PARALLEL IMPLEMENTATION OF FINITE DIFFERENCE SCHEMES FOR THE PLATE EQUATION ON A FPGA-BASED MULTI-PROCESSOR ARRAY

E. Motuk, R. Woods, and S. Bilbao

Sonic Arts Research Centre, Queen's University of Belfast
Belfast, Northern Ireland
email: e.motuk, r.woods, s.bilbao @.qub.ac.uk

ABSTRACT

The computational complexity of the finite difference (FD) schemes for the solution of the plate equation prevents them from being used in musical applications. The explicit FD schemes can be parallelized to run on multi-processor arrays for achieving real-time performance. Field Programmable Gate Arrays (FPGAs) provide an ideal platform for implementing these architectures with the advantages of low-power and small form factor. The paper presents a design for implementing FD schemes for the plate equation on a multi-processor architecture on a FPGA device. The results show that 64 processing elements can be accommodated on a Xilinx X2VP50 device, achieving 487 kHz throughput for a square FD grid of 50x50 points.

1. INTRODUCTION

Among the methods for the digital synthesis of sound, physical modelling approaches involve modelling of the sound production mechanism rather than the actual sound. This brings advantages of greater expressivity and wider ranges of sounds. Sound production mechanisms are described by partial differential equations (PDEs). Finite difference (FD) methods are the most obvious way to solve PDEs iteratively on a computer. The methods involve discretization of time and space to transform the PDEs to difference equations that can be implemented digitally. However, the major drawback of these methods is the massive computational requirements arising from the high space and time sampling rates due to the conditions on stability and convergence of the finite difference schemes. The computational complexity exceeds the capabilities of a single computer implementation and as a result parallel implementations should be sought.

A particular class of FD schemes, named explicit schemes, naturally lend themselves to parallel execution. As a result explicit FD schemes have traditionally been implemented on parallel computer networks or massively parallel computers for various applications involving the solution of PDEs. In the sound synthesis context, using parallel computer networks can be impractical due to the trend towards smaller audio hardware devices. Field Programmable Gate Arrays (FPGAs) are programmable logic devices having a large number of logic gates and dedicated units for signal processing such as RAMs and multipliers on chip. With these properties these devices are suitable for implementing massively parallel processing element (PE) networks for parallel implementation of FD schemes. With the added advantages of low-power and small form-factor, they can be used as hardware accelerators for computationally demanding physical modelling sound synthesis applications.

The sound production mechanism in plates can be described by the classical Kirchoff plate model which governs the small transverse displacements of a thin plate. This model can be used for synthesizing plate-based instrument sounds such as gongs and guitar bodies [2] and for digitally implementing plate reverberation. The PDE can be solved numerically by a number of explicit FD schemes.

In this paper we will describe the implementation of an FD scheme for the solution of the plate equation on an array multi-processor designed on a Xilinx Virtex II FPGA device. This implementation can be connected to a host device or a computer to provide the acceleration needed for real-time musical performance. In a previous paper [1], we have described FPGA implementation of wave equation. The FD schemes for the plate equation differ from those of the wave equation in terms of computation and communication patterns which will be explained in Sections 2 and 3. In addition, in this paper we discuss the aspects of boundary conditions and excitation of the scheme and their implications for hardware implementation.

2. A FINITE DIFFERENCE SCHEME FOR THE PLATE EQUATION

The plate model that can be used for sound synthesis is the classical Kirchoff model with added simple linear damping [2].

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2\nabla^4 u - 2\sigma\frac{\partial u}{\partial t} + f(x,y,t) \quad \nabla^4 = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)^2 \tag{1}$$

In this equation, $u(x,y,t)$ is the transverse plate deflection, and x,y,t are the space coordinates and time respectively. $f(x,y,t)$ represents a time varying driving term. κ^2 denotes the stiffness parameter, and σ is the coefficient that controls the decay rate of plate oscillation.

2.1 Algorithm

In order to solve equation (1) by a FD method, a grid function, $u_{i,j}^n$ that represents an approximation to $u(x,y,t)$ is defined. The space coordinates are discretized as $x = i\Delta x$,and $y = j\Delta y$, and time is sampled as $t = n\Delta t$, where $1/\Delta t$ is the sampling rate. We take $\Delta x = \Delta y$. The differential operators are approximated by centered and second-order accurate operators as

$$\begin{aligned}
\frac{\partial^2 u}{\partial t^2} &\approx \delta_{t0}^2 u_{i,j}^n = \frac{1}{\Delta t^2}\left(u_{i,j}^{n+1} - 2u_{i,j}^n + u_{i,j}^{n-1}\right) \\
\frac{\partial^2 u}{\partial x^2} &\approx \delta_{x0}^2 u_{i,j}^n = \frac{1}{\Delta x^2}\left(u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n\right)
\end{aligned}$$

$$\frac{\partial^2 u}{\partial y^2} \approx \delta_{y0}^2 u_{i,j}^n = \frac{1}{\Delta x^2}\left(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n\right)$$
$$\frac{\partial u}{\partial t} \approx \delta_{t0} u_{i,j}^n = \frac{1}{2\Delta t}\left(u_{i,j}^{n+1} - u_{i,j}^{n-1}\right)$$

The Laplacian and the biharmonic operators are approximated by $\nabla^2 \approx \delta_+^2 = \delta_{x0}^2 + \delta_{y0}^2$, and $\nabla^4 \approx \delta_+^2 \delta_+^2$.

When the above operators are substituted in equation (1), the following FD scheme is obtained

$$\delta_{t0}^2 u = -\kappa^2 \delta_+^2 \delta_+^2 u - 2\sigma \delta_{t0} u + f \qquad (2)$$

Writing equation (2) as an explicit recursion we get the formula below

$$u_{i,j}^{n+1} = \eta \sum_{|k|+|l| \leq 2} \beta_{|k|,|l|} u_{i+k,j+l}^n + \eta(\sigma\Delta t - 1) u_{i,j}^{n-1} + \Delta t^2 f_{i,j}^n \qquad (3)$$

where $\eta = 1/1 + \sigma\Delta t,\ \mu = \kappa\Delta t/\Delta x^2$, and

$$\begin{aligned} \beta_{0,0} &= 2 - 20\mu^2 \\ \beta_{0,1} &= \beta_{1,0} = 8\mu^2 \\ \beta_{1,1} &= -2\mu^2 \\ \beta_{0,2} &= \beta_{2,0} = -\mu^2 \end{aligned}$$

The stability condition below for this FD scheme can be obtained from von Neumann analysis [3].

$$\Delta x^2 \geq 4\kappa\Delta t \qquad (4)$$

2.2 Boundary Conditions and Excitation

For the boundary conditions of the equation (1) we took the clamped conditions where $u = \frac{\partial u}{\partial x_n} = 0$ on the boundaries, where x_n is a coordinate normal to the boundary.

For the FD scheme this translates to $u_{i,j}^n = 0$ for the grid points on the boundary. As the FD scheme involves access to the values of the grid function at a previous time step at most two spatial steps away in x and y directions, for the grid points adjacent to the boundary, such as $i, j = 1$, the values of the points outside the grid can be found from, $u_{i,1}^n = u_{i,-1}^n$, and $u_{1,j}^n = u_{-1,j}^n$.

In order to excite the plate, the driving function $f(x,y,t)$ in the equation (1) is used. In the FD scheme this function is discretized as $f_{i,j}^n = f(i\Delta x, j\Delta y, n\Delta t)$, and applied as an adder term for updating a particular grid point.

2.3 Computational Requirements

The computational requirements of the FD scheme can be calculated from equation (3). In order to update a particular grid point, 5 multiplication, and 13 addition operations are needed. For the excitation one extra addition is needed. Taking the operation count for updating a grid point as 18, the total number of operations per second (OPS) is $18L_xL_y/\Delta x^2\Delta t$.

The stability condition in equation (4) puts constraints on the Δx according to the sampling rate, $1/\Delta t$ chosen. Assuming equality in equation (4), $18L_xL_y/4\kappa\Delta t^2$ OPS is needed for the plate update. For a square steel plate of side length 2 m and thickness 2mm with $1/\Delta t = 44.1$ kHz, the computational requirement is approximately 12x10^9 OPS. In addition to the high operation count, handling the boundary conditions and memory accesses puts the FD scheme above the capabilities of a single processor implementation.

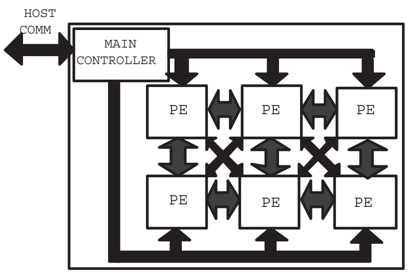

Figure 1: The network configuration

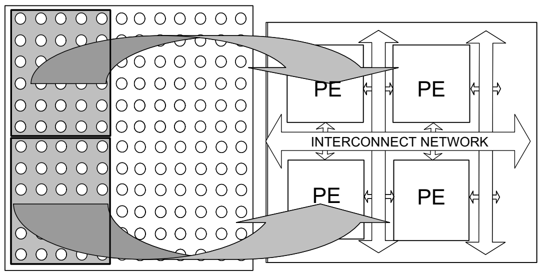

Figure 2: Partitioning by domain decomposition

3. PARALLEL IMPLEMENTATION

The explicit FD schemes have properties that can be exploited for a parallel implementation to tackle the high computational complexity. The most important one is the temporal independence in a certain iteration period, that is updating a grid point only involves neighbouring grid points previous values. Thus, all the grid points can be updated concurrently in an iteration period, and the order of the update does not matter. In addition to this, same operations are applied to different data in the problem domain, implying data parallelism.

Implementation of the FD scheme in parallel requires a network of processing elements (PEs) operating concurrently, and a main controller responsible from connection to the host device to output the results and to receive excitation. This configuration is shown in Fig. 1. Each PE in the network has its own memory to hold the values associated to the grid points in its sub-domain. The following subsections explains the steps toward a parallel implementation on such a configuration.

3.1 Partitioning the Grid for Parallel Implementation

In order to parallellize the computation, the domain of the FD scheme is partitioned into blocks through domain decomposition [4]. As a result of this, individual sub-domains can be mapped onto the PEs. When applying domain decomposition, the major issues are balancing the computational workload among the PEs, and reducing the communication links in between. Since the domain we apply domain decomposition is regular and rectangular, the method is to divide the problem domain into N rectangular blocks, where N is the number of PEs, each sub-domain having equal number of nodes. Fig. 2 shows block partitioning of a rectangular domain and mapping the sub-domains to the PEs.

3.2 Communication and Computation

For the particular difference scheme in equation (3), a grid point update requires access to the previous iteration values of the grid function up to two spatial steps away in vertical and horizontal directions, and one step away in diagonal direction as shown in Fig. 3a. Thus, when the domain of the FD scheme is block partitioned, each sub-domain has to be augmented with the boundary points (ghost points) which are originally mapped to the neighbouring sub-domains [5] as shown in Fig. 3b. This forces the transfer of the ghost points between neighbouring sub-domains in each iteration period.

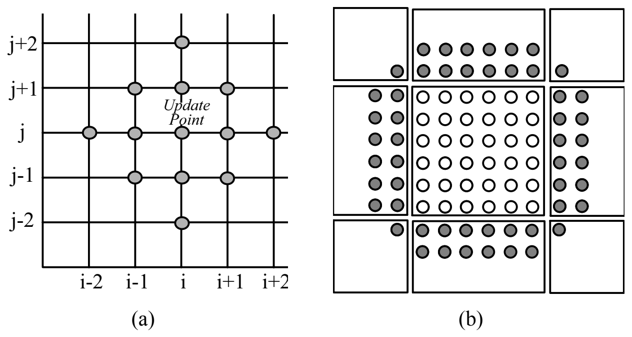

Figure 3: (a) Spatial dependencies for a grid point , (b) Ghost points corresponding to a sub-domain

In order to exploit this locality of the communication, neighbouring sub-domains are mapped on to neighbouring PEs. In this way the communication will be mostly local. The global communication is between the main controller and the PEs which involves taking the output from the grid, the excitation of the grid, and initialization of the PE network.

For a PE in which the computation and communication are not interleaved, in each iteration period of the FD scheme the computation can only start after all the ghost point values are transferred. Computation to communication ratio represents the efficiency of the parallel algorithm

3.3 Taking the Output and Excitation

The main controller in the multi-processor array is responsible from sending the output value to the host upon receiving a request from the host. In addition to the output request, the host also sends the output location information. In turn, the main controller creates a request signal, and address to the corresponding PE. The output is read from the PE's memory.

For the excitation of the scheme, the host sends the main controller a control signal, the location information and the excitation value. The main controller creates a control signal and an address signal for the corresponding PE, and also transfers the excitation value. In the PE the excitation value is added to the final sum of the excited node as previously explained in Section 2.

3.4 Parallel Algorithm

The resulting parallel algorithm that will be implemented on each PE in the network is stated below

```
- Do one time initialization
        Receive initialization parameters from the main controller
```

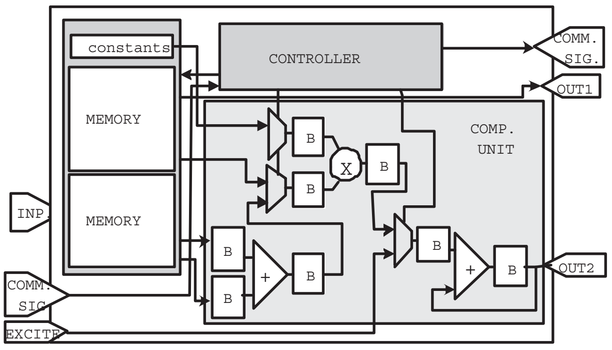

Figure 4: Architecture of a PE

```
- For n(iteration period no.)=1 to nmax do
      Start communication:
        Transfer/Receive ghost point values
      Start computation:
        For i=1 to last point do
          Fetch data from memories for computation
          Update grid points
          If excitation then
            Add excitation value to the point excited
        End
  End
```

4. HARDWARE DESIGN

4.1 Design of the Processing Elements

Fig. 4 shows the architecture of a PE in the network. The main part of the PE is the controller, which is responsible for local and global communication, address generation to access the grid point values stored in the memories, and providing start signals to the computation units. The controller also generates the memory addresses and control signals to write and read from the local memories for taking the output and excitation upon receiving such commands from the main controller.

Local communication consists of a simple handshaking protocol between neighbouring PEs, and transferring of values from one memory to the other. The memories in the PEs are augmented such that they also have locations for the grid points in the neighbouring sub-domains, which are called ghost points. The simple handshaking protocol is made up of receive and transfer signals which are produced by the controller for each neighbouring PE at the beginning of the iteration period.

The computation unit consists of two adders and a multiplier each having input and output buffers. The computational units have states inside, and they get data from their input buffers after receiving start signal from the controller, and send signals to the controller when they output their results to their output buffer. This configuration comes from the representation of the algorithm in dataflow networks model of computation which is explained in [1]. The multiplier has two multiplexers (MUXs) connected to its input buffers which are controlled by the controller. The first MUX selects among the different coefficients, and the second one selects between the ouput of the first adder and data coming from the memory blocks. The MUX connected to the input buffer of the second adder selects between the output of the multiplier and the excitation input.

The memory block that holds the grid point values consists of two separate blocks of RAM, each having one input and two output ports. One memory block holds the values corresponding to the previous iteration ($u^n_{i,j}$), and the second one holds the values corresponding to two previous iteration ($u^{n-1}_{i,j}$) as dictated by the FD scheme. The newly calculated values are written over the two previous iteration values after these values are used. The inputs and outputs of the memories are interchanged in each iteration period to avoid the overhead of transferring values between them.

The overall design of a PE is parameterized, so that a whole network can be built from one design template. This, in addition to the locality of the communication, provides the scalability of the network. New processors can be added without additional design effort and the design of the PE does not change when the geometry of the problem, or partitioning of the domain changes.

4.2 Design of the Main Controller

For the initialization of the PE network, the main controller employs a state machine to send initialization values step by step to the PEs. These values denote the number of nodes assigned to PEs and the communication sequence between neighbours. After the initialization period a start signal is sent to all PEs in the network. The initialization time does not affect the computation performance. The main controller also has conditional routines to respond to the request from the host for excitation and taking the output. These routines create the signals and shape the data coming from the host to be sent to the corresponding PE.

5. PERFORMANCE RESULTS

The main controller and the PEs are coded in VHDL hardware design language, and functional simulation is done using ModelSIM 5.8a. The network is synthesized for Xilinx X2VP50 FPGA [6] using SynplifyPro 7.6. The multipliers and Block RAM memories on the device are used in PEs.

The size of the PE network that can be implemented on a FPGA device depends on the size of the device. From the logic synthesis results, the number of logic slices used for a PE is 454. This constitutes 4% of logic slices on X2VP50. Computation of a grid point value takes 7 clock cycles, and the transferring takes 1 clock cycle. The total number of clock cycles N_{total} for an iteration period can be found by the formula, $N_{total} = n_s \times 7 + l_{comp.} + n_g + l_{comm.}$, where n_s is the number of points in the sub-domain (excluding boundary points), and n_g is the number of ghost points. $l_{comp.}$ and $l_{comm.}$ are the computation and communication latencies respectively. $l_{comp.}$ is equal to 13 and $l_{comm.}$ depends on the granularity of the partitioning. Table 1 lists the number of communication and computation clock cycles ($N_{comm.}$,$N_{comp.}$), total number of clock cycles for an iteration period, communication to computation ratio (CCR), and percentage of logic slices used corresponding to different PE network configurations for implementing a 50x50 square grid.

When the FPGA device is clocked at 170 MHz (according to synthesis results), table 2 lists the data output rate attainable (f_{out}), and the time it takes to produce 1 s of sound sampled at 44.1 kHz (t_{1s}) for each of the configurations also listed in table 2.

From the results we see that the output rate is faster than the real-time audio rate in most of the configurations. When the number of PEs in the network increases, in addition to the increase in output rate, there is an increase in the communication to computation ratio. Although this situation does not reduce the performance much in the example, it can do so when the number of ghost points and the partitioning granularity are higher as in the case for bigger domains. The solution to this is interleaving the communication and computation which we are currently working on.

The maximum size of a network that can be implemented on a X2VP50 device is approximately 64 PEs and a main controller. This puts a limit on the size of domains that can be implemented in real-time. In order to implement bigger domains larger FPGA devices can be chosen.

Table 1. Results for different network configurations

Network Conf.	$N_{comm.}$	$N_{comp.}$	N_{total}	CCR	Logic Slices
4x4	132	1021	1153	0.13	24%
2x8	130	1021	1151	0.13	24%
4x8	108	517	625	0.20	48%
6x6	100	461	561	0.22	54%
8x6	92	349	441	0.26	72%
8x8	84	265	349	0.32	96%

Table 2. Throughput Results

Network Conf.	f_{out}(kHz)	t_{1s}
4x4	147.4	0.3
2x8	147.7	0.3
4x8	272.0	0.16
6x6	303.0	0.14
8x6	385.5	0.11
8x8	487.1	0.09

6. CONCLUSION

In this paper, we presented the design of a hardware platform for accelerating physical modelling of plates by FD schemes. The performance results show that we can achieve real-time performance, which is not possible in the case of software implementation on a single computer. Parallel implementation on a multi-processor architecture on FPGA is more preferable to computer networks for its advantages of low-power and much smaller form factor.

REFERENCES

[1] E. Motuk, R. Woods, and S. Bilbao, "Implementation of Finite Difference Schemes for the Wave Equation on FPGA, " to be published in *Proc. ICASSP 2005*, Philadelphia, USA, March 18-23. 2005.

[2] A. Chaigne and C. Lambourg, "Time Domain Simulation of Damped Impacted Plates. I. Theory and Experiments," *J. of Acoust. Soc. Am.*, vol. 109(4), pp. 1422–32, Apr. 2001.

[3] S. Bilbao, "A Finite Difference Plate Model," submitted to *ICMC 2005*, Barcelona, Spain,

[4] E. Acklam and H. P. Langtangen, *Parallelization of Explicit Finite Difference Schemes via Domain Decomposition*. Address: Oslo Scientific Computing Archive, 1999.

[5] A. Malagoli, A. Dubey, and F. Cattaneo, "Portable and Efficient Parallel Code for Astrophysical Fluid Dynamics," in *Proc. Parallel CFD 95*, Pasadena, USA, June 1995, available from http://astro.uchicago.edu/Computing/On-Line/cfd95/camelse.html

[6] Xilinx, *Virtex II Pro FPGA User Guide*, www.xilinx.com