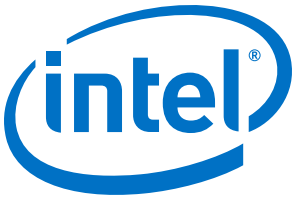


# AN 870: Stencil Computation Reference Design



AN-870 | 2018.10.10
Latest document on the web: PDF | HTML

# Contents

# 1. AN 870: Stencil Computation Reference Design

Stencil computations represent a class of matrix-iterative kernels that are used to solve or approximate several algorithms. For example, solving partial differential equations by applying algorithms such as Jacobi Iteration, Gauss-Seidel iteration, and Successive Over Relaxation (SOR); to perform simulations of physical systems such as fluid dynamics and electromagnetic modeling; and to perform image processing and cellular automata.

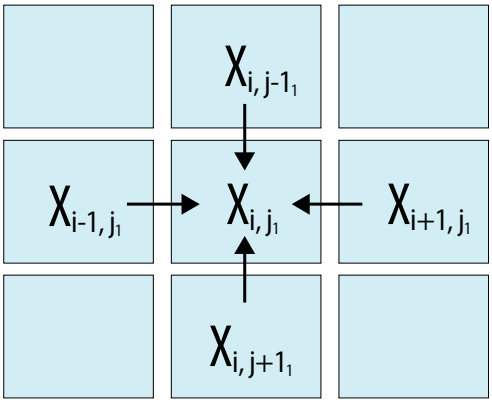
In all versions of a stencil code, a specific algorithm is applied to every element in a matrix in a sweep. Newly calculated values are piped forward into the next sweep. Cells within the same sweep have no dependencies and are calculated in parallel. Sweeps can also be calculated in parallel if the results from one sweep are continuously piped into the next.

Stencil codes are highly parallelizable but, more importantly, they are memory bound applications. The applications are memory bound due to the small operational intensity of the calculation when compared to the much larger external memory bandwidth required to support these calculations. FPGAs provide two large advantages in this area:

- Large internal and external bandwidth
- A highly pipelined and connected fabric that allows the massive parallelization inherent in a stencil code to be exploited to achieve minimal execution times

## 1.1. Four-Point 2D Stencil / 2D Jacobi Iteration Algorithm

In the 1D representation of a 2D matrix, assuming the length of a row is *N* and the number of columns is *M*, the 4-point neighborhood/stencil of any individual point is given by : $\{x\text{-}N, x\text{+}N, x\text{-}1, x\text{+}1\}$, where all values outside of the boundary of the matrix are assumed to be 0. Assume we have matrix A with values $\{x_0, x_1, x_2, x_3, \ldots\}$, where x is in the range 0 to *N*x*M*. The matrix initially begins at timestep $T_0$. The values at $T_1$ are defined as: $T_1(x_i) = 0.25 * (T_0(x_{i+1}) + T_0(x_{i-1}) + T_0(x_{i+N}) + T_0(x_{i-N}))$.




Now assume we have a 2D representation of a 2D matrix B with values $\{x_0y_0, x_1y_0, x_0y_1, x_1y_1, \ldots\}$ where x in the range of 0 to N, and y is in the range of 0 to M. The matrix initially begins at timestep $T_0$. The values at $T_1$ are defined as follows:

$$T_1(x_iy_j) = 0.25 * (T_0(x_iy_{j-1}) + T_0(x_iy_{j+1}) + T_0(x_{i-1}y_j) + T_0(x_{i+1}y_j))$$

The main idea behind this specific stencil pattern is to iteratively update the data within a matrix until the values converge (Jacobi Iteration). Performance is optimized by using caching and feed-forward techniques to reduce to reads from global memory (DDR) as much as possible, ideally to 1. Data is typically retrieved from real world measurements, but for the sake of testing all data in the associated reference design is pseudorandom.

## 1.2. OpenCL Design

Stencil computations in general are memory-bound applications. The optimizations included in this OpenCL*[(1)(2)] reference design seek to leverage the power of an FPGA by both parallelizing as much as possible and using channels/pipes to saturate the bandwidth from on-board DDR to maximize GFLOPS and minimize execution time.

Data is initially read from global memory by the compute unit (CU) named "feeder" and written back to global memory by the "writer"' CU. Data is read and written in blocks of 16 floats at a time, and the net data bandwidth depends on the frequency of the kernel and supported data read/write rate of the on-board DDR and memory controller.

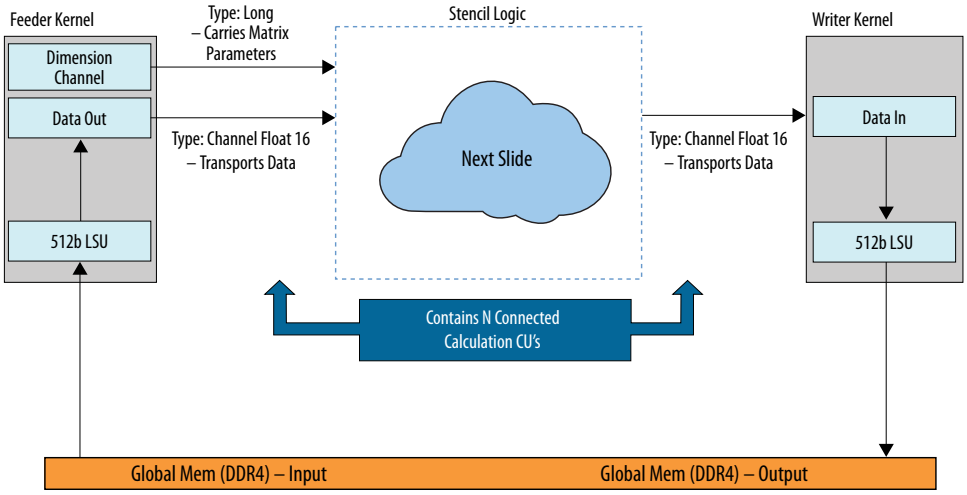
Matrix data is read continuously from 0 up to the size of the matix, and data is written in that same order. Two different memory objects are allocated in device memory for this kernel:

- A source location where data from the host is sent via PCIe to be calculated
- An output location where data is read by the host after kernel execution

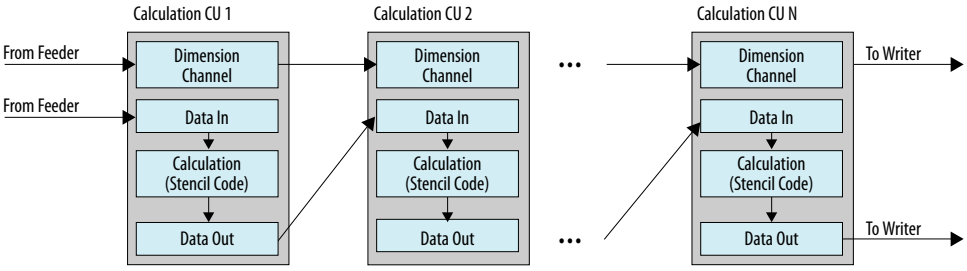
(1) OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission of the Khronos Group™.

(2) The Intel® FPGA SDK for OpenCL is based on a published Khronos Specification, and has passed the Khronos Conformance Testing Process. Current conformance status is available at www.khronos.org/conformance.

The following diagram captures the flow of data into and out of the kernel system:




Between the feeder and writer CUs is the main part of the kernel system – a series of chained and replicated calculation CUs. Data flows into the first calculation CU in the same order it was read from DDR, and the calculated data is piped into the next CU in the chain in the same order. Data coming into a CU is cached, boundaries are updated, and the result of each calculation is immediately sent into the next kernel. In this way every single CU, and hence every iteration, can be calculated in parallel once enough data has been sent through the chain.




Within each of these calculation CUs is a local memory system called "cache" that is composed of M20k blocks adjacent to the CUs. The cache size must be large enough to store an entire row of the incoming matrix. The height of the matrix can be as large as device memory allows. Matrix attributes are fed forward through the system before stencil computations begin.

Because of the nature of the computation, only 14 of the 16 floating point values being forwarded can be calculated at a time, so boundary conditions are updated between CUs. This is because each element requires both its left and right neighbors to perform a stencil computation, so the first and last elements in the block cannot be calculated. The entire computation requires a total of 3 blocks of 16 floats to be loaded into private registers, the row being calculated and their top/bottom neighbors. After a calculation is performed one block with 14 new and 2 outdated values is piped to the next CU.

## 1.3. Performance

The performance[1] on an Intel Arria® 10 GX FPGA Development Kit (A10GX1150) was compared against experimental results reported by the academic paper *OpenCL-Based FPGA-Platform for Stencil Computation and Its Optimization Methodology*[2], published May 2017. Data in the paper was collected by running 15360 sweeps of the stencil pattern. The pattern was optimized for the following GPUs and CPUs:

*GPUs*
- NVIDIA* Tesla* C2075 companion processor (C2075)
- NVIDIA* GeForce* GTX 760 graphics card (GTX760)
- NVIDIA* GeForce* GTX 960 graphics card (GTX960)

*CPUs*
- Intel Xeon® Processor E5-1650 V3 (E5-1650 V3)
- Intel Core® i7-4960X Processor Extreme Edition (i7-4960X)

Thirty kernels were chained together in a feed-forward approach in order to perform 30 iterations of the stencil algorithm in parallel. Each individual kernel began execution as soon as it was sent enough information from the previous kernel.

If optimized correctly, the kernel can be altered easily to alternate between reading/writing from both global memory objects and run the 30 sweeps as many times as wanted. The execution time in this case is the same as is reported for the non-repeating case. The execution time might be less if other optimizations are applied. The following table outlines the execution time over all 15360 sweeps on a 32768x4096 float data set. Lower execution times are better.

**Table 1. Execution Time Comparison**

| Processor | Execution Time (s) |
|---|---|
| A10GX1150 | 42.455 |
| C2075 | 232.4 |
| GTX760 | 176.5 |
| GTX960 | 111.7 |
| E5-1650 V3 | 258.9 |
| i7-4960X | 260.2 |

(1) Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks

(2) H. M. Waidyasooriya, Y. Takei, S. Tatsumi and M. Hariyama, "OpenCL-Based FPGA-Platform for Stencil Computation and Its Optimization Methodology," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1390-1402, May 1 2017.

doi: 10.1109/TPDS.2016.2614981

URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7582502&isnumber=7894348

The execution times shown in the following chart were recorded using the stencil computation reference design running on an Intel Arria 10 GX FPGA Development Kit board. If the following optimizations were to be applied, execution time can be improved significantly. The single largest performance boost would be to upgrade to a larger device, such as moving to a Intel Stratix® 10.

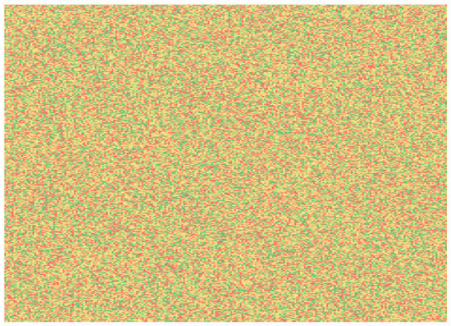
- Compiling with a newer version of Intel Quartus® Prime Design Suite
- Fitting more calculation kernels in the chain
- Using an FPGA device that is larger, faster, or both
- Removing profiling hardware

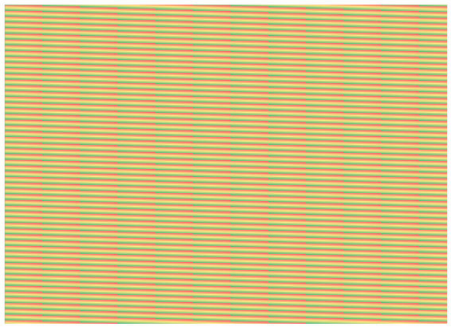
The following table shows the complete chart of execution times on an Intel Arria 10 GX FPGA Development Kit for three data set sizes:

| No. of Kernels | Data set 4088x65536 | | | Data set 4088x32760 | | | Data set 4088x4088 | | |
|---|---|---|---|---|---|---|---|---|---|
| | Exec. Time (ms) | GFLOPS | Stall % (Worst) | Exec. Time (ms) | GFLOPS | Stall % (Worst) | Exec. Time (ms) | GFLOPS | Stall % (Worst) |
| 1 | 151.34 | 7.081040518 | 29.9 | 75.75 | 7.0718352 | 29.25 | 9.71 | 6.8843 | 34.41 |
| 2 | 148.89 | 14.39511951 | 34.74 | 74.56 | 14.369408 | 34.67 | 9.62 | 13.898 | 38.03 |
| 3 | 150.16 | 21.41005605 | 32.45 | 75.1 | 21.399129 | 32.44 | 9.2 | 21.798 | 30.05 |
| 10 | 150.33 | 71.28614861 | 35.03 | 75.23 | 71.207167 | 34.99 | 9.51 | 70.291 | 35.23 |
| 20 | 164.24 | 130.4974028 | 19.78 | 82.12 | 130.46554 | 20.09 | 10.38 | 128.8 | 33.22 |
| 28 | 165.77 | 181.0101394 | 15.41 | 82.88 | 180.97686 | 15.13 | 10.45 | 179.11 | 22.77 |
| 29 | 162.62 | 191.1062322 | 22.53 | 81.4 | 190.84833 | 22.78 | 10.42 | 186.04 | 15.94 |
| 30 | 165.8 | 193.9043435 | 17.46 | 82.92 | 193.81025 | 17.35 | 10.43 | 192.27 | 12.03 |

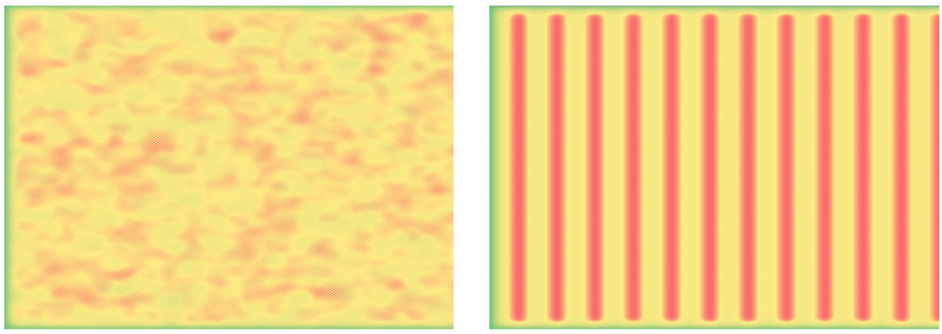
The following heat maps show the convergence of values after running the kernel. The first pair of images represents the unprocessed raw input. The heat maps illustrate 280x280 grids, where the values in the left maps were initially created with a pseudorandom function, and the values in the right maps were created by looping from 1 to 100.

To calculate the execution time for a system requiring more than 30 iterations, divide the desired number of iterations by 30 multiplied by the time it takes 30 iterations to run $((desired number of iterations \div 30) \times (time for 30 iteration to run))$ For example, 300 iterations applied to a 4088x65536 data set should take around 1658 ms to run.





The following heat map shows the result of running the initial values through a kernel with 30 chained calculation CUs. You can see the beginnings of convergence emerge.



# 2. Document Revision History for AN 870: Stencil Computation Reference Design

| Document Version | Changes |
|---|---|
| 2018.10.10 | Initial release. |