Accelerating 3D convolution using streaming architectures on FPGAs

Haohuan Fu, Robert G. Clapp, Oskar Mencer, and Oliver Pell

ABSTRACT

We investigate FPGA architectures for accelerating applications whose dominant cost is 3D convolution, such as modeling and Reverse Time Migration (RTM). We explore different design options, such as using different stencils, fitting multiple stencil operators into the FPGA, processing multiple time steps in one pass, and customizing the computation precisions. The exploration reveals constraints and tradeoffs between different design parameters and metrics. The experiment results show that the FPGA streaming architecture provides great potential for accelerating 3D convolution, and can achieve up to two orders of magnitude speedup.

INTRODUCTION

The oil industry has always been one of the leading consumers of high performance computing systems. With the increasing of the CPU clock frequencies coming to an end, we can no longer double our computation speed by purchasing updated computers every eighteen months and need to adapt to new computation architectures, such as multi-core processors, General Purpose Graphic Processing Units (GPGPUs), and Field Programmable Gate Arrays (FPGAs).

Recent research work has shown that FPGAs can provide a customized solution for a specific application and achieve more than two orders of magnitude speedup compared to a single-core software implementation. Examples include cryptology applications (Cheung et al. 2005), finance and physics simulations (Zhang et al. 2005; Gokhale et al. 2004) as well as seismic computations (Nemeth et al. 2008).

The major difference between FPGA and other computation platform is the reconfigurability of the processing and storage units in the device, which enables an FPGA to be configured into arbitrary processing units and circuit structures. The reconfigurability of the FPGA leads to two major advantages over other computation platforms:

(1) A streaming computation architecture. While CPUs and GPGPUs take in a sequence of instructions that operate on corresponding data in memory, in FPGAs the instructions are mapped into circuit units along the path from input to output. The

FPGA then performs the computation by streaming the data items through the circuit units. The streaming architecture makes efficient utilization of the computation device, as every part of the circuit is performing an operation on one corresponding data item in the data stream.

(2) Customizable number representations. While CPUs and GPGPUs can only handle 8-, 16-, 32- or 64-bit variables, FPGAs support arbitrary bit width for each variable in the design. By adjusting the bit widths according to the precision requirement, we can often achieve significant reduction in the silicon area cost of arithmetic units and the bandwidth requirement between different hardware modules, thus improving the overall throughput of the entire system.

To investigate FPGA's capability on solving the convolution problem, we explore design options such as: (1) using different stencils; (2) fitting multiple stencil operators into the FPGA; (3) processing multiple time steps in one pass; (4) customizing the computation precisions. The exploration demonstrates constraints and trade-offs between different design parameters and metrics. Experiment results show that the streaming computation architecture of FPGAs can provide up to two orders of magnitude speedup compared to a single-core software implementation.

STREAMING ARCHITECTURE FOR CONVOLUTION

Target Application

Our target application is a 512 by 512 by 512 finite difference problem, with 6th to 8th order in space and 2nd order in time accuracy. Each time step of the computation takes the current wave-field state, the wave-field state from the previous time step and the velocity model as inputs, and produces the next wave-field state as the output.

FPGA Platform

Current Xilinx FPGAs contain three major categories of resources: (1) reconfigurable logic slices with 6-input lookup tables (LUTs) and flip flops (FFs); (2) DSP48E arithmetic units that can perform 18×25 multiplications; (3) 36-KBit Block RAM (BRAM)s used as local storage or FIFOs.

In our work, we use the Maxeler MAX2 acceleration card, which contains two Virtex-5 LX330T FPGA chips, 12 GB onboard memory, and a PCI-Express x16 interface to the host PC. Table 1 and 2 show the resource summary of our current FPGAs and the recently released Virtex-6 SX475T FPGA, and the basic cost for implementing single-precision floating-point units on FPGAs.

FPGAs	#LUTs	#FFs	#DSP48Es	#BRAMs
LX330T	207,360	207,360	196	324
SX475T	287,600	595,200	2,016	1,064

Table 1: Resource summary of the Virtex-5 LX330T and Virtex-6 SX475T FPGA.

Operations	#LUTs	#FFs	#DSP48Es	#BRAMs
$+/-$	425	557	0	0
$\times$	122	173	2	0

Table 2: Costs for single-precision floating-point units.

Streaming Architectures

Finite difference based convolution operators normally perform multiplications and additions on a number of adjacent points. While the points are neighbors to each other in a 3D geometric perspective, they are often stored relatively far apart in memory. For example, in the 7-point 2D convolution performed on a 2D array shown in Figure 1, data items $(0, 3)$ and $(1, 3)$ are neighbors in the y direction. However, suppose the array uses a row-major storage and has a row size of 512, the storage locations of $(0, 3)$ and $(1, 3)$ will be 512 items (one line) away. For a 3D array of the size $512 \times 512 \times 512$, the neighbors in z direction will be 512×512 items away. In software implementations, this memory storage pattern can incur a lot of cache misses when the domain gets larger, and decreases the efficiency of the computation.

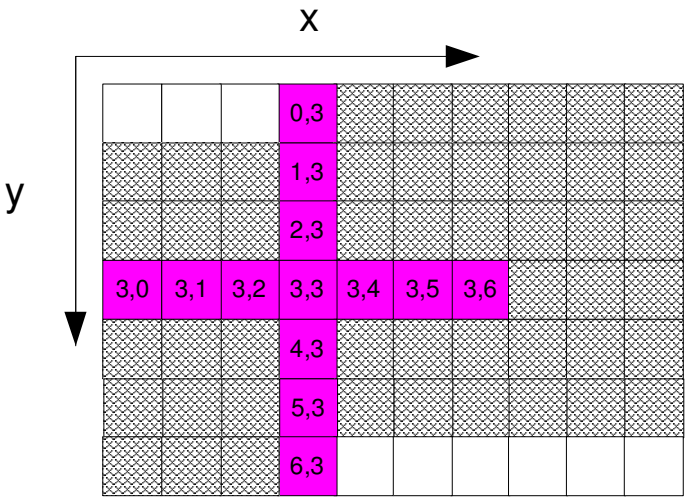


Figure 1: A streaming example of 2D convolution. [**NR**]

In an FPGA implementation, we use a streaming architecture that computes one result per cycle. As shown in Figure 1, suppose we are applying the stencil on the data item $(3, 3)$, the circuit requires 13 different values (solid, dark-color), two of

which ($(0,3)$ and $(6,3)$) are three lines away from the current data item. As the data items are streamed in one by one, in order to make the values of $(0,3)$ and $(6,3)$ available to the circuit, we put a memory buffer that stores all the six lines of values from $(0,3)$ to $(6,3)$ (illustrated by the checker board pattern on the grid). For a row size of 512, this incurs a storage cost of 512×6 data items.

Similarly, for a 7-point 3D convolution on a $512 \times 512 \times 512$ array, the design requires a buffer for $512 \times 512 \times 6$ data items. Assume each data item is a single-precision floating-point number, the buffer size amounts to 6 MB for the $512 \times 512 \times 512$ example. The FPGA chip we currently use provides 1.4 MB of potential buffer size, which is not enough to store all the streamed-in values. We solve this problem by 3D blocking, i.e. dividing the original 3D array into smaller-size 3D arrays, and performing convolution on them separately.

3D blocking reduces the buffer requirement for an FPGA convolution implementation at a cost. Given a convolution stencil with ns non-zero lags in each direction, we must send in a $(nx+ns) \times (ny+ns) \times (nz+ns)$ block to produce a $nx \times ny \times nz$ output block. As nx, ny, nz becomes small, the blocking overhead can dominate. Meanwhile, the initialization cost for setting up the memory address registers and start the streaming process is also increased as we need to stream multiple blocks.

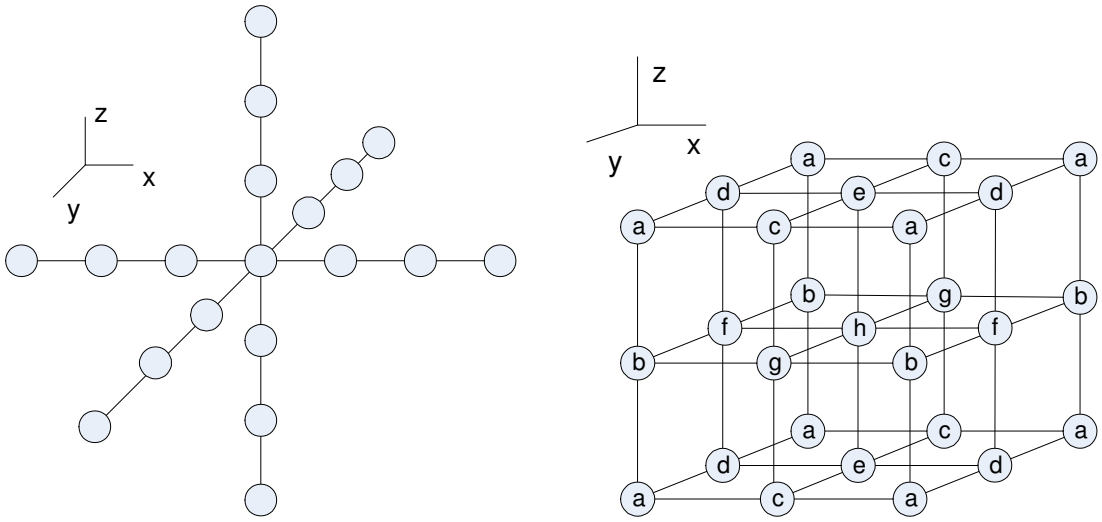
EXPLORATION OF DESIGN OPTIONS

Different Stencils

Our target application uses a a 7-point 'star' stencil (Figure 2(a)) to perform the 8th order finite difference. In our exploration, beside the 'star' stencil, we also consider a 3-by-3-by-3 'cube' stencil (Figure 2(b)), which performs a 6th order finite difference (Spotz and Carey 1996).

In software implementations, the 'cube' and the 'star' stencils provide a similar performance. For the FPGA implementations, the resource costs for the 'star' and the 'cube' stencils are different. The upper part of Table 3 shows the straightforward implementations of 'star' and 'cube' stencils for a 120x120x120 array. The 'cube' consumes 20% more DSP48E arithmetic units than 'star', as it involves more multiplications. Meanwhile, the memory cost (BRAM) of the 'cube' is one third of the 'star', as the data buffering requirement decreases from 6 slices to 2 slices.

For the FPGA designs, we can reduce the count of arithmetic operations by exploiting the symmetry of the coefficients. For example, in the 'cube' stencil shown in Figure 2(b), the stencil coefficients are the same for the points marked with the same letters, as both the Laplace derivatives and the scaling ratio determined by the sampling rate of different axes are the same for these points. Therefore, instead of computing $a1 \times c + a2 \times c + a3 \times c$, we compute $(a1 + a2 + a3) \times c$. Applying this technique, the computation for the 'cube' stencil reduces from 27 multiplications and



(a) (b)

Figure 2: Different 3D stencils: 'star' vs. 'cube'. [**NR**]

Normal Stencils		'star'	'cube'
FPGA resource costs	#slices	5618	7072
	#BRAMs	87	30
	#DSP48Es	50	60
Optimized Stencils		'star'	'cube'
FPGA resource cost	#slices	5207	6256
	#BRAMs	87	30
	#DSP48Es	32	18

Table 3: Resource costs of 'star' and 'cube' 3D stencils for 120x120x120 3D arrays.

26 additions to 8 multiplications and 26 additions, while the 'star' stencil reduces from 19 multiplications and 18 additions to 10 multiplications and 18 additions. The lower part of Table 3 shows the resource costs for multiplication-reduced 'cube' and 'star'. While the cost of BRAMs remains the same, the number of DSP48Es reduces significantly for the 'cube'. After the multiplication reduction, the 'cube' consumes much less than the 'star' for both DSP48Es and BRAMs. The 'star' consumes less logic slices as it involves fewer additions.

As the stencil operator only consumes 8 or 10 multiplications and 26 or 18 additions, the FPGA has the capacity for multiple copies of the stencil operators. Therefore, we have two different ways to improve the performance of the FPGA: (1) using multiple stencil operators to work on multiple data items in parallel; (2) processing multiple time steps in one pass. The following sections discusses these two options in more detail.

Multiple Stencil Operators

To make a full utilization of all the units on an FPGA, we can try to fit as many stencil operators as possible into the chip. For the example shown in Figure 1, instead of processing only (3,3), we can process consecutive data items (such as (3,2), (3,3), and (3,4)) in parallel.

However, increasing the number of stencil units does not always improve the overall performance due to the constraint of the bandwidth between the FPGA and the onboard memories, which is approximately 13 GB/s in our platform. Considering the controlling overheads, the bandwidth for pure input and output data is around 8 GB/s. When the input streams for the multiple stencil operators approach the saturation point of the memory bandwidth, increasing the number of stencil operators may not improve the performance any more.

Using measured experiment results, we built a software tool that models the costs and performance of various FPGA designs. Figure 3 shows the estimated performance for processing a $512 \times 512 \times 512$ 3D convolution using different number of computation cores on an FPGA. The FPGA circuit is running at 125 MHz. The speedup is calculated against a single-core software implementation running on Intel Xeon 2.0 GHz. Due to the constraint of logic slices, the FPGA can fit six concurrent 'cube' stencils or eight concurrent 'star' stencils. For all the different number of stencil operators, the 'cube' provides a slightly better performance than the 'star'. Both the 'cube' and the 'star' arrive at the saturation point of around 25x speedup with four stencil operators.

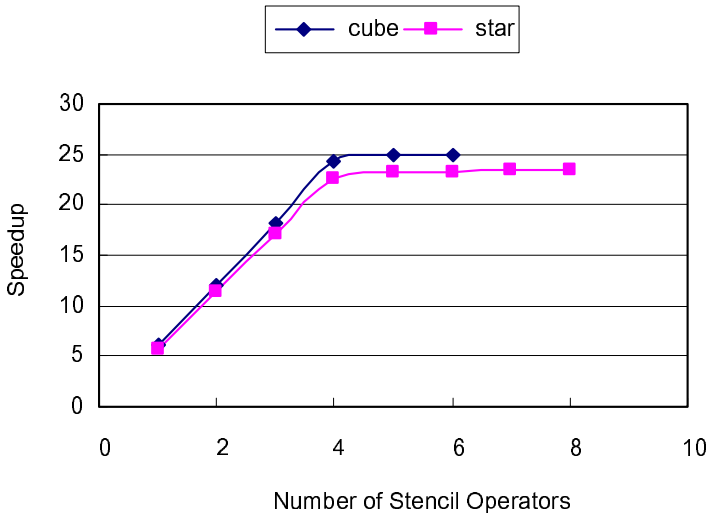


Figure 3: Speedups for processing a $512 \times 512 \times 512$ 3D convolution using multiple stencil operators. [**NR**]

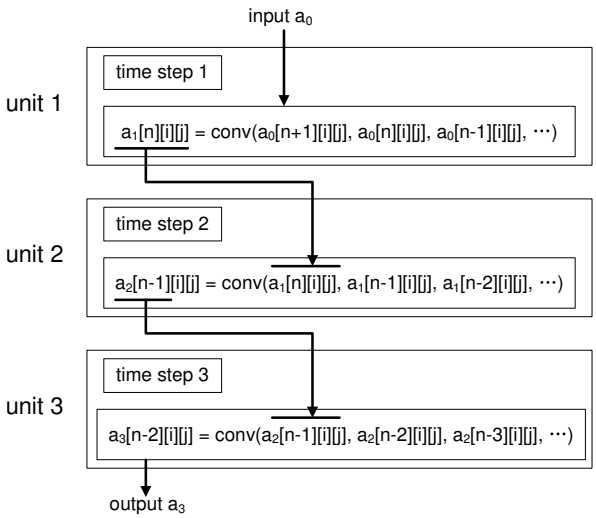


Figure 4: Basic circuit structure for processing multiple time steps (a_i denotes the wave-field data in the time step i). [**NR**]

Processing Multiple Time Steps

Instead of putting concurrent cores, another strategy is to process multiple time steps in one pass. Figure 4 shows the basic structure of a circuit that processes three time steps in one pass. The three units process three time steps separately with the output of each unit as the input of the next unit. The example in the figure uses a 3-by-3-by-3 'cube' stencil. In general, the computation of a wave-field data in slice n requires the wave-field data in slices $(n+1)$, n, and $(n-1)$ in the previous time step. Therefore, when the unit 1 starts processing slice n, the unit 2 can start processing slice $(n-1)$. Meanwhile, unit 2 needs intermediate buffers to store the results for slices $(n-1)$ and n from unit 1.

An advantage of processing multiple time steps over putting multiple stencil operators is that the performance will not be constrained by the memory bandwidth, as the unit for each time step is getting inputs from the previous time step, and does

not consume the memory bandwidth of the FPGA.

However, on the data side, as we are doing a 3D blocking of the array, processing multiple time steps requires extra data items to start with. Given a convolution stencil with ns non-zero lags in each direction, to process n time steps in one pass for a $nx \times ny$ array, we need to start with an array of the size $(nx + 2 \times n \times ns) \times (ny + 2 \times n \times ns)$. Considering doing 10 time steps for a 100x100 size, the data overhead is 44% for the 'cube' and 156% for the 'star'.

Meanwhile, as the unit at each time step needs to store the results of the previous time step, this approach also increases the requirement for BRAM resources. Therefore, to increase the number of time steps, we need to reduce the blocking size, and thus increasing the cost of streaming overlapping data items and doing a larger number of streams.

Another advantage of this multiple-time-step architecture is that we can improve the order of time accuracy with relatively small costs. For example, for the unit 3 in Figure 4, instead of only getting the previous wave-field data a_2 from unit 2, we can get in the wave-field data a_2 and a_1 from both units 2 and 1 to achieve 4th order in time accuracy. The cost for improving the time order is the extra buffer to store the wave-field data from unit 1 and the increased number of adders and multipliers.

Figure 5 shows the estimated performance for FPGA convolution designs that process multiple time steps in one pass. The 'star', the 2nd and 4th order 'cube' are compared here. For this approach, the 'cube' stencil shows a much better performance than the 'star' stencil due to its smaller requirement for BRAM resources ('star' needs to buffer six slices for the convolution operation, while 'cube' only needs to buffer two). Due to the constraint of logic slices, the FPGA can fit eight time steps for the 'star', six and five steps for the 2nd and 4th order 'cube'. The 'star' gets its peak performance of 11x speedup with four time steps. After that, the performance becomes worse with more time steps. The 2nd order 'cube' stencil increases all the way to 29x speedup with 6 time steps. The 4th order 'cube' achieves 25x speedup with 5 time steps.

Different Precisions

As mentioned above, one of FPGA's advantages is the support for customizable number representations. Our previous work (Fu et al. 2008) has shown that, in certain cases of seismic computations, reduced precision provides equivalent results within acceptable tolerances. For FPGA designs, a reduced precision can significantly reduce the area cost and I/O bandwidth of the design, and multiply the performance with more computation units on the FPGA.

Figure 6 shows the performance we can achieve using a reduced floating-point precision. With a 16-bit floating-point precision, the multiple-stencil approach provides 49x speedup and the multiple-time-step approach provides 46x speedup.

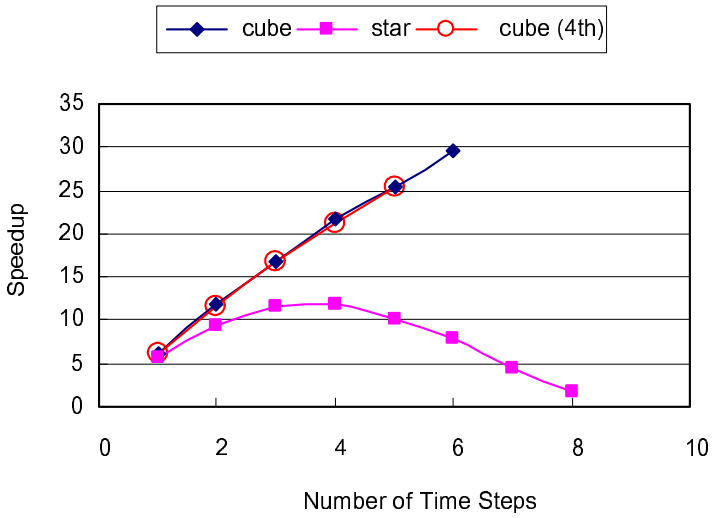


Figure 5: Speedups for processing different number of time steps processed in one pass. [**NR**]

ACCELERATION RESULTS

We have implemented the 2nd order 'cube' with 6 time steps and the 4th order 'cube' with 5 time steps onto the Maxeler acceleration card. The 2nd order 'cube' processes 6 time steps in 1.383 seconds, and the 4th order 'cube' processes 5 time steps in 1.346 seconds. Compared to the 6.36 seconds to process one time step in 2nd order, the 2nd and 4th order 'cube' designs provide 27.5x and 23.5x speedups, slightly lower than our estimated performance.

The speedup discussed so far is achieved by using one FPGA of the acceleration card. The acceleration card contains two FPGAs of the same settings. There is also a inter-FPGA link which can update the overlapping boundaries between the FPGAs in parallel with the computation performed on the FPGAs. Therefore, by dividing the array into two parts and computing in two FPGAs concurrently, we can get another 2x and achieve up to 55x and 47x speedup in total.

Note that the FPGAs we are using are Xilinx Virtex-5 LX330T chips released several years ago. Projecting our designs into the recently announced Xilinx Virtex-6 SX475T FPGAs (shown in Table 1), we can fit up to 13 time steps in one FPGA and achieve up to 55x speedup. With two FPGAs working concurrently on an acceleration card, we can achieve up to 110x speedup compared to a single-core CPU version.

CONCLUSIONS

Our exploration on FPGA convolution designs shows that, the 'cube' stencil fits the FPGA streaming architecture much better than the 'star' stencil. We especially investigate the architecture that processes multiple time steps in one pass. This approach removes the constraints of the memory bandwidth, and improves the performance at the cost of extra data buffering and streaming overhead. Experiment results show

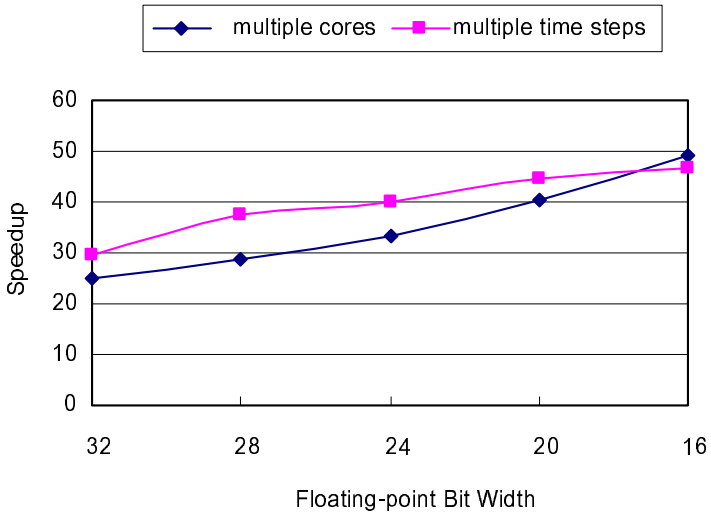


Figure 6: Speedups for different floating-point precisions. [**NR**]

that the FPGA streaming architecture provides great potential for accelerating 3D convolution, and can achieve up to two orders of magnitude speedup.

ACKNOWLEDGMENTS

We would like to thank Maxeler Technologies for providing the hardware device and the Center for Computational Earth and Environmental Science in Stanford University for funding this research.

REFERENCES

Cheung, R., N. Telle, W. Luk, and P. Cheung, 2005, Customisable elliptic curve cryptosystems: IEEE Transactions on VLSI Systems, **13**, 1048–1059.

Fu, H., W. Osborne, R. Clapp, and O. Pell, 2008, Accelerating seismic computations on fpgas: From the perspective of number representations: Presented at the .

Gokhale, M., J. Frigo, C. Ahrens, J. Tripp, and R. Minnich, 2004, Monte Carlo radiative heat transfer simulation on a reconfigurable computer: Proc. FPL, LNCS 3203, 95–104.

Nemeth, T., J. Stefani, W. Liu, R. Dimond, O. Pell, and R. Ergas, 2008, An implementation of the acoustic wave equation on FPGAs: Presented at the .

Spotz, W. and G. Carey, 1996, A high-order compact formulation for the 3d poisson equation: Numerical Methods for Partial Differential Equations.

Zhang, G., P. Leong, C. Ho, K. Tsoi, C. Cheung, D. Lee, R. Cheung, and W. Luk, 2005, Reconfigurable Acceleration for Monte Carlo based Financial Simulation: Proc. FPT, 215–222.