Stanford Exploration Project, Report 129, May 6, 2007, pages 253–??

Accelerating subsurface offset gathers for 3D seismic applications using FPGAs

Oliver Pell and Robert G. Clapp

ABSTRACT

For shot profile migration construction of subsurface offset gathers can be the dominant cost. By implementing the subsurface offset gather computation on a MAX-1 accelerator card with a Xilinx Virtex-4 Field Programable Gate Array (FPGA), we obtain 20x speed-up over a state-of-the-art Opteron system. By reducing data precision further speed-up can be achieved, at minimal image degradagation.

INTRODUCTION

Downward continued based migration often provides higher quality migration results. For velocity estimation and lithological determination they reqire a change in strategy compared to Kirchoff based approaches. In Kirchhoff based approaches moveout and amplitude information are evaluated as a function of offset. For downward continuation based methods angle gathers are constructed from subsurface offset gathers (de Bruin et al., 1990; Prucha et al., 1999; Biondi and Symes, 2004).

The cost of constructing the subsurface offset gathers is trivial for source-receiver migration methods based on the Double Square Root (DSR) equation (Claerbout, 1985) but can be a dominant cost in shot profile and plane wave methods (Rickett and Sava, 2002). With the increased use of wide azimuth geometries (Michell et al., 2006) and the resulting 3-D angle gathers, over 90% of CPU cycles can be spent in constructing the subsurface offset gathers.

Hardware accelerators are emerging as a powerful solution to computationally intensive problems. A standard desktop PC or cluster node can be augmented with additional hardware dedicated to providing substantially increased performance for particular applications. Research projects have shown (He et al., 2004; Zhang et al., 2005; Cheung et al., 2005; Sano et al., 2007) that FPGA-based hardware accelerators can offer order-of-magnitude greater performance than conventional CPUs, provided the algorithm to be accelerated performs a large number of operations per data point.

Construcing subsurface offset gathers involves a significant number of operations for each data point, making it an ideal candidate for acceleration. We implement subsurface offset gather construction on a FPGA. We show that a 20x speed-up is achievable using 32-bit precision. Further we demonstrate that 40x speed-up can be achieved by using a lower precision

representation of the data, with minimal image degredation.

ANGLE GATHERS FROM SHOT PROFILE MIGRATION

Claerbout (1971, 1985) noted that you could simulate sources and receivers in the subsurface by applying the DSR equation to the wavefield recorded at the surface. For 3-D source-receiver based migration methods a 4-D volume (cmpx,cmpy, hx, hy) is downward continued and the the zero-time, zero-offset portion of the volume is the image of the subsurface at that depth. In shot profile migration the source and receiver wavefields are downward continued seprately by the Single Square Root (SSR) equation. To obtain an image I at a given depth z and location $\mathbf{x}$:

$$I(\mathbf{x},\mathbf{z}) = \sum_{\mathbf{s}} \sum_{\mathbf{w}} \mathbf{S}(\mathbf{x},\mathbf{z},\mathbf{w},\mathbf{s})\mathbf{G}^*(\mathbf{x},\mathbf{z},\mathbf{w},\mathbf{s}), \tag{1}$$

where S is the source wavefield, G is the receiver wavefield, w is the temporal frequency, and s is the shot index.

de Bruin et al. (1990), Prucha et al. (1999) and Biondi and Symes (2004) all provided mechanisms to create reflectivity as a function of angle based on the focusing of the energy arround zero-offset. In source-receiver migration, the analysis is naturally done by analyzing the focusing in the **h** plane. In shot profile migration **h** does naturally occur. Rickett and Sava (2002) explained how to create subsurface offset for shot profile migration by cross-correlating the source and receiver fields by various shifts. As a result subsurface offset volumes can be created by applying:

$$I(\mathbf{h},\mathbf{x},\mathbf{z}) = \sum_{\mathbf{s}} \sum_{\mathbf{w}} \mathbf{S}(\mathbf{x}-\mathbf{h},\mathbf{z},\mathbf{w},\mathbf{s})\mathbf{G}^*(\mathbf{x}+\mathbf{h},\mathbf{z}.\mathbf{w},\mathbf{s}). \tag{2}$$

The cost of constructing these sub-surface offset gathers can be significant. The cost of downward continuing a single shot a single depth step is dominated by the FFT cost which grows by $(n \log n)$ with the size of the data n. The cost of constructing the subsurface offset gathers is on the order of $nh * n$, where nh is the number of subsurface offsets. Constructing a 2-D subsurface offset gather can be the dominant cost in shot profile migration. To construct a 3-D subsurface offset gather as part of the migration, over 95% of the compute time is common.

STREAMING PROCESSORS

Traditionally, performance increases have come from microprocessor frequency scaling. However, due to power and other constraints, scaling looks to only deliver modest performance improvements in the future. In the future large performance improvements demanded by computationally intensive applications must come from exploiting parallelism. Intel and AMD

are scaling up the number of cores per chip and processors per node in order to higher degrees of Symetric Multi-Processor (SMP). Existing software has to be modified to take advantage of potentially modest speed improvements that remain limited by a machine's memory bandwidth. The change in software presents an opportunity to move beyond conventional processors to custom accelerators. These accelerators offer the potential of much higher performance by delivering parallelism that is tailored to a particular application. In particular, streaming processors offer a route around the "memory wall" by maximising operations performed per data item retrieved from memory. Stream processors can be implemented using Field-Programmable Gate Arrays (FPGAs) and can speed up highly parallel applications by over an order of magnitude. FPGA acceleration has been successfully demonstrated in a variety of application domains including computational finance (Zhang et al., 2005), fluid dynamics (Sano et al., 2007), cryptography (Cheung et al., 2005) and seismic processing (He et al., 2004).

COMPUTING WITH FPGAS

FPGAs are Complementary Metal Oxide Semiconductor's (CMOS) technology based chips containing logic which can be configured to any digital circuit and a limited number of memory elements including RAMs and registers. In fact, FPGAs can be re-configured several times per second, offering a flexible substrate for application specific circuits. The price of reconfigurability is a 10x slower clock frequency compared to today's state-of-the-art Pentium and Opteron processors. Modern FPGAs contain on the order of 10^5 independent logic cells, all of which can operate in parallel. This massive parallelism more than compensates for the 10x reduction in clock frequency versus a state-of-the-art CPU, delivering orders of magnitude more compute power within a reasonable power budget. FPGAs have shown excellent potential as hardware accelerators for a wide class of applications. Compute-intensive algorithms can be mapped directly into parallel FPGA hardware, tightly coupled to a conventional CPU through a high-speed I/O bus, enabling key hotspots in an application to be accelerated by over an order-of-magnitude. The performance potential of FPGAs arises from exploiting *stream processing*. In a typical CPU, instructions are executed sequentially (Figure 1). Despite the high clock frequency, data throughput can be quite limited since there is limited scope for parallelism, even in modern superscalar processors with vector (SIMD) units. For many algorithms a streaming approach (Figure 2) delivers significant benefits. FPGA stream processors operate continuously on streams of data. Data is transferred to the accelerator once, over a high-speed I/O bus such as PCI Express, then it passes from one processing element to the next as it is required for each operation. The FPGA circuit computes one or more results each and every cycle without any of the control overhead associated with CPU conditionals, loops, etc. On-chip memory implements a custom "perfect cache" which retains data on-chip for precisely as long as it is required for the computation. A large number of compute units operating in parallel overcome the compute limitations of the CPU, while the on-chip storage structure and MISD (multiple instruction, single data) operation significantly mitigate the memory limitations of the CPU. Stream processors show potential for accelerating seismic applications operating on large datasets, since only a small fraction of the data needs to be stored on-chip

at any one time. This makes the approach scalable to multi-dimensional problems with tens of gigabytes of data, since the primary storage medium remains CPU main memory. FPGAs are usually regarded as hard to program, with building FPGA accelerators essentially being a matter of hardware design. We develop this accelerator at a higher level of abstraction using the ASC (Mencer, 2006) compiler. ASC, A Stream Compiler for FPGAs, provides a software-like interface to FPGA design based on C++, while retaining the performance of hand-designed circuits. At the top level, ASC code closely resembles C code, allowing a relatively low cost transition from a C-based software implementation to the FPGA hardware implementation. One key difference between ASC and a conventional imperative programming language is that the standard semantics for all operations performed in parallel and all operators are vector operations performed on streams of data. To transfer code to an FPGA accelerator we identify loops to be accelerated, then re-write those loops in ASC code, replacing the original loop with code which transfers data to/from the accelerator. For example, a C loop can describe a vector increment operation as below:

```
int i;
int a[SIZE], b[SIZE];
for (i = 0; i < SIZE; i++)
    b[i] = a[i] + 1;
```

This can be rewritten for FPGA implementation as:

```
STREAM_START;
    HWint a(IN), b(OUT);
    b = a + 1; // Loop is implicit
STREAM_END;
```

The loop has been replaced with `STREAM_START` and `STREAM_END` declarations, which identify the boundaries of the code to be implemented on the FPGA. The integer arrays *a* and *b* are declared as Hardware Integer type variables, one input and one output. This ASC code can be compiled using GCC producing an executable which, when executed, generates an FPGA circuit.

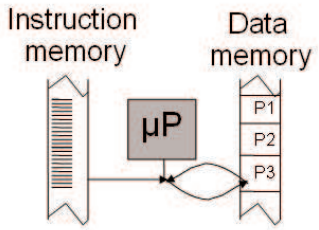

Figure 1: When computing with a microprocessor, instructions are executed sequentially on data items retrieved from memory. bob1-microproc [NR]

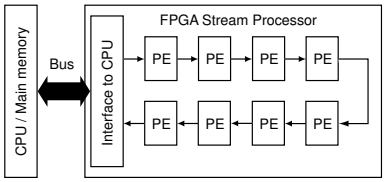

Figure 2: Computing with an FPGA involves "streaming" data through an FPGA which has been configured to implement a function. Here two input arrays are transferred, processed, and a single output array is produced. bob1-streaming [NR]

FPGA ACCELERATION

FPGA's require that memory is accessed through the processor and transfered to the FPGA. In order to obtain a meaningful speed advantage a large number of operations must be performed for each data point. The density of arithmetic operations per data item is the key to the potential for acceleration. Algorithms which use a transferred data item only once (such as the vector add example above) are unlikely to accelerate, since the overhead of transferring the data across the bus is significant, however algorithms such as an offset gather which use each data item many times will accelerate significantly. Because FPGA accelerators dedicate specific resources to each operation executed, there is a maximum size to the code segment that can be executed on-chip. This depends on not only the size of the FPGA but also the complexity of the operations. Additions and multiplications can be implemented more densely than divisions, square roots and complex functions (*sin*, *cos* etc) so algorithms in which adds and multiplies are dominant will accelerate particularly well. This is common in seismic applications. In contrast to conventional processors, which support a fixed set of data representations (typically integer and IEEE floating point) FPGAs offer the potential for the data representation to be customised to the application. This allows acceleration to be maximised subject to desired accuracy constraints. The dynamic range of migration data is such that floating point representations are not necessary, so our FPGA implementation uses fixed-point data. Fixed point arithmetic can be implemented more densely and with lower latency on FPGAs than floating point, so allowing for increased acceleration without loss of accuracy.

SUBSURFACE OFFSET IMAGING CONDITION ON THE FPGA

The shot profile imaging condition has a high arithmetic density, meeting the requirements for FPGA acceleration. The zero time part of the imaging condition (Equation 2), which requires a summing over all frequencies *nf*. In addition each input point is going to be used in *nh* cross-correlations. Despite the high arithmetic density of the offset gather, the arithmetic capabilities of the FPGA are substantially in excess of that required so acceleration is limited by the rate at which data can be streamed across the bus from main memory to the accelerator. In this case, the performance is limited by:

$$max(nf \times 2, nh) \times bw \tag{3}$$

where *bw* is the number of bits used to represent each value. This condition arises because the PCI-Express bus is symmetric, providing limited input and output bandwidth. Two arrays (**S** and **G**) containing *nf* data items per coordinate are sent to the FPGA and one array (*I*) is sent from the FPGA back to the processor, containing *nh* data items per coordinate. By reducing the number of bits stored for each value from 32 to 16 the performance of the operation can be doubled, with negligible degredation in the output image. The on-chip memory requirement is $O(nf \times nh)$, well within the capabilities of modern FPGAs for hundreds of frequencies and dozens of subsurface offsets. This allows very large datasets to be processed easily with only a small fraction stored on-chip at any one time.

RESULTS

To test the applicability of this approach we compared the result of constructing angle gathers for the 2-D Marmousi synthetic dataset. Figure 3 show the zero-subsurface offset image obtained from implementing the imaging step of shot profile migration on both the processor and the FPGA. The images are indistinguisable.

The left panel of Figure 4 shows an angle gather constructed from the CPU implementation of the imagining condition. The remaining panels show the same angle gather obtained from the FPGA implemented imaging condition with decreasing floating point precision. Note that visually the kinematics are identical.

To test the speed-up offered by the FPGA implementation we ran a larger 3-D problem. Specifically the cost of constructing 41 subsurface offset gathers from 500 inline CMPS, 400 crossline cmnps, 200 frequencies, and 41 subsurface offsets. We compare our FPGA implementation to a 2.8Ghz AMD Opteron-based PC with 12GB of RAM. The software implementation was written in C and compiled using both *gcc* and the Intel C Compiler with full optimization, the average of three runs was selected. The FPGA accelerator was implemented on a Maxeler MAX-1 FPGA platform equipped with a Xilinx Virtex-4 FX100 FPGA. The accelerator circuit consumes 58% of the logic resources of the device and runs at 125Mhz. Table 1 shows the runtimes for the gather operation at a single depth and shot, carried out both in software and on the FPGA. The FPGA computes the gather 19–21 times faster than the software using 32-bit data, or 35–42 times faster than the software using 16-bit data. This degree of acceleration transforms the application space, instead of the subsurface offset gather being dominant the time spent computing it is now insignificant as a portion of the overall runtime.

FUTURE WORK

In the future we plan to implement other portions of the shot profile algorithm on an FPGA. The streaming approach should offer significant speed-up for both applying the SSR equation and the 2-D FFTs.

The same subsurface imaging condition is used in both plane wave and reverse time mi-

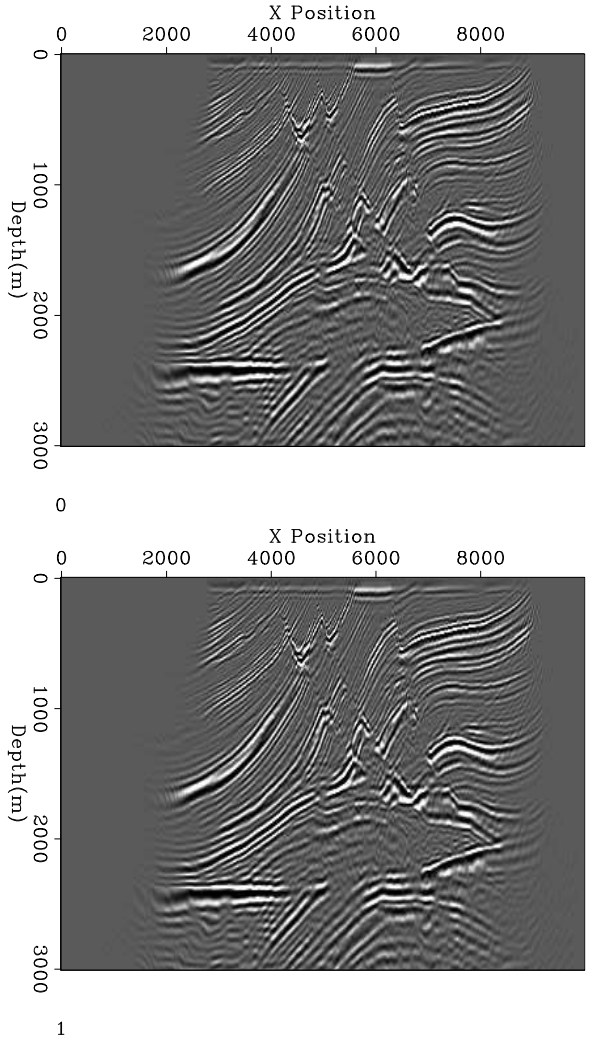

Figure 3: Comparison of the zero-subsurface offset image from implementing the shot profile imaging condition on the processor (top) and the FPGA (bottom). bob1-mig [NR]

ny	T_{sw}	T_{fpga32}	Speed-up	T_{fpga16}	Speed-up
1	0.041	0.002	21x	0.001	41x
50	1.48	0.073	20x	0.042	35x
100	2.76	0.149	19x	0.075	37x
200	6.40	0.311	21x	0.150	42x

Table 1: Performance comparison of FPGA and 2.8GHz AMD Opteron. T_{sw} is the time in seconds for the software version. T_{fpga32} is the time for the FPGA processing 32-bit data, T_{fpga16} is the time for the FPGA processing 16-bit data. Speed-up is shown for both data sizes.

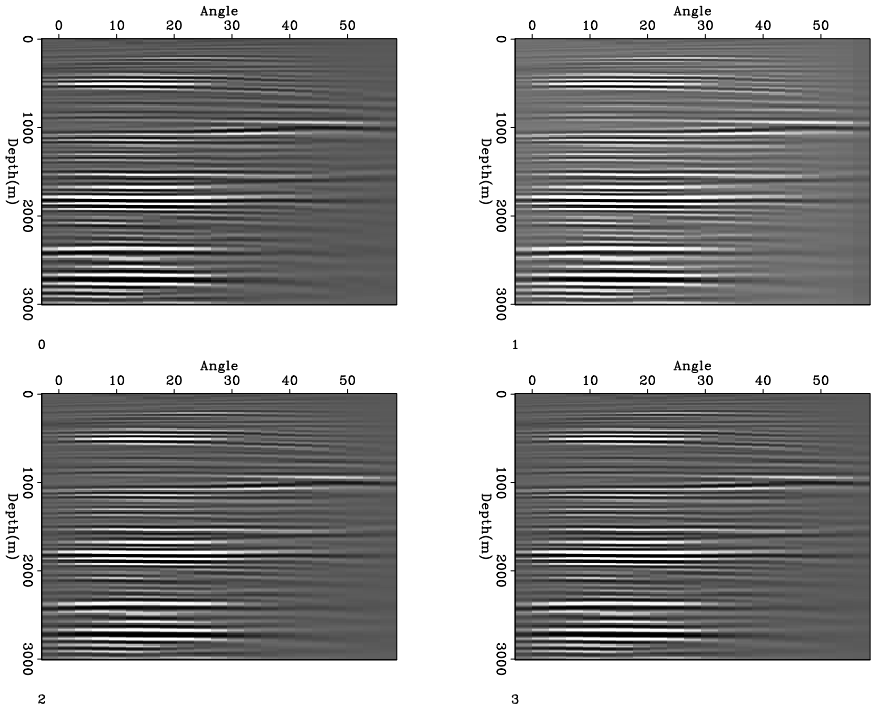

Figure 4: The same angle gather obtained by various implementation of the shot profile imaging condition. The top-left panel shows the result from a CPU based implementation. The remaining panels show various FPGA implementations. Note that they are visually kinematically identical. bob1-gath [NR]

gration. Incorporating a FPGA based imaging condition into either would be trivial. Reverse time migraton, where the cost of propogating the source and receiver wavefield is order and magnitude more than Fourier based methods, is another exciting oppotunity for a FPGA based solution.

CONCLUSION

We implemented the sub-surface offset imaging condition for shot profile migration on a FPGA. We showed that a factor of 40x speed up can be obtained compared to a conventional processor.

ACKNOWLEDGEMENTS

The second author would like to thank the sponsors of the Stanford Exploration Project for financial support.

REFERENCES

Biondi, B., and Symes, W., 2004, Angle-domain common-image gathers for migration velocity analysis by wavefield-continuation imaging: Geophysics, **69**, no. 5, 1283–1298.

Cheung, R., Telle, N., Luk, W., and Cheung, P., 2005, Customisable elliptic curve cryptosystems: IEEE Transactions on Very Large Scale Integration Systems, **13**, no. 9, 1048–1059.

Claerbout, J. F., 1971, Toward a unified theory of reflector mapping: Geophysics, **36**, no. 36, 467–481.

Claerbout, J. F., 1985, Imaging the Earth's Interior: Blackwell Scientific Publications.

de Bruin, C. G. M., Wapenaar, C. P. A., and Berkhout, A. J., 1990, Angle-dependent reflectivity by means of prestack migration: Geophysics, **55**, no. 09, 1223–1234.

He, C., Lu, M., and Sun, C., 2004, Accelerating seismic migration using FPGA-based coprocessor platform: Accelerating seismic migration using FPGA-based coprocessor platform:, IEEE Symp. Field Programmable Custom Computing Machines 2004.

Mencer, O., 2006, ASC: A stream compiler for computing with FPGAs: IEEE Transactions on Computer-Aided Design, **25**, no. 9, 1603–1617.

Michell, S., Shoshitaishvili, E., Chergotis, D., Sharp, J., and Etgen, J., 2006, Wide azimuth streamer imaging of mad dog; have we solved the subsalt imaging problem?: Wide azimuth streamer imaging of mad dog; have we solved the subsalt imaging problem?:, Soc. of Expl. Geophys., 76th Ann. Internat. Mtg., 2905–2909.

Prucha, M., Biondi, B., and Symes, W., 1999, Angle-domain common image gathers by wave-equation migration: Angle-domain common image gathers by wave-equation migration:, Soc. of Expl. Geophys., 69th Ann. Internat. Mtg, 824–827.

Rickett, J. E., and Sava, P. C., 2002, Offset and angle-domain common image-point gathers for shot-profile migration: Geophysics, **67**, no. 03, 883–889.

Sano, K., Iizuka, T., and Yamamoto, S., 2007, Systolic architecture for computational fluid dynamics on FPGAs: Systolic architecture for computational fluid dynamics on FPGAs:, IEEE Symp. Field Programmable Custom Computing Machines 2007.

Zhang, G. L., Leong, P. H. W., Ho, C. H., Tsoi, K. H., Cheung, C. C. C., Lee, D.-U., Cheung, R. C. C., and Luk, W., 2005, Reconfigurable acceleration for monte carlo based financial simulation: Reconfigurable acceleration for monte carlo based financial simulation:, IEEE Intl. Conf. Field Programmable Technology 2005.