The Successive Over-Relaxation Method in Reconfigurable Hardware

Safaa J. Kasbah, Ramzi A. Haraty, and Issam W. Damaj

Abstract—**This paper presents the first hardware implementation of the Successive Over-Relaxation (SOR) method for the solution of a 2D Poisson equation. We use Handel-C, a high level language for the implementation of algorithms on hardware, to code and implement our design which we map onto high-performance Field Programmable Gate Arrays (FPGAs), such as, Virtex II Pro, Altera Stratix, and Spartan3L. We use the FPGA vendors' proprietary software to analyze the design implementation performance. Besides, we implement SOR using C++ and compare our timing results with the obtained C++ version results. Our findings prove that SOR in hardware outperforms SOR in software.**

Index Terms—**Hardware Design, High Performance Computing, Iterative Methods, Parallelization.**

I. Introduction

A large number of physical phenomena can be expressed as systems of linear equations. Numerical solutions for these equations allow us to glean valuable information about the system at hand. There are two basic approaches for solving linear systems: Direct Methods and Iterative Methods. In the first approach, a finite number of operations are performed to find the exact solution. In the second approach, an initial approximate of the solution is generated, then this initial guess is used to generate another approximate solution, which is more accurate than the previous one [13]. The robustness of applying iterative methods over direct methods is shown in different areas including: circuit analysis and design, weather forecasting and analyzing financial market trends.

The well known iterative methods are: Gauss-Seidel, Multigrid, Jacobi and *SOR* which is of a particular interest in this paper. *SOR* has been devised to accelerate the convergence of Gauss-Seidel and Jacobi, [4] by introducing a new parameter, ω, referred to as the relaxation factor. The *SOR* rate of convergence is highly dependent on the relaxation factor. The main difficulty of using *SOR* is finding a good estimate of the relaxation factor [13]. A number of techniques have been proposed for determining the exact value of ω which accelerates the rate of convergence of the method [4], [13].

All available iterative methods packages, including *SOR*, are done in software. Examples are the: *ITPACK 3A*, *ITPACK 3B*, *ITPACK 2C*, *ITPACK 2D*, and the *ELLPACK* package [2], [9]. Several sequential and parallel techniques were used in these packages to accelerate the method [14].

The emergence of the new computing paradigm, Reconfigurable Computing (*RC*), introduces novel techniques for accelerating certain classes of applications including signal processing (e.g., weather forecasting, seismic data processing, Magnetic Resonance Imaging (*MRI*), adaptive filters), cryptography and *DNA* matching[11]. *RC*-systems combine the flexibility offered by software and the performance offered by hardware [5]. It requires a reconfigurable hardware, such as an *FPGA*, and a software design environment that aids in the creation of configurations for the reconfigurable hardware [11].

In [7], we present the first hardware implementation of an iterative method, the Multigrid. The speedup achieved demonstrates that hardware design can be suited for such computational intensive applications. Toward proving the hypothesis that accelerated versions of the iterative methods can be realized in hardware, we undertook the first hardware implementation of the Successive Over-Relaxation method; using the same *FPGAs* that were used in [6]-[8].

In this paper, we study the feasibility of implementing *SOR* in reconfigurable hardware. We use *Handel-C*, a higher level design tool to code our design, which is analyzed, synthesized, and placed and routed using the *FPGAs* proprietary software (*DK Design Suite, Xilinx ISE 8.1i* and *Quartus II 5.1*). We target *Virtex II Pro, Altera Stratix* and *Spartan3L* which is embedded in the *RC10 FPGA*-based system from *Celoxica*. We report our timing results when targeting *Virtex II Pro* and compare them with

Manuscript received November 9, 2006. This work was supported by the Lebanese American University.

Safaa J. Kasbah is with the Division of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon (phone: 961-3-788402; email: safaa.kasbah@lau.edu.lb).

Ramzi A. Haraty is with the Division of Computer Science and Mathematics, Lebanese American University, Beirut, Lebanon (e-mail: rharaty@lau.edu.lb).

Issam W. Damaj is with the Department of Electrical and Computer Engineer, Dhofar University, Salalah, Sultanate of Oman (e-mail: i_damaj@du.edu.om).

a software version results written in *C++* and running on a General Purpose Processor.

The remainder of this paper is organized as follows: In Section 2, the successive over-relaxation method is explained. In Section 3 we study the feasibility of implementing *SOR* on reconfigurable hardware. We present a parallelized version of the algorithm using the '*par*' construct from *Handel-C*. The performance results are then analyzed and presented in Section 4. In Section 5, we conclude the paper and point out possible future work.

II. DESCRIPTION OF THE ALGORITHM

The successive over-relaxation method is an iterative method used for finding the solution of elliptic differential equations. *SOR* has been devised to accelerate the convergence of Gauss-Seidel and Jacobi [4], by introducing a new parameter, ω, referred to as the relaxation factor.

Given the linear system of equations:

$$A\phi = b$$

the matrix A can be written as

$$A = D + L + U$$

where D, U and L denote the diagonal, strictly upper triangular, and strictly lower triangular part of matrix A [13].

Using the successive over relaxation technique, the solution of the PDE is obtained using:

$$x^{(k)} = (D - \omega L)^{-1}[\omega U + (1-\omega)D]x^{(k-1)} + \omega(D - \omega L)^{-1}b \quad (1)$$

where x^k represents the k^{th} iterate.

The SOR rate of convergence strongly depends on the choice of the relaxation factor, ω [2]. Extensive work has been done on finding a good estimate of this factor in the [0, 2] interval [2] [9].

Recent studies have shown that for the case where:

- $\omega = 1$: *SOR* simplifies to Gauss-Seidel method [10].
- $\omega \leq 1$ or $\omega \geq 2$: *SOR* fails to converge[10].
- $\omega \succ 1$: *SOR* used to speedup convergence of a slow-converging process [13].
- $\omega \prec 1$: helps to establish convergence of diverging iterative process [13].

III. IMPLEMENTATION

The successive over-relaxation method was designed using *Handel-C*, a higher-level hardware design tool. *Handel-C* comes packaged with *DK Design Suite* from *Celoxica*. It allows the designer to focus more on the specification of the algorithm rather than adopting a structural approach to coding [3]. *Handel-C* syntax is similar to the *ANSI-C* with additional extensions for expressing parallelism [3]. One of the most important features in *Handel-C* which is used in our implementation is the '*par*' construct that allows statements in a block to be executed in parallel and in the same clock cycle.

Our design has been tested using the *Handel-C* simulator; afterwards, we have targeted a *Xilinx Virtex II Pro FPGA*, an *Altera Stratix FPGA*, and *Spartan3L* which is embedded in an *RC10 FPGA*-based system from *Celoxica*. We have used the proprietary software provided by the devices vendors' to synthesize, place and route, and analyze the design [1] [3] [12].

In Fig. 1 and Fig. 2, we present a parallel and a sequential version of *SOR*. In the first version, we used the '*par*' construct whenever it was possible to execute more than one instruction in parallel and in the same clock cycle without affecting the logic of the source code. The dots in the combined flowchart/concurrent process model which is shown in Fig. 1 represent replicated instances. Fig. 2 shows a traditional way of sequentially executing instructions on a general purpose processor. Executing instructions in parallel have shown a substantial improvement in the execution of the algorithm.

To handle floating point arithmetic operations which are essential in finding the solution to *PDE* using iterative methods, we used the Pipelined Floating Point Library provided by *Celoxica* [3]. However, an unresolved bug in the current version of the *DK* simulator limited the usage of the floating point operations to four in the design. The only possible way to avoid this failure was to convert/Unpack the floating point numbers to integers and perform integer arithmetic on the obtained unpacked numbers. Though it costs more logic to be generated, the integer operations on the unpacked floating point numbers have a minor effect on the total number of the design's clock cycles.

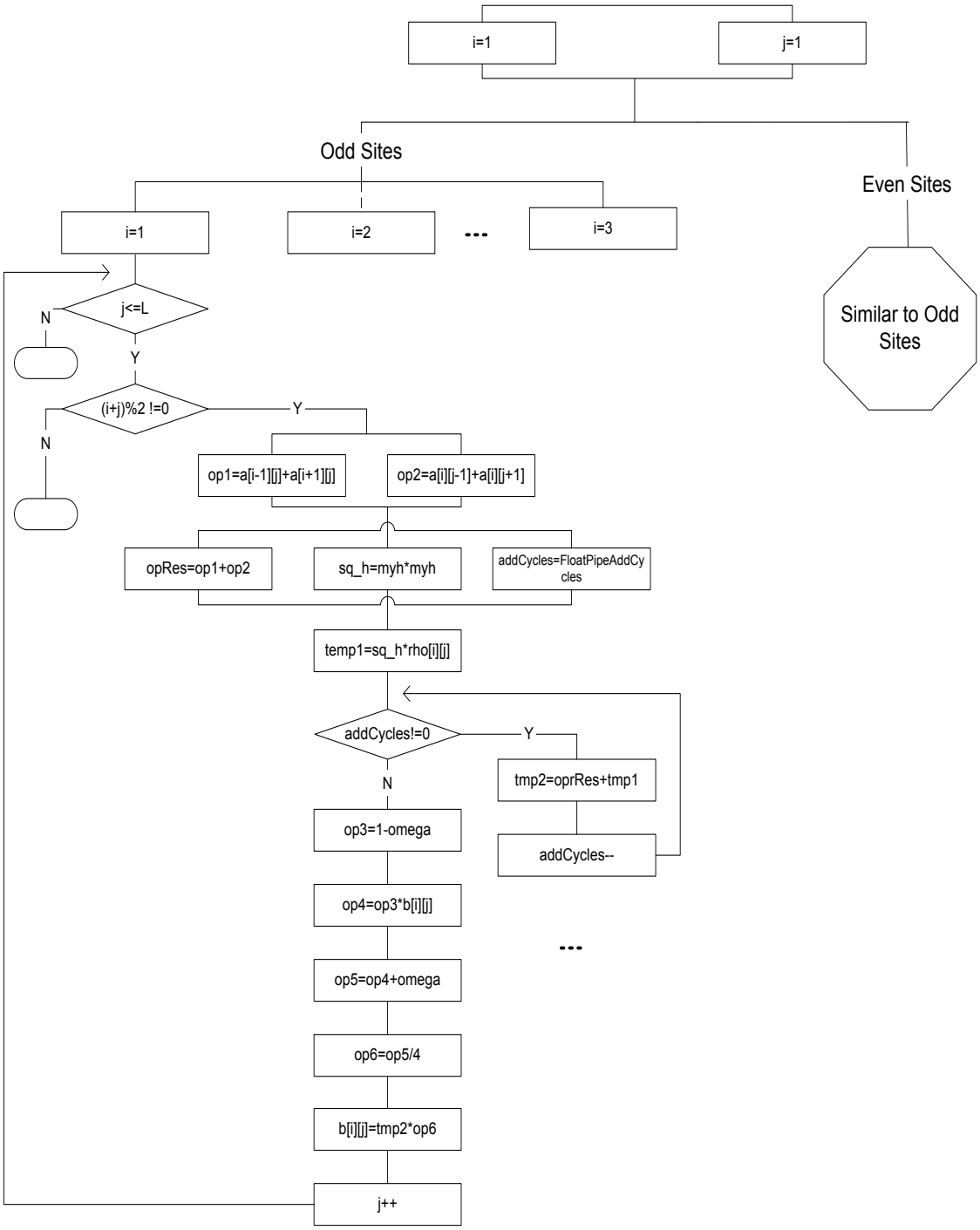

Fig. 1: SOR parallel version, showing the combined flowchart/concurrent process model. The dots represent replicated instancess.

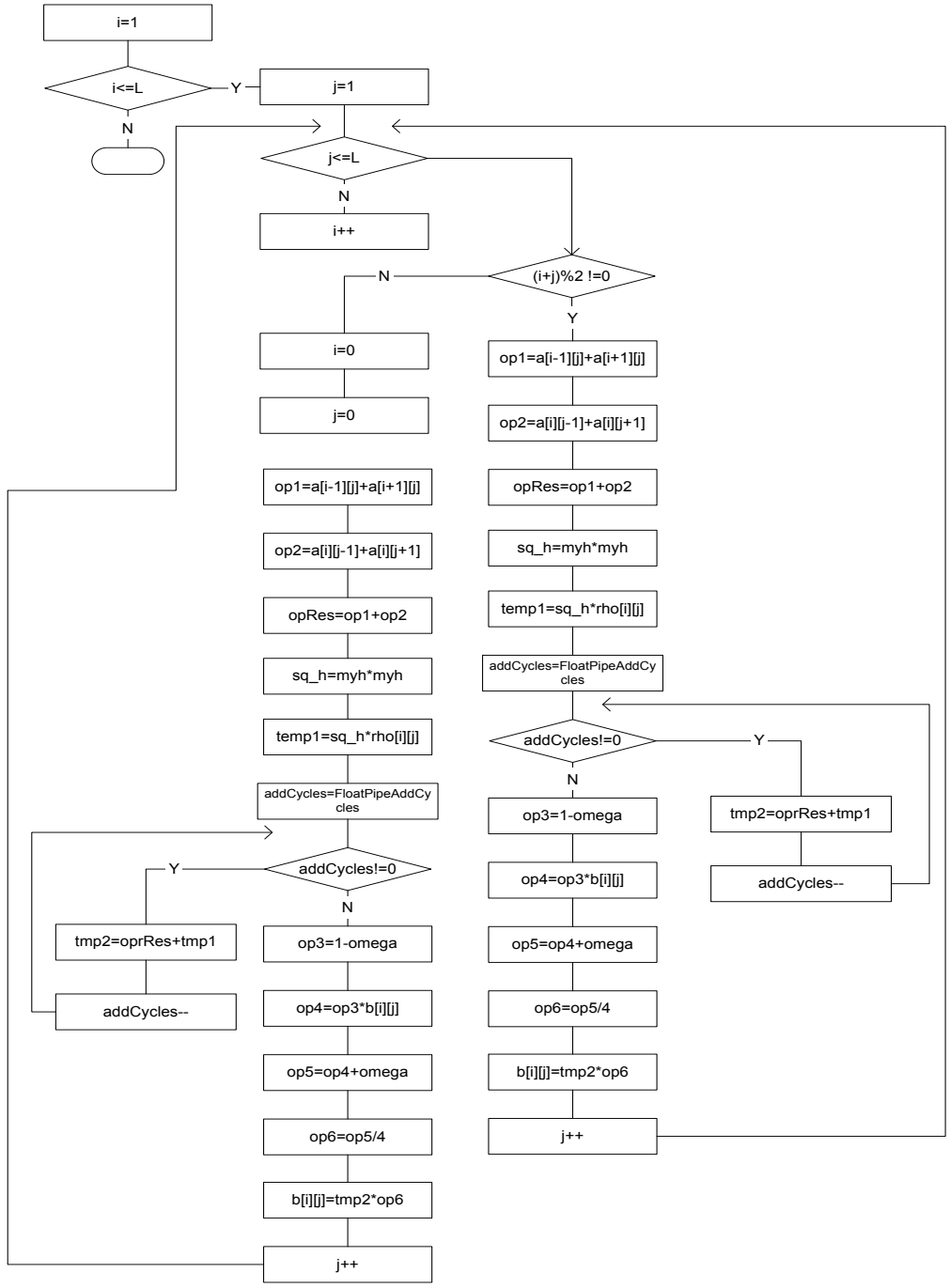

Fig. 2: SOR flowchart, sequential version

IV. EXPERIMENTAL RESULTS

As mentioned before, the main objective of this paper is: i) studying the feasibility of implementing *SOR* method in hardware and ii) realizing an accelerated version of the method.

The first objective is met by targeting high-performance *FPGAs*: *Virtex II Pro* (2vp7ff672-7), *Altera Stratix* (ep1s10f484c5), and *Spartan3L* (3s1500lfg320-4) which is embedded on *RC10* board from *Celoxica*. The second objective is met by comparing the timing results obtained, with a software version written in *C++* and compiled using *Microsoft Visual Studio .Net*.

All the test cases were carried out on a Pentium (M) processor 2.0GHz, 1.99GB of *RAM*. The relaxation factor ω was chosen to be 1.5 [10].

We use the *FPGA* vendor's tools to analyze and report the performance results of each *FPGA*. The synthesis results obtained, for different problem sizes, when targeting *Virtex II Pro, Altera Stratix, and Spartan3L* are reported in Tables 1, 2 and 3, respectively.

Table 1: Virtex II Pro Synthesis Results

Mesh Size	Occupied Slices	Total Equivalent Gate Count
8x8	128	2,918
16x16	136	3,033
32x32	219	4,807
64x64	265	5,978
128x128	315	7,125
256x256	610	14,538
512x512	1,098	23,012
1024x1024	1,601	31,848
2048x2848	2,289	53,476

Table 2: RC10 Spartan3L Synthesis Results

Mesh Size	Occupied Slices	Total Equivalent Gate Count
8x8	302	279,010
16x16	499	281,001
32x32	589	282,997
64x64	745	284,000
128x128	877	285,872
256x256	1,201	297,134
512x512	2,010	299,858

Table 3: Altera Stratix Synthesis Results

Mesh Size	Total Logic Elements	Logic Element usage by nb. of LUT inputs	Total Registers
8x8	519	250	120
16x16	601	310	155
32x32	810	501	199
64x64	999	637	280
128x128	1,274	720	347
256x256	1,510	890	948
512x512	2,286	1,087	501
1024x1024	2,901	1,450	569
2048x2848	3,286	1,798	640

Fig. 3 shows *SOR* execution time when targeting *Virtex II Pro FPGA* versus the execution time of *SOR* in *C++*. We started with a problem size of 8x8 and reached 2048x2048. Obviously, one can notice the acceleration of the method when moving from software implementation to hardware implementation. The speedup of the design, for different problem sizes, is shown in Table 4 and calculated as the ratio of Execution Time (*C++*) / Execution Time (*Handel-C*).

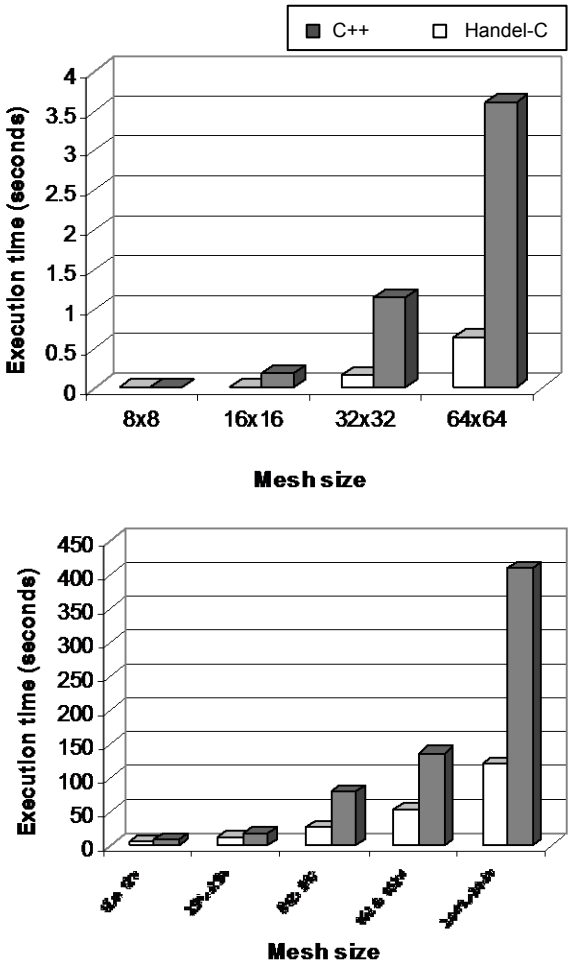

Fig. 3*: SOR execution time results in both versions, Handel-C and C++.*

Table 4: Design Speedup

Mesh Size	Speedup
8x8	1.76
16x16	188
32x32	6.71
64x64	5.70
128x128	1.51
256x256	1.49
512x512	3.03
1024x1024	2.58
2048x2848	3.38

V. CONCLUSION

In this paper, we have studied the feasibility of implementing the Successive Over- Relaxation (*SOR*) method on reconfigurable hardware. We used a hardware compiler, *Handel-C*, to code and implement our design which we map onto high-performance *FPGAs*: *Virtex II Pro*, *Altera Stratix*, and *Spartan3L* which is embedded in the *RC10* board from *Celoxica*. We used the *FPGAs* vendor's tool to analyze the performance of our hardware implementation. For testing purposes, we designed a software version of the algorithm and compiled it using *Microsoft Visual Studio .Net*.

The software implementation results were compared to the hardware implementation results. The synthesis results prove that *SOR* is suitable for *FPGA* implementation; the timing results prove that *SOR* on hardware outperforms *SOR* on *GPP*. In the near future, in addition to realizing other versions of *SOR*, Modified *SOR* (*MSOR*), Symmetric *SOR* (*SSOR*) and Unsymmetric *SOR* (*USOR*), on different classes of reconfigurable hardware, we plan to implement a third iterative method, Jacobi, on the same classes of *FPGAs* that were used in [8] and in this paper.

Once the three methods are realized on reconfigurable hardware, it will become possible to find the modeled system's solution, using the most suitable iterative method, at low cost.

REFERENCES

[1] Altera Inc., www.altera.com. 2006

[2] Bailey W., "The Successive Over Relaxation Algorithm and its application to Numerical Solutions of Elliptic Partial Differential Equations". B.Sc. Project, Dublin Institute of Technology. 2003.

[3] Celoxica, www.celoxica.com. 2006.

[4] Evans G., Blackledge J. and Yardley P., Numerical Methods for Partial Differential Equations. Springer-Verlag. London 2000.

[5] Compton K. and Hauck S. "Reconfigurable Computing: A Survey of Systems and Software". In ACM Computing Surveys, vol. 34, no. 2, pp. 171-210, June 2002.

[6] Kasbah S., Damaj I., "A hardware implementation of Multigrid Algorithms". Poster Session: 17th International Conference on Domain Decomposition Methods, Austria July 2006.

[7] Kasbah S., "Multigrid Solvers in Reconfigurable Hardware". Master Thesis, Division of Computer Science and Mathematics, Lebanese American University. 2006.

[8] Kasbah S., Damaj, I., and Haraty R., "Multigrid Solvers in Reconfigurable Hardware", Journal of Computational and Applied Mathematics., to be published.

[9] Kincaid D., Celebrating Fifty Years of David M. Young's Successive Overrelaxation Iterative Method. Numerical Mathematics and Advanced Applications, M. Feistauer, V. Dolejsi, P. Knobloch, K. Najzar (Eds.), Springer-Verlag, Berlin Heidelberg, pp. 549-558. , 2004.

[10] Kulsrud H. E., "A practical technique for the determination of the optimum relaxation factor of the successive over-relaxation method". communications of the ACM vol. 4 nb. 4. pp.184-187. 1961.

[11] Li Y., Callahan T., Darnell E., Harr R., Kurkure U., and Stockwood J., "Hardware-Software Co-Design of Embedded Reconfigurable Architectures," In 37th Design Automation Conference, Los Angeles, CA, pp. 507-512, 2000.

[12] Xilinx. www.xilinx.com. 2006.

[13] Young D., "Iterative Methods for Solving Partial Difference Equations of Elliptic Type", Ph.D. Thesis, Department of Mathematics, Harvard University, 1950.

[14] Zarka Cvetonivic, Edward G. Freedman, Charles Nofsinger, "Efficient Decomposition and Performance of Parallel PDE, FFT, Monte Carlo Simulations, Simplex, and Sparse Solvers". Proceedings of the 1990 ACM/IEEE conference on Super Computing. 455-464. 1990.