Design and Implementation of a Two-dimensional Sound Field Solver Based on the Digital Huygens' Model

Tan Yiyu [#1], Yasushi Inoguchi[#1], Yukinori Sato[#1], Makoto Otani[#2], Yukio Iwaya[#3], Takao Tsuchiya[#4]

[#1]Research Centre for Advanced Computing Infrastructure, Japan Advanced Institute of Science & Technology

[#2] Faculty of Engineering, Shinshu University

[#3] Faculty of Engineering, Tohoku Gakuin University

[#4] Faculty of Science and Engineering, Doshisha University

Abstract

Sound field analysis is complicated and computationally intensive. In this paper, a two-dimensional sound field solver based on the Digital Huygens' Model (DHM) is designed and implemented by a Field Programmable Gate Array (FPGA). In this sound field solver, the original DHM and its boundary condition are extended to reduce operations and hardware resource consumption. The computation is completed locally, and external memory access is avoided. In a two-dimensional space with both length and width being 1.28m, when boundaries are rigid walls, the FPGA-based analysis system enhances performance from 44 to 217 times, and from 37 to 179 times against the software simulations based on the original DHM and Standard Leapfrog Finite-difference Time-domain (SLF-FDTD), respectively. Compared with the General-purpose Graphic Processing Unit (GPGPU) Tesla C1060, it speeds up by 1223 times in computation and by 114 times in overall performance in the case of time steps being 20,000. When the node scales are different and the calculated time steps are 32,000, the FPGA-based sound field solver achieves about 1,795 and 1,190 times faster in computation, 218 and 179 times enhancement in final performance over the software simulations based on the original DHM and SLF-FDTD, respectively. Furthermore, the proposed system provides high data throughput, and is easily applied in real-time applications.

Key words: Digital Huygens' Model, Sound Field Analysis, FPGA, FDTD

1. Introduction

Sound field analysis, which exhibits numerical methods to model sound propagation phenomena in spatial and time domain, is required widely in many industrial and scientific fields, such as ultra-realistic communication, virtual reality, interactive games and noise control. Generally, a sound space is discretised into small grids. The governing equations of sound propagation are numerically applied on each grid to analyse sound behaviour at discrete time steps. Currently, many numerical methods have been proposed for computer simulation, such as acoustical ray tracing [1], image source method [2], beam tracing method [3], acoustic radiosity [4]， finite element method [5], boundary element method [6], and Finite Difference Time-Domain (FDTD) method [7][8]. Moreover, various software techniques, from high-level thread parallelism on computer clusters [9] to low-level memory and disk optimizations [10] have been developed to accelerate the execution of these simulation tasks. However, current computers are confronted with the difficulty of improving performance at a reasonable cost due to their inefficient hardware-resource utilization. During simulation, huge amounts of data are read from and written into the external memory system. Since no special tailored hardware units are used to accelerate these data access, memory system does not work effectively due to the intensive data requirements and limited memory bandwidth. The simulation is time-consuming even if computers are much faster. Although supercomputers and large computer clusters may eliminate these disadvantages and gain a parallel processing speedup [9][11], they are prohibitively expensive.

In order to eliminate the limitation of memory bandwidth and reduce data access to external memory, GPGPUs [12][13] and FPGAs are currently applied to speed up such type of data-oriented applications [14][15][16][17] by completing all computations inside chips. In a GPGPU, several streaming multiprocessors are connected to each other with a fast interconnection network. Inside a streaming multiprocessor, multiple streaming processors work in parallel to speed up computation, and most of data are exchanged in the on-chip shared memory rather than the external memory. But the GPGPU based solutions are not easily applied for real-time applications because there are no external I/O connectors in general GPU boards and the calculated results require further post-processing. For example, in a virtual audio system, many channels are output at real-time. A sound synthesizer is needed to process the calculation results of GPGPUs.

In the FPGA-based sound field analysis systems, the propagation equations of sound wave are directly implemented by the Configurable Logic Blocks (CLB), and data are kept by the D flip-flops and block memory units inside FPGA chips. By cascading hundreds of arithmetic units together, and coordinating them to work in parallel, FPGA-based sound field analysis system may achieve much higher computation performance than the software simulations on generic computer systems. Since the utilization and interconnection of internal memory resources of a FPGA are explicitly tailored according to the system data flow, data are reused more efficiently, and the external memory access is reduced significantly or eliminated. Particularly, the calculation results can be output by the digital to analogue converters (D/A) directly for real-time applications.

In this paper, a FPGA-based two-dimensional sound field solver is introduced and evaluated. This solver is based on the systolic architecture. It has regularity, locality, and parallelism in computing. The present study mainly discusses the system design and implementation. This design is based on a programmable hardware structure fully tailored for the sound field analysis. Major contributions of this work are as follows:

1) A parallel systolic architecture to avoid external memory access during computation and exploit both the spatial and temporal parallelism of computation for scaling up the system performance. The whole system architecture and function modules inside it are introduced. Design issues are presented. Data flow, memory requirement, and computing flow are analysed.

2) Extension of the original DHM and its boundary condition to reduce operations and hardware resource consumption.
3) Design and implementation of the FPGA-based prototype systems with different node scales, which achieve significant speedups in computations, data throughput, and final performance.
4) Detail analysis and evaluation of system performance based on the prototype system. The hardware resource consumption of the whole system is presented. System performance under two cases is evaluated and analysed, including calculation time, data transfer speed, speedup in computation, and data throughput.

The rest of this paper is organized as follows. The related work is summarized in Section 2. The DHM and its extension are introduced briefly in Section 3. The system design, including system architecture, computing cell design, memory requirement, and computation flow, is described in Section 4. The system performance, such as hardware resource consumption, execution time, and data throughput is discussed in Section 5. Finally, conclusions are drawn in Section 6.

2. Related Work

In FPGA-based sound field solvers, the analysis algorithm and system architecture affect system performance significantly. Since hardware resources are limited inside a FPGA, the analysis algorithm should be as simple as possible to reduce hardware resource consumption. In recent years, many simple algorithms have been derived from the propagation equations of sound waves to investigate acoustical behaviour, such as the DHM, the Digital Waveguide Mesh (DWM)[18][19], and Transmission Line Matrix (TLM)[20][21][22]. The DWM method was in principle a FDTD scheme based on digital signal processing technology. Based on the FDTD algorithm, the compact FDTD schemes were derived [23]. Although some of them are simple and efficient, such as nine-point INT, it requires more data during calculation, which results in increasing of memory bandwidth. The TLM method was a physical equivalent method to analyse sound behaviour in the time domain. The original DHM, proposed by Y. Kagawa and T. Tsuchiya [24][25][26][27], was derived from the TLM method. Compared with other algorithms, the DHM is simpler and more easily implemented by hardware.

As FPGAs have been integrated more and faster hardware resources of CLBs, DSPs, and block RAMs, they have been used for acceleration of numerical simulations based on the FDTD algorithm in thermal propagation, computational fluid dynamics (CFD), electromagnetic analysis, and so on. F. Pardo proposed a FPGA-based solver for thermal simulation [28]. K. Sano, W. D. Smith, and H. Morishita explored FPGA-based accelerators for CFD applications [29][30][31]. Among them, K. Sano researched the systolic computational memory architecture to eliminate the bottleneck of memory bandwidth during computation. W. D. Smith developed baseline system architecture for accelerating the three of the most computation-intensive algorithms in CFD applications by using FPGA. H. Morishita designed FPGA-based arithmetic pipelines with a customized memory system to speed up some CFD subroutines. In addition, C. He and M. shafiq introduced the optimized memory architectures to exploit data reuse on a reconfigurable computing platform for applications based on the finite difference method [32][33]. J. P. Durbano, W. Chen, and R. N. Schneider investigated the FPGA-based FDTD solutions for electromagnetic problems [34][35][36]. Erdem Motuk developed the FPGA-based sound synthesized system based on explicit finite-difference algorithm [37]. In this paper, a FPGA-based solver with parallel architecture is proposed for sound field analysis, where massive data are required along with the numerical algorithms. To reduce hardware resource and improve system performance, the original DHM and its boundary condition are extended. The resolver is easily used for real-time applications.

3. Two-dimensional DHM Algorithm

3.1 Original scheme

In the two-dimensional DHM, a sound space is treated as a grid mesh with each grid being same length Δl. Since the impedance is discontinuous at the connecting node, when a pulse with magnitude being P is incident to a node (Fig. 1a), the sound pulse scatters in four directions as shown in Fig. 1b. This procedure is equivalent to the travelling and scattering of a voltage pulse over an orthogonal mesh made of transmission lines. If the impedance of one direction branch is $Z_{0,}$ the equivalent impedance from one direction branch to the other three branches at the node is $Z_0/3$ because the three branches are connected in parallel. Then the reflection coefficient at the node is given by

$$\Gamma = \frac{Z_0/3 - Z_0}{Z_0/3 + Z_0} = -\frac{1}{2}$$

Thus the pulse with magnitude being $1/2P$ is transmitted and reflected back into the adjacent nodes and the incident node, respectively. When four directions are incident, the scattering matrix is obtained and shown in equation (1) [15][24][25][26], where S_m^n and P_m^n (the subscript m denotes the four directions 1, 2, 3 and 4) are the scattered pulses and incident pulses in different directions at time t, respectively. Time t equals $\frac{n\Delta l}{C_T}$ (n =0,1,2,...), where C_T is the propagation speed of sound, Δl is the grid length, and n is the discrete time steps.

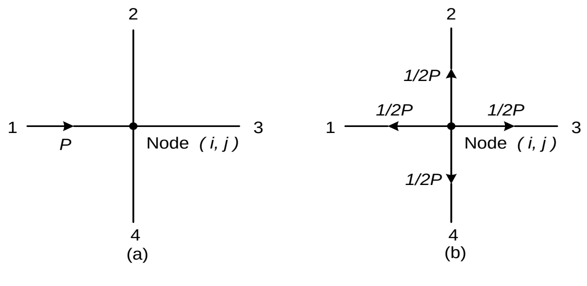

Fig. 1 Two dimensional DHM (a) An incident pulse (b) Scattered pulses

$$\begin{bmatrix} S_1^n \\ S_2^n \\ S_3^n \\ S_4^n \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -1 & 1 & 1 & 1 \\ 1 & -1 & 1 & 1 \\ 1 & 1 & -1 & 1 \\ 1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} P_1^n \\ P_2^n \\ P_3^n \\ P_4^n \end{bmatrix} \qquad (1)$$

Since the coefficient matrix is symmetrical, from the equivalent principle, we have

$$\sum_{m=1}^{4} S_m^n(i,j) = \sum_{m=1}^{4} P_m^n(i,j) \qquad (2)$$

The sound pressure at the node (i, j) is denoted by $P^n(i,j)$ and defined as $P^n(i,j) = \frac{1}{2}\sum_{m=1}^{4} P_m^n(i,j)$ for lossless propagation. Then equation (1) is rewritten as

$$S_m^n(i,j) = P^n(i,j) - P_m^n(i,j) \ (m = 1,2,..,4) \qquad (3)$$

In Fig. 1(b), the scattered pulses at a node are the incidences to the corresponding directions of its neighbours at the next time step. The relations between a node and its neighbours are shown in equation (4):

$$\begin{aligned} P_1^{n+1}(i,j) &= S_3^n(i-1,j) \\ P_2^{n+1}(i,j) &= S_4^n(i,j+1) \\ P_3^{n+1}(i,j) &= S_1^n(i+1,j) \\ P_4^{n+1}(i,j) &= S_2^n(i,j-1) \end{aligned} \qquad (4)$$

Equation (4) shows the mesh connection topology for general nodes. When a node is on a boundary, its scatterings and incidences are affected by the characteristics of the boundary medium. If a boundary is reflective, and its acoustical reflection coefficient is *R*, equation (5) shows the relation between the scatterings and incidences at a node on the boundary.

$$P_m^{n+1}(i,j) = R * S_m^n(i,j) \ (m = 1,2,..,4) \qquad (5)$$

R equals 1 for a rigid boundary while it is $\frac{1-\sqrt{2}}{1+\sqrt{2}}$ in the case of a fully-absorbing boundary [15]. Thus according to equations (1), (3), (4), (5), and the initial incidence, sound pressures of all nodes are calculated through time iteration based on the time step $\frac{\Delta l}{C_T}$. From equations (1) and (3), eight operations, namely three additions, four subtractions, and one right-shift operation, are required to calculate the sound pressure and scatterings at a general node. When a node is on a boundary, an additional multiplication operation is required. During calculation, the scatterings and incidences in four directions are kept for further calculation.

3.2 Extension of the DHM

The formula of sound pressure $P^n(i,j)$ is rewritten by inserting equation (4) and then eliminating the scatterings according to equation (3).

$$\begin{aligned} P^n(i,j) = \frac{1}{2}(&P^{n-1}(i+1,j) - P_1^{n-1}(i+1,j) + P^{n-1}(i-1,j) - P_3^{n-1}(i-1,j) + P^{n-1}(i,j+1) \\ &- P_4^{n-1}(i,j+1) + P^{n-1}(i,j-1) - P_2^{n-1}(i,j-1)) \end{aligned} \qquad (6)$$

By inserting equation (4), and replacing $P_m^{n-1}(i,j)$ with the related scatterings, equation (6) is expressed as

$$P^n(i,j) = \frac{1}{2}(P^{n-1}(i+1,j) + P^{n-1}(i-1,j) + P^{n-1}(i,j+1) + P^{n-1}(i,j-1) - \sum_{m=1}^{4} S_m^{n-2})) \qquad (7)$$

Equation (7) is further rewritten as equation (8) by using equation (2) [15][16].

$$P^n(i,j) = \frac{1}{2}(P^{n-1}(i+1,j) + P^{n-1}(i-1,j) + P^{n-1}(i,j+1) + P^{n-1}(i,j-1)) - P^{n-2}(i,j) \qquad (8)$$

Equation (8) is the same as the FDTD expression with the standard leapfrog stencil for two-dimensional wave propagation [18]. To calculate the sound pressure of a general node requires three additions, one subtraction, and one right-shift operation. Thus three subtractions are reduced against the original DHM scheme. If a node *(i, j)* is on a rigid wall boundary, for example, in the direction 3, due to full reflection,

$$P_3^{n+1}(i,j) = S_3^n(i,j) \qquad (9)$$

By using the same derivation procedure, equation (10) is obtained to calculate the sound pressure.

$$P^n(i,j) = \frac{1}{2}(P^{n-1}(i-1,j) + P^{n-1}(i,j) + P^{n-1}(i,j+1) + P^{n-1}(i,j-1)) - P^{n-2}(i,j) \qquad (10)$$

If nodes are at corners, for example, the corners of the east and north boundaries, equation (11) is derived and applied to calculate the sound pressure.

$$P^n(i,j) = \frac{1}{2}(P^{n-1}(i-1,j) + P^{n-1}(i,j-1) + 2 * P^{n-1}(i,j)) - P^{n-2}(i,j) \qquad (11)$$

Equations (10) and (11) contain only additions, subtractions, and shift operations, and the complex multiplication operations are eliminated. Equations (8), (10), and (11) show that calculating sound pressure of a node only requires the sound pressures of its neighbours at previous time steps.

4. System Design

4.1 System architecture

Although FPGA-based computing engines appear to be attractive for sound field analysis system based on the DHM, the following considerations are taken into account during design and implementation.

(1) System architecture and the physical constraints of the FPGA hardware resources. Once the analysis algorithm is determined, the system architecture affects the hardware resource consumption significantly, particularly system data path. Thus full analysis of the data flow in the DHM is needed to design the computing engine. This not only helps to better understand the DHM algorithm and to exploit the system parallelism, but also optimizes system to reduce hardware resource consumption.

(2) Data transfer speed and data throughput. Data transfer between FPGA and the host computer, and data communication during computation affect system performance. Thus the analysis of the data transfer speed and data throughput is required during implementation.

Based on these considerations, a two-dimensional sound field solver based on the DHM is designed, and its architecture is shown in Fig. 2, which consists of six modules, namely DHM2D, Controller, Incidence ROM, Block RAM, PLX_IF, and PCI Controller PCI9054. The DHM2D is the DHM computing engine, which calculates sound pressures of all nodes at each time step according to the incidence pulses stored into the Incidence ROM. The sound pressure of the observation node is stored into the Block RAM. The computation flow is managed by the Controller. The PLX_IF is the bus interface between the DHM system and the PCI controller PCI 9054. Once the calculation is finished, the calculated results are dumped out from the block memory to the host computer through the compact PCI bus. Thus during computation, data flows from the DHM2D, the block RAM, PLX_IF, and finally to the host computer through the PCI bus controller. In Fig. 2, the modules enclosed by dashed line are implemented by FPGA.

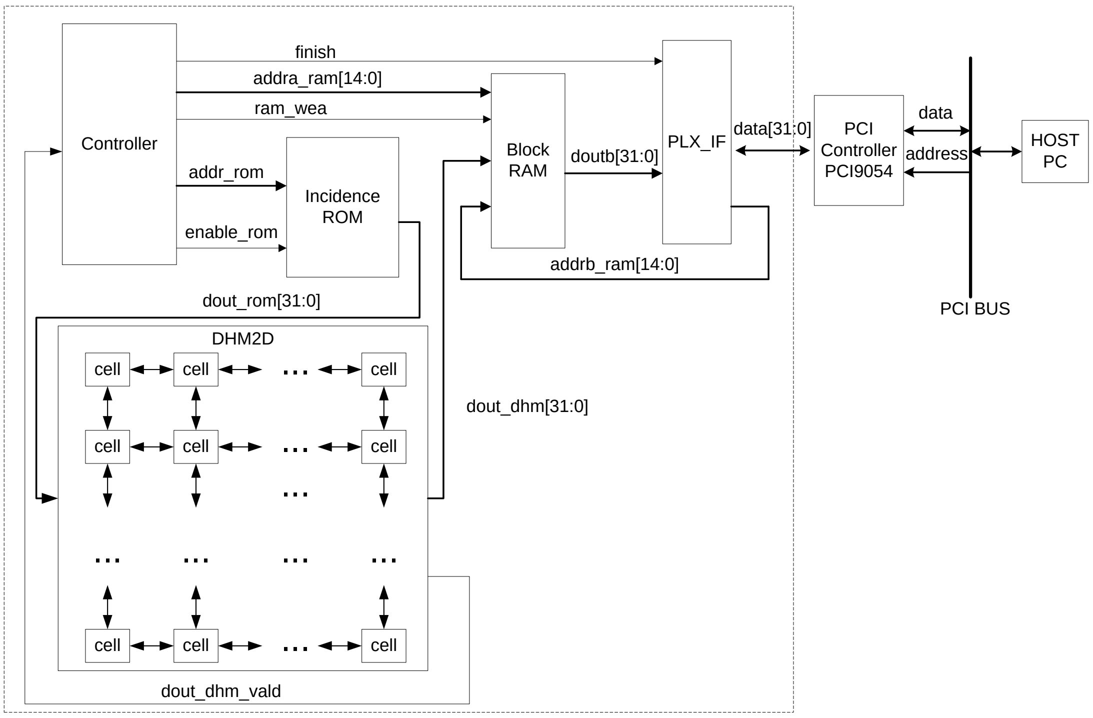

Fig. 2 System diagram

4.1.1 DHM2D

As shown in Fig. 2, the DHM2D is based on a systolic architecture [38][39], where a uniform computing cell exists at each node to calculate its sound pressure, and all computing cells are arranged regularly like an array. This architecture provides a high degree of modularity, regularity, and localized data communication. It has potential to perform computations with high efficiency and robustness. In such architecture, all computing cells work concurrently to speed up computation, and data are kept by the local memories, such as D flip-flops. During computation, data are locally processed and synchronously flow across the array between neighbours. The external memory access is therefore avoided during computation and the performance bottleneck resulting from the memory bandwidth is eliminated. Furthermore, this systolic architecture is suitable for spatially parallel processing, and achieves scalable computing performance by extending the scale of the computing cells. It matches well with the state-of-the-art FPGAs that consist of CLBs and embedded block RAMs. CLBs are arranged like an array, and connected to each other through the programmable routing matrix. Thus a systolic architecture is suitable for high CLBs utilization since computing cells in it are arranged regularly, and each cell is easily implemented by one or more CLBs, and connected together through the inter-connectors inside a FPGA chip. In addition, the locality of the cell control and communication leads to high operation frequency of system. All these lead the DHM2D to achieve high performance in computation.

The DHM2D is the computing unit. At a time step, an incident data (*dout_rom*) is read from the Incidence ROM, computations are then carried out, and finally the sound pressures of all nodes are updated. Since the calculation result of a computing cell is the input of its neighbour cells in the next time step, temporal dependence does not appear in computation in an iteration period. According to equation (8), updating the sound pressure of a node only requires the previously calculated results, and the data flow for such updating is shown in Fig. 3. Due to no data dependency existing during updating, all computing cells are easily cascaded together and work in parallel. Therefore the computing cell structure and the number of the computing cells have a great impact on the system performance and hardware resource consumption. Typically, the number of computing cells is determined by the dimensions of the sound space and the grid size Δl. The computing cell structure is determined by the sound field analysis algorithm and the design techniques.

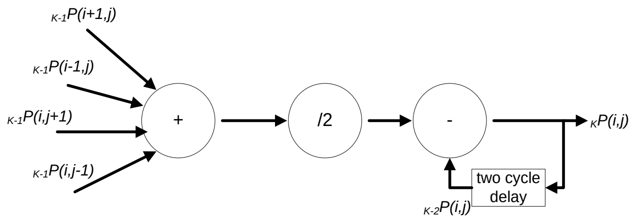

Fig. 3 Data flow for updating the sound pressure of a node

In the DHM2D, when the system clock frequency becomes high, data communication and timing control between computing cells are big challenge. A common technique is to use a hand-shaking signal between computing cells. When a computing cell finishes calculations, its output is updated and the handshaking signal is set. Then its neighbours read the updated results and carry out computations. Although data throughput and timing are improved in this solution, the calculation is inefficient. If the output of a computing cell is not updated on time, all calculations in other computing cells will be suspended gradually. To avoid this and simplify the design of the DHM2D, all operations inside a computing cell are limited to be completed in a clock cycle. Thus all computing cells finish computations and update their outputs synchronously. At the next time step, computing cells read the related data from their neighbours directly, and hand-shaking signals are not required. Although this solution may result in relatively worse timing performance over other solutions, the hardware system is simple, and computing is not suspended. Based on this design issue and equation (8), a computing cell shown in Fig. 4 is designed, which consists of a four-input 32-bit adder, a three-input 32-bit adder, a 32-bit Not gate, and two 32-bit Flip-flops. On average, each computing cell consumed 130 Look-up Tables (LUTs) and 68 D Flip-flops in the FPGA chip XC5VLX330T-FF1738 [15][16].

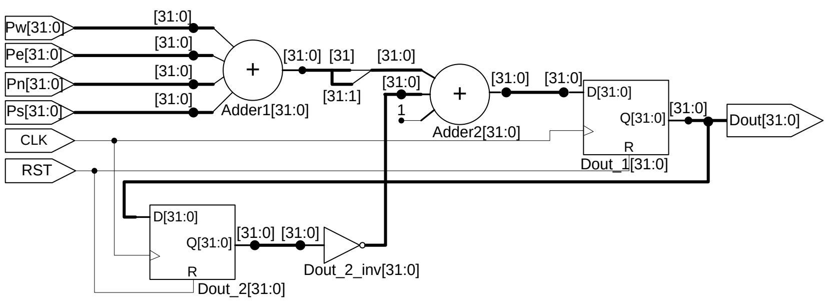

Fig. 4 Structure of a computing cell

4.1.2 Memory requirement

In Fig. 2, D flip-flops in the DHM2D, the Incidence ROM, and the Block RAM are required to store the temporary data, the incidence, and the calculation results, respectively. In Fig. 4, the calculated sound pressure of a node is kept by two 32-bit D Flip-flops for computations in the next time step. If a DHM system contains N × M nodes, 2NM 32-bit D flip-flops are required. The overhead to access data from D flip-flops is much shorter, and the bottleneck of performance improvement due to the bandwidth of external memory is eliminated by accessing data from D Flip-flops in parallel.

The Incidence ROM is used to store the external incident data. Firstly, the incident signal is discretised according to the sample rate. Then with the discretization data, the Incidence ROM is generated by the tool CORE Generator provided by the Xilinx ISE. The sound pressures at the observation point are stored in the Block RAM, and finally output to the external host PC through the compact PCI bus. Since the DHM2D operates on 100MHz, and the PCI controller operates on 50MHz, the block RAM also acts as a synchronization buffer of the data stream. The size of the Block RAM is determined by the calculated time steps, and data-width. In the current system, data are 32-bit fixed-point, and the Block RAM is 128KB in size. If the analysis system is used for real-time applications, the calculation results are output by the D/A board directly, and the Block RAM is not required.

4.1.3 Controller and PLX_IF

The Controller is the system control centre. It generates the related data and control signals according to the system timing diagram and computation flow, such as the reading address of the Incidence ROM (addr_rom), the writing address of the Block RAM (addra_ram), the writing/reading enable signal (ram_wea), and the computation finish flag. The PLX_IF is an interface between the sound field solver and the compact PCI controller PCI9054. Through it, the calculation results are transferred to the external host computer.

4.2 Computation Flow

The calculation flow is based on an iteration procedure, and shown in Fig. 5. After system initialization, an excited datum is read from the Incidence ROM, then calculation is started in the DHM2D, and the sound field of the observation point is stored in the Block RAM. Then the reading addresses of the Incidence ROM and Block RAM are increased by 1. At the next time step, another incident datum is read, and computations occur again. This procedure is repeated until the calculated time steps are over.

Once all calculations are finished, a flag signal (*finish*) is set, and the Incidence ROM and the DHM2D are disabled. At the same time, the system delays three cycles to ensure the last calculation result is stored into the Block RAM. Finally, the PLX_IF generates the reading address of the Block RAM, and all the calculation results are dumped out to the host computer through the compact PCI bus by DMA mode.

5. System Performance

To verify and estimate the performance of the proposed sound field solver, the sound propagation in a small two-dimensional sound space shown in Fig. 6 was examined. In Fig. 6, both the length and width of the sound space surrounded by rigid walls are 1.28m, and Δl is $4.0\times10^{-2}m$. The incident position is at the node (0, 0), and the observation point is at the node (6, 15). The hardware development environment was Xilinx ISE 12.1 and ModelSim SE 6.6C running on the Windows XP platform. For comparison, counterpart systems based on the original DHM and its extension, the SLF-FDTD were developed by C++ programming language, and executed on a personal computer (PC) with 4GB RAM and an AMD Phenom 9500 Quad-core processor running at 2.2 GHz. The operating system of the PC was Windows XP, and the development environment was Microsoft Visual Studio 2008. The reference C++ codes were compiled and optimized for the maximum speed with option of "/O2". The FPGA-based sound field solver ran at 100MHz, and data were 32-bit fixed-point while they were integer in the software simulations.

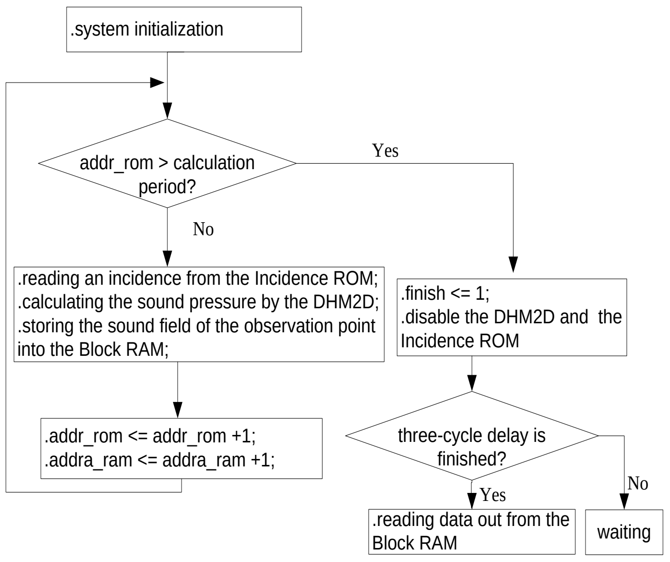
.system initialization
addr_rom > calculation period?
Yes
No
.reading an incidence from the Incidence ROM;
.calculating the sound pressure by the DHM2D;
.storing the sound field of the observation point into the Block RAM;
.finish <= 1;
.disable the DHM2D and the Incidence ROM
.addr_rom <= addr_rom +1;
.addra_ram <= addra_ram +1;
three-cycle delay is finished?
No
Yes
.reading data out from the Block RAM
waiting

Fig. 5 Calculation flow

5.1 Hardware resource consumption

The whole system shown in Fig. 2 was implemented by a processor-based FPGA machine TD-SPP3000 [40], which had a CPU board and several FPGA boards. Each FPGA board had two Xilinx XC5VLX330T-FF1738 FPGA chips, 512MB DDR RAM, and a compact PCI controller. A Xilinx XC5VLX330T FPGA chip had 51,840 slices, and each slice contained four LUTs and four flip-flops. In addition, the FPGA chip included about 3Mb distributed RAMs, and 11Mb block RAMs.

The block RAMs were used to implement the Incidence ROM and Block RAM. On the FPGA board, two FPGA chips communicated each other directly, and one was connected to the compact PCI bus, while another was attached to the Advanced Telcom Computing Architecture (ATCA) bus. The compact PCI bus was for data transfer between the FPGA board and the host PC, and the ATCA bus was mainly for data communication between different FPGA boards. The CPU board, composed of 504MB DDR RAM and an Intel Pentium M processor running at 1.4 GHz, provided an environment to debug the system and dump out the calculation results.

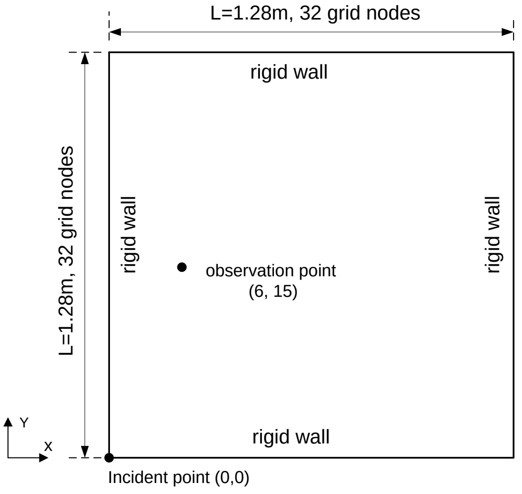

Fig. 6 A two-dimensional sound space

When data were 32-bit fixed-point, and the DHM system contained 1024 (32 × 32) nodes, the sound field solver consumed 200,233 LUTs (96%), 102,423 D flip-flops (49%), and 88 block RAMs (27%) in a Xilinx FPGA XC5VLX330T-FF1738 after implementation, and the maximum clock frequency of system was about 137MHz. Except for the modules shown in Fig. 2, the system also contained communication interface for two FPGAs on the same FPGA board, and two different FPGA boards. In the current parallel architecture, a computing cell locates at each node to speed up computation. The LUTs and D flip-flops inside a FPGA limit system scalability because they are required significantly as the node scale is increased.

5.2 Calculation time

The execution time was measured on the AMD Phenom 9500 platform as a comparison with the proposed FPGA-based sound field solver. The time was calculated by calling the functions QueryPerformanceFrequency() and QueryPerformanceCounter() in Windows XP platform with Microsoft Visual Studio 2008 environment. Then ten measured data were averaged with a precision of 1μs. The system execution performance was evaluated under two cases. One case was different DHM systems at the same time step, and another was a same system at different time steps.

5.2.1 Calculation time vs. node scale

Table I shows the execution time taken by the software simulations and the FPGA systems with different node scales at 32,000 time steps. In each solution, the computational time is shown in the left column, while the total consumed time, including the computational time and the time spent in outputting the calculation results, is listed in the right column. In the FPGA-based sound field solver, the sound pressures of all nodes are obtained and updated synchronously. All computations are completed in a clock cycle at each time step. Therefore the computation time is linearized with the time steps. When the time step is 32,000, the computation takes 3.2×10^{-4}s (32,000 cycles). However, in the software simulation, the computation time is affected by both the number of nodes and the calculated time steps since the sound pressures are calculated node by node at each time step. When the node scale is increased, the computation time becomes much longer. For example, when the node scale is increased from 10 × 10 to 32 × 32, the computational time is increased about 8.4 (0.574809/0.060857 -1) times. In Table I, the FPGA system speeds up computation significantly. When the node scale is 32 × 32, the FPGA system achieves about 1795 (0.574809/0.00032 -1) and 1190 (0.381252/0.00032 -1) times faster in computation than the software simulations based on the original DHM and SLF-FDTD algorithms, respectively.

Except for the computational time, outputting the calculation results will spend time. In the software simulation, the calculation results are written into a binary file while they are output to the host PC through the compact PCI bus by DMA mode in the FPGA system. If all these are taken into account, the FPGA system speeds up execution from 101 times to 217 times against the software simulations based on the original DHM and SLF-FDTD algorithms in the case of different node scales. On the other hand, most of the time is spent in dumping out the calculation results through the compact PCI bus in the FPGA system. When the node scale is 32 × 32, about 7.3% (0.00032/0.004344×100%) of the total time is spent in computation. The system performance is therefore influenced significantly by the data transfer speed through the compact PCI bus. Compared with the software simulations based on the original DHM and the SLF-FDTD, the FPGA system speeds up 1795 times and 1190 times in computation, but the final performance is only enhanced 218 (0.950861/0.004344 -1) times and 179 (0.783141/0.004344 -1) times. Generally, the data transfer speed through the compact PCI bus is affected by the data buffer size, driver, the clock frequency of the PCI controller, and so on. Since the CPU board has small memory (504MB), if an application utilizes high memory, the execution performance of the FPGA machine during data transfer will degrade, which results in a decrease of the data transfer speed. For example, when the anti-virus software Symantec Endpoint Protection 11.0 was run on the FPGA machine, the calculation time taken by the FPGA based rendering system with 32 × 32 nodes is about 0.05s [15] in the case of 20,000 time steps.

Table I Calculation time taken by the system with different node scale (s)

Nodes	Time steps	Software solution				FPGA system	
		Original DHM		SLF-FDTD			
10 × 10	32000	0.060857	0.447203	0.038252	0.441029	0.00032	0.004319
25 × 25	32000	0.376039	0.748787	0.237542	0.63289	0.00032	0.004351
32 × 32	32000	0.574809	0.950861	0.381252	0.783141	0.00032	0.004344

5.2.2 Calculation time vs. time steps

The calculation time of the rendering system with 32 × 32 nodes at different time steps is shown in Fig 7. The software simulation time based on the original DHM and the SLF-FDTD is almost increased linearly with the time steps. The simulation time of the SLF-FDTD is shorter than that of the original DHM due to the reduction of operations. Compared with the software simulations, the FPGA system take much shorter time in computation. As shown in Fig. 7, when the time steps are 28,000, the proposed FPGA system takes about 4.2ms while the software simulations based on the original DHM and the SLF-FDTD spend about 830ms and 680ms, respectively.

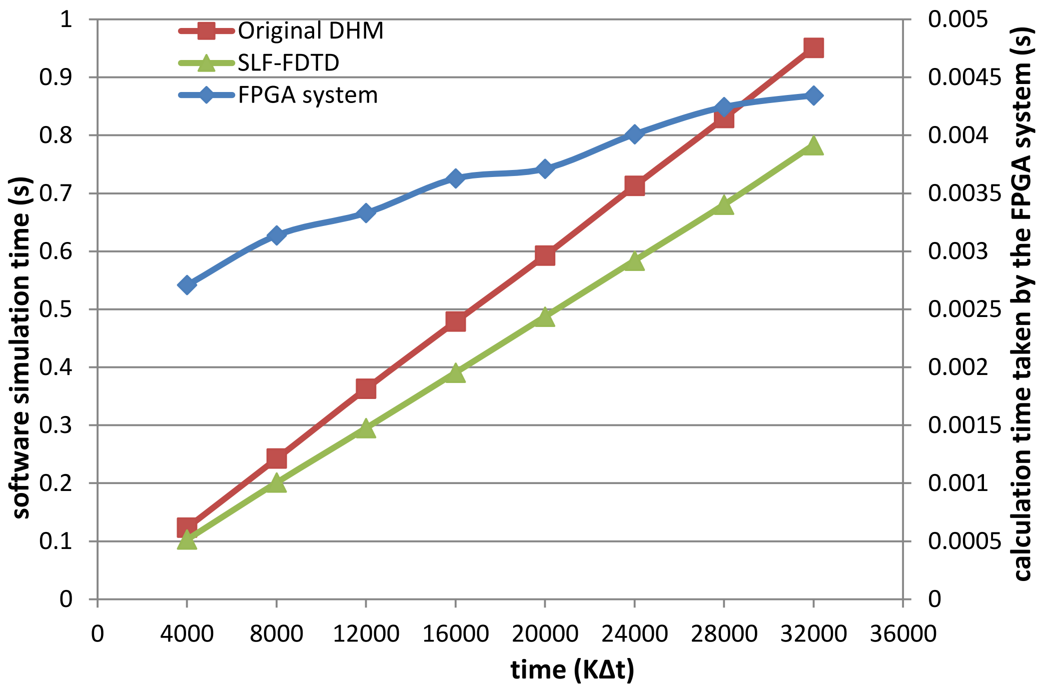

Fig. 7 Calculation time taken by the system at different time steps

As a comparison, the related system was realized by the GPGPU Tesla C1060. When the time steps were 20,000, the Tesla C1060 consumed 0.244977s in computation. But the related computation took about 2×10^{-4}s (20,000 cycles) in the FPGA based system. The FPGA based rendering system hence speeds up by about 1223 ($0.244977/(2\times10^{-4})-1$) times over the Tesla C1060 in computation. If the time spent in outputting the calculation results was taken into account, the Tesla C1060 consumed 0.428461s while the FPGA system took 0.003712s. The overall performance is then enhanced about 114 (0. 0.428461/0.003712 -1) times. In the current FPGA system, the whole system is implemented inside a FPGA. Temporary data are kept by D flip-flops and accessed in parallel. All computations in a time step are finished at a cycle, and external memory access is avoided during computation. However, although the Tesla C1060 provides on-chip shared memory for each streaming multiprocessor to store temporary data, local memory access may occur during calculation. Accessing data from D flip-flops is much faster than accessing data from local memory. Although warps technique is introduced to hide memory access latency in the Tesla C1060, the increasing of threads causes decreasing of the shared memory used by each thread. All these result in performance enhancement in the current FPGA based solver over GPGPU.

From Section 5.1, the LUTs and D flip-flops inside a FPGA limit system implementation. Thus when the sound space becomes larger, more FPGA chips are required to work in parallel to speed up computation due to the limitation of hardware resources. Data exchanges between FPGAs are then big challenge, and they are not completed at a clock cycle due to the bus bandwidth. For example, if multiple FPGAs are used, and each FPGA is applied to implement a subsystem with 32 × 32 nodes, 256 (32×4×2) byte data are exchanged between two FPGAs at a clock cycle. Thus the data transfer speed is about 25.6GB/s (256Bx100M/s). Although a Xilinx XC5VLX330T provides 24 high-speed RocketIO GTP transceivers and each is designed to run maximum 3.75Gb/s, the totally maximum transfer speed is about 11.25GB/s, which is almost half of the required data transfer speed. The data exchange between two FPGAs therefore requires more clock cycles. The same situation will also occur in the data communication between FPGA boards. Thus the calculation time will be increased and affected by the time steps and node scale in the related FPGA-based system. A compromised solution to these problems is to store the temporary data into the block memory instead of D flip-flops, and one or more computing engines are applied to calculate the sound field [41]. The simulated area by a FPGA is enlarged, but the calculation time is prolonged.

As mentioned above, the calculation time spent by the FPGA system is affected by the data transfer speed through the compact PCI bus. Fig. 8 shows the data transfer speed and the speedup of the FPGA system against the software simulations based on the original DHM and the SLF-FDTD. The calculation performance of the FPGA system is enhanced from 44 times to 217 times over the software simulation based on the original DHM while it is improved from 37 times to 179 times against the software simulation based on the SLF-FDTD. Since the computation is increased linearly with the number of nodes and time steps in the software simulations while it is linearized with the number of time steps in the current FPGA system, the speedup is increased as the time steps increase. Since DMA mode is more efficient for large data blocks in the compact PCI bus, data transfer speed becomes faster as the time steps increase, which also results in the speedup enhancement.

5.3 Data throughput

The data throughput is denoted by the number of nodes updated per second, and obtained through the following formula.

$$data\_throughput = \frac{time\_steps \times node\_scale}{computational\_time} \qquad (13)$$

From Table I, the data throughput in the sound field solver with different node scales was calculated and shown in Table II. In the software simulation, because of the limitation of memory bandwidth, the data throughput based on the same analysis algorithm is almost fixed even if their node scales are different. Compared with the software simulation, the FPGA-based solver provides much higher data throughput. When the node scale is 32 × 32 and data are 32-bit, the data throughput of the FPGA solution is 400GB/s, which is beyond the memory bandwidth of current PCs deeply. In another word, such applications are not executed efficiently in the current PCs, and will take a long time.

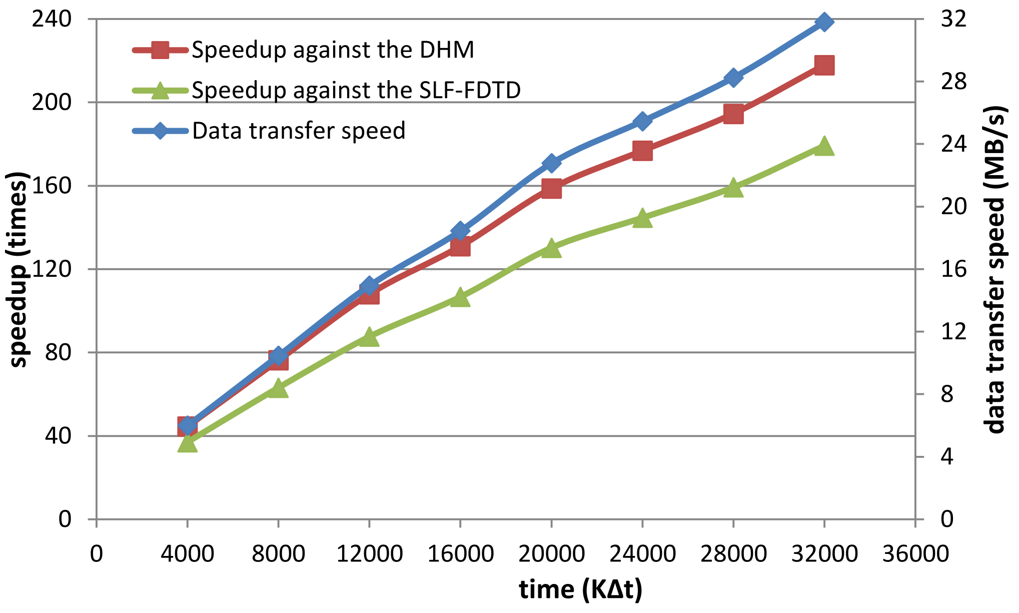

Fig.8 Speedup and data transfer speed

Table II Data throughput (Million nodes/s)

Nodes	Software solution		FPGA system
	Original DHM	SLF-FDTD	
10 × 10	52.58	83.66	1.0×10^4
25 × 25	53.18	84.20	6.25×10^4
32 × 32	57.01	85.95	1.02×10^5

6. Conclusions

Sound field analysis is a data-intensive and computation-intensive application. A two-dimensional sound field solver based on the updated DHM is developed and implemented by a Xilinx FPGA. To speed up calculation, the computing engine is based on a systolic architecture, where a uniform computing cell exists at each node to calculate its sound pressure, and all computing cells are cascaded regularly like an array. All computing cells work in parallel, and all operations are completed in a clock cycle at a time step. Owning to the parallelism of FPGA and on-chip data storage during calculation, system performance of the FPGA-based sound field solver is enhanced significantly. Compared with the software simulations carried out on an AMD Phenom 9500 platform, regardless of the different systems at same time step, or the same system at different time steps, the FPGA system has a significant performance enhancement. But the performance of the proposed FPGA-based system is influenced by the data transfer speed through the compact PCI bus. Optimized PCI driver, high clock frequency of the PCI controller, and large data buffer may result in the improvement of data transfer speed and system performance. Furthermore, the proposed system is easily applied for real-time applications by simply outputting the calculation results through the D/A board. Then the PCI bus is not needed during data transfer, and its effect to the system performance is eliminated.

On the other hand, since the hardware resources are limited inside a FPGA chip. When the sound space becomes larger, more FPGA chips are required to implement the sound field solver. Then the sound space can be divided into different parts, and the sound field of each part is analysed by a FPGA. For each part, all computations are carried out locally by a FPGA, and the sound pressures of nodes on the divided boundary are exchanged with other FPGAs. The key challenge for the scalability is the data communication between FPGAs and FPGA boards due to the bus bandwidth. Because data exchanges between FPGAs and FPGA boards take more cycles, the calculation time will increase and be affected by the number of nodes and time steps. In the future study, the special RocketIO transceivers and receivers inside Xilinx FPGAs will be used for high speed data communication between FPGAs and boards. And a FPGA cluster machine will be developed and evaluated.

Acknowledgements

This work was supported in part by the Strategic Information and Communications R&D Promotion Programme (SCOPE).

References

[1] A. Krokstad, S. Strom, and S. Sorsdal, Calculating the acoustical room response by the use of a ray tracing technique, J. Sound Vib. 8 (1968) 118-125.

[2] J. B. Allen and D. A. Berkley, Image method for efficiently simulating small room acoustics, J. Acoust. Soc. Am. 65 (1979) 943-950.

[3] T. Funkhouser, N. Tsingos, I. Carlbom, G. Elko, A beam tracing method for interactive architectural acoustics, J. Acoust. Soc. Am. 115 (2004) 739-756.

[4] E. M. Nosal, M. Hodgson, and I. Ashdown, Investigation of the validity of radiosity for sound field prediction in cubic rooms, J. Acoust. Soc. Am. 116 (2004) 3505-3514.

[5] J. N. Reddy, An Introduction to the Finite Element Method, McGraw-Hill, USA, 2006, 3rd ed.

[6] P. K. Banerjee, and R. Butterfield, The Boundary Element Methods in Engineering Science, McGraw-Hill, USA, 1994.

[7] D. Botteldooren, Acoustical finite-difference time-domain simulation in a quasi-cartesian grid, J. Acoust. Soc. Am. 95 (1994) 2313-2319.

[8] D. Botteldooren, Finite-difference time-domain simulation of low-frequency room acoustic problems, J. Acoust. Soc. Am. 98 (1995) 3302-3308.

[9] H. Jordan, S. Bokhari, S. Staker, J. Sauer, M. ElHelbawy,and M. Piket-May, Experience with ADI-FDTD techniques on the Cray MTA supercomputer, Proc. of the SPIE - Commercial Applications for High-Performance Computing, vol. 4528, 2001, pp. 68-76.

[10] P. Moczo, M. Lucka, M. Kristekova, 3D displacement finite differences and a combined memory optimization, Bulletin of the Seismological Soc. of Am. 89 (1999) 69-79.

[11] G. A. Schiavone, I. Codreanu, R. Palaniappan, and P. Wahid, FDTD speedups obtained in distributed computing on a Linux workstation cluster, IEEE Int. Symposium on Antennas and Propag., 2000, pp. 1336-1339,.

[12] L. Savioja, Real-time 3D finite-difference time-domain simulation of low and mid-frequency room acoustics, Int. Conf. Digital Audio Effects, 2010, pp.1-8.

[13] A. Southern, D. Murphy, G. Campos and P. Dias, Finite difference room acoustic modelling on a general purpose graphics processing unit, Audio Eng. Soc. Convention 128, May 2010.

[14] Y. Inoguchi, Y. Y. Tan, Y. Sato, et al, DHM and FDTD based hardware sound field simulation acceleration, Int. Conf. Digital Audio Effects, 2011, pp. 69-72.

[15] Y. Y. Tan, Y. Inoguchi, E. Sugawara, et al, A real-time sound field renderer based on digital Huygens' model, J. Sound Vib. 330 (2011) 4302-4312.

[16] Y. Y. Tan, Y. Inoguchi, E. Sugawara, et al, A FPGA implementation of the two-dimensional digital Huygens' model, Proc. Int. Conf. Field Programmable Technology (FPT), 2010, pp. 304 - 307.

[17] G. Campos, and S. Barros, The Meshotron, a network of specialised hardware units for 3D digital waveguide mesh acoustic model parallelisation, 128th Audio Convention, London, May 2010.

[18] G. R. Campos and D. M. Howard, On the computational efficiency of different waveguide mesh topologies for room acoustic simulation, IEEE Trans. Speech Audio Process.13 (2005) 1063-1072.

[19] S. A. Van Duyne, and J. O. SMITH, The 2D digital waveguide mesh, Proc. of IEEE Workshop on Applications of Signal Process. to Audio and Acoustics, 1993, pp. 177-180.

[20] J. Hofmann, and K. Heutschi, Simulation of outdoor sound propagation with a transmission line matrix method, Applied Acoustics 68 (2007) 158 -172.

[21] P. B. Johns, R. L. Beurle, Numerical solution of 2 dimensional scattering problems using a transmission-line matrix, Proc. of IEE 118 (1971) 1203-1208.

[22] P. B. Johns, Application of the transmission-line-matrix method to homogeneous waveguides of arbitrary cross-section, Proc. of IEE 119 (1972) 1086-1091.

[23] M. V. Walstijn, and K. Kowalczyk, On the numerical solution of the 2D wave equation with compact FDTD schemes, Int. Conf. Digital Audio Effects, 2008, pp. 1-8.

[24] Y. Kagawa, T. Tsuchiya, B. Fujii, and K. Fujika, Discrete Huygens' model approach to sound wave propagation, J. Sound Vib. 218 (1998) 419-444.

[25] T. Tsuchiya, and Y. Kagawa, Digital equivalent circuit for acoustic field based on Discrete Huygens' modeling, Jpn. J. Appl. Phys. 44 (2005) 4297-4300.

[26] T. Tsuchiya, Numerical simulation of sound wave propagation with sound absorption using digital Huygens' model, Jpn. J. Appl. Phys. 46 (2007) 4809-4812.

[27] Y. Kagawa, T. Tsuchiya, K. Fujika, and M. Takeuchi, Discrete Huygens' model approach to sound wave propagation – reverberation in a room, sound source identification and tomography in time reversal, J. Sound Vib. 225 (1999) 61-78.

[28] F. Pardo, P. Lopez, D. Cabello, and M. Balsi, FPGA finite difference time domain solver for thermal simulation, Proc. Int. Conf. on Field Programmable Logic and Applications (FPL), 2005,

pp.721-722.

[29] K. Sano, T. IIzuka, and S. Yamamoto, Systolic architecture for computational fluid dynamics on FPGAs, 15th Annual IEEE Symposium on Field Programmable Custom Computing Machines, 2007, pp. 107-116.

[30] W. D. Smith, and A. R. Schnore, Towards an RCC-based accelerator for computational fluid dynamics applications, J. of Supercomputing 30 (2004) 239-261.

[31] H. Morishita, Y. Osana, N. Fujita, and H. Amano, Exploiting memory hierarchy for a Computational Fluid Dynamics accelerator on FPGAs, Proc. Int. Conf. Field Programmable Technology (FPT), 2008, pp. 193-200.

[32] C. He, G. Qin, M. Lu, and W. Zhao, Optimized high-order finite difference wave equations modeling on reconfigurable computing platform, Microprocessors and Microsystems 31 (2007) 103-115.

[33] M. shafiq, M. Pericas, R. D. L. Cruz, et al, Exploiting memory customization in FPGA for 3D stencil computations, Proc. Int. Conf. Field Programmable Technology (FPT), 2009, pp. 38-45.

[34] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, et al, Hardware implementation of a three-dimensional finite-difference time-domain algorithm, IEEE Antennas and wireless propagation letters 2 (2003) 54-57.

[35] W. Chen, P. Kosmas, M. Lesser, and C. Rappaport, An FPGA implementation of the two-dimensional finite-difference time-domain (FDTD) algorithm, Proc. of 12th int. Symposium on Field Programmable Gate Arrays, 2004, pp. 213-222.

[36] R. N. Schneider, L. E. Turner, and M. M. Okoniewski, Application of FPGA technology to accelerate the finite-difference time-domain (FDTD) method, Proc. of 10th int. Symposium on Field Programmable Gate Arrays, 2002, pp. 97-105.

[37] E. Motuk, R. Woods, S. Bilbao, and J. McAllister, Design methodology for real-time FPGA-based sound synthesis, IEEE Trans. On Signal Proces. 55 (2007) 5833-5845.

[38] H. T. Kung, 'Why systolic architecture?' Computer 15 (1982) 37-46.

[39] K. T. Johnson, A. Hurson, and B. Shirazi, General-purpose systolic arrays, Computer 26 (1993) 20-31.

[40] http://www.inrevium.jp/pm/image_signal/spp3000.html

[41] Tan Yiyu, Y. Inoguchi, Y. Sato, M. Otani, Y. Iwaya, H. Matsuoka, and T. Tsuchiya, Design of a FPGA-based timing sharing architecture for sound rendering applications, International Conference on Information Technology: Next Generations (ITNG), 2012, pp. 484-489.

Tan Yiyu received the B.S. degree in Electrical Engineering from Jiangnan University in 1994, M.S. and Ph.D. degrees in Electronic Engineering from Nanjing University and City University of Hong Kong in 2001 and 2006, respectively. He is currently a post-doctoral researcher in the Research Centre for Advanced Computing Infrastructure in Japan Advanced Institute of Science and Technology. His research interests include reconfigurable systems and computer architecture.

Yasushi Inoguchi received the B.S. degree from Tohoku University in 1991, M.S. and Ph.D. degrees from Japan Advanced Institute of Science and Technology in 1994 and 1997, respectively. He is currently a Professor in the Research Centre for Advanced Computing Infrastructure in Japan Advanced Institute of Science and Technology. His research interests include reconfigurable systems, massively parallel computers, and interconnection of multi-processor systems.

Yukinori Sato received the B.S., M.S. and Ph.D. degrees from Tohoku University in 2001, 2003 and 2006, respectively. He is currently an Assistant Professor in the Research Centre for Advanced Computing Infrastructure in Japan Advanced Institute of Science and Technology. His research interests include reconfigurable systems and parallel computer architecture.

Makoto Otani received the B.S., M.S. and Ph.D. degrees in Architectural Engineering from Kyoto University in 1999, 2001 and 2006, respectively. He is currently an Associate Professor in Faculty of Engineering in Shinshu University. His research interests include virtual reality, auditory display, telecommunication, and sound field reproduction.

Yukio Iwaya received the B.S., M.S. and Ph.D. degrees from Tohoku University in 1991, 1993 and 1999, respectively. He is currently a Professor in Faculty of Engineering in Tohoku Gakuin University. His research interests include digital signal processing of acoustics, sound localization, virtual auditory display, and virtual reality.

Takao Tsuchiya received the, B.S., M.E. and Ph.D. degrees from Doshisha University in 1984, 1986 and 1989, respectively. He is currently a Professor in Department of Information Systems Design in Doshisha University. His research interests include computational acoustics, sound and vibration in general ultrasonics.