Accelerating Seismic Migration Using FPGA-based Coprocessor Platform

Chuan He, Mi Lu, and Chuanwen Sun
Department of Electrical Engineering
Texas A&M University, College Station, TX 77843
chuanhe@ee.tamu.edu, mlu@ee.tamu.edu

Abstract

Migration is the most important seismic data processing method that recovers subsurface images of the Earth’s interior using surface-recorded data volumes obtained from seismic reflection surveys. A reconfigurable coprocessor platform called SPACE (Seismic data Processing Accelerator with reConFigurable Engine) using Field Programmable Gate Array (FPGA) technology is proposed in this paper to speed up these computationally demanding and data-intensive seismic migration applications. The proposed SPACE platform is characterized by its simple architecture and abundant on-board memory resources along with ultra-wide memory bandwidth, which also makes the platform suitable for other seismic data processing methods or some large-scale scientific computing applications. The time-consuming kernel part of the Pre-Stack Kirchhoff Time Migration (PSTM) algorithm is programmed into the FPGA-based coprocessor platform, which acts as a hardware accelerator attached to an Intel-based workstation through the local Peripheral Controller Interface (PCI) bus. Improved performance can be achieved by integrating a number of parallel running fully pipelined arithmetic modules into a single FPGA chip. Our simulation results show that the proposed coprocessor platform operating at a conservative speed of 50 MHz can calculate the Kirchhoff summations for 50 million points per second, which is about 15.6 times faster than a referential 2.4 GHz Pentium 4 workstation. The impressive performance of the proposed platform implies its broad applications in seismic data processing industry.

1. Introduction

Seismic migration is a wave-equation-based seismic data processing method that recovers an image of geological subsurface structures by collapsing diffraction energy at discontinuous points and moving reflection events to their geological locations. Today, migration is the culmination and a central step in the seismic data processing flow. [1] Pre-stack Kirchhoff time migration (PSTM) algorithm based on diffraction summation is the most popular migration method because of its simplicity, efficiency, feasibility and target-orientated property [2]. However, this algorithm is still too expensive for the practical 3D pre-stack applications because it is computationally intensive and requires processing large amount of input data. Moreover, the requirement of updating the subsurface acoustic velocity model iteratively makes the situation much worse. For example, it usually takes weeks to get the final 3D migrated subsurface image from the input data volume of a large area on a powerful PC cluster system, which consists of thousands of workstations.

In this paper, we introduce a novel coprocessor platform based on FPGA technology to speed up the PSTM algorithm. Instead of a stand-alone computer system, it has been designed as a new computing resource that is attached to a conventional workstation. The kernel part of the PSTM algorithm, which consumes more than 90 percent of CPU time, is adjusted carefully to maximize its execution speed in the proposed SPACE platform. Operations like process management, I/O management, and network communications are left to the mother machine. The performance simulation shows that the proposed platform is over ten times faster than a referential P4 workstation for this particular algorithm without losing any accuracy.

The rest of this paper is organized as follows: the PSTM algorithm and its software implementation on conventional PC cluster systems are reviewed in section 2. Section 3 introduces the hardware structure of the proposed coprocessor platform. In Section 4, the implementation details of the PSTM algorithm on the SPACE platform are presented. Building on this basic design, section 5 introduces three travel-time solver schemes: the high-precision modified hybrid scheme, the pure fixed-point design, which is area-efficiency with limited accuracy, and the optimized 6th order travel time solver used in our final design. Section 6 is the performance comparison of the proposed SPACE platform...
to a pure software implementation on a referential P4 workstation. Finally, section 7 describes conclusions and directions of our future work.

2. PSTM Algorithm

2.1 Algorithm Description

Seismic reflection survey is the most widely used geophysical exploration method in the petroleum industry. Acoustic waves generated by an intense energy source such as dynamite or air gun are directed into the Earth’s interior and propagate downward at a velocity depending on the elastic properties of the medium. When this wave reach an interface between rock layers where the density or velocity changes, a portion of the energy is reflected back to the surface. An array of thousands of geophones is regularly distributed across the ground surface to detect the intensity and time of the reflected waves. The time-series recorded by each geophone during an exploding experiment (one shot) is called a “seismic trace”, which is characterized by the corresponding position of its source and receiver. The set of traces recorded by all the receivers during one shot forms a common shot gather and the data set collected from a large number of experiments with different shot and receiver positions in an area forms a 5D data volume \( D(s, g, t) \), where \( s = (x_s, y_s) \) and \( g = (x_g, y_g) \) are the surface coordinates of the shot and receiver of each seismic trace, \( t \) is the recording time.

Mathematically, seismic migration tries to find a solution \( P(x, y, z, t) \) of the wave field governed by the following scalar acoustic wave equation using the surface-recorded data volume \( D(s, g, t) \) as boundary conditions:

\[
\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2} = \frac{1}{v^2(x,y,z)} \frac{\partial^2 P}{\partial t^2}. \tag{1}
\]

Then the subsurface image can be extracted from \( P(x, y, z, t) \) by:

\[
I(x, y, z) = P(x, y, z, t)|_{t=0}. \tag{2}
\]

The PSTM algorithm provides a high-order approximate integral solution to Equation (1). It is a mathematical statement of Huygens’ Principle based on Green’s Function theory. Define travel time as the duration of an acoustic impulse traveling from the shot point through the scatter point and back to the receiver point, the total down-up travel time for a hypothetic horizontally layered Earth model is determined by:

\[
T_{SR} = T_s + T_R = \sqrt{\tau^2 + \frac{S^2}{V_s^2}} + \sqrt{\tau^2 + \frac{R^2}{V_r^2}}. \tag{3}
\]

Where \( T_s \) is the travel time from the shot point to the scatter point; \( T_R \) is the travel time from the scatter point to the receiver point; \( S \) and \( R \) are the distances between the surface mirror of the scatter point and the shot point or receiver point respectively; \( \tau \) is the pseudo-depth of this scatter point in the output section; and \( V_r \) is a priori estimation of the Root Mean Square (RMS) velocity at point \( \tau \). Figure 1 schematically shows the relationship between the source, receiver and scatter points.

![Figure 1. The relationship between the source, receiver and scatter points.](image)

The PSTM algorithm assumes that the energy of a sampled point \( t_0 \) in an input trace is the superposition of reflections from all the underground scatter points that have the same travel time \( T_{SR} = t_0 \) for the fixed source and receiver positions. So for one sample point on an input trace with known source and receiver coordinates, its energy should be spread to all possible scatter points according to the travel time \( T_{SR} \). The locus of all possible scatter points with travel time \( T_{SR} \) in a constant velocity medium is also depicted in Figure 1, which is an ellipse in 2D profile following the geometric definition. In order to retrieve all the underground scatter points, the energy of an input trace must be distributed to all possible scatter points correctly, and then the energy from different input traces is summed together at each pseudo-depth point in the output section. For amplitude preserving PSTM, an extra oblique factor calculation step [3] [4] and a multiplication of this factor are needed.

The PSTM algorithm is very computationally intensive when executed on general-purpose computers because of its huge iteration number. The computational complexity of this algorithm is \( O(N_s \cdot N_y \cdot N_r \cdot N_s \cdot N_g) \) for 3D case [5], and those
two square root operations in Equation (3) are very time-consuming for most CPU. A small seismic survey, for example, contains 1000 shot gathers, 1000 traces per shot and 1500 samples per trace. The 3D output section we need to image has 500 by 500-surface positions, by 1500 pseudo-depth points. The execution of PSTM algorithm theoretically contains about 37500 Trillion Floating-point operations, which would cost at least one month for a powerful computer with computational capacity of 1GFLOPS.

2.2 Implementation on PC Cluster

Traditionally, only high performance super computers can finish this time-consuming 3D PSTM algorithm within a reasonable time interval. However, the high cost of running and maintaining such a super-computer is always not affordable for most people. Recently, PC cluster has emerged as a cheap and efficient alternative to super-computer. A PC cluster system consists of one or several servers and a bunch of workstations connected through high-speed networks using the Message Passing Interface (MPI). Taking advantage of Commercial-Off-The-Shelf (COTS) hardware and Linux OS, a PC cluster system can always achieve a better performance at affordable prices for many network non-intensive applications.

<table>
<thead>
<tr>
<th>Figure 2. The program flow of the PSTM kernel code executed in a workstation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>...</strong> Receieve one input trace from server Prepare parameters for this trace For every local output trace in this workstation For every pseudo-depth point on this output trace Calculate travel time ( T_{sr} ) for this output point - -associated with the position of this input trace IF ( T_{sr} &gt; T_{max} ) THEN finish this output trace Fetch data from input trace indexed by ( T_{sr} ) Anti-aliasing filtering Calculate oblique factor Scaling the selected input data by oblique factor Accumulate scaled input data to this output point End End <strong>...</strong></td>
</tr>
</tbody>
</table>

PSTM is an ideal algorithm for a PC cluster system with limited shared communication channel. By parallelizing the migration task in output surface coordinates \((x, y)\) and serial looping over the pseudo-depth coordinate \(\tau\) [6], the communication between server and workstations is minimized. In the meantime, no communication is needed among workstations. The server of the system broadcasts input traces to all workstations and each workstation migrates these input traces into its local output section \((x, y, \tau)\). Once all input traces are migrated, the server collects all partial results from workstations and forms a final migrated image. Figure 2 is the kernel part of PSTM procedure executed on every workstation.

The performance of a PC cluster system can be linear to the number of connected workstations for this parallelized PSTM algorithm. However, when more and more workstations are added into the system, the limited shared communication channel becomes a bottleneck to the system performance, and the reliability problem is inevitable for a system with thousands of connected workstations. Employing more powerful workstations can effectively alleviate these problems at the cost of increasing system price considerably.

3. Reconfigurable Hardware Architecture

Although the programmable hardware resources inside an FPGA chip have increased greatly in recent years, it’s still not rich enough to fit a complicated program into a single chip. The conventional hardware-software hybrid approach is our only choice. The feasibility of applying FPGA technology to PSTM algorithm is based on the fact that over 90 percent of CPU time is consumed by billions of iterations of the procedure’s short kernel code. This short but time-consuming code is very suitable to fit into an FPGA chip. A good design depends greatly on where to place the dividing line between hardware and software. In our design, the dividing line is chosen to saturate the computing power of the mother workstation, which obviously implies a longer product lifecycle of the SPACE platform.

While most general-purpose reconfigurable coprocessor platforms proposed in recent years mainly focus on real-time signal processing for stream-oriented input and output, the fundamental hindrance of seismic data processing is the massive data volume along with the complex processing algorithms. Another important issue is the price if we take into account that there are millions of workstation nodes running seismic data processing software all over the world. Our design stresses the simplicity of the coprocessor architecture and its capability to manipulate huge amount of data. The proposed SPACE platform consists of only three main components: FPGA, memory, and PCI interface circuit. (Figure 3) All these components are Commercial-Off-The-Shelf (COTS) hardware and can be easily replaced or upgraded in the future.

The core component of SPACE platform is an up-to-date Xilinx Vertex II Pro series FPGA chip [7], which contains a large array of programmable Logic Cells (LC)
along with high-density components like block RAM, fast multiplier, and gigabit serial transceivers. The reason that we use only one FPGA chip in our system is based on two considerations: First, the hardware resources inside the Vertex II Pro series FPGA are abundant enough for most potential applications. The routing resources inside an FPGA chip are denser and more reliable than those available on a PCB, especially for VLSI components with thousands of pins. Second, for applications that one FPGA chip cannot meet the requirements, two or more coprocessor boards can be attached to a single workstation and connected to each other through their on-chip gigabit serial transceivers. These boards can be easily configured to different topological structures like star, array, mesh, or cube by external connections for different applications. Moreover, the total memory bandwidth is also enlarged accordingly. As we mentioned above, the data manipulating capability is a pivotal factor to the performance of seismic data processing applications.

4. PSTM on Reconfigurable Platform

Figure 4 is the structure of one arithmetic module (AM) inside FPGA, which evaluates the travel time function (Equation 1) and executes the summation operations. In order to achieve a high data throughput rate, a fully pipelined structure is employed. Several similar AMs can be fitted into one FPGA chip to manipulate their own data sets concurrently. Furthermore, multiple SPACE boards can be attached to a single workstation to increase its computing power dramatically.

The actual implementation of AM is more complex than that shown in Figure 4. Every arithmetic unit should be carefully designed in order to maximize its data throughput. The layout of arithmetic units inside the FPGA chip will affect the paths of data flows, which in turn affects the final execution speed. Several data buffers should be added to guarantee the fluent execution.

Figure 5 is the program flow of the PSTM kernel code executed on every accelerated workstation. The bold part of the program was migrated into FPGA. Here we place two inner iterations instead of one into FPGA to maximize the throughput of the long pipeline. There are only trivial

![Figure 3. Coprocessor board](image)

![Figure 4. Hardware structure of the AM](image)
differences between Figure 2 and Figure 5, which means that the corresponding seismic processing software can be easily migrated into this hardware platform. The PSTM program running on the mother workstation is almost the same as the software version except that it invokes SPACE as a subroutine and transmits input traces and pre-computed parameters into the coprocessor. When SPACE finishes all the migration calculations between one input trace and all local output traces, a signal is sent back to activate the transmission of next input trace. After all input traces are processed, the final output result is read out from SPACE for displaying or further processing steps.

5. DSR Travel-Time Solver

5.1 Hybrid DSR Travel-Time Solver

The evaluation of the Double Square Root (DSR) function (Equation 3) is the most time-consuming part of the PSTM procedure, which contains five multiplications, three additions, and two square-root operations. (Division by velocity in this equation can be replaced by multiplication using slowness table, which is the reciprocal of velocity.)

\[ T_{SR} = T_x + T_R = \sqrt{\frac{\tau^2 + \frac{S^2}{V^2}}{2}} + \sqrt{\frac{\tau^2 + \frac{R^2}{V^2}}{2}}. \]  (3)

This travel time equation will be evaluated for every output pseudo-depth in the whole local output section with respect to almost all input traces (Figure 2). Making the situation worse, square root is always the slowest hardware arithmetic unit whose latency is at least ten times more than a pipelined multiplier.

It’s always not the best solution for FPGA to implement an algorithm by simply mapping the software version into hardware. Considering properties of the algorithm along with characteristics of FPGA can usually produce a more hardware efficient solution. For the PSTM algorithm, CORDIC [8] arithmetic unit can be employed to evaluate \( T_x \) and \( T_R \) in Equation 3 by regarding \( \sqrt{X^2 + Y^2} \) as the norm of a vector in the Cartesian coordinate system. CORDIC is a vector-rotation-based hardware algorithm for trigonometric and other transcendental functions using only shifts and additions. [9] Its highly regular structure makes it amenable to FPGA implementation. Now only two multiplications, two CORDIC, and one addition are needed to solve the DSR function, which achieves more than 50% of area reduction.

\[ x' = x \cos \theta - y \sin \theta \]
\[ y' = x \sin \theta + y \cos \theta \]

\[ r' = x \cos \theta - y \sin \theta \]
\[ \theta' = \theta + \tan^{-1} \left( \frac{y}{x} \right) \]

\[ x_{\text{new}} = x_{\text{old}} - \frac{x_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]
\[ y_{\text{new}} = y_{\text{old}} - \frac{y_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ r_{\text{new}} = r_{\text{old}} - \frac{r_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ \theta_{\text{new}} = \theta_{\text{old}} - \Delta \theta \]

\[ x_{\text{new}} = x_{\text{old}} - \frac{x_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]
\[ y_{\text{new}} = y_{\text{old}} - \frac{y_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ r_{\text{new}} = r_{\text{old}} - \frac{r_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ \theta_{\text{new}} = \theta_{\text{old}} - \Delta \theta \]

\[ x_{\text{new}} = x_{\text{old}} - \frac{x_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]
\[ y_{\text{new}} = y_{\text{old}} - \frac{y_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ r_{\text{new}} = r_{\text{old}} - \frac{r_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ \theta_{\text{new}} = \theta_{\text{old}} - \Delta \theta \]

\[ x_{\text{new}} = x_{\text{old}} - \frac{x_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]
\[ y_{\text{new}} = y_{\text{old}} - \frac{y_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ r_{\text{new}} = r_{\text{old}} - \frac{r_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ \theta_{\text{new}} = \theta_{\text{old}} - \Delta \theta \]

\[ x_{\text{new}} = x_{\text{old}} - \frac{x_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]
\[ y_{\text{new}} = y_{\text{old}} - \frac{y_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ r_{\text{new}} = r_{\text{old}} - \frac{r_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ \theta_{\text{new}} = \theta_{\text{old}} - \Delta \theta \]

\[ x_{\text{new}} = x_{\text{old}} - \frac{x_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]
\[ y_{\text{new}} = y_{\text{old}} - \frac{y_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ r_{\text{new}} = r_{\text{old}} - \frac{r_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ \theta_{\text{new}} = \theta_{\text{old}} - \Delta \theta \]

\[ x_{\text{new}} = x_{\text{old}} - \frac{x_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]
\[ y_{\text{new}} = y_{\text{old}} - \frac{y_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ r_{\text{new}} = r_{\text{old}} - \frac{r_{\text{old}}}{r_{\text{old}}} \cdot \Delta \theta \]

\[ \theta_{\text{new}} = \theta_{\text{old}} - \Delta \theta \]
our application, we designed a modified hybrid CORDIC unit, which can provide better error limits than single-precision floating-point arithmetic with nearly the same area occupation as fixed-point CORDIC scheme. Figure 6 shows the corresponding hardware structure.

Our modified hybrid CORDIC unit differs from previous design [10] as follows:

- The CORDIC unit in our design employs fully pipelined hardwired add-and-shift stages and can achieve a much higher throughput than the recursive implementation in [10].
- The very first stages of our design accept two single-precision floating-point input data and convert them into internal aligned floating-point format. The same exponent of these two aligned internal data keeps constant while two mantissa parts are feed into a pipelined fixed-point CORDIC unit as two inputs. The word width of the mantissa parts is extended to provide enough error limits.
- The alignment operation in our design guarantees the same dynamic range as floating-point arithmetic and the extended word width of mantissas guarantees the same or even better error limit.
- The physical character of our application implies that the inputs to our CORDIC unit are two positive numbers. By parallel comparing two exponents and two mantissas, we can decide which operator is larger in one pipeline stage. So an extra exchanging stage is placed in front of those alignment stages to eliminate one costly barrel-shifter because only the smaller input needs this operation.
- Ordinary CORDIC algorithm needs two extra leading bits to prevent overflow because its processing gain is 2.33. The exchanging scheme of our design eliminates the first 45° rotation step in ordinary CORDIC algorithm. Now the processing gain of our CORDIC unit is 1.647, which means only one extra leading bit is needed. Moreover, the final normalization stage is simplified.
- No Leading-Zero-Detector (LZD) is needed in our design. LZD is usually used for normalizing intermediate results of floating-point arithmetic. Its implementation is not as easy as it looks and some complex hardware structures like array or tree are involved. [11]
- The final normalization step of the floating-point multiplication \( X \cdot \frac{1}{V} \) is eliminated because the carry bit of the exponent subtractor in exchanging stage can absorb the possible exponent increment. So this floating-point multiplication is almost the same as fixed-point arithmetic and can be implemented efficiently by using pipelined array multiplier or fast multiplier blocks available in Vertex II FPGA.
- Only the vector norm result of the CORDIC unit is needed, so we can save about 33% FPGA resources by omitting the Z channel completely.

In order to analyze the error property of our design, a C program was designed to simulate the exact iterative operations of the proposed CORDIC unit. In every single experiment, we create one million pairs of random single-precision floating-point positive numbers as the initial input to X and Y. The outputs of the proposed CORDIC unit are compared with corresponding double-precision floating-point results. Maximum errors and average errors are used as the performance metrics and their values have been amplified for \( 2^{20} \) times to make them easy to read and compare.

Table 1. Error property of the hybrid CORDIC unit with different guarding bits

<table>
<thead>
<tr>
<th>Wordwidth /Guarding Bits</th>
<th>Average Error (ppm)</th>
<th>Max. Error (ppm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25/0</td>
<td>0.0376</td>
<td>0.305</td>
</tr>
<tr>
<td>25/1</td>
<td>0.0181</td>
<td>0.170</td>
</tr>
<tr>
<td>25/2</td>
<td>0.0092</td>
<td>0.075</td>
</tr>
<tr>
<td>25/3</td>
<td>0.0046</td>
<td>0.034</td>
</tr>
<tr>
<td>25/4</td>
<td>0.0023</td>
<td>0.018</td>
</tr>
<tr>
<td>25/5</td>
<td>0.0012</td>
<td>0.009</td>
</tr>
<tr>
<td>floating-point</td>
<td>0.0260</td>
<td>0.120</td>
</tr>
</tbody>
</table>

Table 1 is the simulation results of the proposed CORDIC unit with word-width b=25 (One bit is added for preventing possible overflow.) and different guarding bits. The error limit of single-precision floating-point arithmetic for the same operations is also listed as a reference. In order to keep the same precision as single-precision floating-point arithmetic, a 25-bit CORDIC unit with two extra guarding bits is needed in the design. If the same average relative error is acceptable, a 25-bit CORDIC with one extra guarding bits is enough.

The final floating-point addition of \( T_s \) and \( T_R \) is simpler than an ordinary floating-point addition unit because all these two operators are positive so that subtraction doesn’t need to be taken into account.

5.2 Fixed-point Travel-time Solver

We can do better if we consider the physical meaning of travel time. The value of travel time is used in the final accumulation stage as a time index to fetch the proper sample point in an input trace. Its value should be bounded by the time of the last sample in an input trace \( T_{\text{max}} \), which is less than 16 seconds in most cases. The sample interval of input trace is usually coarser than
1ms. A more ambitious approach tries to employ pure fixed-point arithmetic units to evaluate this travel time equation. Figure 7 shows the fixed-point format used in this approach and Figure 8 are the structure of the fixed-point DSR travel time solver.

**Figure 7. Output format of the conversion stage**

<table>
<thead>
<tr>
<th>Integer Bits</th>
<th>Fraction Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Radix-Point

**Figure 8. Hardware structure of the fixed-point DSR travel-time solver**

The worst-case absolute error of the final time results should be larger than 0.5ms, which means the error limit for every CORDIC channel is less than 0.25ms. There are two error sources: one is the rounding error comes from the first conversion stage that converts two floating-point numbers into aligned fixed-point values; the other is the computing error caused by the fixed-point CORDIC unit. Table 2 lists average and maximum rounding errors caused by the conversion stage with different fraction word width. Implementation results of the fixed-point CORDIC unit with different word-width and guarding bits are listed in Table 3.

Table 2. Rounding error of the conversion stage with different fraction word-width

<table>
<thead>
<tr>
<th>Fraction Wordwidth</th>
<th>Ave. Rounding Error</th>
<th>Max. Rounding Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.000236</td>
<td>0.000691</td>
</tr>
<tr>
<td>11</td>
<td>0.000118</td>
<td>0.000345</td>
</tr>
<tr>
<td>12</td>
<td>0.000059</td>
<td>0.000174</td>
</tr>
<tr>
<td>13</td>
<td>0.000030</td>
<td>0.000087</td>
</tr>
</tbody>
</table>

5.3 Optimized 6th order Travel-time Solver

The travel time calculated by Equation 3 is a 2nd order approximation to the following Taner’s travel time equation [12] for a horizontally stratified medium model:

\[ T_{SR} = T_S + \frac{cc S^6}{T_S} + T_R + \frac{cc R^6}{T_R} \]  

(5)

Where \( T_X ^2 = c_1 + c_2 X^2 + c_3 X^4 + c_4 X^6 + \Lambda \).  

(4)

\[ X \] is the offset and \( c_k \) are priori-estimated coefficient tables associated with the output section. (For example, \( c_1 = \tau^2 \) and \( c_2 = \frac{1}{V^2} \) )

The accuracy will be improved significantly when we include higher order series into the travel time calculation. People always adopt the 4th or 6th truncation schemes because the evaluation of the coefficients for higher than 6th order terms are not practical. In our design, we use an optimized 6th order scheme proposed by Sun [13] as follows:

\[ T_{SR} = (T_S + \frac{cc S^6}{T_S}) + (T_R + \frac{cc R^6}{T_R}) \]  

(5)

Where

\[ T_S = \sqrt{c_1 + c_2 S^2 + c_3 S^4} \]  

(6)

\[ T_R = \sqrt{c_1 + c_2 R^2 + c_1 R^4} \]  

(7)
The definitions and values for the first three coefficients are the same as Equation 4, but the \( c_4 \) term is modified by taking other higher order terms into account.

The hardware design of the optimized 6th order travel time solver is a direct expansion of our previous design depicted in Figure 4. The evaluation of Equation 5 is straightforward. Equation 6 and 7 can be rewritten as:

\[
T_X = \sqrt{(\sqrt{c_1 + c_2X^2})^2 + c_3(X^2)^2}. \tag{8}
\]

Obviously, two cascaded CORDIC units can finish this calculation.

Three coefficients table are needed in this scheme comparing to only one RMS velocity table in previous design, i.e. one evaluation of \( T_{se} \) needs to access three coefficients from memory comparing to one previously. So the memory capacity and bandwidth is the main bottleneck to this scheme.

### 6. Performance Comparison

The aim of this section is to show the speedup of the proposed SPACE platform comparing to a referential Intel P4 2.4GHz workstation without losing process accuracy. The performance comparison contains precision comparison and speed comparison. A real 3D input data volume is used for the comparison which contains 186512 input traces and each trace has 1500 samples with 4ms sample interval. The 3D output image cube contains 90 by 500 surface positions, by about 1500 pseudo-depth points per output trace. Figure 9 shows the image of a vertical in-line section selected from the stacked input data. Figure 10 is the migrated image for the same output section created by a simulation program, which imitates the same operations and precision as the hardware design on the SPACE platform. This migrated image is the same as the result of the pure software version of the PSTM algorithm running on the referential workstation. Notice that migration provides people a much reliable underground image and makes detailed subsurface structures easy to tell. For example, the inclination of the reflection event A is increased and clarified in Figure 10, and the vague event B in Figure 9 is turned into a syncline at the same position in Figure 10.

Define an elementary computation as all the operations required for each input-output points pair to calculate the migration travel time, oblique factor, anti-aliasing filtering and output accumulation, the number of all elementary computations for this data volume is about 644 billions, considering migration aperture and the \( T_{max} \) limitation. The total execution time on the referential Intel workstation is 206570 seconds, in which more than 98% (202468 Seconds) are consumed by elementary computations and less than 2% (4102 Seconds) by other operations. In the following quantitative performance analysis, we use \( T_{CORE} \) as the elapsed time for all the elementary computations, \( T_{OTHER} \) as the time for other assistant works including initialization, data preparation, communication, etc. We have:

\[
T_{SOFT} = T_{CORE} + T_{OTHER}. \tag{4}
\]
The proposed coprocessor accelerates only the elementary computation part of the program and leaves all other operations to its mother machine. So the new total execution time will be:

$$T_{\text{HARD}} = T_{\text{CORE}}^* + T_{\text{OTHER}}.$$  (5)

According to Amdahl’s Law [14], the overall speedup is:

$$\text{Speedup} = \frac{T_{\text{SOFT}}}{T_{\text{HARD}}} = \frac{1}{(1 - \text{CoreFraction}) + \frac{\text{CoreFraction}}{\text{CoreSpeedup}}}.$$  (6)

Table 4 shows the theoretical performance comparison results for the designated task between the referential Intel workstation and the proposed SPACE platform with different configurations.

<table>
<thead>
<tr>
<th>Configurations</th>
<th>Clock Frequency (Hz)</th>
<th>FPGA Resources Occupation (%)</th>
<th>Kernel Code Speed (million/s)</th>
<th>Kernel Code Speedup</th>
<th>Overall Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel P4 Workstation</td>
<td>2.4G</td>
<td>NA</td>
<td>3.2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>SPACE with One AM</td>
<td>50M</td>
<td>18.6</td>
<td>50</td>
<td>15.6</td>
<td>10.8</td>
</tr>
<tr>
<td>SPACE with Two AM</td>
<td>50M</td>
<td>32.8</td>
<td>100</td>
<td>31.2</td>
<td>16.4</td>
</tr>
<tr>
<td>SPACE with Four AM</td>
<td>50M</td>
<td>61.4</td>
<td>200</td>
<td>62.4</td>
<td>22</td>
</tr>
</tbody>
</table>

Following observations can be drawn from Table 4:

- The execution speed of a single arithmetic module (AM) inside the FPGA is 15.6 times faster than the referential Intel workstation with less than 20% hardware resources usage. This impressive result comes from the fully pipelined structure of the AM.
- The speedup of the kernel code of PSTM algorithm will increase linearly with the number of AMs inside the FPGA but the overall speedup is limited by $T_{\text{OTHER}}^*$, which is constant for the designated task. This is partly because of the small data volume we used for the comparison. A bigger task will increase the proportion of elementary computations and the overall speedup will rise accordingly.
- The density (hardware resources) of an FPGA will bound the number of AMs integrated inside it. On the other hand, performance improvement can be gained easily by integrating more AMs into denser FPGAs in the future.
- Memory bandwidth is another bottleneck especially when more AMs are integrated into the chip. Employing faster memory modules (for example, DDR400) can partly alleviate this problem.
- This comparison doesn’t take into account the speed degradation introduced by some hardware behaviors such as pipeline stall. For the PSTM algorithm, switching to next output trace will lead to control hazard that is similar to the branching stall in a pipelined generic CPU platform. This control hazard is hard to predict or avoid. Theoretically, simply flashing pipeline will cause at most 10% performance loss considering the big difference between the number of pseudo-depth points per output trace and our pipeline stages.

7. Conclusions and future work

A novel coprocessor platform is proposed to accelerate the 3D pre-stack Kirchhoff time migration. To the best of our knowledge, this is the first attempt to implement a practical seismic data processing procedure using FPGA technology. The simulation result shows an impressive speedup comparing with a referential Intel P4 2.4 GHz workstation. The proposed SPACE platform can be easily attached to present PC cluster systems booming their performance for more than ten times with modest software migration work.

The main motivation for introducing reconfigurable computing resources to seismic data processing industry is that the same hardware resources can be reconfigured for different algorithms used in different processing stages. Future work in this field will concentrate on extending the utilization of the proposed SPACE platform to Kirchhoff depth migration and other migration methods.

Acknowledgement

The guidance of Prof. Philip Rabinowitz is gratefully acknowledged.

References


