Preliminary results using Acceleware's AxRTM API

Babatunde Arenrin, Gary Margrave, and Rolf Maier

ABSTRACT

We describe a seismic modelling program based on modules designed to perform forward modelling and reverse time migration, and show results derived from the code. The software we used demands that we provide a velocity model and necessary parameters, which must be fed to a central repository in predefined structures from which the forward modelling and migration software read the required information.

We perform forward modelling and reverse time migration on a simple velocity model consisting of three horizontal layers. To demonstrate the output to be expected from the forward modelling and reverse time migration software, we migrated a shot record with a split spread configuration, having 800 receiver locations spaced 10 meters apart.

The migrated shot record from AxRTM produced desirable results. However, there are some low frequency artefacts that need to be filtered for improved result. The forward modelling and reverse time migration program have been benchmarked and compared with corresponding programs in CREWES Matlab toolbox.

INTRODUCTION

A few modelling packages are available at CREWES to obtain synthetic shots. Among these are Tiger from SINTEF (3D FD, elastic), Norsar 2D and 3D from Norsar, the locally written codes of Peter Manning (2D and 3D FD), and the Matlab codes of Gary Margrave and his students.

The software on which the work presented here is based on is new to CREWES; AxRTM from Acceleware became available to us as a result of Acceleware becoming a CREWES sponsor. The software is not a single application with its various input lists pertaining to a series of tasks it can perform, but a library of compiled objects that are to be used inside a C program or package a user wants to develop. This allows for experimentation with both prebuilt and locally built modules.

AxRTM is the first piece of software available to us which utilizes CUDA devices, i. e. programmable graphics cards or GPUs.

The purpose of the code described here is to build a forward modelling program which can be built upon. The program is in part based on existing sample codes. The software arrives with a simple migration code that is meant to be adapted and extended as required.

Elements of AxRTM forward modelling and migration algorithms

Introducing a little terminology simplifies explaining how the software works. Since the user interacts with the software not via a GUI, but a program written in the C

language, including objects written in C/C++ which are bundled into a library, it is necessary for their working together that some conventions are being followed.

Functions have fairly descriptive names. When there is more than one way to perform a task, the functions are told what exactly to do via enumerated integers. Although there is synoptic help available for all functions in the API reference, the best, and often clearest information on certain details is found in the major include file in the include directory.

The include file is particularly useful in deciphering the meaning of the vehicles the software uses to keep information, little structures the software calls modules. Modules are defined as extended data types with typedef statements. The modules are filled via sometimes repeated function calls, each time a descriptor called a handle is returned in the form of an integer. Note the rather colloquial use of the term handle.

For instance, receiver coordinates are given to the underlying runtime in x, y, z triplets, hence a location at which a trace is desired is indicated by three coordinates to which AxRTM attaches a tag it calls a handle. This handle is to be used later to retrieve the data trace which the forward modelling mode will associate with the given coordinates for a given shotpoint and velocity field. Any kinds of data are referred to in this manner.

On a different note, a velocity profile must be regularly spaced. Hence, only a few pertinent numbers are required; the velocities are then read from a binary file in a single call. In summary, the package consists of functions which are directed by and talk to each other via enumerated constants, and which refer to portions of a central data bank via handles. Each handle is a numerical tag referring to a small structure, called a module, which is of one of several predefined types.

Code snippets

The first sample code shows how the forward modelling code performs time steps, printing an occasional status message. The last call finishes the job before the result is extracted from the internal buffers and written to disk.

```
/*-------------------------------------------------------------------------*
 * 3F. Forward Pass                                                        *
 *-------------------------------------------------------------------------*/
 printf (" * Forward Pass\n");

 /* Perform timestepping */
 ax_uint_t       timestep = 0;
 ax_uint_t       batchSize = 50;
 ax_uint_t       remaining;

 AX_CALL (AxGetNumTimestepsRemaining (jobHandle, &remaining));

 do {
     if (remaining < batchSize)
         batchSize = remaining;

     printf ("   - Performing forward timesteps %d to %d\n", timestep,
             timestep+batchSize-1);
     AX_CALL (AxDoTimesteps (jobHandle, batchSize));
```

```
    timestep += batchSize;

    AX_CALL (AxGetNumTimestepsRemaining (jobHandle, &remaining));
} while (remaining > 0);

AX_CALL (AxFinalizeForwardPass (jobHandle));/* Finalize Forward Pass stage*/
```

FIG. 1. Code snippet for the forward modelling engine.

If the dominant frequency of the chosen wavelet is low then the migrated data need to be filtered to remove the low frequency noise typical for RTM. We wrote a filtering program based on utilizing functions provided with the software to do this; the operative part is shown below.

```
AX_CALL (AxCreateLowCutFilter (dimensionality, &filterHandle));

float filterDomFreq = 10;    /* Typically 2x the lowest temporal frequency */
                             /* in your input data */
AX_CALL (AxFilterSetDominantFrequency (filterHandle, filterDomFreq));

AX_CALL (AxFilterSetDeltas (filterHandle, dx, dy, dz));

/* image and velocity buffers are */
AX_CALL (AxApplyFilter (filterHandle, imageBuffer, velocityBuffer,
    outputBuffer));
```

FIG. 2. Code snippet for the 10 Hz low cut filter.

During forward modelling and reverse time migration, AxRTM manages work in multiple CPUs and other computer resources in child processes. It is designed to run parallel jobs in the case of large volumes of data.

Synthetic examples

To demonstrate what is obtainable using AxRTM, we produced a shot gather and a migrated image for a horizontal three-layer model. As mentioned earlier, we benchmarked the results with the forward modelling (Afd_shotrec_alt) and migration program (PSPI) in Matlab that is available in the CREWES toolbox. The migrated images are presented in Figures 9 and 10.

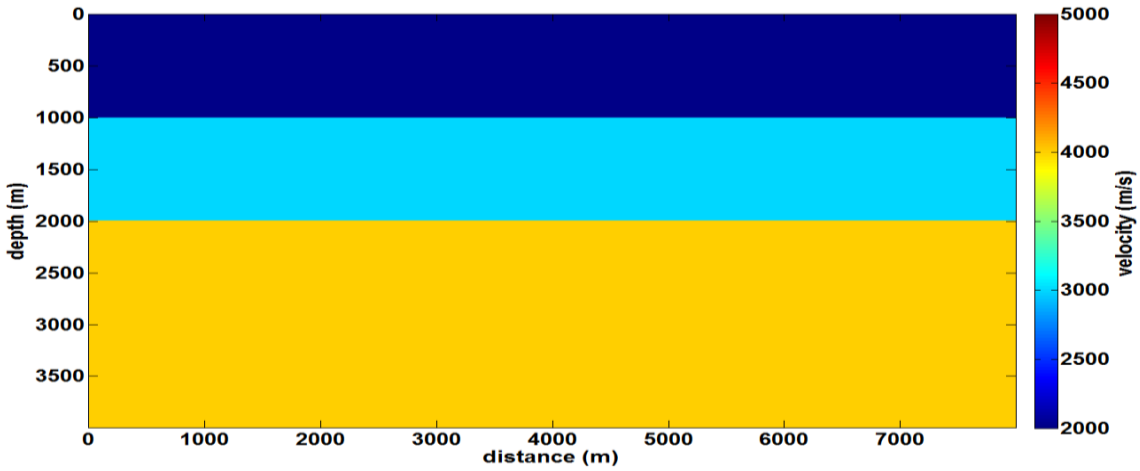

FIG. 3. Velocity model for the tests.

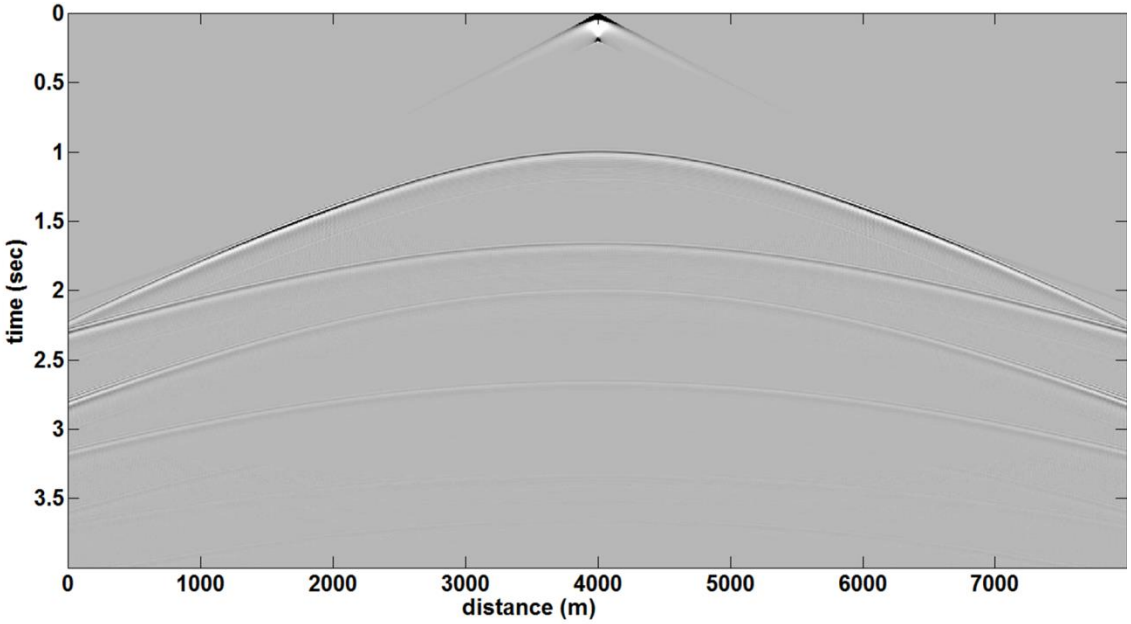

FIG. 4. Shot record from AxRTM.

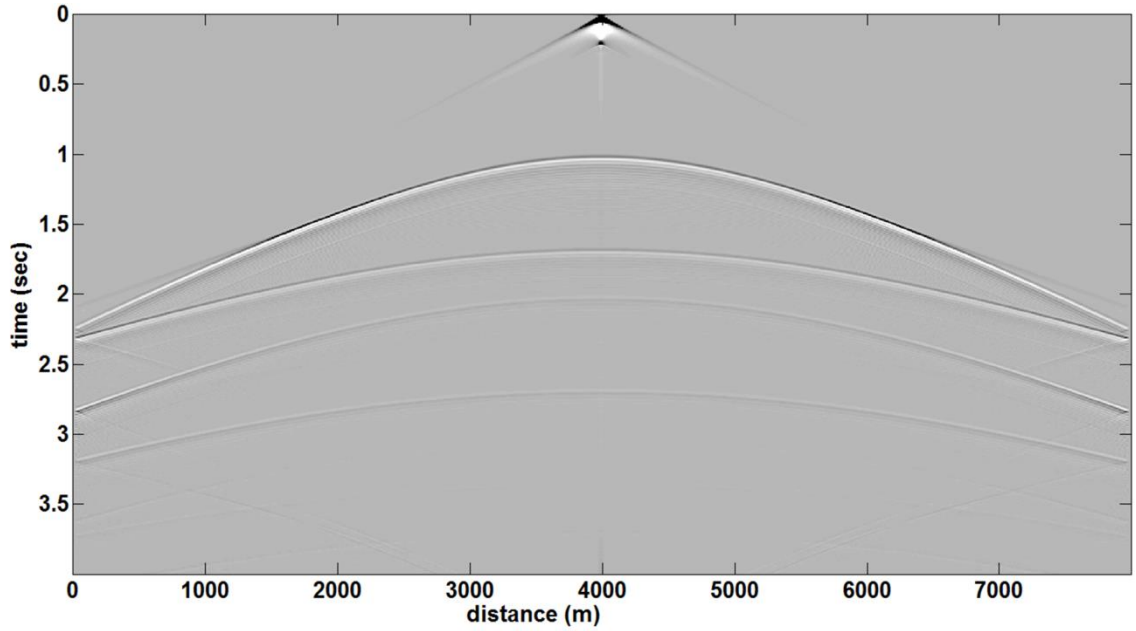

FIG. 5. Shot record from Afd_shotrec_alt from the CREWES Matlab toolbox.

Figure 3 shows the velocity model. Figure 4 is a shot record using AxRTM, and Figure 5 is from Afd_shotrec_alt from CREWES Matlab toolbox. In both cases the parameterization set are exactly the same i.e. similar propagation steps, sample intervals, and wavelet (minimum phase). Notice the similarities in the plots. Figure 6 is a plot of a trace extracted at 500 meters from AxRTM and Afd_shotrec_alt. Notice the difference in the amplitudes between the two traces. Figure 7 shows the two traces after amplitude balancing. Figure 8 shows an overlay of the amplitude spectrum of the traces in Figure 7.

We observed from Figure 8 that despite the bandwidth of the two traces being slightly different, they are in phase with each other.

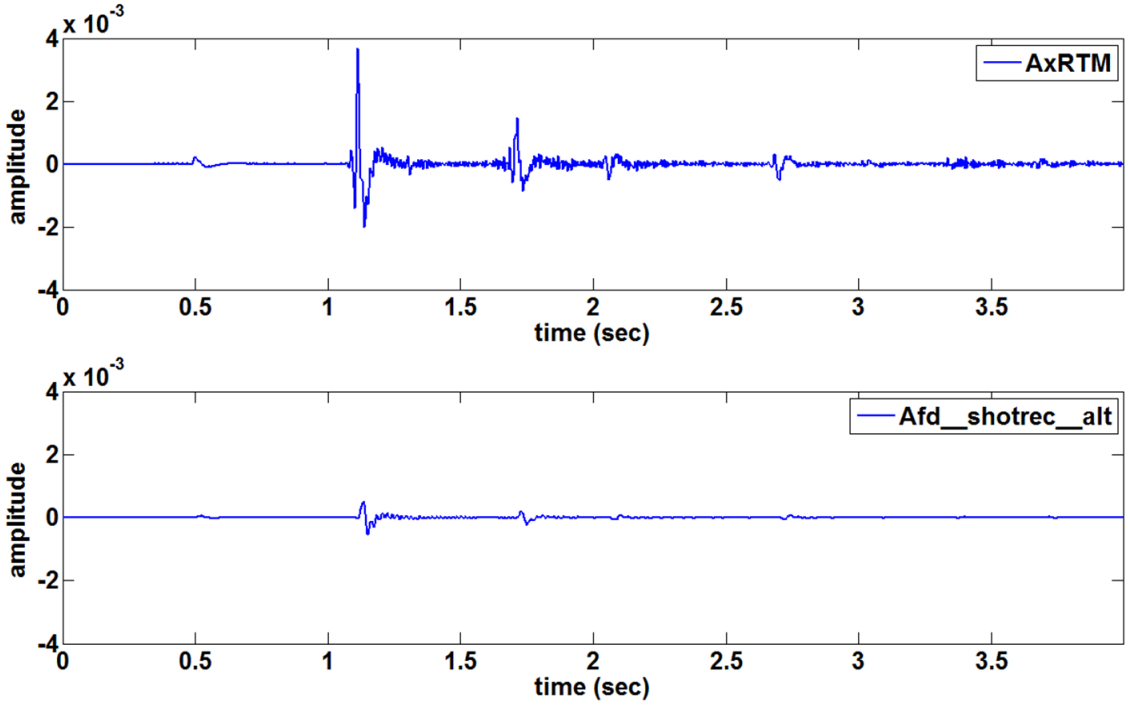

FIG. 6. The same traces from AxRTM and Afd_shotrec_alt.

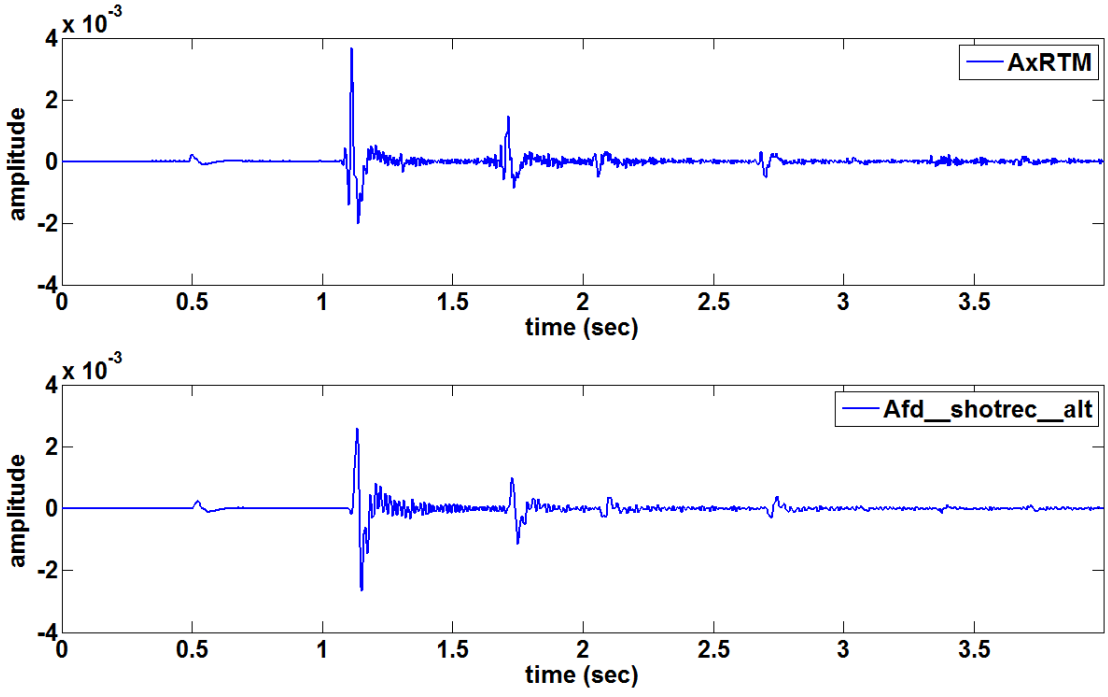

FIG. 7. The same traces from AxRTM and Afd_shotrec_alt after amplitude balancing.

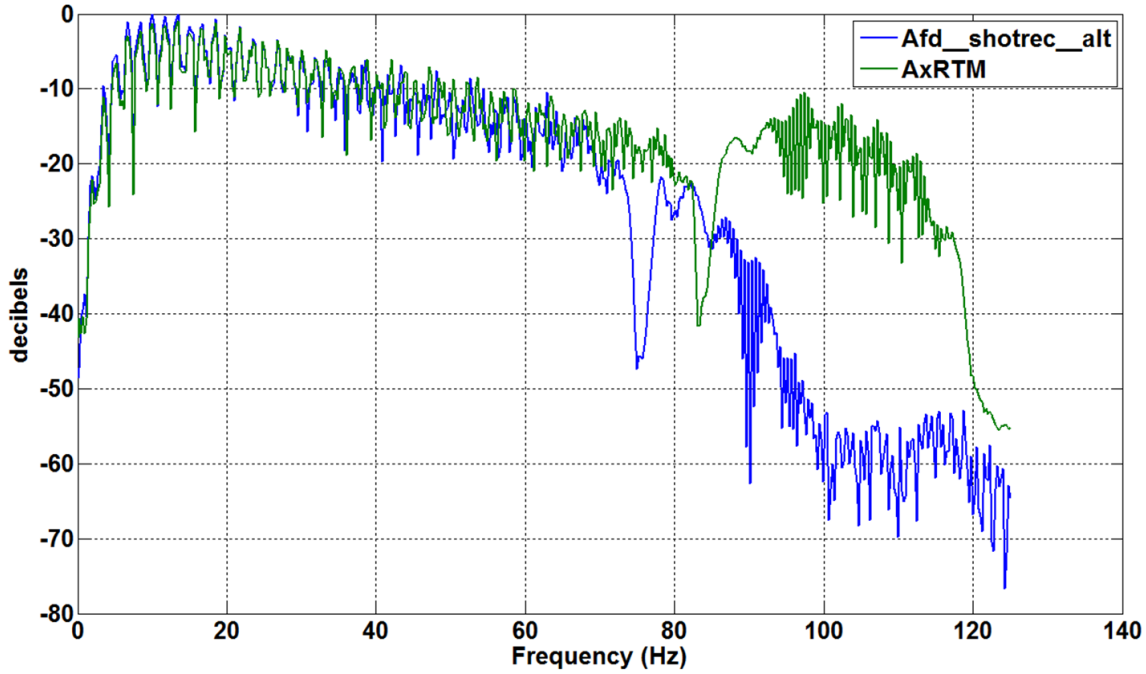

FIG. 8. Amplitude spectrum of the trace at 500 meters. Spectrum of AxRTM (green) and Afd_shotrec (blue).

Finally we show the migrated shot using AxRTM (Figure 9) and PSPI (Figure 10).

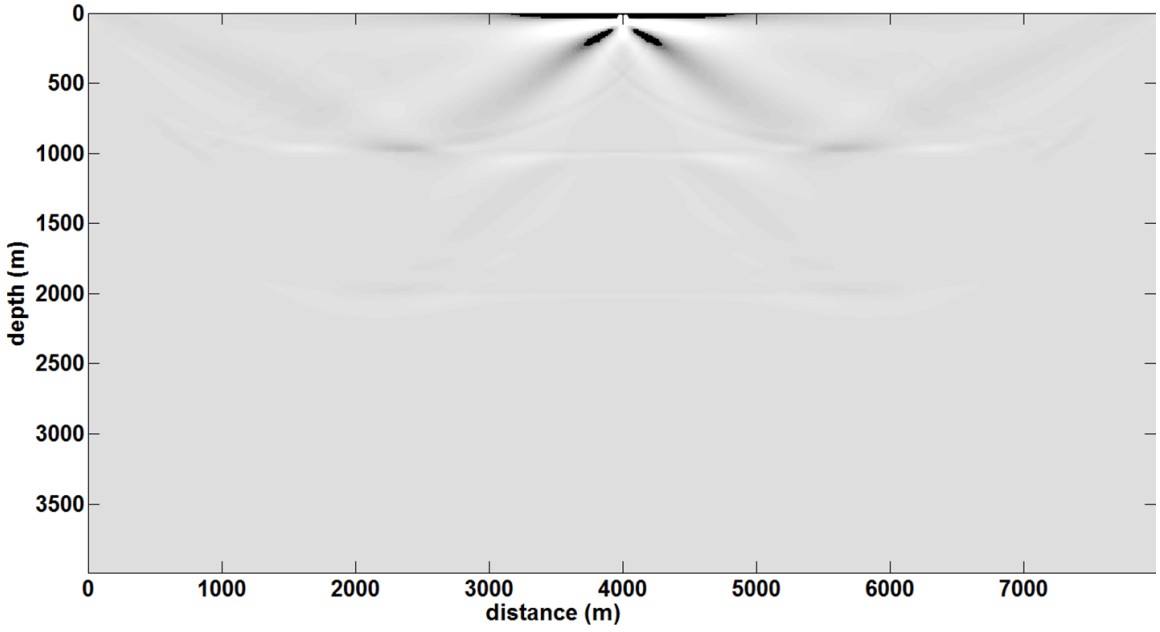

FIG 9. Migrated shot from AxRTM.

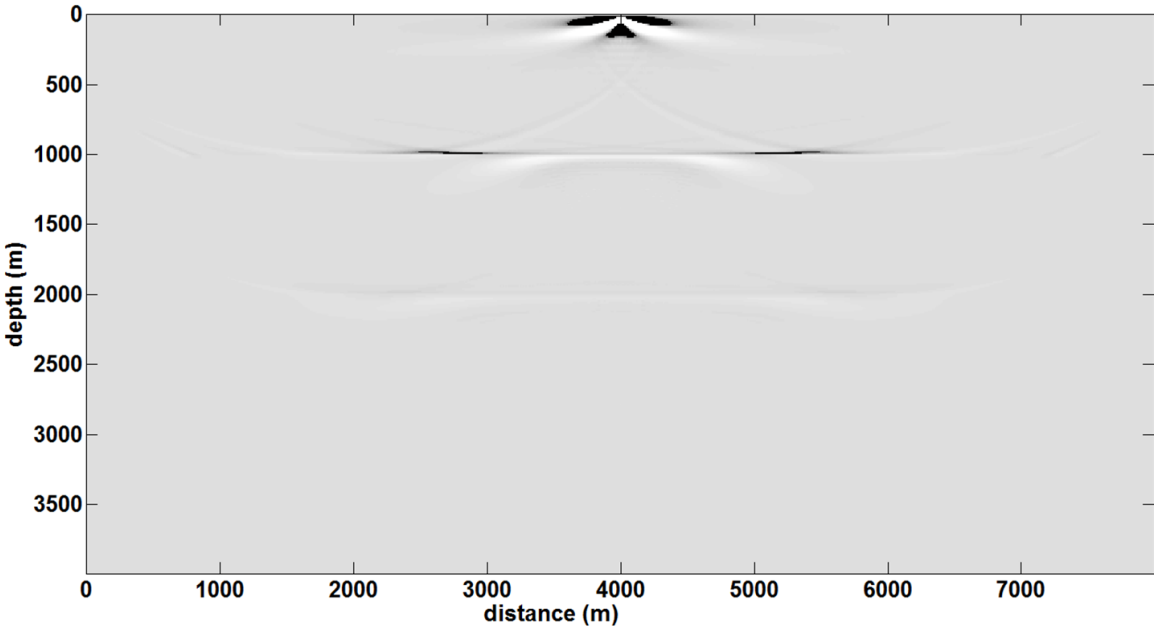

FIG. 10. Migrated shot from PSPI code written in Matlab.

Comparing the migrated images above, the migrated image from AxRTM's reverse time migration appears to suffer from low wavenumber artefacts in the upper section of the image. This is not a huge problem as a filter can be used to get rid of the artefacts. The PSPI algorithm on the other hand seems to be less affected by cross correlation artefacts. However, both algorithms seem to have produced a decent image of the shot gather.

DISCUSSION

There are a few things a new user needs to be aware of. Once the terminology, as explained above, is understood, parts of programs become reusable. The use of words with a fixed meaning in programming in a nonstandard way proved to be a bit of a hurdle. The documentation does not prepare a user of these little details, but the API reference guide provides a synoptic description of all functions, and descriptions of all their arguments.

Tests showed that Acceleware's forward modelling and reverse time migration program (AxRTM) runs at least ten times faster than the corresponding Matlab programs. However, the Matlab programs ran on CPUs while Acceleware's programs were made to run on a single graphics card. Allowing for multiple graphics cards in the case of a single shot test slows down a run significantly because of the time taken to initialize the cards. We have not done large array examples.

CONCLUSION

We have been able to show that AxRTM API from Acceleware can be used for forward modelling and reverse time migration. The results from AxRTM are comparable to the results from forward modelling and migration codes written in Matlab from the

CREWES toolbox. However, the migrated image from AxRTM suffers from artefacts or crosstalk inherent in any reverse time migration algorithm. However the package comes with the flexibility of applying a filter to the migrated image.

ACKNOWLEDGEMENTS

We like to acknowledge the continued support of CREWES sponsors and NSERC. We like to thank Darren Foltinek from Acceleware for readily pointing us in the right direction when we needed help. We also want to thank Kevin Hall of CREWES for providing us with the computing resources to do this work, and CREWES staff and students.