ResearchGate

See discussions, stats, and author profiles for this publication at: http://www.researchgate.net/publication/3320564

Design Methodology for Real-Time FPGA-Based Sound Synthesis

ARTICLE *in* IEEE TRANSACTIONS ON SIGNAL PROCESSING · JANUARY 2008

Impact Factor: 3.2 · DOI: 10.1109/TSP.2007.898785 · Source: IEEE Xplore

CITATIONS
3

DOWNLOADS
9

VIEWS
122

4 AUTHORS, INCLUDING:

Erdem Motuk
University College London
10 PUBLICATIONS **28** CITATIONS
SEE PROFILE

Roger Woods
Queen's University Belfast
165 PUBLICATIONS **704** CITATIONS
SEE PROFILE

Stefan Bilbao
The University of Edinburgh
120 PUBLICATIONS **670** CITATIONS
SEE PROFILE

Available from: Stefan Bilbao
Retrieved on: 23 June 2015

Design methodology for real-time FPGA-based sound synthesis

Erdem Motuk, *Member, IEEE,* Roger Woods*, *Senior Member, IEEE,* Stefan Bilbao, *Member, IEEE,* John McAllister, *Member, IEEE,*

Abstract

Explicit finite difference (FD) schemes can realise highly realistic physical models of musical instruments but are computationally complex. A design methodology is presented for the creation of FPGA-based micro-architectures for FD schemes which can be applied to a range of applications with varying computational requirements, excitation and output patterns and boundary conditions. It has been applied to membrane and plate-based sound producing models, resulting in faster than real-time performance on a Xilinx XC2VP50 device which is 10 to 35 times faster than general purpose and DSP processors. The models have developed in such a way to allow a wide range of interaction (by a musician) thereby leading to the possibility of creating a highly realistic digital musical instrument.

Index Terms

FPGA implementation, physical models, finite difference, digital instruments.

I. Introduction

Sound synthesis is an indispensable tool in musical composition and performance, and effect generation in computer games and virtual reality applications. Digital-based techniques range from the reproduction of recordings e.g. sampling and the wavetable synthesis techniques and the generation of sounds from an absolute abstraction such as FM synthesis [1]. Spectral and physical models fall between these two extremes with spectral model-based techniques dealing with the construction of the perceived sound and physical models aiming to create the sound itself by reproducing transient features of the instrument sounds. This allows more expressive and intuitive interactions with the models thus giving rise to the creation of new instruments that the user can interact with in different geometrical ways [2] and using varying gestures [3].

The general structure of physical modelling sound synthesis is given in Fig. 1 [4]. The exciter provides the stimuli to the resonator block which represents the vibration mechanism of the instrument. The latter can be realised by a

Manuscript received. This work was supported by the Support Program for University Research.

E. Motuk, R Woods and J McAllister are with the Sonic Arts Research Centre, Queen's University Belfast, 4 Cloreen Park, BT3 5HN, UK (phone: +44 (0)28 90974081, fax: +44 (0)28 90974828) (e-mail: e.motuk, r.woods, j.mcallister@qub.ac.uk).

S. Bilbao is with the University of Edinburgh, School of Arts, Culture and Environment, Alison House, 12 Nicolson Square, Edinburgh, EH8 9DF, UK (e-mail: sbilbao@staffmail.ed.ac.uk).

physical model defined as a set of coupled partial differential equations (PDEs) which involve partial derivatives in both time and space dimensions, and that are defined with initial and boundary conditions. Typically, these PDEs are solved using explicit finite difference (FD) schemes that describe structured time and space discretisation and which involve huge computational complexity, particularly for 2-D and 3-D. This represents the main obstacle in realising a digital instrument based on this approach as it is well beyond the capabilities of modern desktop computers.

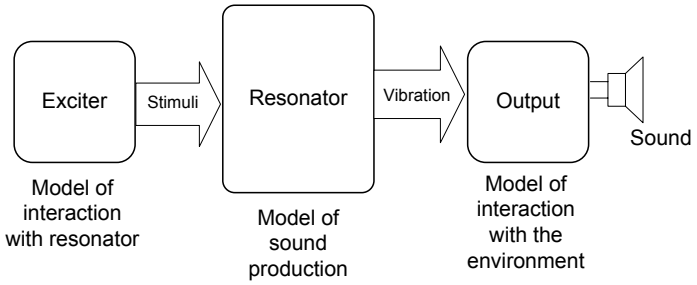


Fig. 1. General structure for physical modelling sound synthesis

The work in this paper presents the first attempt at developing a design methodology for the real-time sound synthesis and effects generation based on physical models using FD methods. Modern FPGAs offer numerous hardware resources in form of multipliers, accumulators, microprocessors and RAM blocks and are ideal platforms to allow the efficient exploitation of the high level of concurrency available in many FD schemes. The paper starts by describing FD methods and then introduces the systematic approach to the design of FPGA-based micro-architectures before applying it to membrane and plate-based sound producing models. The existence of the methodology leads to the possibility of designing the computational engine for a digital instrument which can be played in a novel manner by the various input and output mechanisms that are supported. For example, it is possible to drive the model using a sound file rather than an impulse which is synonymous to striking the instrument.

II. FINITE DIFFERENCE METHODS

In linear FD schemes, a $N+1$ uniform grid of points is defined with N space dimensions and 1 time dimension. The function u defined on the grid, $u^n_{i,j}$, corresponds to u at the grid point (i, j, n) for a (2+1)-D grid and is the discretised version of the continuous function $u(x, y, t)$ at coordinates $x = i\Delta x$, $y = j\Delta y$, and $t = n\Delta t$, where Δt, Δx, and Δy correspond to the time and space sampling periods respectively. For the partial derivatives involving time and space dimensions, FD schemes with difference approximations and properties have been proposed for various applications.

The FD approximations are obtained by using truncated Taylor series shown in equations (1) and (2) where the function $u(x, y, t)$ is expanded on the x coordinate,

$$u(x + \Delta x, y, t) = u(x, y, t) + \Delta x \frac{\partial u}{\partial x} + \frac{\Delta x^2}{2!}\frac{\partial^2 u}{\partial x^2} + \frac{\Delta x^3}{3!}\frac{\partial^3 u}{\partial x^3} + O(\Delta x^4), \quad (1)$$

$$u(x - \Delta x, y, t) = u(x, y, t) - \Delta x \frac{\partial u}{\partial x} + \frac{\Delta x^2}{2!}\frac{\partial^2 u}{\partial x^2} - \frac{\Delta x^3}{3!}\frac{\partial^3 u}{\partial x^3} + O(\Delta x^4). \quad (2)$$

The first and second order approximations to $\partial u(x,y,t)/\partial x$ are shown in equations (3) and (4) respectively where the order corresponds to the highest order of Δx to be truncated. In order to reduce the truncation error, either a high order approximation is chosen or the sampling period is decreased.

$$\frac{u(x+\Delta x,y,t)-u(x,y,t)}{\Delta x} = \frac{\partial u(x,y,t)}{\partial x}+O(\Delta x) \quad (3)$$

$$\frac{u(x+\Delta x,y,t)-u(x-\Delta x,y,t)}{2\Delta x} = \frac{\partial u(x,y,t)}{\partial x}+O(\Delta x^2) \quad (4)$$

The dispersion relation of an FD scheme is different than that of the PDE which causes dispersion errors. In this case, for a non-dispersive PDE, the phase velocity of the numerical solution becomes dependent on the frequency and the direction of propagation [5] whereas in the dispersive case, the error between the actual and the numerical phase velocities corresponds to different wavenumbers which are dependent on frequency and direction of propagation. Deviations in the phase velocity due to numerical dispersion correspond to deviations in eigen (modal) frequencies, which are of great perceptual importance in musical acoustics can be reduced by increasing the order or decreasing the sampling period [6].

The computational efficiency and memory requirements increase with order size but for a given order, smaller step sizes in time and space result in more calculations per second for a given domain. Therefore, the choice of an FD scheme for a given problem requires a trade-off between accuracy, sampling period and computational efficiency. Another important aspect is whether the FD scheme is explicit i.e. the state can be solved as a function of the earlier states or implicit i.e. the FD formula contains more than one non-zero terms with time level $n+1$, requiring the inversion of a large but sparse matrix [5].

III. Design methodology for hardware implementation

A design methodology taking the form of the Y-shape diagram [7] is proposed (Fig. 2). Separate models of application and architecture are built for the initial stage of the design with a gradual lowering of the level of abstraction during the design process. The key stages are:

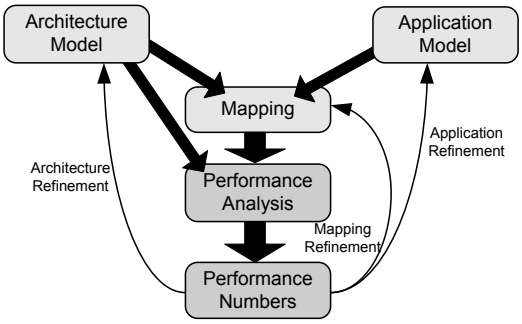


Fig. 2. Y-chart methodology

- Build a general application model for FD algorithms which considers detailed hardware implementation.

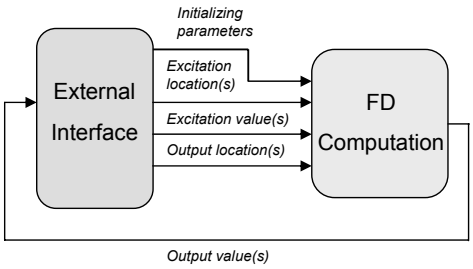


Fig. 3. General application model

- Investigate parallelism available in the general application model representing the FD algorithm.
- Build a parameterised architecture model to be matched to the particular application.
- Produce application instances with parameters for specific sound synthesis and effects applications.
- Create a set of FPGA-based micro-architectures that are parameterised in terms of performance and resource usage characteristics for the application sets.

IV. General application model

The general application model is represented by a communicating process corresponding to the computation of the FD algorithm which represents the resonating mechanism and, an external interface process which is responsible for initializing the computation and processing input and outputs (Fig. 3). The FD algorithm (Fig. 4) consists of nested loops defined in space and time that compute update equations at each point in the grid. The numbers of time steps (T_{MAX}) (given as $t_f/\Delta t$ where t_f is the sound file duration), points in x and y coordinates (Nx and Ny) and coefficients (α, β), are calculated in the initialisation step. Nx and Ny are calculated as $l_x/\Delta x$ and $l_y/\Delta x$ respectively where one spatial sampling rate, Δx, is used for both coordinates. Boundary points and those adjacent, are treated differently from the regular inner points. For sound synthesis, the FD grid is initialised and then executed by applying excitation to and taking output from, multiple points in the model. It is this latter aspect that is of most interest to composers as it allows the creation of new sounds.

A. High-level representation of FD algorithms

Dataflow representation [8] exposes the different levels of concurrency and the 5X5 grid that corresponds plate FD scheme to be introduced later is given in Fig. 5 with each update computation corresponding to a node and data being distributed and stored as internal states. The edges between the nodes in the graph show the spatial dependencies for the update operation implying local communication. For simplicity, input and output edges are shown for only the node in the middle and connection to the external interface is ignored. The node functionality (Fig. 6) corresponds to the membrane FD scheme given later and shows local storage node states that provide data tokens in order to eliminate zero-delay cycles in the dataflow graph, guaranteeing deadlock avoidance.

```
Begin
  Initialize
  for t=0 to T_MAX
     Get output location(s)
     Get excitation
     Update outer points according to boundary conditions
     For i=1 to Nx
        For j=1 to Ny
          Calculate u_{i,j}^{n+1} according to update equation
          If excitation then
             u_{i,j}^{n+1} += f_{i,j}^{n}
          If output then
             Send u_{i,j}^{n+1}
        End for
     End for
  End for
End
```

Fig. 4. General FD algorithm

All the different sound synthesis and effects applications based on a particular sound producing model that employ the general FD algorithm form an application set. The parameters that are common to all the instances of a particular application set are the number of:

- Operations per point update,
- Memory reads and writes per point update,
- Time steps involved in the update of the FD scheme.

The parameters that define the different instances of an application set are: domain size; time and spatial sampling rates; excitation and output patterns and; boundary conditions.

V. General architecture model

A. Exploiting parallelism on a hardware platform

Coarse grain parallel execution makes use of algorithmic concurrency, allowing scalable, low communication overhead implementations. During run time, nodes exchange data as determined by the FD scheme and execute concurrently. Thus, parallelism can be exploited by assigning each node to a PE but as the FPGA processing data rate is much higher than the sampling rate, PE hardware sharing via domain decomposition method [9] is used which involves splitting the FD algorithm into smaller algorithms. The computation of the boundary points are handled via "'ghost points"' which are shown in Fig. 7 for a rectangular partition for two different FD stencils used later. Fine grained parallel execution is employed by choosing the levels of concurrency within the PE spatially, by implementing operations in parallel and temporally, by employing pipelining.

B. High level memory design

The FD algorithms are memory bound, operating over large memories with high access bandwidth requirements which scale with grid size with the need to support simultaneous memory access for parallel execution. Memory

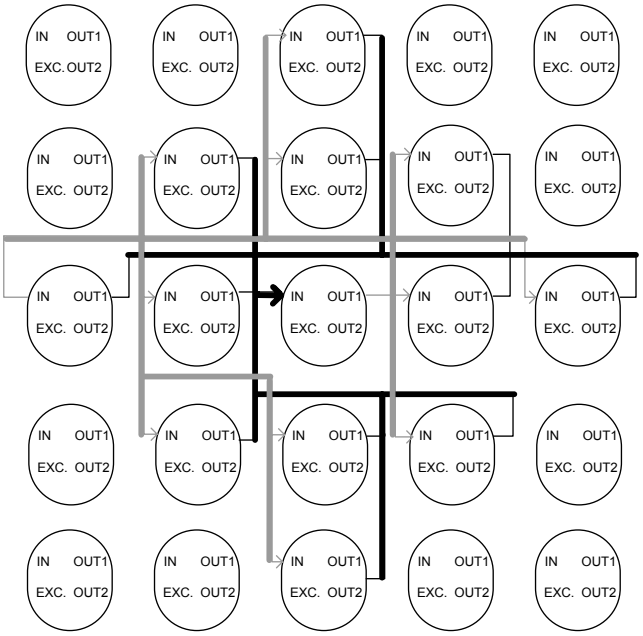


Fig. 5. Dataflow representation of a general FD algorithm for a domain of size 5x5

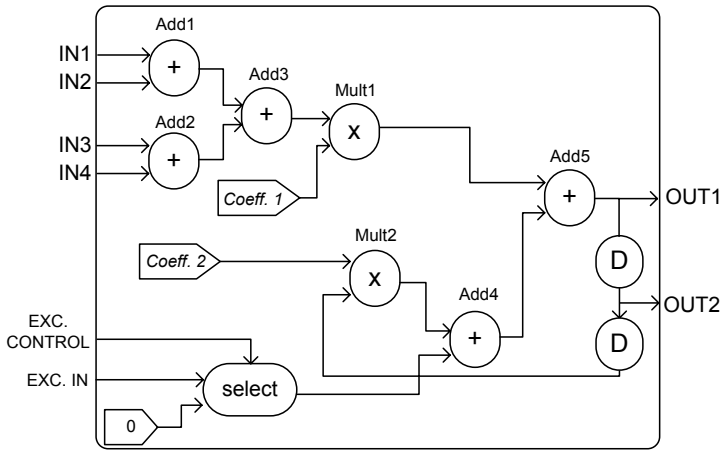
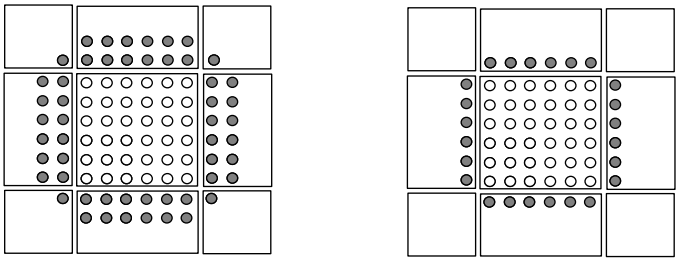


Fig. 6. Dataflow representation corresponding to a node in figure 5



(a) Plate (b) Membrane

Fig. 7. Ghost points for a rectangular partition for two stencils

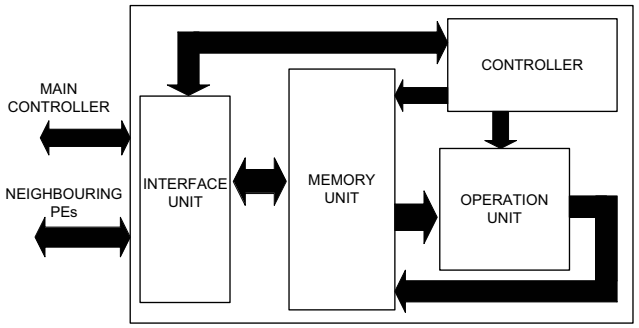


Fig. 8. Architecture of a processing element

can be implemented using distributed registers giving good performance (local control) or a shared memory with a complex controller which is good for larger grids but which reduces scalability. With the domain decomposition, a distributed memory parallel processing architecture is used with registers forming memory blocks for grid values, the size of which will be K (number of PEs) times smaller than the shared memory blocks.

C. Architecture model

1) PE network: A network of PEs is used which transfer grid point values with their neighbours, according to the partitioning of the FD grid. Inter-PE communication is point-to-point operating on many channels with a channel for each input such that no contention occurs when all inputs arrive simultaneously, and global communication is used for initialization and external data transfer.

2) Main controller: The main controller is responsible for: PE initialisation (sending parameters e.g. FD algorithm coefficients and "start" signal to each PE); communication with the external interface; PE synchronisation and; distribution of excitation and output data. The main controller parameters are number and size of the FSMs and buffer sizes for external interface and on-chip communication.

3) Processing elements: Each PE (Fig. 8) has a controller that is responsible for the memory address generation, scheduling of computations and communication with neighbouring PEs and the main controller. The computation unit implements the functionality of Fig. 6 using registers to act as input and output buffers and as coefficient storage. The computation unit parameters include: number of processors and registers and number and size of multiplexers. Memory unit parameters include: number of blocks for grid point values, block size, number of read/write ports, latency associated with read/writes. The interface unit parameters are size and buffer structure e.g. memory blocks or FIFOs, number of registers for the global communication and number of channels for point-to-point communication.

VI. Refining the architecture model

The general architectural model is next refined by mapping model parameters to FPGA relevant constraints, specifically throughput, area and most importantly memory. This then allows selection of the most suitable micro-architecture that matches the required real-time performance with minimum resource usage.

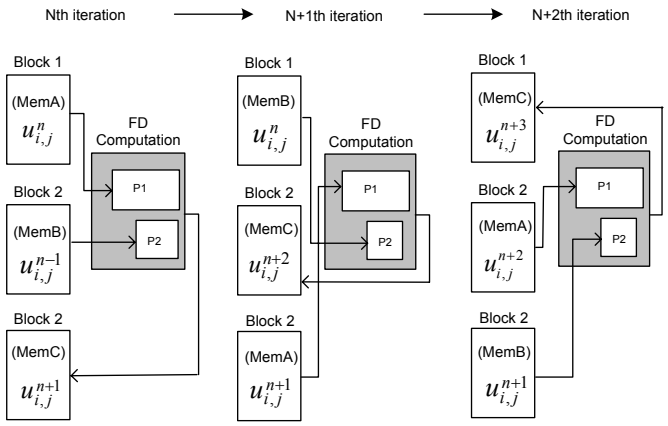


Fig. 9. Memory access pattern

A. Formalising the performance

The throughput rate of the FD grid update, f_{OP}/N_{total} must be greater than the sampling rate of the FD scheme, f_{FD}, where f_{OP} is the operating clock frequency of the FPGA device and N_{total} is the total number of clock cycles for the FD grid update and given by,

$$N_{total} = max_{1 \leq i \leq P}(N_i) + O_{network} \tag{5}$$

where P denotes the number of PEs in the network, N_i the number of clock cycles for the i^{th} PE to update its sub-partition and $O_{network}$ denotes the overhead corresponding to PE network. The N_i term is given by the number of cycles for communications, $N_{comm.}$, given by number of ghost points divided by the inverse of the number of clock cycles to transfer one point value, namely $(n_{ghost\_points}/comm.rate + comm.latency)$ plus the cycles for computation $N_{comp.}$ given by an equivalent expression $(n_{points}/comp.rate + comp.latency)$. The latencies include the pipelined datapath latency and local communication setup times.

B. Memory structure

PE memory depends on the number of points in the sub-partition and time steps required by the FD algorithm. For second order FD algorithms, the update equations involve three time steps and need three different blocks of memory required for storing $u_{i,j}^{n+1}$, $u_{i,j}^{n}$, and $u_{i,j}^{n-1}$ (Fig. 9) although in some cases, this can be reduced to 2 as $u_{i,j}^{n-1}$ can be written over by $u_{i,j}^{n+1}$.

C. Computation unit

Pipelining is employed by first determining the number and type of the operational units and scheduling the computations accordingly. Modulo scheduling involves the initiation interval (II) $(1/comp.\_rate.)$ [10] which depends on the number of grid function values to be accessed, number of the operational units and the scheduling.

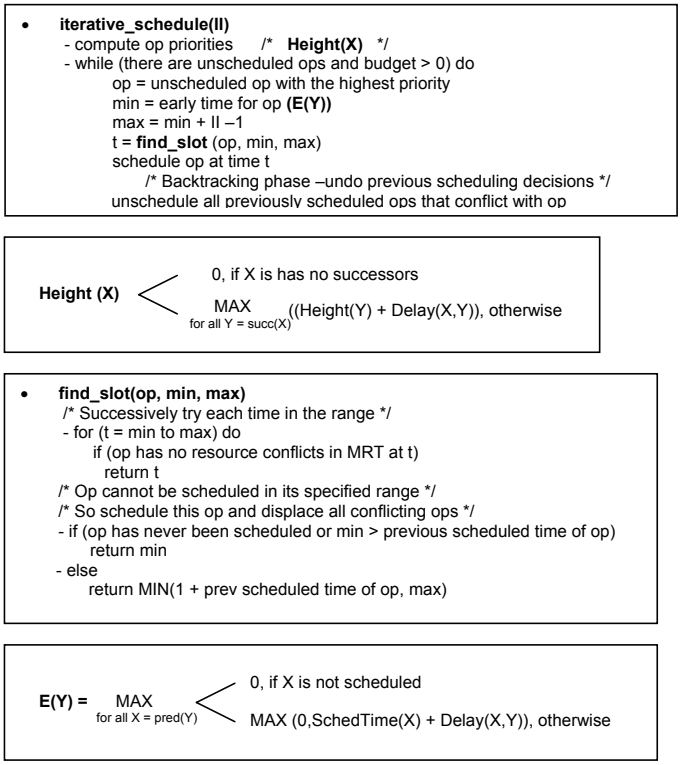


Fig. 10. Modulo Scheduling algorithm

Since the FD algorithms are explicit, the loop schedule is obtained by repeating a schedule of II cycles. As opposed to using resource constraints to get a minimum II length [11], the approach is to use a particular II value - determined by the memory access schemes which give the number of grid function values that can be accessed by the computation unit at each clock cycle, and then calculating the number of operational units by computing the number of additions and multiplications required and dividing this by the II value. The actual schedule is given by a variation of the algorithm presented in [11](Fig. 10). The heuristics used to determine the scheduling height based priority min_time, and choice of time slots are also shown. In the case of units with multiple clock cycle latencies, the length of each iteration will increase, changing the actual scheduling times but not the computation rate.

D. Controller

The PE controller consists of hierarchical FSMs which control the datapath, local data transfer and memory block access. The top level FSM consists of the 'communication', 'computation' and 'wait' states along with signals 'start' and 'iteration finished'. In the 'computation state' FSM, the number of sub-states is equal to the length of the prologue and the epilogue of the whole schedule plus the sub-states, given by the length of II and an extra 'boundary' state, needed for updating the boundary conditions. The 'communications' FSM controls the regeneration of the transmit and receive signals.

E. Interface Unit

The interface unit comprises buffers of size, M_{buffer} given as $n_{trans.} \times bit - width$ where $n_{trans.}$ is the total number of points to be transferred and equal to $2 \times n_{ghost}$. The buffers can be implemented by using the on-chip memory or registers and to reduce size, only received values are stored in the buffers as data to be sent can be read from the memory that stores grid function values.

VII. FPGA MAPPING

The last stage of the design process involves estimating performance and choosing the FPGA micro-architecture comprising PE networks of a particular PE type that meets the resource and real-time performance requirements. The requirements determine the parameters such as computation and communication rates, numerical representation, boundary conditions, number of grid points assigned to a PE, number of communication interfaces and FPGA device clock frequency. Given a FD grid of size $N_{points} = N_x \times N_y$ and sampling rate f_{FD}, the following factors determine the FPGA performance.

- Size of the memory on the device, $M_{storage}$, given by $2 \times N_{points} \times bit\text{-}width$
- Number of external memory links and external memory access rate
- Numerical representation
- Clock frequency of the device
- Number of PEs that can fit on the device - dependent on logic size
- PE network overhead

If the memory fits on the FPGA, it is divided among the PEs with at least 2 memory blocks for each PE, defining number of grid points that it can support. If external memory has to be used, then memory interfaces will depend on the pin count of the device and logic size of the memory interface, thus possibly limiting the number of PEs as one 2-port external memory bank can be assigned to 2 PEs. A library of PE architectures with varying logic size and performance characteristics is constructed using the parameters: grid points assigned to a PE, computation rate of the datapath, inter-PE communication channels, number of bits used and FPGA device clock frequency for the synthesised architecture namely $f_{dev.\_clock}$.

Fig. 11 presents a procedure that maps an application onto an FPGA. The PE network is chosen such that it provides the required update rate of the grid, f_{update}, with certain resource usage characterised by the number of logic slices, N_{logic}, block RAMs, N_{mem}, and embedded multipliers, N_{mult}. The network is formed by a single type of PE that corresponds to the i^{th} element of the set of available PE architectures, P, with the characteristics $n_{points\_max\_i}$, n_{logic_i}, n_{mem_i}, n_{mult_i}. The result of the procedure is a set of micro-architectures, M, with j^{th} element of M having K PEs of type $P(i)$. $f_{dev.\_clock}$ and the total resource usage is estimated from a single PE with actual figures obtained post place and route.

```
Procedure FPGA mapping
Given an application and an FPGA device with
    N_logic : Total no. of logic slices
    N_mem : Total no. of memory blocks
    N_mult : Total no. of multipliers
Determine numerical representation
Determine the storage memory requirement (M_storage)
if M_storage < device memory size then
        Memory type ← Internal memory
        K_max = No. of blocks/2
else
        Memory type ← External memory
        K_max = No. of external memory interfaces x 2
end
while i ∈ P do
        Calculate N_total_max   {freq_dev._clock / f_update}
        K = 4
        while K < K_max do
                n_points = N_points/K
                if n_points ≤ n_points_max_i then
                        Calculate N_total_i { N_comp._i + N_comm._i + L_comp._i + L_mov._i + L_comm._i }
                        Calculate no. of logic slices { N_logic_nw = n_logic_i x K }
                        Calculate no. of memory blocks { N_mem_nw = n_mem_i x K }
                        Calculate no. of multipliers { N_mult_nw = n_mult_i x K }
                        if (N_logic_nw ≥ N_logic) or (N_mem_nw ≥ N_mem) or (N_mult_nw ≥ N_mult) then
                            Exit while
                        end
                        if N_total_i ≤ N_total_max then
                            M(j) ← K PEs of type i
                            j = j + 1
                            Exit while
                        end
                end
                K = K + 1
        end
        i ← next PE type in the library
end
EndProcedure
```

Fig. 11. The procedure for FPGA mapping

VIII. Application to membrane-based sound synthesis

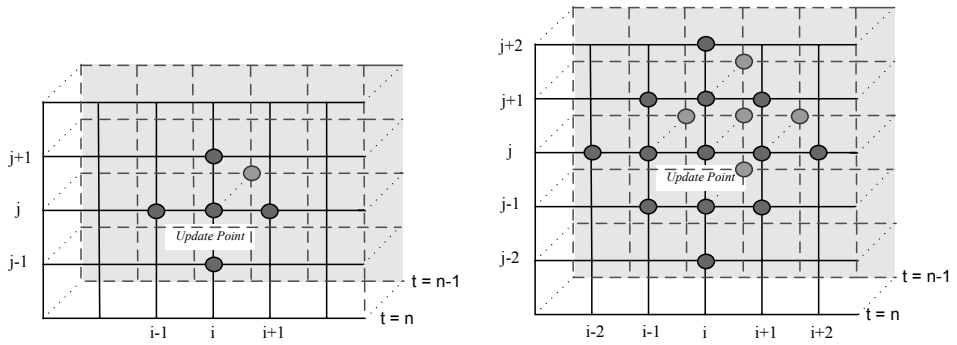
The methodology is now applied to the design of both a membrane and a plate-based synthesis models to show its range of applicability. Detailed descriptions of the applications can be found in [12] and [13].

A. Membrane model and Finite difference representation

The model for a vibrating membrane can be described by the wave equation in its 2-D form plus a simple linear damping term,

$$\frac{\partial^2 u}{\partial t^2} = \nu^2 (\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}) - 2\sigma \frac{\partial u}{\partial t} + f(x, y, t) \tag{6}$$

where $u(x, y, t)$ denotes the transverse displacement of the membrane, defined over $0 \leq x \leq L_x$, $0 \leq y \leq L_y$ and time $t \geq 0$, ν is the wave speed, and σ represents the coefficient for the damping term which can be adjusted



(a) Membrane model (b) Plate model

Fig. 12. Stencils for FD schemes

Parameter	Feature	Membrane	Plate
Computation	No. of multiplies	2	6
	No. of adds	5	18
Memory	Reads	5	18
	Writes	1	1
No. of FD steps		3	3

TABLE I

APPLICATIONS PARAMETERS

to represent a particular mechanism. For musical applications, a driving term, $f(x, y, t)$, is added to the model to represent the external excitation of the membrane. This gives the explicit FD algorithm representation [12] below

$$u_{i,j}^{n+1} = \frac{1}{1+\sigma\Delta t}\left(u_{i+1,j}^{n} + u_{i-1,j}^{n} + u_{i,j+1}^{n} + u_{i,j-1}^{n}\right) \quad + \quad \frac{\sigma\Delta t - 1}{1+\sigma\Delta t}\, u_{i,j}^{n-1} + \Delta t^2 f_{i,j}^{n}. \tag{7}$$

The stencil (Fig. 12(a)) shows the spatial and temporal data dependencies for updating a point. For this application set, values of the parameters are given in Table I.

B. FPGA implementation

The iterative procedure in Fig. 11 is used for the final mapping on to an FPGA device.

1) Memory structure: Two memory blocks MemA and MemB, are assigned to the PE along with the moving window (Fig. 13). A moving window structure consisting of two shift registers of length $(n_x - 2)$, where n_x is the number of points in the horizontal direction of the partition that is assigned to a PE, is used to allow multiple grid points to be made available in one cycle. Table II gives the memory access scheme options with number of cycles needed to access the grid function values for the update of a point defined as, n_{cycles_mem}.

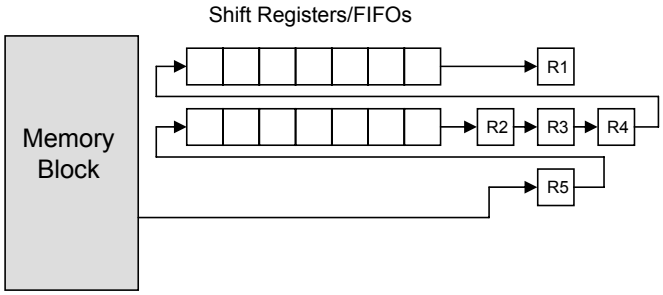


Fig. 13. Moving window for the membrane-based applications

Memory access type	Accesses at each cycle	n_{cycles_mem}
Moving window 2-port MemB	4 reads MemA 1 read/write MemB	1
2-port MemA 2-port MemB	2 reads MemA 1 read/write MemB	2
1-port MemA 2-port MemB	1 read MemA 1 read/write MemB	4

TABLE II

LIST OF MEMORY ACCESS TYPES

2) Computation unit: The possible minimum II lengths can be 1, 2, and 4. Using adders, multipliers and extra registers for intermediate results, table III lists the iteration schedules in the pipelined execution when operational unit latencies are one clock cycle. Fig. 14 shows the possible scheduling configurations.

3) Interface Unit: In the case of 2-D domain decomposition, for a partition of size $n_x \times n_y$, the number of values to be transferred, $n_{trans.}$, is $4 \times (n_x + n_y - 2)$ with half of this number sent to, and the other half received from, the neighbouring PEs. In the 1-D case, $n_{trans} = 4 \times n_x$ or $4 \times n_y$ and M_{buffer}, is $n_{trans.} \times bit - width/2$.

4) List of PE types: Table IV lists the types of PEs available with the number of clock cycles for computation and communication, $N_{comp.}$ and $N_{comm.}$, needed to update a partition of size $n_{points} = n_x \times n_y$. Number of clock cycles for computation ($L_{comp.}$), moving window ($L_{mov.}$), and communication latencies ($L_{comm.}$) are also shown. FPGA resources and speed is determined by the bit widths and the maximum number of grid points that the PE can support, n_{points_max}. The number of logic slices (n_{logic}), Block RAMs (n_{mem}) and 18X18 multipliers (n_{mult}), and the clock frequencies for the PEs of type I and II in Table IV supporting up to 2048 grid points, are listed in Table V for a Xilinx Virtex II Pro device. Distributed memory (PE type I) and block RAM (PE type II) is used for the communication FIFOs. Two PE networks made up of 20 PEs of type I and II implementing a 200x200 grid at 44.1kHz, a wordlength of 16 bits, implemented on a Xilinx XC2VP50 device were created. With FPGA clock rate in excess of 170MHz, both were able to produce 1 second of sound (t_{1s}) in faster than real-time compared to the same data bit-width implementations on a laptop with a Pentium-M processor running at 1.6 GHz, and a DSP

II	Cyc. 1	Cyc. 2	Cyc. 3	Cyc. 4	Cyc. 5	Cyc. 6	Cyc. 7	Cyc. 8	Cyc. 9
1	Read $u_{i,j}^{n-1}$, $u_{i+1,j}^{n}$, $u_{i-1,j}^{n}$ $u_{i,j+1}^{n}$, $u_{i,j-1}^{n}$	Add1 Add2 Mult2	Add3 Add4	Mult1	Add5	Write $u_{i,j}^{n+1}$			
2	Read $u_{i+1,j}^{n}$, $u_{i-1,j}^{n}$, $u_{i,j}^{n-1}$	Read $u_{i,j+1}^{n}$ $u_{i,j-1}^{n}$, Add1, Mult2	Add2 Add4	Add3	Mult1	Add5	Write $u_{i,j}^{n+1}$		
4	Read $u_{i+1,j}^{n}$ $u_{i,j}^{n-1}$	Read $u_{i-1,j}^{n}$ Mult2	Read $u_{i,j+1}^{n}$ Add1, Add4	Read $u_{i,j-1}^{n}$	Add2	Add3	Mult1	Add5	Write $u_{i,j}^{n+1}$

TABLE III

SCHEDULES FOR THE THREE II VALUES

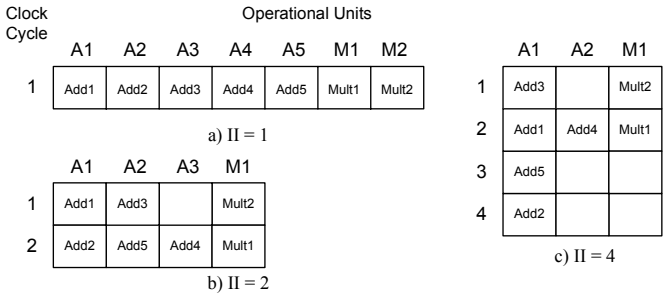


Fig. 14. Kernels for the schedules in table 3

processor, TI C6415 running at 720 MHz (Table VI). For the Pentium-M implementation, C code for the algorithm was written as a nested loop as in Fig. 4 with the optimization level set to -o3 for best optimization on a compiler based on the Mingw port of GCC. The TI C6415 implementation was based on optimized C code supplied on the development environment, Code Composer Studio 3.1, with compiler optimization level set to highest.

IX. APPLICATION TO PLATE-BASED SOUND SYNTHESIS

A. Plate model

The model for a stiff plate is a variation of the classical Kirchoff model [14] with a term for the frequency dependent damping, which models the damping due to thermoelasticity, viscoelasticity, and radiation [15], and a term for the membrane like characteristics in addition to the first model [16]. It is given below as,

$$\frac{\partial^2 u}{\partial t^2} = -\kappa^2 \nabla^4 u + c^2 \nabla^2 u - 2\sigma \frac{\partial u}{\partial t} + b_1 \frac{\partial}{\partial t} \nabla^2 u + f(x, y, t), \tag{8}$$

where $u(x, y, t)$ is the transverse plate deflection as before, $\kappa^2 = \frac{Eh^2}{12\rho(1-\nu^2)}$ denotes the stiffness parameter, where E, h, ρ, and ν are Young's modulus, plate thickness, density, and Poisson's ratio respectively, and are assumed to be constant. The term ∇^4 is the biharmonic operator. The term involving the parameter c represents a contribution to

PE type	$N_{comp.}$	$N_{comm.}$	L_{comp}	$L_{mov.}$	$L_{comm.}$
PE I (comp. rate = 1, 4-channels)	n_{points}	$2 \times (n_x + n_y)$	14	$2 \times n_x + 5$	12
PE II (comp. rate = 1, 2-channels)	n_{points}	$2 \times n_y$	14	$2 \times n_x + 5$	6
PE III (comp. rate = 1/2, 4-channels)	$n_{points} \times 2$	$2 \times (n_x + n_y)$	18	-	12
PE IV (comp. rate = 1/2, 2-channels)	$n_{points} \times 2$	$2 \times n_y$	18	-	6

TABLE IV

TYPES OF PES AND THE CORRESPONDING PERFORMANCE FIGURES

PE type	Logic slices	Block RAMs	18x18 Multipliers	Clock freq. (MHz)
PE I , 16-bit fixed-point	639	4	2	180
PE I , 24-bit fixed-point	1200	6	8	173
PE II, 16-bit fixed-point	505	6	2	182
PE II, 24-bit fixed-point	851	8	8	172

TABLE V

RESOURCE USAGE AND DEVICE CLOCK FREQUENCIES FOR DIFFERENT PE TYPES THAT CAN CALCULATE UP TO 2048 GRID POINTS

the dynamics due to constant applied tension, and the term with the parameter b_1 provides the frequency dependent damping. The PDE is of 2nd order in time and 4th order in space, therefore it needs two initial conditions, displacement $u(x, y, 0)$ and velocity $\partial u/\partial t(x, y, 0)$ and two boundary conditions at any edge which can be one of three types clamped, pinned and free boundary conditions [14] which are given in more detail in [13]. The explicit recursion FD scheme for the plate model is shown below,

$$u_{i,j}^{n+1} = \eta \sum_{|k|+|l|\leq 2} \beta_{|k|,|l|} u_{i+k,j+l}^{n} + \eta \sum_{|k|+|l|\leq 1} \gamma_{|k|,|l|} u_{i+k,j+l}^{n-1} + \Delta t^2 f_{i,j}^{n}, \tag{9}$$

where $\eta = 1/1+\sigma\Delta t,\ \mu = \kappa\Delta t/\Delta x^2$ and $\beta_{0,0} = 2-20\mu^2-4(\lambda^2+\nu)$, $\beta_{0,1} = \beta_{1,0} = 8\mu^2+\lambda^2+\nu$, $\beta_{1,1} = -2\mu^2$, $\beta_{0,2} = \beta_{2,0} = -\mu^2$, $\gamma_{0,0} = -1+4\nu+\sigma\Delta t$, $\gamma_{0,1} = \gamma_{1,0} = -\nu$, and $\mu = \kappa\Delta t/\Delta x^2$, $\lambda = c\Delta t/\Delta x$, $\eta = 1/(1+\sigma\Delta t)$ and $\nu = b_1\Delta t/\Delta x^2$. The stability condition for this scheme [16] is

$$\Delta x^2 \geq 2b_1\Delta t + c^2\Delta t^2 + \sqrt{(2b_1\Delta t + c^2\Delta t^2)^2 + 16\kappa^2\Delta t^2}. \tag{10}$$

PE network	$N_{comp.}$	$N_{comm.}$	$N_{overhead}$	N_{total}	f_{update} (kHz)	t_{1s} (s)	Pentium-M perf. (s)	C6415 perf. (s)
I (PE type I)	2000	180	131	2311	73.5	0.60	10	7.39
II (PE type II)	2000	400	45	2445	69.5	0.63		

TABLE VI

PERFORMANCE RESULTS OF THE IMPLEMENTATIONS

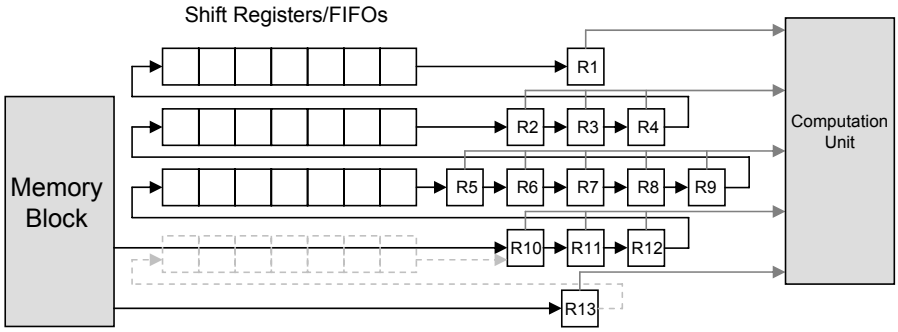


Fig. 15. Moving window for the plate-based applications

The discretised forms of the boundary conditions are listed as below.

$$u^n_{i,j} = \delta_{n0} u = 0, \quad (clamped) \tag{11}$$

$$u^n_{i,j} = \delta^2_n u = 0, \quad (pinned) \tag{12}$$

$$\delta^2_n u + \nu \delta^2_s u = \delta_{n0}\delta^2_n u + (2-\nu)\delta_{n0}\delta^2_s u = 0, \quad (free) \tag{13}$$

where n and s refer to the coordinates normal and tangential to the edge on the boundary. The stencil of the FD scheme that determine the spatial data dependencies are shown in Fig. 12(b).

B. FPGA implementation

Application parameters were given earlier in Table I. As with the membrane example, the time $(1/\Delta t)$ and spatial $(1/\Delta x)$ sampling rates determine the computational and storage needs. The stability condition gives the upper bound on $1/\Delta x$ which should be high to reduce the dispersion and increase accuracy.

1) Memory Structure: Unlike the membrane, the $(n+1)th$ step result cannot be written over the $(n-1)th$ value, so 3 memory blocks (MemA, MemB, and MemC) are needed. A similar structure to Fig. 13 is used with a reduced register cost (Fig. 15), shown in shadow, by allowing multiple separate address generation units to be used as shown completely in Table VII. Table VIII lists the memory access types and corresponding n_{cycles_mem} values for the plate model. In the table, the moving window structure for accessing MemA and MemB are labelled as MWA and MWB respectively. As access to MemC involves only 1 write operation, its number of ports does not affect the value, n_{cycles_mem}.

2) Computation Unit: As before, the minimum II values depend on the memory access schemes (Table VIII), so II is taken as 1, 2, 3, 4, 5 and 7 respectively with the best performance schedules (2, 3 and 7) for a given number of multipliers (3, 2 and 1), shown in Table IX. Adder and multiplier latency are 1 and 2 clock cycles, respectively. The possible schedule and operation allocation in Fig. 16 results in a datapath architecture consisting of units whose inputs are multiplexed and outputs fed to the shift registers whose lengths depend on the particular schedule.

Moving window type	No. of SRs/FIFOs	No. of address generations	No. of clock cycles	
			Single-port mem.	Dual-port mem.
MW I	4	1	1	1
MW II	3	2	2	1
MW III	2	3	3	2
MW IV	1	4	4	2
MW V	-	5	5	3

TABLE VII

LIST OF MOVING WINDOW STRUCTURES AND THE CORRESPONDING MEMORY ACCESS TYPES

MemA	MemB		
	MWB	2-port	1-port
MWAI/MWAII (2-port)	1	3	5
MWAII (1-port)/MWAIII (2-port)/MWAIV (2-port)	2	3	5
MWAIII (1-port)/MWA V (2-port)	3	3	5
MWA IV (1-port)	4	4	5
MWA V (1-port)	5	5	5
2-port	7	7	7

TABLE VIII

NO. OF CLOCK CYCLES FOR DIFFERENT MEMORY ACCESS TYPES FOR THE PLATE MODEL WITH 2-PORT MEMC

Clock cycle	Schedule 1 (II=2)	Schedule 2 (II=3)	Schedule 3 (II=7)
1	Add1, Add2, Add3, Add7, Mult1	Add1, Add2, Add7	Add1, Add7
2	Add4, Add5, Add6, Add8, Add9, Mult2	Add3, Add4, Add8, Add9, Mult1	Add2, Add8
3	Add10, Add11, Mult3	Add5, Add6, Add10, Add12, Mult2, Mult3	Add3, Add9, Mult2
4	Add12, Add13, Mult4, Mult5	Add11, Mult4, Mult6	Add4, Add12, Mult3
5	Add14, Mult6	Add13, Mult5	Add5, Add10, Mult6
6	Add15	Add14, Add15	Add6, Mult4
7	Add16	Add16	Add11, Mult1
8	Add17	Add17	Mult5
9	Add18	no. op.	Add15
10	write $u_{i,j}^{n+1}$	Add18	Add13
11		write $u_{i,j}^{n+1}$	Add14
12			Add16
13			Add17
14			Add18
15			write $u_{i,j}^{n+1}$

TABLE IX

THE SCHEDULES FOR THE PLATE MODEL

Clock Cycle — Operational Units

	A1	A2	A3	A4	A5	A6	A7	A8	A9	M1	M2	M3
1	Add1	Add2	Add3	Add10	Add11	Add7	Add16	Add18	Add14	Mult3	Mult6	Mult1
2	Add4	Add5	Add6	Add9	Add12	Add8	Add15	Add17	Add13	Mult4	Mult5	Mult2

a) II = 2

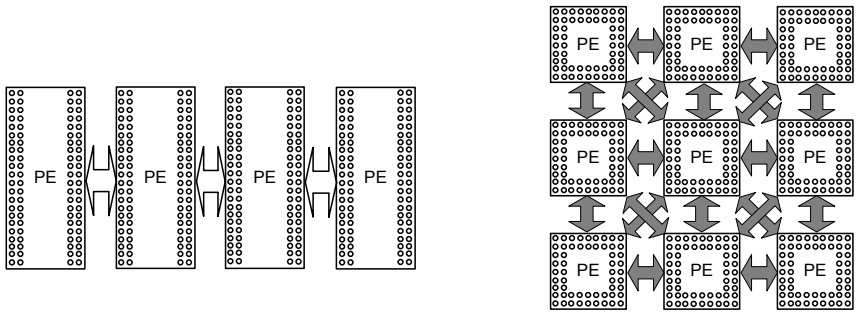
	A1	A2	A3	A4	A5	A6	M1	M2
1	Add1	Add2	Add11	Add16	Add7	Add18	Mult6	Mult4
2	Add3	Add4	Add9	Add17	Add8	Add13	Mult1	Mult5
3	Add5	Add6	Add10	Add15	Add12	Add14	Mult2	Mult3

b) II = 3

	A1	A2	A3	M1
1	Add1		Add7	Mult5
2	Add2	Add15	Add8	
3	Add3	Add9	Add13	Mult2
4	Add4	Add12	Add14	Mult3
5	Add5	Add10	Add16	Mult6
6	Add6	Add17		Mult4
7	Add11	Add18		Mult1

c) II = 7

Fig. 16. Kernel parts for the schedules in table 11



(a) 1-D (b) 2-D

Fig. 17. Communication patterns corresponding to 1-D and 2-D domain decompositions

3) Interface Unit: In 1-D domain decomposition, 2 communication channels are needed with one send and receive buffer for each channel (Fig. 17(a)) and buffer sizes of $2 \times n_x \times bit\text{-}width$ (vertical) or $2 \times n_y \times bit\text{-}width$ (horizontal). In 2-D, 8 communication channels (Fig. 17(b)) with buffer sizes for channels 1 and 2 of $2 \times n_x \times bit\text{-}width$, 3 and 4 of $2 \times n_y \times bit\text{-}width$ and single registers for channels 5 to 8 as they involve only a single point. Four extra channels are needed to transfer the data for the $(n-1)th$ time step with sizes $n_x \times bit\text{-}width$ for 1 and 2 and $n_y \times bit\text{-}width$ for 3 and 4. In order to halve the number of buffers in the interface unit, only receive buffers can be included with data to be sent being read directly from the memory blocks.

4) List of PE types: Table X shows the performance figures for two PE architectures, PE types I and II that have been designed to implement the plate model.

A grid size of 100x100 with 'clamped' boundary conditions, and a time sampling rate of 44.1 kHz was implemented on a Xilinx Virtex II Pro, XC2VP50 device (Table XI), a Pentium-M 1.6GHz processor and a TI

PE type	$N_{comp.}$	$N_{comm.}$	L_{comp}	$L_{mov.}$	$L_{comm.}$
PE I (comp. rate = 1/7, 4-channels interface)	$n_{points} \times 7$	$4 \times (n_x + n_y - 4)$	24	-	12
PE II (comp. rate = 1/3, 2-channels interface)	$n_{points} \times 3$	$6 \times n_y$	18	$3 \times n_x + 13$	6

TABLE X

THE PES FOR THE PLATE MODEL AND THE CORRESPONDING PERFORMANCE FIGURES

PE network	$N_{comp.}$	$N_{comm.}$	$N_{over.}$	N_{total}	f_{update} (kHz)	t_{1s} (s)	Pent. (s)	C6415 (s)
I (25 by 25)	2800	144	36	2980	60.4	0.73	32	5.97
II (10 by 100)	3000	600	67	3667	49.1	0.89		

TABLE XI

PERFORMANCE RESULTS OF THE IMPLEMENTATIONS

C6415 DSP processor running at 720 MHz. The details of the Pentium-M and TI C6415 processor implementations are the same as in the previous example in Section VIII.B. Network I is made up of 25 PEs each processing 400 grid points, arranged as a 5x5 mesh and Network II has 10 PEs, each processing 1000 grid points, arranged as an array. The results show that better than real-time performance is achievable on an average FPGA device without using its full resources.

X. CONCLUSION

A design methodology for implementing a range of FD schemes along with examples and application details have been presented. The results show that real-time performance is achievable for both membrane and plate examples and sets the scene for allowing an FPGA-based digital instrument to be realised using these models which can be driven in a number of different ways. Work has been carried out on FPGA accelerators for speeding up explicit FD algorithms in the area of 2-D seismic wave propagation [17] and 3-D FDTD calculations [18] and modelling sound synthesis involved 1-D waveguides [19] but this work represents the first attempt at a systematic design approach for music applications.

REFERENCES

[1] X. Serra, "Current perspectives in the digital synthesis of musical sounds," *Formats*, vol. 1, 1997.

[2] N. Castagne and C. Cadoz, "10 criteria for evaluating physical modelling schemes for music creation," in *Proc. of the 6th Int. Conference on Digital Audio Effects (DAFX-03)*, London, UK, September 2003.

[3] G. Eckel, F. Iovino, and R. Causse, "Sound synthesis by physical modelling with modalys," in *Proceedings of the International Symposium on Musical Acoustics*, Dourdan, France, 1995, p. 479.

[4] M. Karjalainen, C. Erkut, and L. Savioja, "Compilation of unified physical models for efficient sound synthesis," in *Proc. IEEE In. Conf. Acoust. Speech and Sig. Proc. (ICASSP03)*, Hong Kong, China, April 2003, pp. 433–436.

[5] L. N. Trefethen, "Finite difference and spectral methods for ordinary and partial differential equations," 1996, available online at http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/pdetext.html.

[6] S. Schedin, C. Lambourg, and A. Chaigne, "Transient sound fields from impacted plates: Comparison between numerical simulations and experiments," *Journal of Sound and Vibration*, vol. 221, pp. 471–490, 1999.

[7] B. Kienhuis, E. F. Deprettere, P. van der Wolf, and K. A. Vissers, "A methodology to design programmable embedded systems - the y-chart approach," in *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*. London, UK: Springer-Verlag, 2002, pp. 18–37.

[8] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: Formal models, validation and synthesis," *Proceedings of the IEEE*, vol. 85, no. 3, pp. 336–390, March 1997.

[9] I. Foster, "Designing and building parallel programs," online book, available from "http://www-unix.mcs.anl.gov/dbpp/".

[10] B. R. Rau, "Iterative modulo scheduling: An algorithm for software pipelining loops," in *Proceedings of the 27th Annual International Symposium on Microarchitecture*, November 1994, pp. 63–74.

[11] ——, "Iterative modulo scheduling," Hewlett-Packard Company, Tech. Rep., November 1995.

[12] E. Motuk, R. Woods, and S. Bilbao, "Implementation of finite difference schemes for the wave equation on fpga," in *Proc. of IEEE Conf. On Acoustics, Speech and Signal Processing (ICASSP05)*, Philadelphia, USA, April 2005, pp. 237–240.

[13] ——, "Parallel implementation of finite difference schemes for the plate equation on a fpga-based multi-processor array," in *Proc. of 13th European Signal Processing Conference (EUSIPCO 2005)*, Antalya, Turkey, September 2005.

[14] K. F. Graff, *Wave Motion in Elastic Solids*. New York, USA: Dover, June 1991.

[15] A. Chaigne and C. Lambourg, "Time-domain simulation of damped impacted plates. i. theory and experiments," *The Journal of the Acoustical Society of America*, vol. 109, p. 1422, 2001.

[16] S. Bilbao, "Finite difference schemes for plate synthesis," in *Proceedings of the International Computer Music Conference*, Madrid, Spain, 2005.

[17] C. He, G. Qin, and W. Zhao, "Time domain numerical simulation for transient wave equations on reconfigurable coprocessor platform," in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2005.*, April 2005, pp. 127–136.

[18] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, D. W. Prather, and M. S. Mirotznik, "Hardware implementation of a three-dimensional finite-difference time-domain algorithm," *IEEE Antennas and Wireless Propagation Letters*, vol. 2, no. 1, pp. 54–57, January 2003.

[19] J. Gibbons, D. Howard, and A. Tyrrell, "Fpga implementation of the 1d wave equation for real-time audio synthesis," *IEE Proceedings of Computers and Digital Techniques*, vol. 152, no. 4, pp. 619–631, August 2005.