	7.60 e$-$2	11860	7.40 e$-$2	26766	2.80 e$-$2
	Parallel restart $= 50$	62	9.10 e$-$2	161	4.00 e$-$2	452	3.80 e$-$2
	Speedup$_1$	82		73		59	
	Parallel$_2$ restart $= 5000$	3165	7.30 e$-$6	476	2.98 e$-$5	15875	3.73 e$-$6
	Speedup$_2$	1.6		24		1.7	

Tab. 2. Numerical results for PIC FIT tests

Problem		PIC FIT1		PIC FIT2		PIC FIT3	
No. of DoFs		110		2104		3332	
GMRES	Results	Time [s]	Norm	Time [s]	Norm	Time [s]	Norm
	Sequential	145	6.94 e$-$11	210	5.15 e$-$11	1459	3.70 e$-$11
	Parallel$_1$ restart $= 50$	11	0.99	2	0.99	5	1.00
	Speedup$_1$	13.63		116.90		316.92	
GMRES restart $= n$, $k = 1$ Parallel$_2$		16	0.71	68	0.28	269	0.25
Speedup$_2$		8.80		3.09		5.42	
GMRES restart $= 2n$, $k = 2$ Parallel$_3$		58	8.85 e$-$11	273	7.15 e$-$11	1084	5.91 e$-$11
Speedup$_3$		3.04		0.77		1.35	

Tab. 3. CPU memory usage for PIC FIT tests

Problem		PIC FIT1		PIC FIT2		PIC FIT3	
No. of DoFs		110		2104		3332	
GMRES	Results	Memory [MB]	Norm	Memory [MB]	Norm	Memory [MB]	Norm
	Sequential	0.19	6.94 e$-$11	298.11	5.15 e$-$11	2289.18	3.70 e$-$11
	Parallel restart $= 2n$	1.24	8.85 e$-$11	1.58	7.15 e$-$11	1.96	5.91 e$-$11
	Ms/Mp	0.16		188.90		1169.68	

Regarding PIC FIT tests (Table 2) due to matrix A of system $(A \times x = b)$ is very sparse, complex and ill-conditioned, calculations were more demanding than in case of FEM tests. To obtain rewarding results, regarding the computation time and also accuracy, preliminary tests were conducted. These tests consisted of checking different iterative solvers (GMRES, CG, BiCGstab) with and without preconditioners, as well as changing values of three factors: the required tolerance, the number of iterations and the restart number.

For comparison purposes the sequential computations were executed on CPU of the classical PC and parallel computations were made on the GPU Tesla C2070 card which was placed in this classical PC. Regarding sequential computations of PIC FIT for all tests the following factors were set: the relative tolerance rel_tol $= 1$ e-9, the number of iterations no_it $= k \times n$ (where n is the matrix dimension), the restart $= n$.

Preliminary tests of the parallel GPGPU computations revealed that the LSE $(A \times x = b)$ was successfully solved with the GMRES and the GMRES with an identity preconditioner iterative solver. The rest of iterative solvers available in the CUSP-library failed to calculate the x vector, which means that the LSE was not solved. Due to that fact, the GMRES method was used for further tests.

Table 2 depicts the results of the sequential computations, the parallel preliminary tests (with different values of factors) and final results obtained after setting proper values of the factors. Based on this table it has been observed that for the same values of the factors (rel_tol $= 1$ e-9, no_it $= k \times n$, $k = 1$, restart $= n$) which were set for the sequential and the parallel computations, the parallel computations were running much faster than the sequential one but the accuracy, defined as a value of the residual norm, was extremely wrong comparing to sequential accuracy. The perfect result for accuracy is when the residual norm equals zero. To improve the accuracy of the GPGPU computing, the number of iterations and the value of restart were doubled to no_it $= k \times n$, $k = 2$, restart $= 2n$. With such values of parameters the accuracy of the parallel computing in PIC FIT tests was just slightly worse than in the sequential case. Better accuracy affected longer computation time of GPGPU, which in case of PIC FIT2 test means longer computation time for the GPU than for the CPU. However, it can be observed that with an increase of the matrix dimension, there is increasing computation accuracy, and also for PIC FIT1 and PIC FIT3 tests the computation time is lower than in the sequential computations even though values of factors are twice bigger in the GPU than in the CPU computations. It is expected that, as in FEM tests also in FIT tests, higher performance of the GPU than in the CPU computations will be seen for bigger matrices. Supremacy of the GPU will increase with the increase of the matrix dimension.

Table 3 depicts CPU memory usage for PIC FIT tests. As it was expected during the parallel computations, the CPU memory usage is relatively small because mainly GPU memory is used. Due to that fact, the classical PC equipped with the GPU card is a useful machine to solve relatively large computation problems. Also, parallel solvers of the CUSP-library are able to solve the linear systems that require a bigger quantity of memory than the physical memory installed on the GPU card, because memory that these implementations are using to execute the dot product operation is the main RAM memory.

VII. CONCLUSIONS

The solver is one of the most important components for the modeling process, because the final results depend on its accuracy. That is why the tolerance for parallel solvers tested in this paper has to be adapted to the application in order to obtain precise results and improved execution time.

Conducted FEM and FIT tests revealed that the parallel computations with the use of the CUSP-library speed-up computations for RFIC and PIC problems. From iterative methods available in the CUSP-library, the best results were obtained for the GMRES iterative solver. For the biggest matrices in case of FEM tests the GPU achieved 1.7 times faster results of matrix calculation, and in case of FIT tests 1.35 times faster results of matrix calculation than during the sequential CPU computations.

By using GPU cards the acceleration of computation was achieved with the high level of accuracy. An important part presented in the paper is the created code based on the CUSP-library [10], which was prepared to be applied in solving problems of the RFIC and PIC design, modeling and computation. These problems reveal in the form of large, very sparse, complex and ill-conditioned matrices of the LSE $(A \times x = b)$. Without any doubts, the parallel computing results obtained with the code confirm usefulness and reasonableness of using GPU cards in design, modeling and simulation of the integrated circuits which are working in the HF.

References

[1] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, *NVIDIA Tesla: A Unified Graphics and Computing Architecture,* Micro, IEEE, 39-55 (2008).

[2] K.K. Matam, S.R.K. Bharadwaj and K. Kothapalli*, Sparse Matrix Matrix Multiplication on Hybrid CPU+GPU Platforms,* in Proc. of 19th Annual International Conference on High Performance Computing (HiPC), Pune, India, (2012).

[3] F. Ellinger, *Radio Frequency Integrated Circuits and Technologies,* Springer, 2nd edition 2008.

[4] I.-A. Lazar, G. Ciuprina and D. Ioan, *Effective extraction of accurate reduced order models for hf-ic using multi-CPU arhitectures*, Inverse Problems in Science and Engineering, 1-13 (2011).

[5] T. Weiland, *A discretisation method for the solution of Maxwell's equations for six-component fields,* International Journal of Electronics and Communication AEU 31 116-120 (1977).

[6] G. Ciuprina, D. Ioan, D. Mihalache, *Magnetic Hooks in the Finite Integration Technique: A Way Towards Domain Decomposition,* Proceedings of the IEEE CEFC, 2008.

[7] G. Ciuprina, D. Ioan, D. Mihalache and E. Seebacher, *Domain partitioning based parametric models for passive on-chip components,* Scientific Computing in Electrical Engineering, in the series Mathematics in Industry (J. Roos, L. Costa Eds), Vol. 14, pp. 37-44, Springer, 2010.

[8] *LiveLin for MATLAB User's Guide* COMSOL 2011.

[9] B. Gustavsen and A. Semlyen, *Rational approximation of frequency domain responses by vector fitting,* IEEE Trans. Power Delivery, 1052-1061 (1999).

[10] I.-A. Lazar, M.-I. Andrei, E. Caciulan, G. Ciuprina and D. Ioan, *Parallel algorithms for the efficient extraction of fitting based reduced order models,* Proceedings of the 7th International Symposium on ADVANCED TOPICS IN ELECTRICAL ENGINEERING, 1-13 (2011).

[11] T. Davis, *Algorithm 832: Umfpack, an unsymmetric-pattern multifrontal method,* ACM Transactions on Mathematical Software (TOMS), 196-199 (2004).

[12] Y. Saad, *Iterative Methods for Sparse Linear Systems,* Society for Industrial and Applied Mathematics, 2003. Second edition with corrections.

[13] Y. Saadand M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems,* SIAM J. Sci. Stat. Comput., 856-869 (1986).

[14] J. Sanders and E. Kandrot, *CUDA by Example: An Introduction to General-Purpose GPU Programming,* Addison-Wesley Professional; 1st edition 2010.

[15] D. B. Kirk and W. mei W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series),* Morgan Kaufman Elsevier, 2010.

[16] N. Bell and M. Garland, *Cusp: Generic parallel algorithms for sparse matrix and graph computations,* 2012. Version 0.3.0.

[17] H.H.J.M. Janssen, J. Niehof, W.H.A. Schilders, *Accurate Modeling of Complete Functional RF Blocks: CHAMELEON RF,* Scientific Computing in Electrical Engineering, Mathematics in Industry, 81-87 (2007).

[18] S. Kula, *Interconnect Elements Propagation Quantities in PIC,* Poznan University of Technology Academic Journals, Series Electrical Engineering, Iss. 70, 83-89 (2012).

Mihail-Iulian Andrei is a Research Assistant – Numerical Modeling Laboratory, Faculty of Electrical Engineering, University “Politehnica”of Bucharest, Romania. He received his MSc degree in Electrical Engineering in 2009, and currently he is finishing his PhD thesis entitled “Electromagnetic modeling of integrated inductors using multiprocessor systems”. His research interests are electromagnetic modeling, high performance computing and parallel computing.

Sebastian Kula is an Assistant Professor at the Kazimierz Wielki University in Bydgoszcz, Poland. He received his MSc Eng. degree in Electronics and Telecommunication in 2004. In 2010, he received his PhD Eng. degree in Electrical Engineering at the University “Politehnica” of Bucharest as part of the FP6 RTN Comson project; the thesis focused on reduced order models of high frequency integrated circuits. In 2011/2012 he was a postdoc at Chalmers University, Goteborg, Sweden. His research interests are in design of RFIC and PIC, parallel, GPU computing and electromagnetic modeling.

DOI:10.12921/cmst.2013.19.02.89-97