Revisiting Convolution and FFT on Parallel Computation Platforms

Haohuan Fu, Robert G. Clapp, Stanford University and Olav Lindtjorn, Schlumberger

SUMMARY

Due to the reduction in computational complexity, FFTs are usually performed to enable a faster convolution implementation in seismic computations. However, on current parallel computation platforms, such as multi-core processors, Graphic Processing Units (GPUs), and Field Programmable Gate Arrays (FPGAs), the performance is not only determined by the computational complexity but also relates to other factors, such as the parallelism and the memory access pattern of the algorithm. In our work, we investigate optimized designs of convolutions and FFTs on different parallel computation platforms with different problem and stencil sizes. Experiment results show that, for many stencil sizes used in practical seismic applications, the direct convolution approach demonstrates a better performance than the FFT-based approach on parallel platforms. For 1D cases, the parallel performance of the FFT-based approach is limited by the data dependency of the FFT algorithm. 3D FFT-based approaches use cache poorly. Only 2D FFTs scales well with the parallel computation capacity of modern architectures. The technological trends indicate that these findings will continue.

INTRODUCTION

In seismic computations, convolutions and Fast Fourier Transforms (FFTs) are two of the most frequently-used signal processing operations for computing derivatives or applying filters. For example, in Reverse Time Migration (RTM) (Yoon et al., 2003), convolving finite-difference based stencils is the major kernel that consumes most of the computation cycles. In downward continued based migration (Gazdag and Sguazzero, 1984), multi-dimensional FFTs, which convert the wave fields between time domain and frequency domain, are among the most costly parts. There are also methods that perform RTM (Reshef et al., 1988) or waveform inversion (Pratt, 1999; Pratt and Shipp, 1999) in the frequency domain, which also involve FFTs as a major cost.

In most cases, FFTs are used to convert the convolution in time domain into a dot product in frequency domain (Reshef et al., 1988; Pratt, 1999; Pratt and Shipp, 1999). Due to the reduced computational complexity, the FFT-based approaches are supposed to provide a faster solution. However, in practical systems, especially in the context of parallel computation platforms, such as multi-core CPU, Graphic Processing Unit (GPU), and Field Programmable Gate Array (FPGA), the performance is not only determined by the computational complexity but also relates to other factors, such as the parallelism and the memory access pattern of the algorithm. Besides, different platforms also require different designs and different optimization techniques to achieve good performance (Clapp et al., 2010).

In our work, we revisit the design issues of convolution and FFT on multi-core CPUs, GPUs and FPGAs. We develop optimized 1D, 2D, 3D direct convolutions and FFT-based convolutions on these different platforms, and investigate their performance with different problem sizes and different stencil sizes.

Our experiment results show that, for many stencil sizes used in practical seismic applications, the direct convolution approach demonstrates a better performance than the FFT-based approach on parallel platforms. Meanwhile, with the parallel computing power increasing and new computational architecture emerging, in 1D and 3D cases, the direct convolution approach also shows a more significant performance improvement compared to the FFT-based approach. Only in 2D cases, the FFT-based approach shows a similar scalability over multiple cores to the direct convolution approach.

CONVOLUTION AND FFT IMPLEMENTATIONS ON DIFFERENT COMPUTATION PLATFORMS

Multi-core CPU

For the multi-core CPU platforms, we investigate the current-generation Intel Nehalem quad-core CPU (Gainestown E5520, 2.27 GHz, 8 MB L3 cache). We measure the performance on a system with two quad-core CPUs and 48 GB memory.

For the direct convolution approach on multi-core CPU, we use OpenMP to parallelize the computation through multiple threads. As the data access is linear and there is no communication needed between different threads, the automatic thread spawning and scheduling generated by OpenMP compiler already achieve a close-to-linear performance improvement.

For the FFT-based approach, we build our implementation based on the FFTW library (version 3.2.2). The FFTW library provides well-optimized FFT implementations for various sizes and dimensions, as well as multi-threaded FFT implementations that make utilization of multiple cores through OpenMP.

GPU

For GPUs, we use the NVIDIA Tesla C1060 GPU card, which consists of 30 multiprocessors (240 arithmetic cores in total) and 4 GB of memory. GPUs parallelize the computation by executing a same computation kernel through hundreds of threads. The threads are organized in 1D or 2D grids of 1D or 2D blocks, which provide the geometry coordinates that enable different threads to work on different parts of the data.

An added feature of the GPU platform is that users can control the read and write of a 16 KB local memory shared among a block of maximum 512 threads. The shared memory provides a similar access speed as the registers, and can be used

as a user-controlled cache to maximize the memory reusing and improve the overall performance.

For the convolution implementation, as we assume all the stencil coefficients are constants, we store them in the constant memory to achieve a fast access speed. For the other data, we apply the similar idea to P. Micikevicius's work (Micikevicius, 2009) to load the data based on the threads' coordinates and share the data among the threads in the same block to maximize the memory reuse.

For the FFT designs on GPU, we use CUFFT, the official FFT library supported by NVIDIA. The library supports 1D, 2D, and 3D FFT with a wide range of transform sizes.

FPGA

We use the Maxeler MAX2 FPGA acceleration card, which contains two Virtex-5 LX330T FPGA chips, 12 GB onboard memory, and a PCI-Express x16 interface to the host PC. The FPGA card comes with a high-level compiler called MaxCompiler, which enables users to describe the hardware circuit using a high-level Java-based description (Flynn et al., 2008).

In contrast to the CPU and GPU platforms, FPGAs take a streaming approach to perform computation. The data items are pushed in and out as sequential streams, while the instructions are mapped into programmable circuit units along the path from the input ports to output ports. Therefore, instead of fetching instructions and data back and forth from the memory, the computation gets performed as the data streams flow through the circuit units in one pass.

For the direct convolution design, we simply stream the data through an FPGA circuit that performs all the stencil arithmetic operations in parallel (Fu et al., 2009). On the memory side, using the distributed Block RAMs on the FPGA, we can implement a window buffer to cover all the data items required by the stencil operator, so as to guarantee concurrent access to all the data needed as operands.

There are two different ways to parallelize FFT computation on an FPGA. The first way is to unroll the outer loop, i.e. to compute different stages of the transform in parallel. The other way is to unroll the inner loop, i.e. to compute different butterfly operations in the same stage in parallel. In our implementation, we apply unrolling of the outer loop. We build a pipeline of butterfly units which deals with different stages of the transform. In between of consecutive butterfly units, we use Block RAMs to implement commutation units that reorganize the data stream into different strides.

Another design problem for FFT is that we need to either access the data item in the slowest axis or transpose the slowest axis to the fastest axis before the computation. This problem can be partly solved with the strided memory access pattern supported by the Maxeler memory controller. In the strided mode, as long as we operate in blocks of 96 sequential bytes, we can achieve 80% of the maximum memory performance.

PERFORMANCE RESULTS

In our experiments, we change both the size of the data (n) and the size of the stencil (m), and we assume that $m < n/4$.

Cross-over Points

For an optimized FFT design, the number of floating-point operations (FLOP) for a 1D n-point FFT is around $4n\log_2(n) - 6n$ (Yavne, 1968). Convolving a 1D array of size n with a 1D stencil of size m involves around $2n \cdot m$ operations. The FFT-based approach consists of two n-point FFTs (forward and backward) and one n-point dot production, which has a FLOP count of around $2 \cdot (4n\log_2(n) - 6n) + n = n \cdot (8\log_2(n) - 11)$. Similarly, 2D direct convolution and FFT-based convolution involve $2n^2 \cdot m^2$ and $n^2 \cdot (16\log_2(n) - 23)$ operations respectively. For a m by m by m 3D cube stencil, with m increasing, the number of involved arithmetic operations will soon increase to an impractical large extent. Therefore, we use a 3D star stencil that involves $3m - 2$ points. 3D direct convolution and FFT-based convolution involve $2n^3 \cdot (3m - 2)$ and $n^3 \cdot (24log_2(n) - 35)$ operations respectively.

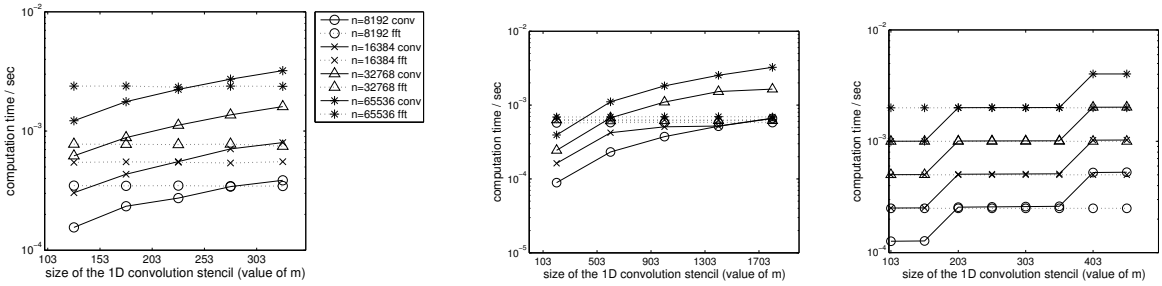
Table 1 shows the cross-over points between the two different approaches. For stencil sizes smaller than than cross-over points, the direct convolution approach provides a better performance than the FFT-based approach. The first row shows the FLOP cross-over points computed based on the FLOP numbers discussed in the previous paragraph.

problem size	1D 65536	2D 4096×4096	3D $256 \times 256 \times 256$
FLOP	59	7×7	$3 \times 25 - 2$
single-core CPU	77	15×15	$3 \times 23 - 2$
eight-core CPU	225	15×15	$3 \times 31 - 2$
GPU	403	13×13	$3 \times 35 - 2$
FPGA	353	27×27	$3 \times 51 - 2$

Table 1: The theoretical and experimental cross-over points between the direct convolution and FFT-based convolution.

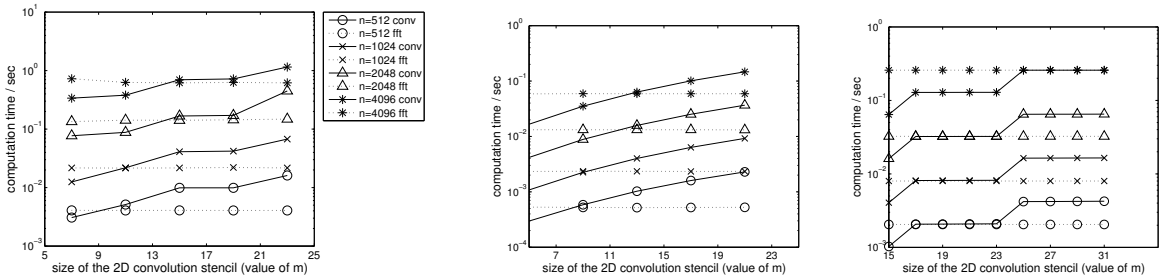
For 1D and 2D cases, the single-core implementation shows a larger cross-over point than the theoretical value computed based on the FLOP number, which suggests that the memory access pattern of the direct convolution approach leads to a better cache behavior than the FFT-based approach. For the 3D case, the single-core implementation shows a smaller cross-over point than the FLOP one, which is possibly because that accessing data items in different planes starts to cause a large number of cache misses in the direct convolution approach.

As shown in Table 1, for 1D and 3D cases, the increase of the parallel computation capacity leads to the increase of cross-over points, which shows that the direct convolution approach scales much better than the FFT-based approach and achieves a more significant performance gain with through parallel computing. For the 2D case, the two approaches show a similar scalability over parallel computation capacity, and cross-over point does not show a straight increase. As an n by n 2D FFT is generally performed as 1D FFTs in rows and columns, the



(a) two quad-core CPUs (b) GPU (c) FPGA

Figure 1: Computation time for convolving a 1D array of size n with a stencil of size m. 'conv' refers to the direct convolution approach while 'fft' refers to the FFT-based convolution approach.



(a) two quad-core CPUs (b) GPU (c) FPGA

Figure 2: Computation time for convolving a 2D array of size $n \times n$ with a stencil of size $m \times m$.

embarrassingly parallel workload fits the multi-core structures quite well. Besides, the size of the data is still within the size of cache, which helps alleviate the performance penalty on memory access.

For the FPGA, the direct convolution is even more favorable than the FFT-based approach as it requires only one streaming of the data while FFT-based approaches requires more.

Figure 1, 2 and 3 shows more detailed results about the comparison of computation time between the direct convolution approach and the FFT-based approach in 1D, 2D, and 3D cases. Note that for 3D GPU designs, due to constraint of the onboard memory size, the maximum size for FFT-based approach is $256 \times 256 \times 256$, and the maximum size for direct convolution approach is $512 \times 512 \times 512$. For 3D FPGA designs, the maximum size for FFT-based approach is $512 \times 512 \times 512$. We project the performance results for $1024 \times 1024 \times 1024$ FFTs based on the assumption that we have enough memory to hold two copies of the 3D complex array.

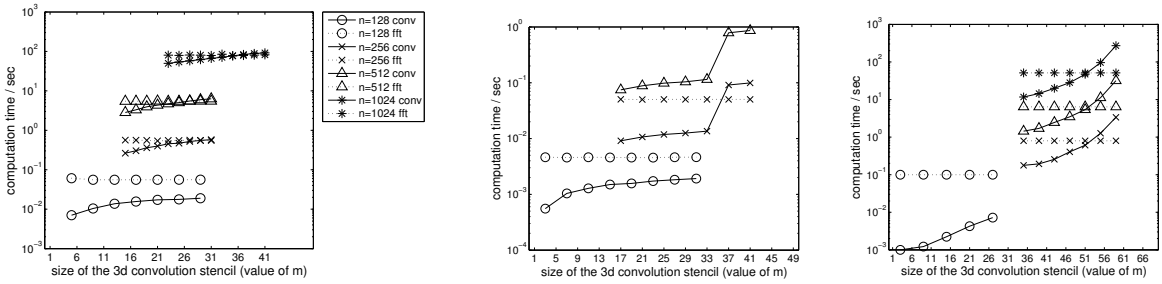
In general, comparing the two approaches on three different parallel platforms, the direct convolution approach provides better performance than the FFT-based approach for most of the stencil sizes that we apply in practical applications.

Trend on Number of Cores

Figure 4 shows the performance of the direct convolution and FFT-based convolution using different number of cores in the two quad-core Nehalem CPUs. In the 2D case, we apply a 13×13 stencil, while in the 3D case, we apply a 15th order in space star stencil. In 2D cases, the direct convolution approach and the FFT-based approach show a similar scalability over the number of cores, with the direct convolution approach slightly better than the FFT-based approach. In 3D cases, the direct convolution approach still shows a close-to-linear performance improvement over the number of cores, while the FFT-based approach does not scale well. The result again shows that the linear memory access pattern the direct convolution approach is more suitable to parallelize with multiple cores.

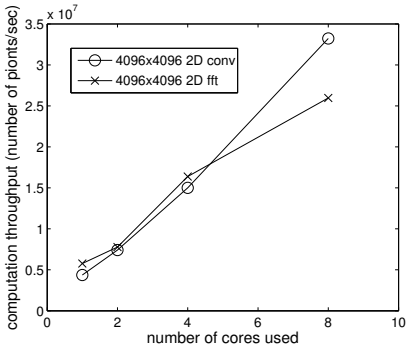
Trend on Technology Advancement

Figure 5 shows the computation throughput of convolution and FFT approaches on different generations of devices. For multi-core CPUs, we compare the performance between the previous Intel Core architecture and the current Intel Nehalem architecture. The Nehalem architecture improves the performance of direct convolution by 36%, while the performance of FFT-based approach remains almost the same. For FPGA platforms, we compare between Virtex 5 LX330 and Virtex 6 SX475T. The new generation has 2 to 3 times more resources than the previous one. With the increased resources, both the convolution approach and the FFT-based approach achieve a performance improvement of around two times. However, the performance advantage of the direct convolution approach also gets magnified by two times.

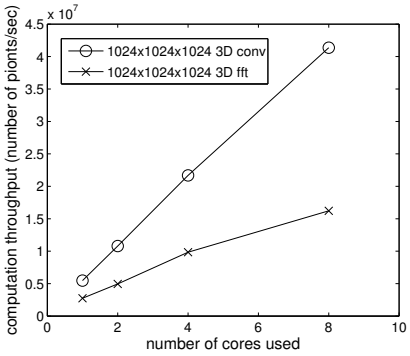


(a) two quad-core CPUs (b) GPU (c) FPGA

Figure 3: Computation time for convolving a 3D array of size $n \times n \times n$ with a star stencil of size $3m-2$.



(a) 2D cases



(b) 3D cases

Figure 4: Performance with different number of cores.

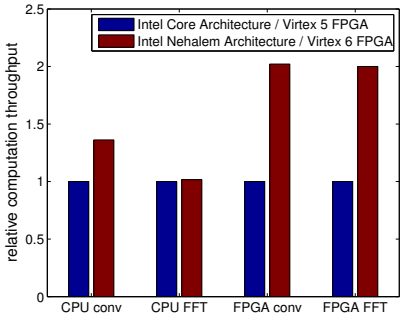


Figure 5: Comparison of between two generations of CPUs and FPGAs (the throughput on the previous-generation architecture is normalized to one).

We do not have the performance results of the new Fermi GPU yet. However, considering GPU's similar SIMD approach to CPU and the significant increase on number of cores and size of shared memory in the new architecture, we expect the convolution approach to benefit more.

CONCLUSIONS

Our experiment results show that for many stencil sizes used in practical applications, the direct convolution approach demonstrates a better performance than the FFT-based approach on parallel platforms. Compared among different parallel computation platforms, multi-core CPU and GPU shows a similar behavior as they both take an SIMD approach to parallelize the computation. For the FPGA, the direct convolution is even more favorable as it requires only one streaming of the data while FFT-based approaches requires more.

Our results also demonstrate that, in 1D and 3D cases, the direct convolution approach achieves a significantly larger performance gain through the increase of parallel computation capacity than the FFT-based approach. Only in 2D cases, the FFT-based approach shows a similar scalability over number of cores to the direct convolution approach. The direct convolution approach also seem to benefit more from the introduction of new generations of computational platforms.

Considering the current parallel platforms and the possible advancement in the future, Fourier-based 3D approaches, such as pseudo-spectral methods and wave inversions in frequency domain, would become expensive on parallel platforms. In contrast, convolution-based approaches would achieve a more efficient performance that scales well with the parallel computation capacity. Migrations that rely on 2D FFTs will continue to perform well. Source-receiver based approaches that rely on higher dimensional FFTs will become relatively less efficient on future hardware.

EDITED REFERENCES

Note: This reference list is a copy-edited version of the reference list submitted by the author. Reference lists for the 2010 SEG Technical Program Expanded Abstracts have been copy edited so that references provided with the online metadata for each paper will achieve a high degree of linking to cited sources that appear on the Web.

REFERENCES

Clapp, R., H. Fu, and O. Lintjorn, 2010, Selecting the right hardware for reverse time migration: The Leading Edge, **29**, no. 1, 48–58, doi:10.1190/1.3284053.

Flynn, M., R. Dimond, O. Mencer, and O. Pell, 2008, Finding Speedup in Parallel Processors: Proc. International Symposium on Parallel and Distributed Computing, 3–7.

Fu, H., R. Clapp, O. Mencer, and O. Pell, 2009, Accelerating 3D Convolution Using Streaming Architectures on FPGAs: SEG Expanded Abstracts, 3035–3039.

Gazdag, J., and P. Sguazzero, 1984, Migration of seismic data by phase shift plus interpolation: Geophysics, **49**, 124–131, doi:10.1190/1.1441643.

Micikevicius, P., 2009, 3D Finite Difference Computation on GPUs using CUDA: Proc. 2nd Workshop on General Purpose Processing on Graphic Processing Units, 79–84.

Pratt, R., 1999, Seismic waveform inversion in the frequency domain, Part 1: Theory, and verification in a physical scale model: Geophysics, **64**, 888–901, doi:10.1190/1.1444597.

Pratt, R., and R. Shipp, 1999, Seismic waveform inversion in the frequency domain, Part 2: Fault delineation in sediments using crosshole data: Geophysics, **64**, 902–914, doi:10.1190/1.1444598.

Reshef, M., D. Kosloff, M. Edwards, and C. Hsiung, 1988, Three-dimensional Acoustic Modeling by the Fourier Method: Geophysics, **53**, 1175–1183, doi:10.1190/1.1442557.

Yavne, R., 1968, An economical method for calculating the discrete Fourier transform: Proc. AFIPS Fall Joint Computer Conf., 115–125.

Yoon, K., C. Chin, S. Suh, L. Lines, and S. Hong, 2003, 3D Reverse-time Migration Using the Acoustic Wave Equation: An Experience with the SEG/EAGE Data Set: The Leading Edge, **22**, no. 1, 38–41, doi:10.1190/1.1542754.