The Jacobi Method in Reconfigurable Hardware

Safaa J. Kasbah and Issam W. Damaj

***Abstract*—Linear equations provide useful tools for understanding the behavior of a wide variety of phenomena—from science and engineering to social sciences. A number of techniques have arisen to find the solution of these systems; examples are Jacobi, Gauss-Seidel, Successive Over Relaxation, and Multigrid. In this paper, we present an accelerated version of the Jacobi algorithm by implementing it on reconfigurable hardware devices- Field Programmable Gate Arrays such as *Virtex II Pro, Altera Stratix* and *Spartan3L*. The design presented is implemented using *Handel-C*, a hardware compiler. The implementation results obtained are compared with a software version results written in *C*++ and running on a general purpose processor. Final results illustrate that *Jacobi* on a reconfigurable hardware can outperform a software version of the same algorithm.**

***Index Terms*— *FPGA;* Hardware Design, Iterative Methods, Parallelization, *PDE*.**

I. Introduction

The problem of finding the solution of Partial Differential Equations (*PDE*s) plays a central role in modeling real world problems varying from physics, chemistry, economics, computer vision and engineering. The broad field of modeling real systems has drawn the researchers' attention for designing efficient algorithms for solving *PDEs* [6].Examples of such algorithms are: Gauss-Seidel, Multigrid, Successive Over relaxation and Jacobi.

The simplest iterative method for solving a linear system of equations, $A\Phi = b$, is the Jacobi method. It is relatively easy to understand and is usually considered as a starting point for understanding more useful but complicated iterative methods. The advantage of the Jacobi method over other iterative methods lies in its computation which can be, trivially, done in parallel. However, for large system of linear equations, it is not preferable to use the Jacobi technique since the other iterative methods have proved to be more powerful and converge faster than the Jacobi [2]. For this reason, all the effort has been reserved for implementing and designing accelerated versions of SOR, Multigrid and Gauss-Seidel, leaving few versions of Jacobi for solving problems of small sizes. In an attempt to benefit from the simplest iterative method (Jacobi) which is primarily used in education; we benefit from the technological advanced to accelerate the method.

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 [5]. Besides, the well known direct *PDE* solver, fast fourier transform, which is used in a number of signal and image processing algorithms, has been successfully accelerated using RC concepts [7]. RC-systems combine the flexibility offered by software and the performance offered by hardware [4]. 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 [5].

This paper provides the first hardware implementation of the Jacobi method for computing the solution to a *2D* Poisson equation. We use *Handel-C*, a hardware compiler, to code our design which is mapped into *Virtex II Pro*, *Altera Stratix* and *Spartan3L* Field Programmable Gate Arrays. The design is analyzed, synthesized and placed and routed using the *FPGAs'* propriety software. The results obtained demonstrate that Jacobi in hardware outperforms Jacobi in software.

The rest of the paper is organized as follows: Section 2 introduces the Jacobi method. Section 3 introduces our hardware implementation of the method. Section 4 gives the experimental results. The conclusion and the future work are drawn in Section 5.

II. Description of the Algorithm

The simplest iterative method for solving a linear system of is the Jacobi method. As it is the case with the other iterative methods (Gauss-Seidel, SOR and Multigrid), the Jacobi technique starts with an initial estimate for the true solution and at each step, the current approximate solution is used to produce a better approximation for the true solution. This iterates continues until the approximate solution is sufficiently close to the true solution. Unlike Gauss-Siedel and SOR strategies where the update of the $(i+1)^{th}$ element depends on the update of the i^{th} element, the Jacobi's strategy is updating all the elements at the same time [2].

Given the linear system of equations:

$A\Phi = b$

where A can be split into three matrices: the diagonal (D), an upper triangular (U) and a lower triangular (L); having D

Manuscript received March , 2007.

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).

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).

the diagonal part of A, $-U$ the upper part of A, and $-L$ the lower part of A. A can be written as: $A = D - L - U$
Therefore:
$(D - L - U)\Phi = b$ can be rewritten as:
$D\Phi = (L + U)\Phi + b$ and
$\Phi = D^{-1}(L + U)\Phi + D^{-1}b$

This leads to the iterative Jacobi technique:

$$\Phi^{(k)} = D^{-1}(L + U)\Phi^{(k-1)} + D^{-1}b \quad \text{where } k = 1,2,...$$

The convergence of the Jacobi technique is guaranteed if the matrix A is diagonally dominant; i.e., in every row of the matrix, the magnitude of the diagonal entry in that row is larger than the sum of the magnitude of all the other entries in that row [2].

III. Implementation

The Jacobi algorithm was designed and implemented using a higher level hardware design tool, *Handel-C*. Our choice of the tool was based on the language's features for rapid and efficient prototyping. *Handel-C* syntax is similar to *ANSI-C* with additional extensions for expressing parallelism, e.g. '*par*' construct [3]. The *Handel-C* compiler comes packaged with *Celoxica Design Suite*.

We tested our Jacobi design using the *Handel-C* simulator. We targeted available high-performance *FPGAs*: *Xilinx Virtex II Pro*, *Altera Stratix*, and *Spartan3L* which is hosted on an *RC10* board from *Celoxica*. We synthesized and placed and routed the design using the devices' vendors' tools [1] [3] [8].

Unlike Gauss-Seidel and SOR, in the Jacobi iteration, all elements are updated at the same time and the computation can be done in parallel. In our *Handel-C* design we used the 'par' construct whenever it was possible to execute more than one instruction in parallel and in the same clock cycle. The traditional l way of executing instructions on a *GPP* is shown in Figure 1, where a sequential version of the Jacobi algorithm was design to be compared with our hardware design of the algorithm. Figure 2 shows the combined flowchart/concurrent process model of our design. Estimating the number of clock cycles needed for each version, one can easily expect that the hardware version would outperform the software version.

It is well known that Floating point operations are far more complex than fixed point operations. They consume more area on *FPGA*, for this reason, *Handel-C* does not support floating point type. To handle floating point arithmetic operations in our design, we used the Pipelined Floating Point Library provided in the Platform Developer's Kit from Celoxica. The library contains: the Float type, macro procedures needed to perform arithmetic operations on floating point numbers, and macros for converting from integers to floating points and vice versa.

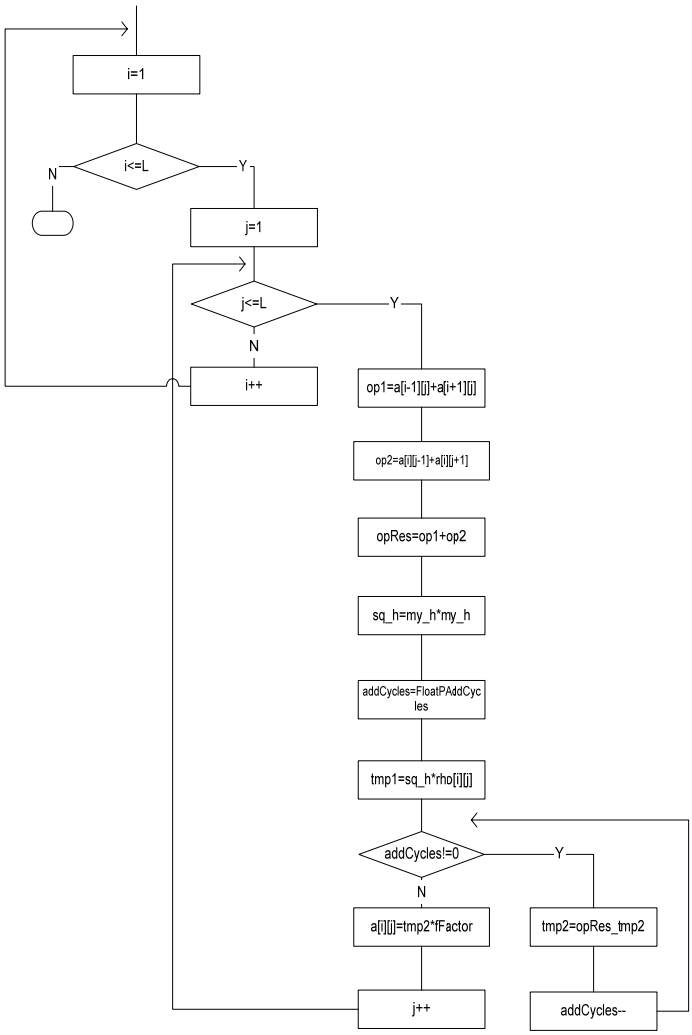

Fig. 1: Jacobi Flowchart, Sequential Version

Unfortunately, a failure in the current version of the *Handel-C* simulator persists whenever the number of floating point arithmetic operations exceeds four. The only possible way to avoid the bug in the simulator was to convert/unpack the floating point numbers to integers and perform integer arithmetic on the 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.

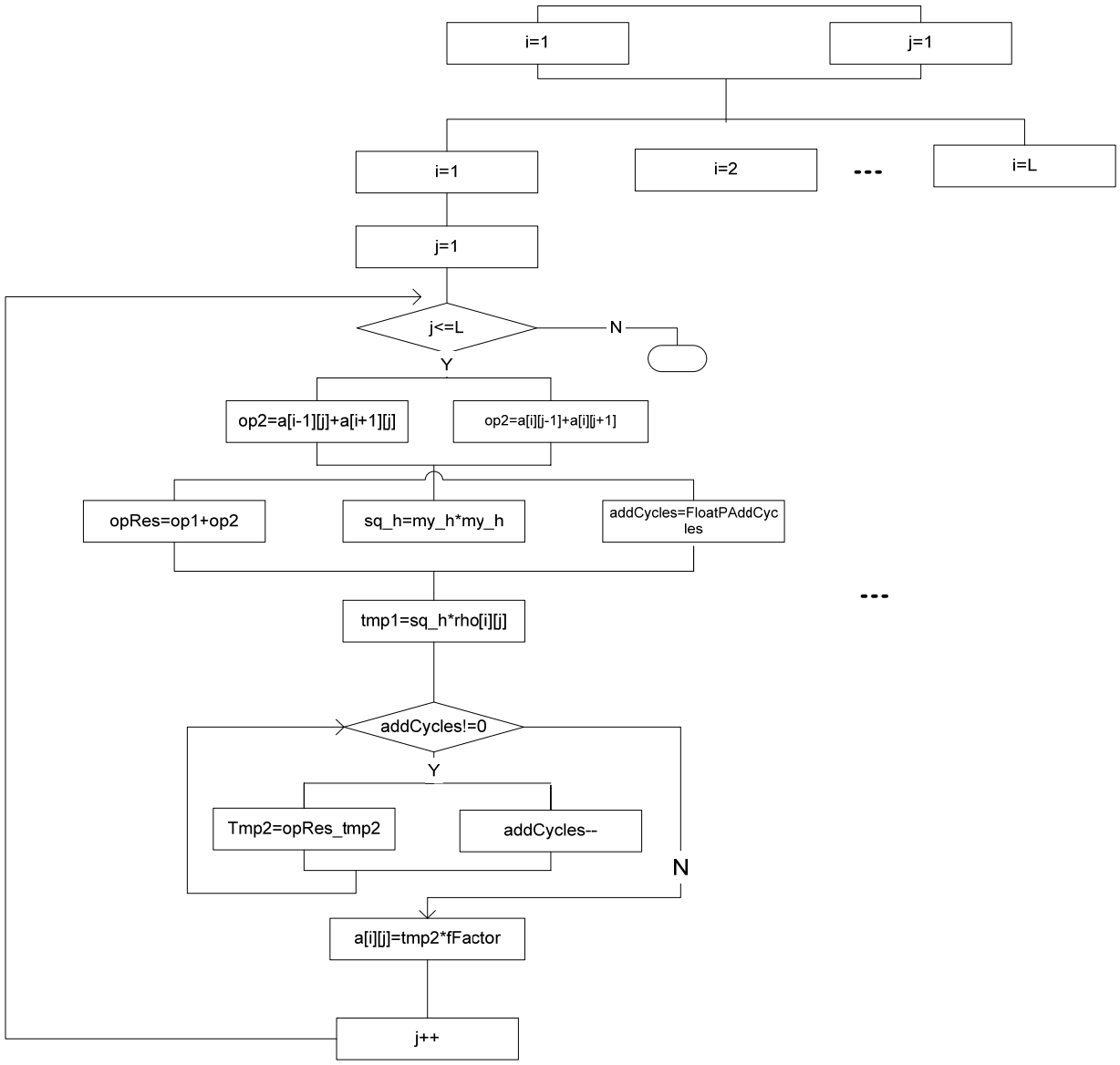

Fig. 2: : Jacobi parallel version, showing the combined flowchart/concurrent process model. The dots represent replicated instances.

"Magnetization ($A \cdot m^{-1}$)," not just "A/m." Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)," not "Temperature/K."

Multipliers can be especially confusing. Write "Magnetization (kA/m)" or "Magnetization (10^3 A/m)." Do not write "Magnetization (A/m) × 1000" because the reader would not know whether the top axis label in Fig. 1 meant 16000 A/m or 0.016 A/m. Figure labels should be legible, approximately 8 to 12 point type.

IV. Experimental Results

The hardware implementation results were obtained using the *Handel-C* simulator and the *FPGA* vendor's tools (*DK Design Suite*, *Xilinx ISE 8.1i* and *Quartus II 5.1*). The software version was written in C++, compiled using *Microsoft Visual Studio .Net* and running on a Pentium (M) processor 2.0 GHz, 1.99 GB of *RAM*.

We targeted *Xilinx Virtex II Pro* (*2vp7ff672-1*) *FPGA*, simulated our design and calculated the execution time using clock cycles of the design divided by the frequency at which the design operates at.

The execution time of Jacobi in hardware (*Handel-C*) and in software (*C++*) is shown in Figure 3 for different problem sizes. As the Figure shows, a significant improvement in the execution time of the hardware implementation of the algorithm over the software implementation. This acceleration is directly presented in Table 1.

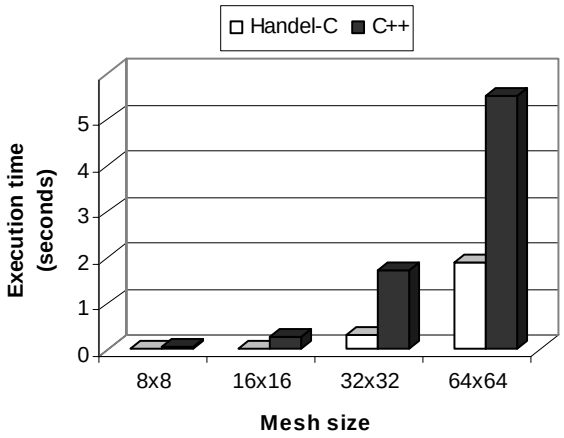

Fig. 3(a): Jacobi execution time results in both versions *Handel-C* and *C++*

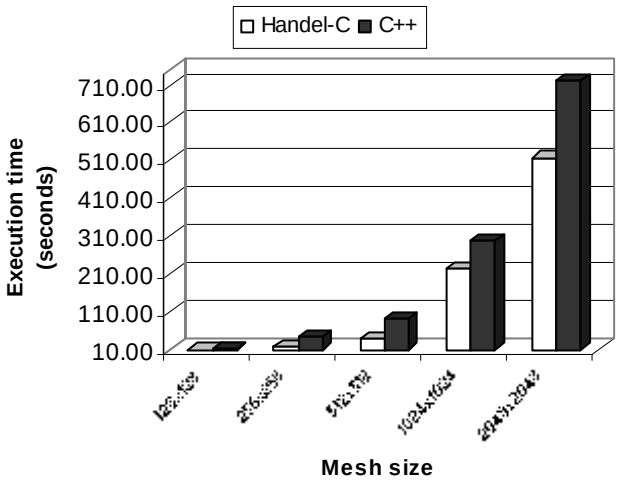

Fig. 4(b): Jacobi execution time results in both versions *Handel-C* and *C++*

Table 1: Design Speedup: Execution Time (C++) / Execution Time (Handel-C)

Mesh Size	Speedup
8x8	223.8095
16x16	56.2
32x32	5.676667
64x64	2.89267
128x128	1.409887
256x256	2.287887
512x512	2.386364
1024x1024	0.753898
2024x2024	1.391351

The Handel-C code was synthesized for *Xilinx Virtex II Pro* (*2vp7ff672-1*), *Altera Stratix* (*ep1s10f484C5*), and *Spartan3L* (*3s15001fg320-4*) which is embedded on *RC10* board from *Celoxica.* Tables 2, 3 and 4 report the obtained synthesis results.

Table 2: Virtex II Pro Synthesis Results

Mesh Size	Number of Occupied Slices	Total equivalent gate count
8x8	146	3,229
16x16	159	3,397
32x32	299	5,090
64x64	380	7,849
128x128	499	11,897
256x256	839	17,864
512x512	1286	23,649
1024x1024	1890	31,327
2048x2048	3198	35,839

Table 3: RC10 Spartan3L Synthesis Results

Mesh Size	Number of Occupied Slices	Total equivalent gate count
8x8	416	356,109
16x16	599	357,631
32x32	7326	359,989
64x64	9010	342,768
128x128	1198	389,999
256x256	1665	397,987
512x512	2810	498,030

Table 4: Altera Stratix Synthesis Results

Mesh Size	Total Logic Elements	Logic element usage by number of LUT inputs	Total Registers
8x8	610	354	189
16x16	709	401	232
32x32	880	556	300
64x64	1001	681	385
128x128	1286	801	390
256x256	1590	950	476
512x512	2589	1,101	560
1024x1024	3342	1,499	689
2048x2048	3927	1,941	819

V. Conclusion

In this paper, we presented a hardware implementation of the Jacobi algorithm using a hardware compiler, *Handel-C*, and a reconfigurable hardware, *FPGA*. The design was mapped onto high-performance FPGAs: *Virtex II Pro*, *Altera Stratix* and *Spartan3L* which is embedded in the RC10 board from *Celoxica*. The *FPGAs* vendors' proprietary software were used to analyze the performance of our hardware implementation. Our findings were compared to a software version written in *C++*, compiled using *Microsoft Visual Studio .Net* and running on a general purpose processor. The implementation results prove that *Jacobi* on hardware outperforms *Jacobi* on *GPP*.

REFERENCES

[1] Altera Inc., www.altera.com. 2006.
[2] Bertsekas D. P. and Tsitsiklis J. N., "Some aspects of parallel and distributed iterative algorithms -- a survey," Automatica, vol. 27, no. 1, pp. 3--21, 1991.
[3] Celoxica, www.celoxica.com. 2006.
[4] 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.
[5] 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.
[6] Osher, S. and Fedkiw, R. Level set methods: An overview and some recent results. Tech. Rep. 00-08, UCLA Center for Applied Mathematics, Department of Mathematics,University of California, Los Angeles, 2000.
[7] Uzun, I. S., Amira, A. and Bouridane, A. “FPGA implementations of fast Fourier transforms for real-time signal and image processing”. IEE Proceedings - Vision, Image, and Signal Processing. vol. 152, no. 3, pp. 283-296, 2005.
[8] Xilinx. www.xilinx.com. 2006.
[9] Young D., "Iterative Methods for Solving Partial Difference Equations of Elliptic Type", Ph.D. Thesis, Department of Mathematics, Harvard University, 1950.