Study of a Parallel Algorithm on Pipelined Computation of the Finite Difference Schemes on FPGA

Wenshi Wang, Zhangqin Huang, Shuo Zhang
Beijing Advanced Innovation Center for Future Internet Technology
Beijing Engineering Research Center for IoT Software and Systems
Beijing University of Technology
Beijing, China
zhuang@bjut.edu.cn

***Abstract*—With the rapid increase of data, stream computation contributes to the improvement of data real time processing. One of characters of stream computing is in the pipeline to exploit parallelism in the MIMD way. Moreover, pipelined parallel implementation is a common problem and often yields much better parallel efficiency with respect to other commonly used methods. Plentiful physical modeling in the field of image processing and machine vision deals with the solution of partial differential equations, typical representative by Poisson's equation. The finite difference schemes as the main method for the solution of partial difference equation for physical modeling can be time-consuming and computationally expensive. Constructing highly parallel computation models can greatly solve the problem and it is important that quickly and efficiently carry out it. The paper mainly studies the parallel implementation of the mix of Jacobi iterative and Gauss-Seidel method for the finite difference schemes to solve Poisson equations in a pipelined fashion on FPGA.**

Keywords—parallel computation; pipelined computation; finite difference scheme; FPGA

I. Introduction

The Poisson equation has wide application throughout image process, photography, electromagnetic and the mechanical engineering and so on. Processing the solution of Poisson's equation as an inhomogeneous partial difference equation is complex, even sometimes can't solve the exact solutions. With the development of the computer technologies, modern numerical method that discretize the succession problem and translate it into limited form of linear algebraic equations mainly includes the finite difference schemes and the finite element method. The finite difference method plays an important part of solving partial difference equations in a variety of practical problem and commonly produce large-scale linear algebraic equation [1]. In practice, the effectively method of solving the numerical solution of large-scale algebraic equations adopt the direct or iterative methods, such as Jacobi method or Gauss-Seidel method [2]. The Poisson equation is an N dimensional elliptic partial differential equation having N space dimensions. It describes displacement on a membrane in 2+1-D and sound propagation in spaces in 3+1-D form. Finite difference schemes transform the partial differential equation into a difference equation by discretizing space on an N dimensional grid. Many important scientific calculation involve large computation time, some are greater than a human lifespan. For accurate and stable approximations, these schemes employ high sampling rates resulting in high computation. Parallelism offers to make these previously impractical solution techniques feasible [3], but commonly they are expensive. At present, Field programmable gate arrays (FPGAs) and GPU appear to provide a platform to derive the architecture to best match the computational requirements[4,5]. Constructing parallel model of recursion formula on pipelined fashion often gain access to considerable speedup, especially that suitable to in the serial programs [6].

In this paper, the particular focus is to realize the mixed method of Jacobi iterative and Gauss-Seidel method for the finite difference scheme solving the 2D Poisson equation, and we emphasize on the idea of doing the computations in the pipelined fashion. The paper is organized as follows. Background on finite difference schemes is given in section II and analysis of algorithm and parallel method is presented in Section III to solve the Poisson equation. Section IV presents the idea of pipelined method to construct parallel algorithm of combination of Jacobi iterative and Gauss-Seidel method to solve linear algebra system. Section V describes an idea of design parallelism in an algorithm and summarizes the FPGA implementation. Section VI gives an analysis of experiment result of the simulation example on the Xilinx Artix-7 FPGA.

II. Background on Finite Difference Schemes

Numerous FD schemes exist, which use discretization of space on a structured rectangular grid for the solution of 2D Poisson equation given below, with certain boundary conditions,

$$\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}=f\left(x,y\right),\ \left(x,y\right)\in\Omega \tag{1}$$

where boundary conditions

The work was supported by the Beijing Municipal Natural Science Foundation (No.4122010) and Beijing innovation platform of Science and technology.

$$u(x,y)=g(x,y),\ (x,y)\in\Omega$$

When discretizing the problem on the rectangular grid Ω, with grid spacing h_1, h_2 in x, y directions respectively, Equation (1) can be approximated at any point (x_i, y_j) in many ways by $x_i = x_0 + ih_1$, $y_j = y_0 + jh_2$, for all $i, j = 0, \pm 1\cdots$ is used. Assuming (x_i, y_j) are regular interior point, (1) is solved by the finite difference approximation that is the standard five-point formula as

$$-\left[\frac{u_{i+1,j}-2u_{i,j}+u_{i-1,j}}{h_1^2}+\frac{u_{i,j+1}-2u_{i,j}+u_{i,j-1}}{h_2^2}\right]=f_{ij} \quad (2)$$

where $u_{i,j}$ is an approximation to exact solution $u(x_i, y_j)$ at the grid point (x_i, y_j) and $f_{i,j} = f(x_i, y_j)$. For the particular case, where h_1=h_2=h ,(2)reduces to the following simple formula.

$$4u_{i,j}-\left(u_{i+1,j}+u_{i-1,j}+u_{i,j+1}+u_{i,j-1}\right)=h^2 f_{ij} \quad (3)$$

The finite difference equations shown at above have a local truncation error of order $O(h^2)$ and to each of internal mesh points will result in a large and sparse linear system,

$$Au=b \quad (4)$$

where A and b are a square nonsingular matrix and a column matrix, respectively, while the u is column matrix shows the solution. The solution of (4) can be obtained by direct or iterative methods. Since the equation is large and sparse, particularly iterative method is suitable to solve the problem.

III. Analysis of Parallel Method

The solution of (3) needs four consecutive points and Fig.1 shows the graphical representation of the five point finite difference scheme in (3) [7]. It can be shown that for the finite difference scheme corresponding to a rectangular mesh, the error depend on the number of mesh and also associated with difference of two successive values in the iterative process. This can be reduced by increasing the density of the grid points and interpolation will provide reduction of error at the cost of increased number of operations per grid points [8].

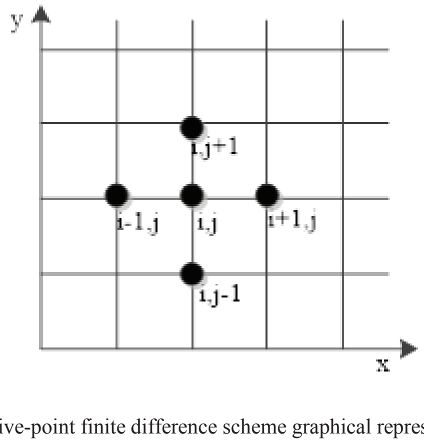

Fig. 1. Five-point finite difference scheme graphical representation.

The computation cost of finite difference scheme result from the number of operations per grid point and the size of the grid. For the finite difference scheme in equation (3), at least 4 operations included 1multiplication, 3 additions are needed to update for a grid point, moreover, additional computation can be increased by the other boundary conditions and increased number of the mesh to require higher accuracy. The finite difference schemes are suitable to parallel operation as the different data are being implemented by the same arithmetic.

Similarly to the idea of image data blocking, we commonly regard the compute process of a grid point as an atomic operation. Based on this reason, adopting parallel computation that process simultaneously grid points to cut down computing time and increase computational efficiency [9]. In fact, we can divide the data points into a quantity of sections in the light of the idea of blocking because the number of processor is certainly much less than the scale of grid points that use to compute and then distributed these different tasks to various process units in order to each node can compute independently allocated data points.

The parallelism strategy which involves partitioning the domain into subdomains is domain decomposition method [10] as shown in Fig.2 for a 2-D rectangular grid. Each subdomain are mapped onto the process element in mesh connectivity. According to the FD scheme formulation, values on the subdomain boundaries have to be transferred between the neighbouring sub-domains in each iteration period because updating the value of a grid point requires the values of the neighbouring grid points. Therefore, this communication locality needs to be exploited by the block partitioning method.

The solution of equation (1) can be obtained by iterative method, especially Jacobi iterative and Gauss-Seidel iterative. Gauss-Seidel iterative method makes the key-dependent new value to take place of previous values and sufficiently make use of new values to increase the speed of iterative computation. Although have more advantage than Jacobi iterative in the serial algorithm, Gauss-Seidel iterative consume more communication times in the parallel process. In fact, the neighbouring progress must theoretically send and

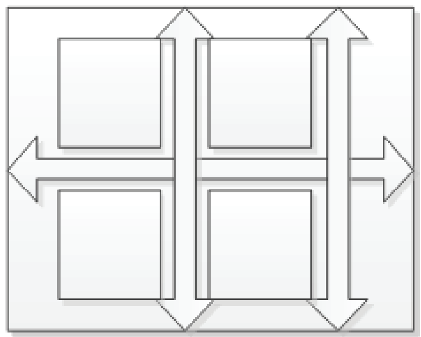

Fig. 2. Domain decomposition strategy.

receive the latest data value from computation each other after work out a new value of boundary point between the neighbouring subdomain and broadcast the derived new value to the next computation process. This feature make the parallel computation process cost greater communication overhead. But Jacobi iterative method have the feature that reserve the latest computation value until all the latest value work out and then process the next iterative computation. Consequently the speed of the iterative computation is relatively slow and its serial efficiency less than Gauss-Seidel iterative in the serial computation process. However the process of Gauss-Seidel parallel iterative just must broadcast the latest value when the grid point belong to boundary and the number of boundary point in a process of iterative determine the number of the latest value need to broadcast. Therefore, the efficiency of iterative in the Gauss-Seidel parallel process is undesirability and slower than Jacobi iterative [11]. To solve this problem, we firstly use precise solution to initialize data point on the boundary and adopt the first boundary condition. Secondly, the iterative strategy is that use Jacobi iterative method for the subdomain boundary point because the process that send data to inner point does not occupy communication time in order to reduce the total computing time. The algorithm diagram is outlined in Fig3.

IV. Pipelined Method

We use the idea of pipelined method to construct parallel algorithm of hybrid of Jacobi iterative and Gauss-Seidel method to solve linear algebra system. The computation is pipelined often yields much better parallel efficiency and speedup compared to other commonly used methods by properly regulate the time of computation and communication when simultaneously compute numbers of grid points [12]. The other advantage of pipelined method simplified the process of program implementation, especially suitable to get parallel program by modifying serial program.

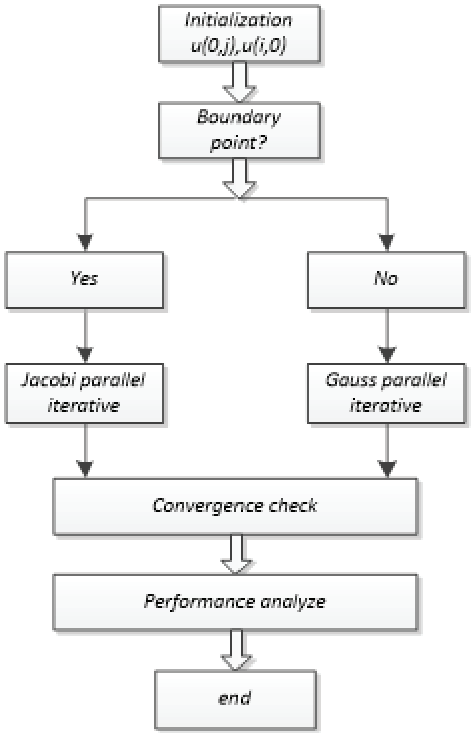

Fig. 3. Parallel algorithm diagram.

Without loss of generality, a set of recurrences are considered as follow.

$$a(i,j) = F(a(i-1,j)),\ i = 1,2,\cdots,n,\ j = 1,2,\cdots,m$$

Data partitioning is processing only along i direction and $a(i,j)$ are divided into p sections, namely $\{a(i,j), i = n(k),\cdots,n(k+1)-1\}$ are distributed to p process units, where $1 = n(0) < n(1) < \cdots < n(p) = n+1$. Let k be the processor number. Then, the process of computation is pipelined are given by the program as follow.

```
For j=1, m
    if process unit {p(k), k= 0,1,⋯,p-1} receive {a(n(k)-1, j)} from p(k-1)
    for i= n(k), n(k+1)-1
    compute a(i, j) :=F(a(i-1, j))
    end
    if process unit p(p-1) send {a(n(k)-1, j)} to p(k+1)
    end
```

The above pipelined method has disadvantage that communication are fine-grained and need many times, however, the solution is only to slightly modify the sequence of recurrent, namely that divided compute grid points into some sections and each process unit simultaneously compute a set of points each time. For example, we give Jacobi iteration algorithm on pipelined fashion as follows.

```
For j(0)=1+hb, h=0,1,…,m/b
    % b is the size of block along j direction
    j(1)=min(n, j(0)+b-1)
    if process unit {p(k), k= 0,1,⋯,p-1} receive {a(n(k)-1, j), j= j(0),⋯,j(1)} from p(k-1)
    for j=j(0), j(1)
    for i= n(k), n(k+1)-1
    compute u(i, j)=0.25*(f(i, j)+ u(i, j-1) + u(i-1, j) + u(i, j+1) + u(i+1, j))
    end
    end
    if process unit p(p-1) send {a(n(k)-1, j), j= j(0),⋯,j(1)} to p(k+1)
    end
```

Obviously, we can conclude that the pipeline length of the algorithm is [m/b] multiply the total times of iterative and the algorithm can derive expect efficiency by properly value of b. When data partitioning is simultaneously processing in two space directions, the feasibility will be remained unless change the cycle range of the second direction into the range on its corresponding processor and increase communication times in the second direction. Compute program will form two pipelines that are constructed by process units from i direction and j direction respectively, moreover, the pipeline length equal to that grid points of each direction divide by size of groups and multiply the total times of iterative.

V. Data Flow Representation and FPGA Implementation

Dataflow process networks that each process consists of repeated "firings" of a dataflow "actor" are a model of computation where writes to the unidirectional FIFO channel are non-blocking and reads are blocking. For data parallel algorithms, dataflow block diagrams representations can completely design parallelism in an algorithm and offer specification that has not related of the hardware. FDs as a algorithms that have locality of compute communication use asynchronous message passing mechanism in data flow networks. In dataflow process network, an algorithm is specified by a directed graph where the nodes represent computations and the arcs represent streams of tokens. The process is typically hierarchical design, in that a node in a diagram may represent different directed graph and also allows the use of a visual syntax to specify the algorithms, which simplifies the design process. The nodes in the graph can be either language primitives or other language, such as C or FORTRAN.

Each process element in the FPGA operates a grid point in the mesh according to the data flow graph and each process element will send and receive signals for the transfer of data values as tokens. Fig.4 shows the process element in block form and the hardware structure inside. The process element includes 4 inputs and an output to be linked to its neighbors. In the design structure of the process element, the memory is designed as registers to store the values of the previous and two previous iterations. The operation module executes the operation included add, subtract, multiply and divide in the pipelined fashion and the control section can generate the scheduling for the timing and flow of data between the modules. The process element is synthesized for Xilinx Artix 7 FPGAs using the synplify synthesis tool. The factors of determine the mapping is the level of parallelism and the FPGA size. Moreover, the storage memory can adopt the block RAM when a large number of nodes are to be mapped onto a process element.

VI. Experiment Analysis

The evaluation of the required advantage of mixed improvement algorithm on a FPGA is performed with the Xilinx Vivado 2016.3. Because the numbers of grid would be very specific to the device, as for large array which do not fit the block RAM on chip, the data must be streamed from outside and then the different number of IO pins and the bandwidth of the connection to the larger RAM become crucial for the overall performance.

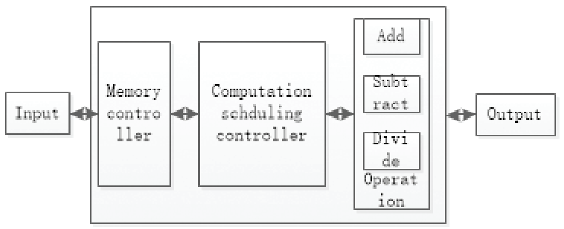

Fig. 4. Inner structure of process element.

Table1 gives the evaluation of resource consumption and percent of total resource for the pipelined improvement method. Table2 illustrates the iterative number of different iterative method for different mesh spacing and demonstrates that the improved pipelined algorithm possesses faster convergence speed than the Jacobi parallel iterative method, consequently cut down the computing time. In the same tolerance, we compared the performance implemented by hardware design and software simulation respectively. Table3 shows the hardware implementation speedup result relative to the result given by software simulation using the programming language C++. Meanwhile, it indicates that with the augment of the grid scale, the pipelined improvement algorithm reduced the computation time and its speedup is a little more than Jacobi method. Meanwhile this means intuitively that the pipelined algorithm improved the computation efficiency.

TABLE I. Evaluation of Resource Consumption and Percent of Total Resource

Resource	Slice LUTs	DSP48Es	Block RAMs	FF
Report Result	17115	67	49	17564
Percent of Total Resource	32%	30%	17%	16%

TABLE II. Iterative Number for Different Mesh Spacing

	h	Jacobi method	Improved algorithm
Iterative number	1/10	203	153
	1/40	3689	2472

TABLE III. Speedup Simulation Result

Tolerance	Grid number	Jacobi method	Improved algorithm
10-6	200	6.9	7.8
10-6	3200	1.2	2.3

VII. Conclution

In this paper, a parallel algorithm on pipelined computation for implementing finite difference schemes to solve the solution of the Poisson equation has been presented. The work suggests that the parallel computation in a pipelined fashion can get access to much better parallel efficiency and speedup than other commonly used methods and can simplify the process of program implementation. In the future work, we need to further improve the methods in the condition of higher double precision.

References

[1] Jiang Li, Japan, Kenichi Takahashi, Hakaru Tamukoh, Masatoshi Sekine, "2D/3D FPGA array for brain process and numerical

computation," Natural Computation (ICNC), 2012 Eighth International Conference on, May 2012, pp. 16-19.

[2] R Strzodka and D Goddeke, "Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components", 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines., 24-26 April 2006, 259 - 270.

[3] Gerhard Zumbusch, "Tuning a Finite Difference Computation for Parallel Vector Processors," 11th International Symposium on Parallel and Distributed Computing, 25-29 June 2012, pp. 63 - 70.

[4] H Kawaguchi, "Improved Architecture of FDTD Dataflow Machine for Higher Performance Electromagnetic Wave Simulation," 2016, 52(3):1-4.

[5] P D' Ambra and S Filippone, "A parallel generalized relaxation method for high-performance image segmentation on GPUs," Journal of Computational & Applied Mathematics, 2015, 293.

[6] LB Zhang, "On pipelined computation of a set of recurrences on distributed memory systems," Journal of Numerical Methods & Computer Applications, 1999, 20(03):184-191.

[7] E. Motuk, R. Woods, and S. Bilbao, "Implementation of finite difference schemes for the wave equation on FPGA," IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 3, pp. 237–240, March 2005.

[8] L Savioja and V Valimaki, "Reducing the dispersion error in the digital waveguide mesh using interpolation and frequency-warping techniques", IEEE Transactions on Speech & Audio Processing, 2000, 8(2):184-194.

[9] V Demir, "Graphics Processor Unit (Gpu) Acceleration of Finite-Difference Frequency-Domain (Fdfd) Method," Progress in Electromagnetics Research, 2012, 23:29-51.

[10] I Foster, "Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering," Addison-Wesley Longman Publishing Co., Inc., 1998, 16(2):88 – 90.

[11] Xu Changfa and Li Hong, "Numrical Solution of Partial Difference Equation," Huazhong University of Science and Technology Press, Sep. 2000.

[12] DJ Jackson and T Blom, "Fractal Image Compression Using a Circulating Pipeline Computation Model," June 1996.