Time Domain Numerical Simulation for Transient Waves on Reconfigurable Coprocessor Platform

Chuan He, Wei Zhao, and Mi Lu
Texas A&M University, College Station, TX 77843
chuanhe@ee.tamu.edu, w-zhao@tamu.edu, mlu@ee.tamu.edu

Abstract

A successful application-oriented reconfigurable coprocessor design requires not only a powerful FPGA-based computing engine along with suitable hardware architecture, but also an efficient algorithm tailored for this special application. In this paper, we present our hardware architecture and numerical algorithms designed to speedup the time-domain finite-difference simulation of linear wave propagation problems in 2D and 3D space on FPGA-based reconfigurable platforms. Application fields of this work include seismic modeling and migration, computational electromagnetics, aeroacoustics, marine acoustics, to name a few.

By writing first-order linear wave equations into second-order form, we halve the number of unknowns and simplify the treatment of parameters. We also adopt higher-order finite-difference (FD) schemes to further reduce the number of unknowns at the cost of increasing floating-point computations per discrete grid point. By doing so, we relief the bandwidth requirements between the FPGA and onboard memories but put more burden on the computing engine to take full advantage of FPGA's computational potentials. The speed of our design implemented on a Xilinx ML401 Virtex-4 evaluation platform is about 1.5~4 times faster than a pure software implementation of the same algorithm running on a 3.0GHz DELL workstation. This impressive result is mainly attributed to the memory architecture design, which is well-tuned for our numerical higher-order FD algorithms and can utilize onboard memory bandwidth more wisely. Furthermore, the good scalability of our design makes it compatible with most commercial reconfigurable coprocessor platforms and correspondingly, the performance would be proportional to their onboard memory bandwidth.

1. Introduction and related work

Time domain numerical simulation can improve people's understanding on dynamic behaviors of complex time-evolution problems, so plays an important role in scientific research and engineering design. In the past decades, efforts of simulating linear wave phenomena, including acoustic, electromagnetic, and elastic waves, have grown rapidly with the performance improvement of digital computers. Pure software acceleration methods, from low-level instruction reordering to high-level process parallelism, are all exhausted to speedup these numerical simulations. However, because the computational requirements of these problems in 2D or 3D space are extraordinarily high, especially when the geometrical size under study is much larger than the wavelength of sources, this kind of simulations is still limited in institutes that can afford high costs of running and maintaining supercomputers or large PC-cluster systems.

Recently, with the great improvement on reprogrammable hardware resources inside an FPGA chip, people start showing their interests in accelerating time domain numerical simulations for wave-like equations with FPGA-based reconfigurable hardware platform. Comparing with pure software procedures running on general-purpose computers or fully-customized VLSI hardware chips, FPGA technology can provide people a compromise between the best flexibility of software and the highest performance of hardware implementations. The idea of accelerating acoustic wave simulations using hardware platforms for geophysical applications can be traced back to 1990s [1]. The first effort of implementing such a stand-alone system was described in [2]. In [3], a reconfigurable coprocessor platform using high density FPGA was proposed to speedup seismic migration problems. For computational electromagnetics problems, several authors proposed their FPGA-based solutions to accelerating the standard Yee's Finite-Difference Time-Domain (FDTD) algorithm from the early 1990s [4-5]. Recent work in this field can be found in [6-9].

While most reconfigurable coprocessor platforms proposed in recent years mainly focus on real-time signal processing for stream-oriented input and output, the fundamental hindrance of simulating wave propagation problems numerically is the massive data volume along

Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)
0-7695-2445-1/05 $20.00 © 2005 IEEE

with the complex numerical algorithms. Specifically, memory bandwidth available between the computing engine (FPGA) and onboard memory modules has been proven a bottleneck preventing people taking full advantage of FPGA's computational potentials [3, 8, 9]. In this paper, we try to alleviate this bottleneck from two approaches: First, we rewrite the prevailing first-order linear wave equations into second-order form and adopt higher-order FD schemes to greatly reduce the number of discrete grid points inside the simulation area. Second, we propose a new memory architecture design for our applications. The time domain higher-order FD numerical algorithms can be effectively mapped into this memory architecture without changing memory bandwidth requirements and sustained high computational throughput can be achieved.

The rest of this paper is organized as follows: in Section 2, we first provide a brief review of wave equations in second-order derivative form and the corresponding standard second-order accuracy FD scheme. We then introduce the maximum order FD schemes and analyze their advantage over the standard one. In Section 3, after summarizing common hardware architecture properties of conventional reconfigurable coprocessor platforms, we propose our memory architecture design for the simplest standard second-order FD scheme in 2D space. We then extend our design to higher-order schemes and 3D space to show its simplicity and scalability. Section 4 provides our simulation results and performance comparisons between the hardware accelerator implemented on Xilinx ML401 evaluation platform and its pure software counterpart on a referential 3.0GHz P4 workstation based on two simple seismic modeling problems in 2D and 3D space, respectively. Finally, conclusions and a discussion of future research direction are presented in Section 5.

2. Numerical algorithms for wave equations

Wave equations are generally presented as linear system equations in first-order derivative form. It is well known that those governing equations can also be written as second-order derivative form without losing generality [10]. Representing wave equations in second-order derivative form has no benefit for conventional Finite-Difference Time-Domain (FDTD) algorithms executed on general-purpose computers. However, as we will see in this section and section 3, it plays a key role in our FPGA-based implementation to increase the efficiency of memory accesses.

2.1 Wave equations in second derivative form

Seismic modeling (forwarding) governed by acoustic or elastic wave equations in geophysics are a class of numerical methods that simulate the scattering field arising when an impulsive source excites an underground region with known physical properties like density, velocity, anisotropy, elasticity, etc. Let's consider the simplest 3D scalar acoustic case in the form of second-order linear Partial Differential Equation (PDE), which relates the temporal and spatial derivatives of the vertical pressure field as,

$$\frac{\partial^2 P(x,y,z,t)}{\partial t^2} - \rho(x,y,z)v^2(x,y,z)\nabla \bullet \left(\frac{1}{\rho(x,y,z)} \nabla P(x,y,z,t) \right) = f(x,y,z,t) \tag{2.1}$$

where P is the time-variant scalar pressure field (pressure in vertical direction) excited by an energy impulse $f(x,y,z,t)$; $\rho(x,y,z)$ and $v(x,y,z)$ are the density and acoustic velocity of underground media, which are already known as input parameters.

Define the gradient of a scalar field S as: $\nabla S \equiv \frac{\partial S}{\partial x}\vec{x} + \frac{\partial S}{\partial y}\vec{y} + \frac{\partial S}{\partial z}\vec{z}$ and the divergence of a vector field $\vec{V}$ as, $\nabla \bullet \vec{V} \equiv \frac{\partial V_x}{\partial x} + \frac{\partial V_y}{\partial y} + \frac{\partial V_z}{\partial z}$, equation (2.1) describes the propagation of acoustic waves inside 2D or 3D heterogeneous media with known physical properties. The numerical modeling problem we are facing here is to simulate the time evolution of the scalar pressure field P at each discrete grid points in 2D or 3D space accurately. For computational electromagnetical problems, the classical 3D Maxwell's equations can also be rewritten as three scalar second-order wave equations in x, y, or z direction respectively with a similar but more complex form as equation (2.1). So without losing generality, in this paper we will limit our discussion to the simplest second-order acoustic wave equations and focus on seismic modeling problems in geophysics field. It is straightforward to extend the numerical methods and corresponding hardware designs proposed in this paper to electromagnetic numerical simulations.

We make a constant density assumption to further simplify equation (2.1) as,

$$\frac{\partial^2 P(x,y,z,t)}{\partial t^2} - v^2(x,y,z)\Delta P(x,y,z,t) = f(x,y,z,t) \tag{2.2}$$

where $\Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ stands for the Laplace operator. Notice that the input and output of this Laplace operator are all scalars and the vector field $\vec{V}$ disappears. Equation (2.2) is still very practical and widely used for 2D and 3D acoustic modeling problems in seismic data processing industry.

Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)
0-7695-2445-1/05 $20.00 © 2005 IEEE

2.2 Second-order and higher-order Finite Difference Schemes

Finite difference time-domain methods are the simplest but the most popular approach to solve time evolution problems governed by PDEs or ODEs numerically. Given the values of a function on a set of discrete points, the finite difference approximation to the derivative of the function at one grid point can be expressed as a linear combination of function values at neighboring points. The theory of finite difference approximations for first order hyperbolic system equations is by now well developed. However, rewriting the first-order wave equations into second-order form halves the number of unknowns and simplifies the treatment of coefficients so will benefit our FPGA-based hardware implementation.

We discretize equation (2.2) by standard second-order FD scheme as,

$$P_{i,j,k}^{n+1} = 2 \cdot P_{i,j,k}^{n} - P_{i,j,k}^{n-1} + (dt)^2 \cdot v_{i,j,k}^2 \cdot \Delta^{(2)} P_{i,j,k}^{n} + f_{i,j,k}^{n} \tag{2.3}$$

and,

$$\Delta^{(2)} P_{i,j,k}^{n} = \left(\frac{P_{i+1,j,k}^{n} - 2 \cdot P_{i,j,k}^{n} + P_{i-1,j,k}^{n}}{(dx)^2} \right) + \left(\frac{P_{i,j+1,k}^{n} - 2 \cdot P_{i,j,k}^{n} + P_{i,j-1,k}^{n}}{(dy)^2} \right) + \left(\frac{P_{i,j,k+1}^{n} - 2 \cdot P_{i,j,k}^{n} + P_{i,j,k-1}^{n}}{(dz)^2} \right) \tag{2.4}$$

where the subscript marks the spatial position of discretized unknown pressure fields or parameters, superscript marks the time point when those unknowns are evaluated, dx , dy , and dz define the spatial interval between two adjacent grids in x , y , or z direction respectively, dt is the time-marching step, and $\Delta^{(2)}$ represents the second-order accuracy FD approximation of the spatial Laplace operator.

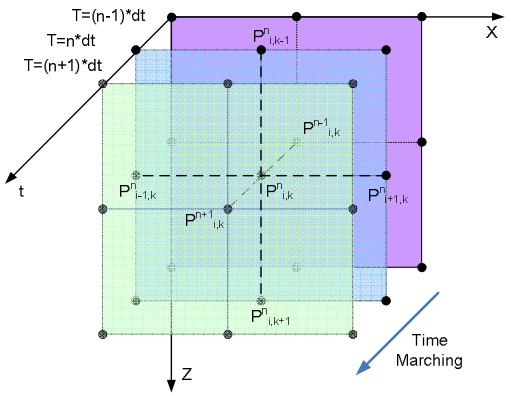

Figure 1. Second-order time-marching stencil for the 2D acoustic equation

Equation (2.3) shows us the second-order time-marching scheme and equation (2.4) is the second-order FD scheme evaluating the spatial Laplace operator. Figure 1 depicts the time-marching stencil of equation (2.3) in 2D space. We also draw the 3D spatial stencil of $\Delta^{(2)}$ in figure 2. All grid points that are involved in calculation are marked with subscripts in these figures. From these two figures, we can conclude that eight pressure values are needed to progress the evaluation of 3D pressure field P at grid point (i, j, k) to a future time, seven of them come from the present pressure field at this spatial point and its six orthogonal neighbors, the last one is the pressure value at the same grid point but from previous time step.

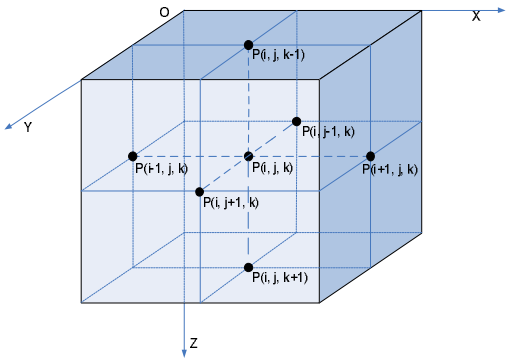

Figure 2. Second-order FD stencil for the 3D Laplace operator

Numerical errors arise from both the temporal and spatial discretizations. The errors associated with linear wave propagation problem involve mainly dispersion, dissipation, and anisotropy errors. Here, we omit detailed numerical theories but give the reader an intuitive result that numerical errors will cause the high frequency wave components propagating in slower speeds, damped amplitudes, or wrong directions in numerical simulations than in the reality. These errors will accumulate gradually and finally destroy the original shape of wave sources after propagating over a long distance or time. The FD scheme (2.3) and (2.4) is of second order accuracy with respect to time and space (a so-called (2, 2) FD scheme). Assuming the temporal derivative term can be calculated precisely by decreasing time-marching step and choosing the spatial discretization interval as 20 points per shortest wavelength (corresponds to the highest frequency component), the simulation results of the (2, 2) FD scheme are considered satisfactory only for wave propagating in an area with moderate size, generally on the order of 10 wavelengths [11]. For waves propagating over longer distances, the spatial interval required by this (2-2) scheme should be further refined, leading to enormous number of grid points in 2D or 3D space, impractical memory requirements, and unfeasible computational costs. This is the main motivation of the

Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)
0-7695-2445-1/05 $20.00 © 2005 **IEEE**

development of higher-order FD schemes. We must point out that the famous Yee's FDTD method, which has been widely adopted for electromagnetic modeling problems, is also a (2, 2) FD scheme but for the first-order derivative Maxwell equations discretized on staggered spatial grids. So it suffers the same numerical errors we discussed above, although they are generally less serious than the standard (2, 2) FD scheme.

We consider only spatial higher-order FD schemes and keep the second-order time-marching stencil in equation (2.3) unchanged. Numerical derivatives of a function defined on discrete points can be derived from Taylor expansion. The goal of the maximum order FD schemes [12] we adopted here is to make the approximation accurate by canceling as many the lower order terms in Taylor expansion formula as possible. The first un-cancelled Taylor series term determines the formal truncation error and the accuracy order of the finite difference approximation. For example, the one-dimensional Taylor expansions along x-axis at $x = (i \pm 1) \cdot dx$ for P are,

$$P(x_{i\pm1}) = P(x_i) \pm dx \cdot \frac{\partial P(x_i)}{\partial x} + \frac{(dx)^2}{2}\frac{\partial^2 P(x_i)}{\partial x^2} \pm \frac{(dx)^3}{6}\frac{\partial^3 P(x_i)}{\partial x^3} + \frac{(dx)^4}{24}\frac{\partial^4 P(x_i)}{\partial x^4} + \cdots \quad (2.5)$$

Add these two equations together to eliminate odd derivative terms at the right hand side,

$$\frac{\partial^2 P(x_i)}{\partial x^2} - \frac{P(x_{i-1}) - 2P(x_i) + P(x_{i+1})}{(dx)^2} = -\frac{(dx)^2}{12}\frac{\partial^4 P}{\partial x^4} + \mathrm{O}\left((dx)^4\right) \quad (2.6)$$

Equation (2.6) shows us that the difference (truncation error) between the second-order derivative of P and the three term FD scheme $\frac{P(x_{i-1}) - 2P(x_i) + P(x_{i+1})}{(dx)^2}$ is proportional to $(dx)^2$. That is the name (2, 2) FD scheme of equation (2.3) and (2.4) comes from. Using the same idea at more discrete points along x-axis, we can make higher order terms canceled and our approximation to the second-order derivative will be more accurate in the sense of truncation error. Systematically, we can approximate $\frac{\partial^2 P}{\partial x^2}$ to $(2m)$ accurate order by a linear combination of the values of P at $(2m+1)$ discrete grid points as follows,

$$\left(\frac{\partial^2 P(x_i)}{\partial x^2}\right)^{(2m)} = \frac{\alpha_0^m \cdot P_i + \sum_{r=1}^{m} \alpha_r^m \cdot (P_{i+r} + P_{i-r})}{(dx)^2} + O\left((dx)^{2m}\right) \quad (2.7)$$

where $\alpha_0^m = -2 \cdot \sum_{r=1}^{m} (-1)^{r-1} \frac{2m!^2}{r!(m-r)!(m+r)!}$

and $\alpha_r^m = (-1)^{r-1} \frac{2m!^2}{r!(m-r)!(m+r)!}$,

which are chosen to maximize the order of the un-cancelled truncation term.

Expanding the higher-order FD schemes to y and z-axis is straightforward, so a class of $(2m)^{th}$-order FD approximation of the Laplace operator in 2D or 3D space can be easily obtained and we finally get our higher-order (2-2m) FD schemes. The benefit is obvious: adopting higher order FD scheme means higher order of the un-cancelled truncation term, which leads to less approximation errors. Put it another way, by using higher-order FD schemes, we can enlarge the spatial discretization interval so that reduce the number of grid points without deteriorating our error criterions.

We designed a simple experiment to show the effectiveness of higher-order FD schemes. We simulate an exponentially-attenuated single-frequency sine wavelet propagating in 1D homogenous media (constant velocity) along x-axis. Setting the time-marching step small enough to make temporal discretization errors neglectable, we try to find out a suitable spatial discretization interval that reduce the power of numerical errors to about 0.1 percent of the energy of the original wavelet after it propagating a distance of 400 wavelengths. The simulation results are concluded in table 1 for different FD schemes. We can observe that the (2, 16) FD scheme decreases the total number of spatial grids in our test to 1600 (a propagation distance of 400 wavelengths times four points per wavelength), which is about five times less than (2, 4) scheme or ten times less than the standard (2, 2) scheme. This reduction will become much more significant if we apply this scheme to 2D ($10^2 = 100$ times less grids) or 3D （$10^3 = 1000$ times less grids）cases.

Table 1. Performance Comparison for different HD Schemes

FD schemes	Propagation Distance (Wavelength)	Grid Density (Grid/ Wavelength)	Total Number of Grid Points	Relative Error Power
(2, 2)	40	40	1600	0.0024
(2, 4)	400	19	7600	0.0037
(2, 8)	400	7	2800	3.8e-4
(2, 16)	400	4	1600	0.0010

Note: The standard (2, 2) FD scheme is incapable of simulating this wavelet propagating for hundreds of wavelengths accurately with a reasonable spatial sampling interval.

Similar to the standard (2-2) FD scheme, we draw a 3D spatial stencil of $\Delta^{(4)}$ in figure 3. This figure shows us

Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)
0-7695-2445-1/05 $20.00 © 2005 IEEE

that thirteen grid values around the grid point (i, j, k) are needed to evaluate the Laplacian value at this position, six points more than the $\Delta^{(2)}$ case.

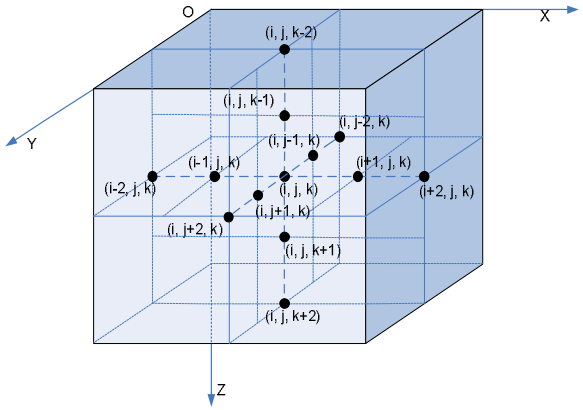

Figure 3. Fourth-order FD stencil for the 3D Laplace operator

3. Memory Architecture Design

We introduce now our hardware implementations for higher-order FD numerical algorithms based on but not limited to the reconfigurable hardware platform we proposed in [3]. Generally speaking, DDR-SDRAM module, which is the prevailing choice as large capacity onboard memories for general-propose computers and most commercial reconfigurable coprocessor platforms, has high potential data throughput at relatively low price, but the bandwidth utilization is usually poor in practice due to random-access natures of most applications. Based on data dependency properties of the higher-order FD algorithms and our applications, we introduce a special buffering system between the computing engine and onboard memory modules using on-chip memory blocks inside FPGA chips. This buffering system can utilize onboard memory bandwidth more efficiently than previous designs [8-9] and can be fitted into most commercial FPGA-based hardware platforms effectively.

In this section, we first summarize common architecture characters of conventional reconfigurable coprocessor platforms. After analyzing the floating-point computation and memory bandwidth requirements of FD algorithms for our application, we show the basic idea of our design by a simple hardware implementation of the standard second-order FD scheme in 2D space and compare its performance with designs proposed in [8, 9]. Then, we extend our design to higher-order schemes and 3D space to show its simplicity and good scalability.

We have to point out that the memory buffering system we proposed here is specified for higher-order FD schemes of equation (2.2) only. For equation (2.1), more complex memory structure is needed for buffering vector field $\vec{V}$.

3.1 Architecture properties of general FPGA-based hardware platforms

With the emergence of high capacity FPGAs, more and more people notice its potentials as high speed computing engine to accelerate large-scale computation-dominating applications. Consequently, many reconfigurable hardware platforms have been designed for fields like scientific computing, genetic computing, cryptography, image signal processing, radar signal processing, to name a few. Similar objectives make these hardware platforms have the same characters as follow: First, almost all of them were designed as a coprocessor attached to PC or workstation to enhance their flexibility for multiple application fields. Second, similar hardware architecture are adopted, including one or several high capacity FPGA chips acting as computing engine, one or several memory blocks to buffer or store data and parameters, high speed I/O channels interfacing the coprocessor with its host machine. Third, pipelining and parallelism are extensively used in these designs to increase processing speed and data throughput.

Our application of numerical simulation for wave propagating problems can be classified into scientific computing, so its FPGA-based hardware implementations should bear all those characters we mentioned above. Moreover, the application's unfeasible data manipulation requirements between computing engine and on-board memories force people integrating as many dedicated memory channels as possible into their designs. This special hardware property can be seen clearly from the recent works presented in [3] and [9], which accidentally proposed two almost the same FPGA-based hardware platforms applied to seismic acoustic wave and electromagnetic wave simulations, respectively. However, because of the limited number of I/O pins of an FPGA chip, only a few dedicated memory channels could be implemented in practice, and we can predict that the restricted memory-bandwidth would always be a bottleneck preventing people taking full advantage of FPGA's computational potentials in the near future.

3.2 Memory architecture design for second-order FD scheme in 2D space

We rewrite the standard second-order FD scheme (2.3) and (2.4) in 2D space and ignore the source term,

$$P_{i,k}^{n+1} = 2 \cdot P_{i,k}^{n} - P_{i,k}^{n-1} + (dt)^2 \cdot v_{i,k}^2 \cdot \Delta^{(2)} P_{i,k}^{n} \qquad (3.1)$$

Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)
0-7695-2445-1/05 $20.00 © 2005 IEEE

$$\Delta^{(2)}P_{i,k}^{n}=\left(\frac{P_{i+1,k}^{n}-2\cdot P_{i,k}^{n}+P_{i-1,k}^{n}}{(dx)^{2}}+\frac{P_{i,k+1}^{n}-2\cdot P_{i,k}^{n}+P_{i,k-1}^{n}}{(dz)^{2}}\right) \tag{3.2}$$

We need six pressure values and one velocity to evaluate these two equations for one grid point at position (i,k). As for the computational costs, we need five additions and two multiplications to approximate the 2D Laplace operator in equation (3.2), another one multiplication and two additions are needed to calculate the final result in equation (3.1). (We ignore three multiply-by-two operations because they can be easily merged into other hardware arithmetic units.) So, we totally need seven additions and three multiplications operated on seven pressure or velocity values for the evaluation of one grid point at one time-marching step. Of cause, the number of memory accesses is one more than the number of operands because the new results should be saved back for the next time-marching step.

From this analysis, we can find out that the number of floating-point computations and memory accesses are nearly balanced for the standard second-order FD scheme (similarly the Yee's FDTD scheme). So the execution speed of these schemes on general-propose computers is mainly decided by available memory bandwidth but not the CPU's nominal speed. Although higher-order schemes reduce the number of grid points and complicate the computations at each grid point, they do require more operands in their computational stencils so that the ratio of computations and memory accesses is almost kept constant. So higher-order schemes result in little benefit on generic computers and are seldom put into practice in reality.

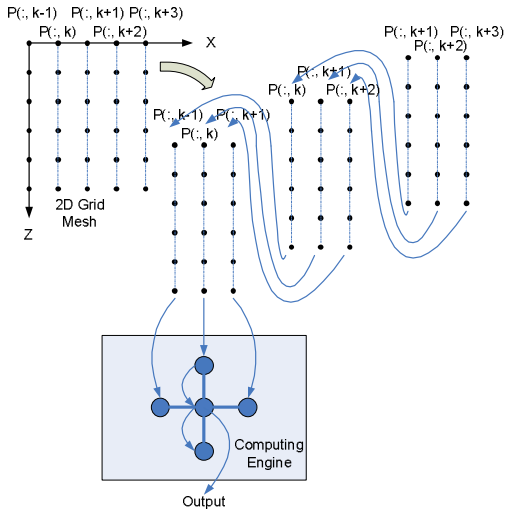

Figure 4. Stripped 2D operands feed into the fixed computing engine through three input ports

The maximal memory channels available on reconfigurable coprocessor platforms could be significantly more than but still comparable to generic computers. For example, an up-to-date PC has two DDR memory channels compared with four of them on the coprocessor platform presented in [9], which is the top record to our best knowledge. Previous designs in [8, 9] tried to migrate the software version of Yee's FDTD algorithm directly into their FPGA-based hardware platforms. Their efforts concentrated on integrating more hardware arithmetic units into FPGA so that the aggressive computational speed of their designs would exceed generic computers. This approach works great and all these works reported impressive speedup comparing with PCs. However, the memory bandwidth bottleneck will finally be reached and after that, no more speed benefit will be gained.

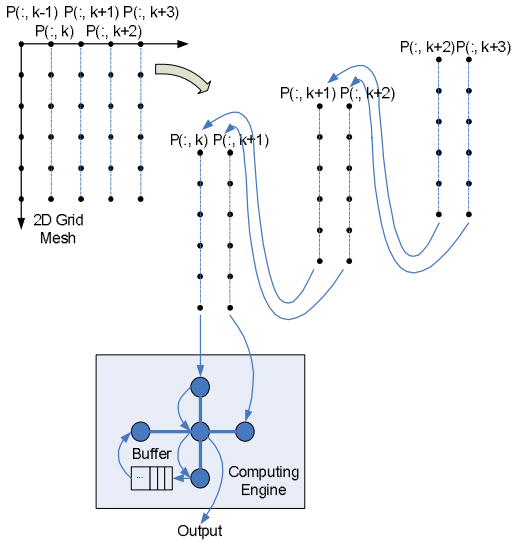

Figure 5. Stripped 2D operands feed into the computing engine through two input ports

In our design, we try to find suitable on-chip memory architecture to utilize onboard memory bandwidth more wisely. We define "row" as a line of spatial grids along the X-axis and "column" as a line of grids along Z-axis in 2D space. Because little optimization can be applied to equation (3.1) to reduce its computations and memory accesses, we consider only equation (3.2) here and suppose we evaluate this equation grid by grid along each column. Our approach can be imagined as moving a striped 2D operands mesh into the fixed computing engine through one or several input ports. Figure 4 shows a trivial implementation of this idea, which explores few data dependencies and corresponds to the execution of the (2, 2) FD algorithm on generic computers. If we calculate $P_{i,k}^{n+1}$ when the operand at grid point (i,k) reaches the center of our computing engine, we observe that almost all those pressure values we needed to calculate $P_{i,k}^{n+1}$ have been encountered

Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)
0-7695-2445-1/05 $20.00 © 2005 IEEE

except $P^{n}_{i,k+1}$. This observation implies that if we could store some used grid values in our FPGA chip temporarily, we may avoid accessing the same data repeatedly from onboard memories and save memory bandwidth a lot. This idea is reflected in the implementation in figure 5, where values at grid points of a whole column $(:, k-1)$ are saved in the computing engine. Notice two input ports are needed here, one less than the previous case.

This basic idea is almost the same as on-die data or instruction caches appearing in architectures of most modern generic CPUs. However, the caching mechanism is too complex to implement on reconfigurable hardware platforms. We need to find out a much more efficient on-chip memory structure to implement this "buffering" idea.

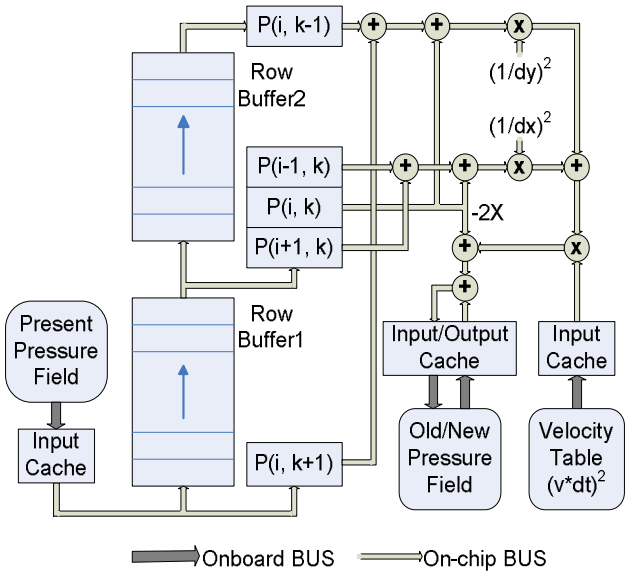

Figure 6. Block diagram and dataflow of the buffering structure and computing engine in 2D space.

Figure 6 illustrates the block diagram and dataflow design of our buffering structure and computing engine inside FPGA. We use two cascaded FIFOs as our data buffer, each of which has the capacity to contain a whole row of grid values. Generally, this number is less than a couple of thousands and can be efficiently implemented by one or several memory blocks inside FPGA. Pressure values are feed into the FPGA chip from one input port at the bottle of the first FIFO and discard at the top of the last one. We delay the calculation of $P^{n+1}_{i,k}$ a whole row until the grid value $P^{n}_{i,k+1}$ enters our data buffer structure so that all operands we needed to evaluate equation (3.2) are available inside FPGA chip. Considering the grid value $P^{n-1}_{i,k}$ at previous time step and the parameter $v_{i,k}$ that are needed for calculating equation (3.1) and taking the inevitable save back operation into account, we need only four memory accesses to evaluate one time-marching step at one grid point, which is the best a complex cache system of modern generic CPUs could achieve. We also introduce simple input caching circuits after SDRAM modules so that input data can be feed into the buffering structure at a constant speed and the computing engine can be fully pipelined to achieve high computational throughput. We will revisit this input cache design in section 4.

3.3 Extension to higher-order FD schemes and 3D space

Consider the higher-order FD stencils we derived in section 2, we can conclude that (4m+1) present pressure values at grids around position (i,k) for 2D case or (6m+1) values at grids around (i,j,k) for 3D case are needed to evaluate the Laplacian value at this center point up to (2m) accuracy order. The requirements of one old pressure value and one velocity parameter keep unchanged for second-order time-marching scheme. The extension of our design to higher-order schemes is straightforward and easy to implement. (2m) cascaded FIFOs are needed as row buffer, and correspondingly, we delay the calculation of $P^{n+1}_{i,k}$ for m rows to make sure all the operands appear at correct positions in our buffer structure. Inevitable, extra addition and multiplication units should be inserted into the pipeline of our computing engine.

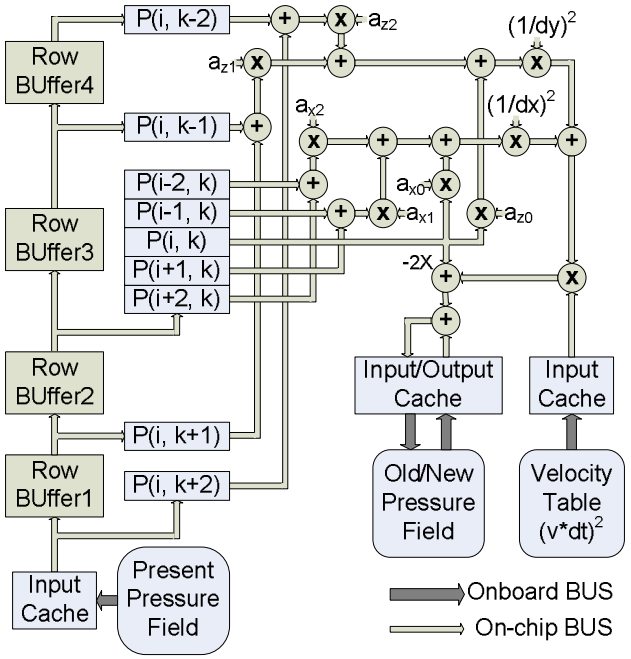

Figure 7. Block diagram of the buffering structure and computing engine for (2, 4) FD scheme in 2D space

Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)
0-7695-2445-1/05 $20.00 © 2005 IEEE

The most exciting observation of our higher-order implementation is although the memory bandwidth requirements for pure software implementations increase linearly with the order of FD schemes, this requirements are unchanged for our design, i.e., the number of memory accesses to evaluate one time-marching step at one grid point is still four. The only costs we pay for the higher-order schemes are on-chip memory blocks and conventional addition or multiplication arithmetic units, which are all abundant inside an up-to-date high density FPGA chip. This result encourages us adopting extraordinarily higher-order FD schemes in our design to further enlarge the spatial discretization interval until reaching the extreme of two samples per wavelength, which is bounded by the famous Nyquist-Shannon sampling theorem. For example, we can easily extend our design to a (2, 16) FD scheme using 16 cascaded on-chip FIFOs and 56 arithmetic units (35 additions and 21 multiplications) and still need four memory accesses per grid point per time step.

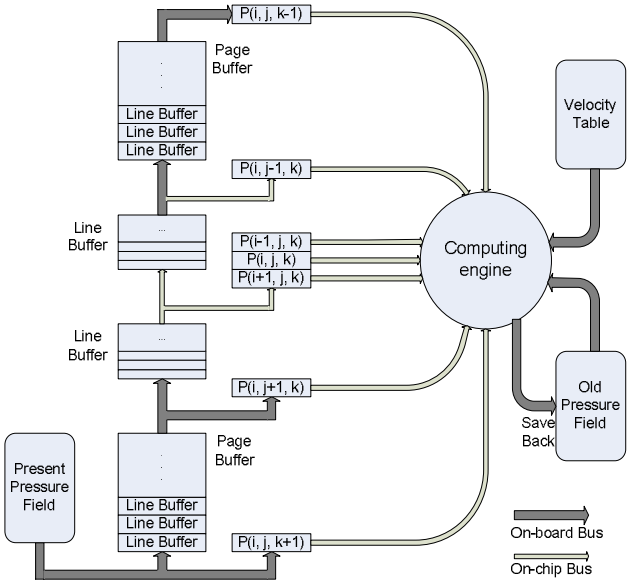

Figure 8. Block diagram of buffering structure for the (2, 2) FD scheme in 3D space

Now, the basic idea of our approach is very clear: Consider the design for the (2, 2) FD scheme in section 3.2, the computing engine for this simplest case consists of ten conventional arithmetic units (seven additions and three multiplications), which cost only a very small portion of hardware resources even for the fastest fully-pipelined implementations. Because the clock frequencies applied to FPGAs and memory channels are within the same range as hundreds of MHz per second, the bandwidth of onboard memories would be saturated rapidly with considerable part of FPGA hardware resources being wasted. By introducing higher-order FD schemes, we could make the computations as complex as necessary to throw the burden back on the computing engine again. Moreover, higher-order FD schemes allow larger discrete intervals in spatial axes so that the number of grid points is considerably reduced, and consequently, memory bandwidth requirements for the same problem decrease in an indirect way. Put it another way, we can always find a point, at witch the utilization of onboard reconfigurable hardware resources and memory bandwidth are well balanced.

Extending our design to 3D space is also straightforward but the hardware implementation will become less efficient than 2D cases. Now, we need several large-capacity FIFO structures to buffer 2D pages instead of 1D grid lines as before. Practical 3D wave simulations contain generally hundreds to thousands of grid points along each spatial axis, so the capacity of page buffers could easily reach several millions of words per page, which approaches the maximal capacity of on-chip block memories inside an up-to-date FPGA chip. So we have to sacrifice some onboard memory bandwidth to meet our buffering requirements, and we still need to spend some extra hardware resources to imitate the FIFO behavior on commercial memory modules. Figure 8 depicts the block diagram of the (2, 2) FD scheme in 3D space.

4. Simulation Results

The aim of this section is to show the correctness and effectiveness of our hardware accelerator design for wave-propagation modeling problems. The target FPGA-based prototyping platform we used is a low-end Xilinx ML401 Virtex-4 evaluation board [13]. Although this device provides very limited onboard hardware resources (one XC4LX25 FPGA chip embedding 24,192 Logic Cells, 48 DSP Slices and 72 18-kb SRAM Blocks; 64MB onboard DDR-SDRAM modules with 32-bit interface to the FPGA chip; and 9Mb onboard ZBT-SRAM with 32-bit interface.), it contains all necessary components we needed to validate our accelerator design. The software development environments are Xilinx ISE 6.3i and ModelSim 6.0 se.

Table 2. Size of Test Problems

	2D Test Problem	3D Test Problem
Number of Spatial Grids	1000×1000	$100\times100\times100$
Total Time Steps	6000	6000
Storage Requirements	4 Million Words	4 Million Words
Number of Grid Computations	6×10^9	6×10^9

Two small seismic modeling problems are selected as our benchmarks in 2D or 3D space, respectively. Table 2 shows their computation and storage sizes. These two problems are chosen carefully to make sure that they can be fitted into our hardware platform. The simulation results are compared with their software counterparts running on a referential Intel P4 3.0 GHz workstation.

Our hardware designs are based on the block diagrams presented in figure 6 and 7 for 2D and 3D cases respectively. As we show in table 2, there are one million discrete grids for each problem so that four million words of onboard DDR-SDRAM are assigned as data storage space. These storages are organized as four data volumes to save information about the previous pressure field, the present pressure field, the unknown future pressure field, and the velocity table, respectively. In order to utilize the bandwidth of DDR-SDRAM more efficiently, an extra cache circuit is constructed for each data volume using two on-chip RAM Blocks at the interface between the onboard DDR-SDRAM modules and our data buffering structure. This input cache contains two parallel 512-word data buffers, each of which can accept a whole physical column of data from SDRAM and works in a swapping manner to hide the irregular data accessing and periodic refreshing behaviors of SDRAM components. This implementation isolates our buffering and computing engine from the memory interface circuits so that a constant computational throughput can be achieved.

Table 3. Performance Comparison for Test Problems

FD scheme	Software Compu-tational Through-put (Million Grid / second)	Hardware Implementations		
		Compu-tational Throughput (Million Grid/second)	Speed-up	Resource Utilizations (RAM Blocks / DSP Slices /Logic Slices)
(2, 2) for 2D case	33.27	49.69	1.49	12/6/4336
(2, 4) for 2D case	27.53	49.57	1.80	16/18/6532
(2, 8) for 2D case	19.90	49.43	2.48	24/26/9098
(2, 16) for 2D case	12.45	49.10	3.94	40/42 /15370
(2, 2) for 3D case	21.37	48.33	2.26	52/8/5482

We modified the single-precision floating-point adder and multiplier design proposed in [14] as our arithmetic units. These functional units are open-source and IEEE 754 compliant. They utilize hundreds of logic cells and can provide a sustained computational speed at over 80 MHz per second, which is fast enough for our design on ML401 platform because of the limitation of onboard memory bandwidth. The simulation results of the software and hardware implementations for these two test problems are shown in table 3.

Utilizing the onboard 100MHz oscillator, we set the clock frequencies applied to onboard DDR-SDRAM modules at 100MHz and the computing engine at 50MHz. (The maximal clock frequency for the DDR-SDRAM modules on the ML401 platform is 133MHz, so the theoretical maximal computational throughput is 66 million grids per second.) Comparing with the 50 million grid-per-second theoretical computational throughput, the speed of our implementation is degraded for less than 2 percent because of pipeline stalls, which occur mainly when we flush the cascaded data FIFOs at the beginning of each time-marching step and when we deal with imaginary boundary points. The corresponding software codes are programmed by ANSI C language and compiled using INTEL C++ compiler v8.1with optimization for speed. All results are obtained on a referential DELL workstation equipped with one Intel P4 3.0 GHz CPU and 1GB memory. We have to admit that the performances of software implementations might be further improved using some low-level tuning and optimization tools. However, this approach is so involved and hard to predict that only specialists could benefit from it [14].

In our experiments, we keep the number of 2D or 3D grid points unchanged to evaluate the speeding-up attributed purely to our hardware implementations. If we take into account the results we concluded in table 1 that higher-order FD schemes can reduce the number of discrete grids considerably, the speeding-up of our design will become much more impressive.

We emphasize again that our FPGA-based design is implemented on a low-end Xilinx ML401 Virtex-4 evaluation platform. The limited onboard memory bandwidth considerably restricts the performance of our hardware accelerator. To prove this point, we designed our own pipelined high-speed floating-point multipliers and adders, which are optimized for Xilinx Virtex II series FPGA. By employing more pipeline stages than the designs in [15], the speeds of our floating-point arithmetic units reported by Xilinx ISE 6.3i after place-and-route (PAR) are 238MHz for multiplier with 7 pipelined stages and 216MHz for adder with 12 stages. Suppose we have a reconfigurable coprocessor platform integrating one 200MHz 72-bit DDR-SDRAM memory module, because the aggressive onboard memory bandwidth is 800 million words per second now, this platform can afford our computing engine working at a sustained speed of 200 million grids per second, which is four times faster than the ML401 platform.

5. Conclusion

In this paper, we proposed an FPGA-based hardware implementation to accelerate time-domain numerical simulations of linear wave propagation problems in 2D and 3D space. By adopting higher-order finite difference numerical algorithms along with efficient on-chip memory architecture, we alleviate the bandwidth bottleneck between the FPGA chip and onboard memories at the cost of much more computational requirements, which fortunately are absorbed into the pipelined computing engine without any speed or memory bandwidth penalty. Our hardware accelerator design takes full advantage of data dependency of the higher-order FD algorithms, its desirable properties of simplicity and scalability make it compatible with most commercial FPGA-based reconfigurable coprocessor platforms and its performance is expected to increase linearly with available onboard memory bandwidth. The simulation results on a low-end Xilinx ML401 Virtex-4 evaluation platform show impressive speeding-up comparing with their pure software counterpart running on a referential Intel P4 3.0 GHz workstation.

To the best of our knowledge, this is the first attempt to solve a practical seismic modeling problem using FPGA technology. The main motivation for introducing reconfigurable logic to seismic data processing industry is its immense computational potentials along with acceptable flexibility so that the same hardware resources can be reconfigured for different algorithms used in different processing stages. Future work in this field will concentrate on improving our design for 3D cases and extending it to more general form of wave equations.

References

[1] R. P. Bordeling, "seismic modeling with the wave equation difference engine", Society of Exploration Geophysicists (SEG) International Exposition and 66th Annual Meeting, 1996

[2] M. Bean, and P. Gray, "Development of a high-speed seismic data processing platform using reconfigurable hardware", Society of Exploration Geophysicists (SEG) International Exposition and 67th Annual Meeting, 1997

[3] C. He, M. Lu, and C. W. Sun, "Accelerating Seismic Migration Using FPGA-Based Coprocessor Platform", 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2004

[4] J. R. Marek, M. A. Mehalic, and A. J. Terzouli, "A dedicated VLSI Archtecture for Finite-Difference Time Domain Calculations", 8th Annual Review of Progress in Applied Computational Electromagmetics, 546-553, 1992

[5] P. Placidi, L. Verducci, G. Matrella, L. Roselli, and P. Ciampolini, "A custom VLSI architecture for the solution of FDTD equations", IEICE Transactions on Electronics, vol. E85-C, 572-577, 2002

[6] R. N. Schneider, L. E. Turner, and M. M. Okoniewski, "Application of FPGA Technology to Accelerate the Finite-Difference Time-Domain (FDTD) Method", 10th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2002

[7] 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, 54-57, 2003

[8] W. Chen, P. Kosmas, M. Leeser, and C. Rappaport, "An FPGA Implementation of the Two Dimensional Finite Difference Time Domain (FDTD) Algorithm", 12th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), 2004

[9] J. P. Durbano, F. E. Ortiz, J. R. Humphrey, P. F. Curt, and D. W. Prather, "FPGA-Based Acceleration of the 3D Finite-Difference Time-Domain Method", 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM), 2004

[10] W. C. Chew, "Waves and Fields in Inhomogeneous Media", IEEE Press, 1995

[11] I. R. Mufti, J. A. Pita, and R. W. Huntley, "Finite-difference depth migration of exploration-scale 3-D seismic data", Geophysics, vol.61, 776-794, 1996

[12] F. Bengt, "Calculation of weights in finite difference formulas", SIAM Review, 40, 685-691, 1998

[13] Xilinx, "ML401 Evaluation Platform User Guide", www.xilinx.com

[14] G. Chaltas and W. R. Magro, "Performance Analysis and Tuning of LS-DYNA for Intel Processor-Based Clusters", 7th International LS-DYNA Users Conference, 2002

[14] G. Marcus, P. Hinojosa, A. Avila, and J. Nolazco-Flores, "A Fully Synthesizable Single-Precision,Floating-Point Adder/Substractor and Multiplier in VHDL for General and Educational Use", 5th International Caracas Conference on Devices, Circuits and Systems (ICCDCS), 2004