Acceleration of Anisotropic Phase Shift Plus Intepolation with Dataflow Engines

Carlo Tomas[1*] *, Luca Cazzola*[1]*, Diego Oriato*[2]*, Oliver Pell*[2]*, Daniela Theis*[3]*, Guido Satta*[3]*, and Ernesto Bonomi*[3]
[1]*ENI E&P Division, Italy,* [2]*Maxeler Technologies,* [3]*CRS4 Imaging and Numerical Geophysics, Italy*

SUMMARY

Although *time-domain depth migration* techniques have been successfully ported to run on modern hardware accelerators, their ultimate obstacle is the I/O overhead present during the imaging step. *Frequency-domain depth migration* algorithms overcome this limitation and can exploit the full potential of new computing technologies. In particular, our implementation of Phase Shift Plus Interpolation (PSPI) method is characterized by fast running time, good quality results under low-signal-to-noise ratio conditions and excellent results for steep dips. We provide a novel computational dataflow scheme to perform acceleration of PSPI on a generic dataflow engine. We present speedup results obtained on the state-of-the-art dataflow technology for synthetic VTI datasets. Our measurements indicate that a dataflow approach can achieve high speedups despite larger and larger computational domains, increased complexity of the anisotropic approach and the I/O overhead during angle-gathers calculation.

INTRODUCTION

The availability of new hardware platforms presents an opportunity and a challenge for seismic computation. The opportunity offers the clear benefit of a faster application and the possibility of implementing more complex and accurate models. The challenge lies in restructuring the algorithm to efficiently exploit the new hardware. Dataflow programming focuses on modeling a program as a directed graph and dataflow engines exploit the intrinsic parallelism of a physical graph of instructions executed concurrently on a continuous stream of data.

Frequency-domain depth migration based on the phase-shift formula utilizes two-dimensional Fourier transform as a fundamental mathematical tool. The performance benefit of this algorithm derives from the fact that, after the Fourier transform, data are decomposed in a series of n frequency panels $\omega_{1,\dots,n}$. Each frequency panel is processed independently from the others as an application of the linearity principle. This reduces the dimensionality of the propagation problem and makes it a natural and efficient domain for code parallelization (Bagaini et al., 1995). To overcome the problem of lateral variations in velocity, the PSPI algorithm applies the phase-shift formula for several reference velocities covering the range of lateral variability found in the layer. Next, the wavefield at a given location is interpolated from the different phase shifts based on the local velocity. We have identified the phase shift of different reference velocities as another axis for code parallelization once the inter-process communication of the interpolation phase is solved.

Our literature search did not find any precedents in the acceleration of the PSPI method using dataflow computing. We believe this is because two-way time-domain depth migration techniques are nowadays more accessible, more obviously benefit from acceleration and are easier to parallelize with current accelerators (Oriato et al., 2010). However, in our implementation of PSPI, the design of an innovative depth extrapolation operator has recently led to significantly better results than conventional methods, proving itself to be particularly effective for imaging very steeply dipping reflector in the presence of severe lateral velocity variations, thus making PSPI a real contender to two-way migration techniques.

On the porting of the algorithm to a dataflow paradigm we believe we have identified the key factors to achieve optimal performance on a dataflow engine. The first step is to efficiently exploit all the code parallelism available in the PSPI algorithm:

- the n frequency panels $\omega_{1,\dots,n}$
- the reference velocities
- the decomposition of a 2D FFT into 1D FFTs
- the depth extrapolation of both source and receivers.

The second step is to apply bespoke wavefield compression techniques to optimize memory bandwidth requirements without degradation in the quality of the results.

ACCELERATION OF PSPI IN VTI MEDIA

PSPI Algorithm Whereas seismic imaging in a stratified medium can be done with plane Phase-Shift (Gazdag, 1978), the case with lateral velocity variations requires more attention. In this context the Fourier representation of the scalar wave equation is meaningless and no straightforward representation of the solution as with the phase shift formula is possible. To overcome this difficulty and yet keep the computational complexity of the migration to a minimum, the wave propagation model is modified in order to construct a pure spectral method for downward continuation in an inhomogeneous medium (Gazdag and Sguazzero, 1984). The starting point is the phase shift formula

$$\hat{P}(k_x,k_y,z+\Delta z,\omega) = \hat{P}(k_x,k_y,z,\omega)e^{\pm ik_z\Delta z}, \qquad (1)$$

split into vertical and horizontal components and then modified to handle wave propagation inside the layer $]z\,;\,z+\Delta z]$ which has a laterally variable velocity field. The resulting first term governs vertically-travelling waves through the layer:

$$\hat{P}_0(x,y,z,\omega) = \hat{P}(x,y,z,\omega)e^{i\frac{\omega}{v}\Delta z}, \quad v = v_z(x,y). \qquad (2)$$

The second term governs the horizontal correction for a reference velocity $v_z^{(j)}$, one of $v_z^{(1)} < v_z^{(2)} < \cdots < v_z^{(n_z)}$:

$$\hat{P}^{(n)}(k_x,k_y,z+\Delta z,\omega) = \hat{P}_0(k_x,k_y,z,\omega)e^{i\left(k_z^{[n]}-\frac{\omega}{v_z^{(n)}}\right)\Delta z}, \qquad (3)$$

with $k_z^{[n]}$, $n = 1, 2, \ldots, n_z$, given by

$$k_z = \sqrt{\left(\frac{\omega}{v_z^{(n)}}\right)^2 - (k_x^2 + k_y^2)}. \qquad (4)$$

The fields $\hat{P}^{(n)}(k_x, k_y, z+\Delta z, \omega)$ are inverse tranformed to the space-frequency domain and the transformed fields $P^{(n)}(x, y, z+\Delta z, \omega)$ serve as reference data from which the final result is obtained by interpolation. Finally, the imaging of the migrated data at $z+\Delta z$ is obtained by the standard condition.

The velocities $v_z^{(n)}$, which play a crucial role, are chosen using an adaptive scheme, based on information theory concepts (Bonomi et al., 1998) : the task is to highlight a minimal set of velocity values that predominate statistically in the propagation process. This approach turned out to be very effective and can significantly diminish the computing cost of PSPI migration.

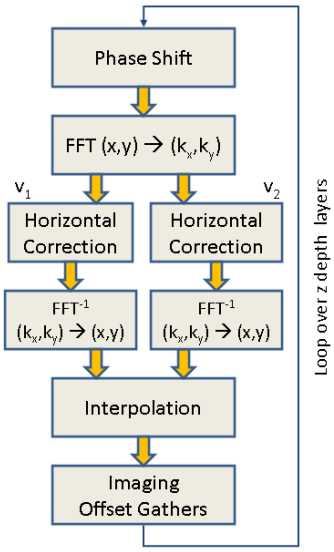

Figure 1: Workflow for PSPI with two reference velocities.

Acceleration Architecture A dataflow engine can be thought as programmable hardware logic (i.e. Field-Programmable Gate Array) connected to a local memory that provides a reservoir of data that can be accessed through a high bandwidth channel. It is also connected to a CPU via a slower connection as depicted in Figure 2.

Partitioning the computation between dataflow engine and CPU is a crucial step to fully exploit the hardware resources available and minimize data transfer bottlenecks. Analysis of the software reveals that the logical components depicted in Figure 1 have similar computational weights. Figure 3 shows an

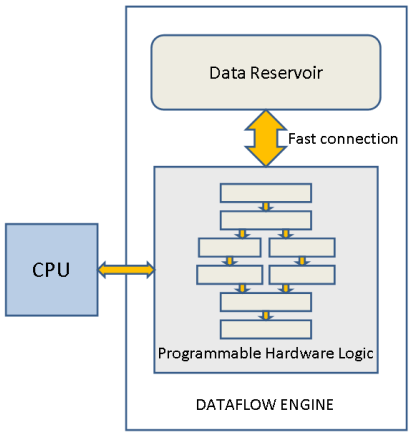

Figure 2: Architecture of a dataflow engine.

example of the software profiling for a very simple layer with two reference velocities.

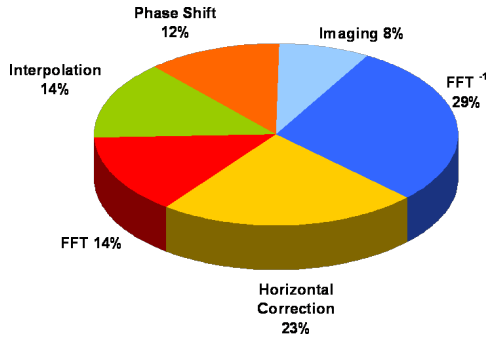

Figure 3: Components Breakdown by CPU utilization for two reference velocities.

Based on this analysis, we determined that all six PSPI components would need to be computed using the dataflow engine to provide maximum performance. All data is moved in the local memory on the dataflow engine during the initialization phase. The image and the angle gathers results were transferred to the CPU at the end of the full shot migration. Since in a dataflow engine each instruction occupies physical space in the silicon device, the accelerated implementation had to settle for a maximum number of reference velocities $v_1, \ldots, v_m$ that could be computed in a single streaming phase of the data. In the case where, for example, a layer had $v_1 < \ldots < v_n$ reference velocities with $m < n < 2m$, the computation was executed in two streaming phases each with overlapping sets of reference velocities $[v_1, v_m]$ and $[v_m, v_n]$. Computation of the reference velocity v_m is performed twice to guarantee that the interpolation block had always two consecutive reference velocities available. The 2D-FFT step has been implemented with 1D-FFT hardware logic blocks. The Fourier transform was first executed along the rows of the computation plane. Then the

data were transposed so that the next Fourier transform step was carried out on the columns of the plane. The most efficient way of performing the data transposition is to write the data to the dataflow engines memory in one direction then read it back in another. This means that a single depth migration step had to be implemented as two streaming steps as shown in Figure 4.

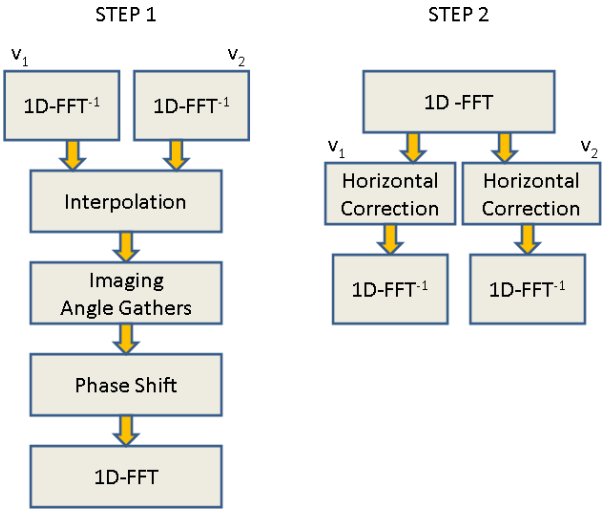

Figure 4: Workflow for PSPI with two reference velocities.

The complex source and receiver wavefields at $z = 0$ were initially transferred from CPU to the dataflow engine memory. From here both source and receiver wavefields were streamed into the hardware logic, extrapolated in parallel and combined together in the imaging logic. The image was accumulated over the frequencies. As it can be observed from Figure 4, the necessity of breaking the 2D-FFT step in two 1D-FFT phases meant that we had to change the order of execution of the PSPI components. In the very beginning of the calculation at step 1 when the first layer $z = 0$ had to be propagated, the blocks 1D-FFT^{-1}, Interpolation and Imaging/Angle Gathers were disabled. Then step 2 was computed followed by the whole step 1 again. During the second invocation of step 1, the depth extrapolation of the first layer was concluded and the output processed directly as input for the next layer within the same streaming step. This cycle repeated until the very last layer where computation stopped at the Imagin/Angle Gather component of step 1.

Wavefield Compression The streaming of wavefields between the dataflow engine memory and the hardware logic utilizes a high bandwidth connection. However during both step 1 and step 2 of Figure 4 not only the complex source and receiver wavefields need to be streamed but also their intermediate values. This intermediate data consist of the extrapolation results for each reference velocity and can be very large. To reduce the bandwidth requirement we implemented a lossy compression mechanism where the wavefields and the intermediate data were compressed before being written back to the memory and decompressed once read from it. A similar technique has been previously deployed on time-domain depth migration algorithms and has given very good results. The technique is based on the compression of a small group of wavefield values that are spatially adjacent on the assumption that their values have similar amplitude. Although this is true for a signal in time and space, we verified that this applies also for a signal in the frequency domain.

Imaging and angle gathers The imaging step and the angle gathers generation was computed in the hardware logic of the dataflow engine. Both are computed in parallel and accumulated over the frequencies of the signals. The result for each layer was stored in the dataflow engines memory. The angle-gather volumes were then streamed back to the CPU once the migration had reached the deepest z-layer. The overhead of this final step is very minimal compared to the full extrapolation time. Since each angle gather was computed in parallel utilizing separate accumulator hardware logic, a fixed maximum number of gathers could be computed during a single depth migration. Again flexibility on the number of gathers could be traded against speed of computation. The angle-gathers data structure is arranged so that their direction is the *fast* dimension of the streaming data.

RESULTS AND SPEEDUPS

In our tests we have used the SEG C3NA velocity model and simulated a new VTI synthetic dataset with a smaller binsize to compare hardware and software results and to compute the speedup.

To implement the dataflow architecture and to test its speed, we have utilized Maxeler Technologies software and hardware platforms. We used two generations of Maxelers hardware. The MaxNode-1842 consists of a 1U compute node with 8 Intel Xeon “Westmere” cores coupled to 4 MAX2 dataflow engines, each of which is equipped with a Xilinx Virtex-5 and has 12GB of DDR2 DRAM. We also tested on the newer MaxNode-1843, a 1U node with 8 Intel Xeon “Westmere” cores coupled to 4 MAX3 dataflow engines. The MAX3 dataflow engine is equipped with a Xilinx Virtex 6, has twice the computational resources of the MAX2 engine and supports up to 48GB of DDR3 DRAM, with substantially greater memory bandwidth.

The PSPI dataflow graph generated for the MAX2 and MAX3 devices had the ability of computing 3 reference velocities in parallel and 40 angle gathers. To exploit the increased resources on the MAX3 solution we doubled the amount of data (i.e. source and receiver wavefields values) processed in one cycle compared to MAX2. We preferred this approach to the possibility of increasing the number of reference velocities computed in parallel only because in our specific test case a large number of layers had 3 or less reference velocities and the extra capacity would have been wasted.

Figure 5 shows how the combination of computation in hardware coupled to lossy compression still gives good agreement with the software results, the difference only starting being visible an order of magnitude below the signal.

To evaluate performance, we compare the MAX2 and MAX3 system to a Fortran software version on a node with 8 Intel Xeon 2.6GHz cores parallelized using MPI. We measured compute time of layers with equal number of reference velocities as reported in Figure 6.

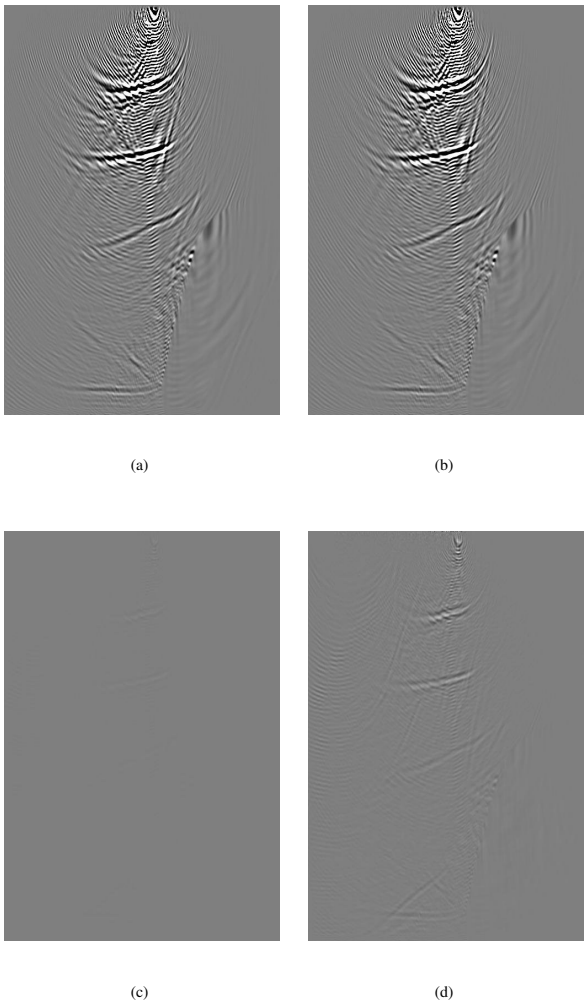

Figure 5: **One-shot imaging results.** (**a**) Software result. (**a**) Hardware result. (**c**) Software-Hardware difference. (**d**) Software - Hardware difference magnified by 10.

CONCLUSION

Computational performance of a given application depends on both hardware capability (computational speed, memory access speed and bandwidth, etc.) and algorithmic choices. The chosen porting strategy of the algorithm optimally balances the load over the different components of the hardware.

As newer dataflow engines hardware become available, we will need to re-calibrate the algorithms for optimal performance but this is a straightforward process of adjusting program parameters and recompiling.

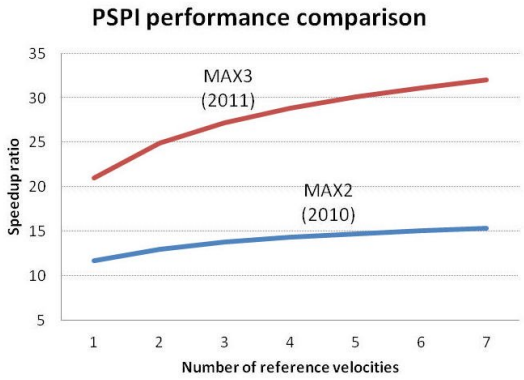

Figure 6: Node-to-node speedup for depth extrapolation of one layer as function of the number of reference velocities. MAX2 and MAX3 nodes were compared to a 8 Intel Xeon 2.6GHz cores with 48GB of DDR3 DRAM.

Our test indicated that computation of depth migration can be highly accelerated maintaining unaltered the result's quality.

ACKNOWLEDGMENTS

The authors acknowledge Eni Exploration & Production Division for giving the permission to publish these results.

REFERENCES

Bagaini, C., E. Bonomi, and E. Pieroni, 1995, Data parallel implementation of 3d pspi: 65th Ann. Internat. Mtg., Soc. Expl. Geophys., Expanded Abstracts, SEG, 188–191.

Bonomi, E., L. Brieger, C. Nardone, and E. Pieroni, 1998, Phase shift plus interpolation: A scheme for high-performance echo-reconstructive imaging: Computers in Physics, **12**, 126–132.

Gazdag, J., 1978, Wave equation migration with the phase-shift method: Geophysics, **43**, 1342.

Gazdag, J., and P. Sguazzero, 1984, Migration of seismic data by phase shift plus interpolation: Geophysics, **49**, 124–131.

Oriato, D., O. Pell, C. Andreoletti, and N. Bienati, 2010, Fd modeling beyond 70hz with fpga acceleration: Presented at the SEG 2010 HPC Workshop.