White Paper

FPGA
Memory Bandwidth, HBM

Addressing Memory-Bandwidth and Compute-Intensive Challenges with Intel Agilex® 7 FPGAs M-Series

Authors

Supriya Velagapudi
Memory IP Product Marketing Manager
Intel Programmable Solutions Group

Mark Honman
FPGA IP SW Design Engineer
Intel Programmable Solutions Group

Table of Contents

Introduction

FPGAs are taking on an increasingly important role in modern applications from the data center to the network to the edge. Their flexibility, power efficiency, massively parallel architecture, and huge input/output (I/O) bandwidth make FPGAs attractive for accelerating a wide range of tasks from high-performance computing (HPC) to artificial intelligence (AI) to storage and networking. Many of these applications put enormous demands on memory, including capacity, bandwidth, latency, and power efficiency.

To handle these high-demand applications, Intel has created Intel Agilex® 7 FPGAs M-Series, which are the sequel to the successful Intel® Stratix® 10 MX device family. M-Series devices are implemented on the Intel 7 process technology, which brings higher programmable fabric capacity and performance while consuming less power.

M-Series devices offer the highest memory bandwidth in the FPGA industry and are the first members of the Intel Agilex device family to provide in-package HBM2e memory. M-Series devices also include hardened controllers for other state-of-the-art memory technologies such as DDR4, DDR5, and LPDDR5. Two hardened memory network-on-chip (NoC) functions provide the FPGA fabric with high-bandwidth, resource-efficient access to both in-package HBM2e and onboard memory resources.

Furthermore, M-Series devices offer class-leading transceiver data rates, critical for systems processing today's enormous data loads. With support for PCI Express (PCIe) Gen5, Compute Express Link, 400G Ethernet, and serial transceivers operating up to 116 Gbps, the M-Series devices can support the throughput requirements of the most demanding applications from the data center to the edge.

Markets that will benefit from M-Series devices include, but are not limited to, test and measurement (arbitrary waveform generators, 5G/6G cellular network test, GHz RF test); data centers (high performance computing (HPC), cloud computing, cryptocurrency mining); wireless and wireline (high data rate (888G+) transmission, optical transport network (OTN), network functions virtualization (NFV), 5G Baseband); and aerospace and defense (radar, electronic warfare (EW)).

Industry Grapples with Memory Challenges

Today's computational workloads are larger, more complex, and more diverse than ever before. The explosion of HPC, AI, machine vision, video streaming, gaming, analytics, and other specialized tasks is driving the exponential growth of data. According to projections from Statista,[1] 74 zettabytes of data will be created in 2021 alone (a zettabyte is a trillion gigabytes). That's up from 59 zettabytes in 2020 and 41 zettabytes in 2019, and the pace is accelerating.

[1] https://www.statista.com/statistics/871513/worldwide-data-created/

Traditionally, DDR memories have been favored by many developers to address their memory needs. In recent years, however, the demand for increased bandwidth, higher capacity, and greater power efficiency has outpaced the growth in DDR performance, bringing us to a situation where more robust solutions are required. Furthermore, flat power budgets and small form factor restrictions mean that it is necessary to do more in the same space.

Addressing Memory Challenges with M-Series Devices

Before we consider how M-Series devices address today's memory bandwidth and capacity requirements, let's first consider a high-level view of the M-Series device architecture. These devices are built using System-in-Package (SiP) technology (Figure 1).

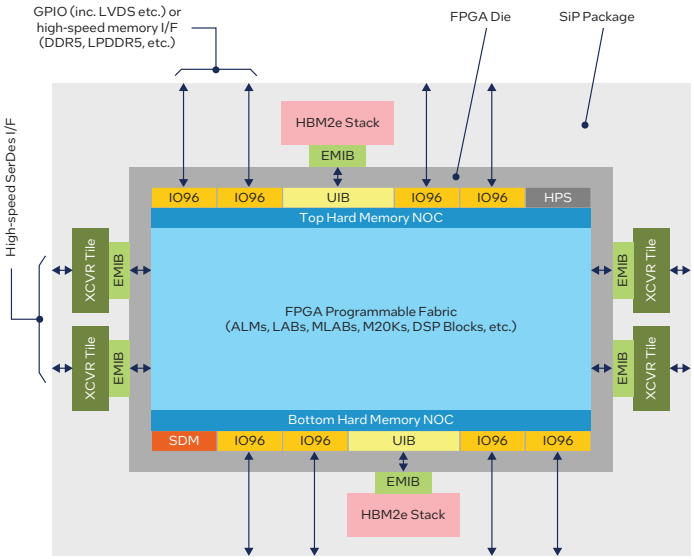

Figure 1. M-Series device floorplan.

In addition to the main FPGA die, there are four transceiver (XCVR) tiles and two HBM2e stacks. The XCVR tiles and HBM2e stacks are connected to the FPGA die using Intel Embedded Multi-die Interconnect Bridge (EMIB) technology, which is an elegant and cost-effective approach to the in-package high density interconnect of heterogeneous chips. The result is that all these chips function as a single large die.

The Memory Hierarchy

Many of today's applications require a hierarchy of memory resources. This hierarchy allows design teams to make latency-versus-capacity trade-offs between ultra-low latency, ultra-high bandwidth on-chip memory (MLAB and M20K blocks); higher-capacity, high-bandwidth in-package memory (HBM2e); and ultra-high-capacity on-board memory (DDR4, DDR5, LPDDR5, etc.) (Figure 2).

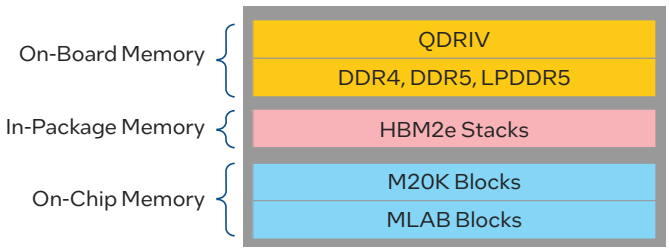

Figure 2. The M-Series device memory hierarchy.

M-Series devices bring considerable resources to bear on the memory bandwidth challenge. Let's examine these from the inside out, starting with the hyper-local on-chip memories in the FPGA fabric, then moving to local in-package memory in the form of the HBM2e stacks, and finally considering the architecture and interfaces for external memories such as DDR5 and LPDDR5.

On-Chip Memory

When designers need the highest locality of memory, nothing competes with the on-chip memory resources in the form of MLAB blocks and M20K block RAMs embedded in the programmable fabric.

In-Package Memory (HBM2e)

In-package HBM2e spans a large and critical gap in the memory hierarchy, enabling today's data-intensive applications. The capacity is far greater (more than two orders of magnitude) than is available with on-chip memory, and the bandwidth is far greater (more than two orders of magnitude) than is possible with off-chip memory.

By integrating high-performance HBM2e in the same package with the FPGA die, we get higher bandwidth, lower power, and lower latency in a small form factor. Also, since the HBM2e is embedded in the package, it does not require the use of external I/O pins, thereby eliminating accompanying board footprint, power consumption, and interconnect delay issues.

Each HBM2e stack can contain 4 or 8 layers, with each layer providing 2 GB, so a single M-Series device can contain 16 or 32 GB of high-bandwidth memory. Each stack has an associated universal interface bus (UIB) function, which includes eight hard controllers and hard PHYs. Each hard controller services one HBM2e channel, and each of these channels is broken down into two pseudo channels (PCs). The result is to maximize performance across all transactions, providing up to 410 GBps memory bandwidth per stack, which is 18X more bandwidth than a DDR5 component and 7X more bandwidth than a GDDR6 component. Combined, the two HBM2e stacks can provide up to 820 GBps peak memory bandwidth.[2] Each HBM2e stack supports 8 channels, or 16 pseudo channels that can be used to route data to and from the stack.

[2] System-level throughput can be improved by12.5% when the error correction code (ECC) bits in the HBM device are utilized for storing data.

Off-Chip Memory (DDR5, LPDDR5, etc.)

For application demands that exceed the HBM2e's capacity, or where the additional flexibility of discrete memory is required, M-Series devices support the latest and highest-performance DRAM variants -- DDR5 and LPDDR5 -- as well as other popular memory architectures.

Massive I/O Capacity

Getting data into and out of the FPGA is critical for today's data-intensive applications. Intel Agilex 7 FPGA M-Series brings massive I/O bandwidth via transceivers capable of supporting up to 116 Gbps PAM4, CXL, PCIe 5.0, 400G Ethernet, and a wide variety of other protocols.

M-Series devices support up to 768 primary I/Os connected to enhanced IO96 subsystems. These pins can function as general-purpose I/Os (GPIOs) that can support a variety of electrical interfaces, such as low-voltage differential signaling (LVDS), or as high-speed interfaces to on-board memory devices. Meanwhile, the XCVR tiles provide high-speed SERDES (serializer/deserializer) interfaces that can implement communications protocols like PCIe 5.0 and 400G Ethernet.

Extreme DSP Capability

M-Series devices contain up to 12,300 variable-precision DSP blocks, each of which contains two 18x19 DSP multipliers. The DSP blocks can support up to 18.5 single-precision TFLOPS, up to 37 half-precision TFLOPS, and up to 88.6 INT8 TOPS. The floating-point capabilities of these DSP blocks allow Intel Agilex FPGAs to outperform conventional FPGAs that have only fixed-point support.

Network-on-Chip (NoC) Functions

A key differentiator for M-Series devices is the addition of two hard memory Network-on-Chip (NoC) functions, which facilitate high-bandwidth data movement between the FPGA's programmable fabric and NoC-attached memories without using existing FPGA routing resources. Each on-chip HBM2e stack communicates with its NoC via its UIB. Off-chip memories (DDR4, DDR5, etc.) communicate with the NoCs via the IO96 subsystems (Figure 3).

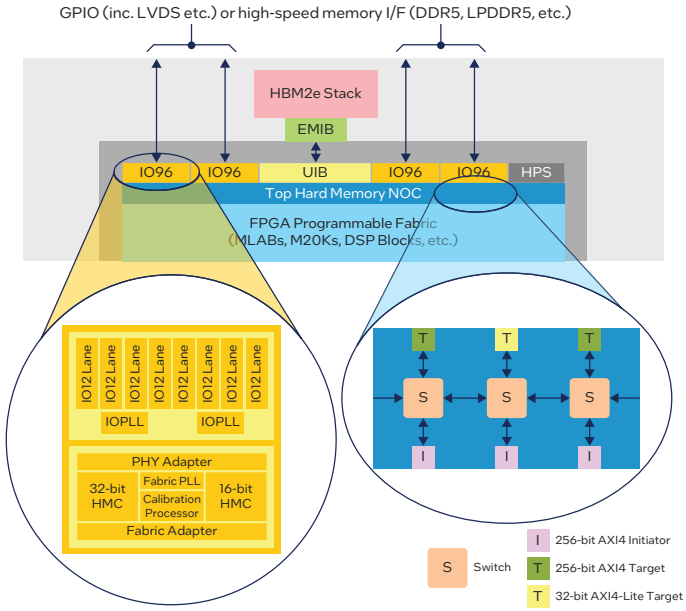

Figure 3. A closer look at a IO96 subsystem and a portion of the top hard memory NoC.

The NoCs route data from its source to its destination via a network that consists of switches (routers), interconnect links (wires), initiators (I), and targets (T).

Horizontal and Vertical Networks

Each NoC provides a horizontal network that connects logic in the programmable fabric via its AXI4 initiators to NoC-attached target memories. Furthermore, each NoC provides a vertical network that can be used to distribute memory read data from the horizontal network paths deep into the FPGA's programmable fabric via optimized routing (Figure 4).

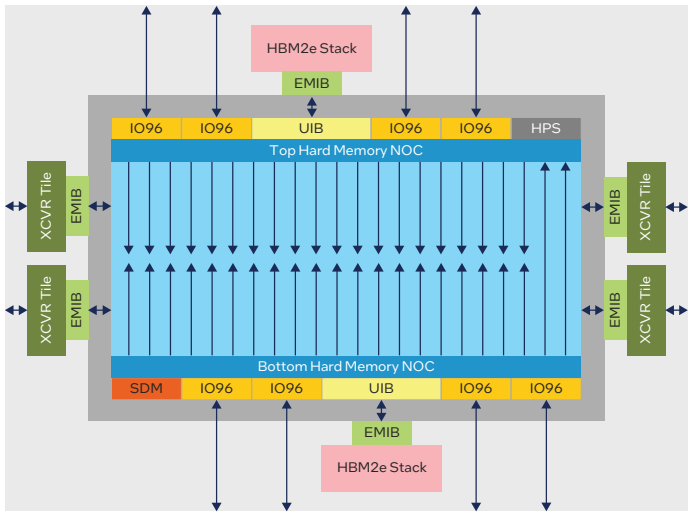

Figure 4. Vertical networks can transport read data from NoC-attached memory to M20K blocks deep in the programmable fabric.

The top NoC has 20 x 256-bit initiators and the bottom NoC has 22 x 256-bit initiators. The bandwidth of each initiator is 256 bits x 700 MHz = 22.4 GBps. With regard to routing read data from NoC-attached memories into the programmable fabric and/or M20K blocks, the top NoC's bandwidth is 256-bit x 700 MHz x 20 initiators = 3.58 Tbps, the bottom NoC's bandwidth is 256-bit x 700 MHz x 22 initiators = 3.94 Tbps, so the aggregate bandwidth is 3.58 + 3.94 = 7.52 Tbps.

The access-points to the NoCs are known as initiators and targets. User logic in the programmable fabric connects to 256-bit AXI4 initiators to initiate requests and send data (each initiator on the fabric side can be clocked independently by user clock), while 256-bit AXI targets provide responses from NoC-attached memories. Every initiator can talk to every target, thereby giving users the flexibility to implement a full hardened crossbar. The switches in the NoC route requests and responses between initiators and targets using a proprietary protocol.

Use Cases

The use cases presented below[3] were selected to reflect the ratio of computation workload to memory traffic and memory access patterns that are exhibited by real-world workloads. These use cases are 5G RF analog hardware-in-the-loop testing and the Shallow Water Model, which is representative of weather forecasting workloads.

5G RF Analog Hardware-in-the-Loop Testing

In the case of companies that make RF and microwave analog devices -- for example amplifiers or other analog circuitry that is used in applications like radar and 5G base stations -- the response of the device must be tested with high input frequencies. Any anomalies in the response will manifest themselves across a wide frequency range, spanning lower frequencies than the operating frequency to higher frequencies resulting from things like harmonic noise.

A common test scenario is to use hardware-in-the-loop (HIL). In this particular use case, a M-Series device forms part of a larger test system (Figure 5).

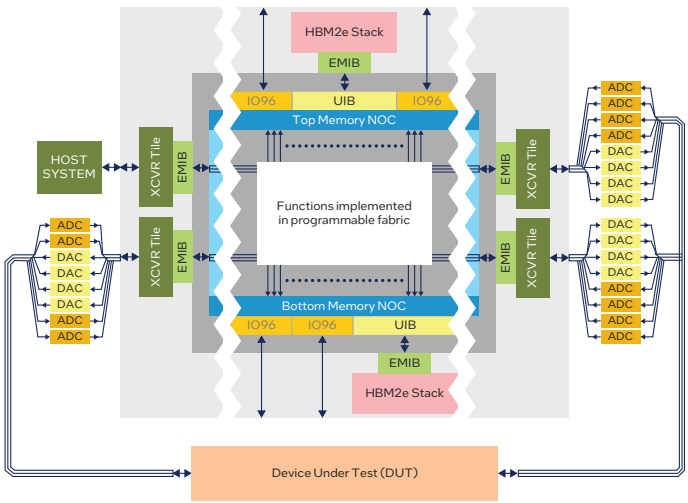

Figure 5. Using a M-Series device as part of a 5G RF HIL testbench.

The host system, which is usually a CPU, generates custom waveforms that typically last for a few milliseconds and are carefully crafted to exercise the analog device under test (DUT) and expose potential problems and weaknesses. These flaws will be manifested as errors in the DUT's responses to the stimulus waveforms.

For highly integrated analog products, the host system will upload (play-in) multiple super-high-frequency waveforms that are all perfectly aligned, download (capture) multiple response output waveforms, and analyze the results. Data loading and retrieval take place concurrently. Based on the results of this analysis, new stimulus waveforms will be applied.

In this particular use case, the FPGA communicates bidirectionally with the host system via one of its XCVR tiles that is configured to act as a PCIe 5.0 x16 interface with 64 GBps bandwidth. The FPGA uses its three remaining XCVR tiles to play these waveforms, which are synchronized with each other, across 12 digital-to-analog converter (DAC) channels while simultaneously capturing the output waveforms from the DUT across 12 analog-to-digital converted (ADC) channels. The data loader and retriever logic that orchestrates all of this is implemented in the FPGA's programmable fabric (Figure 6).

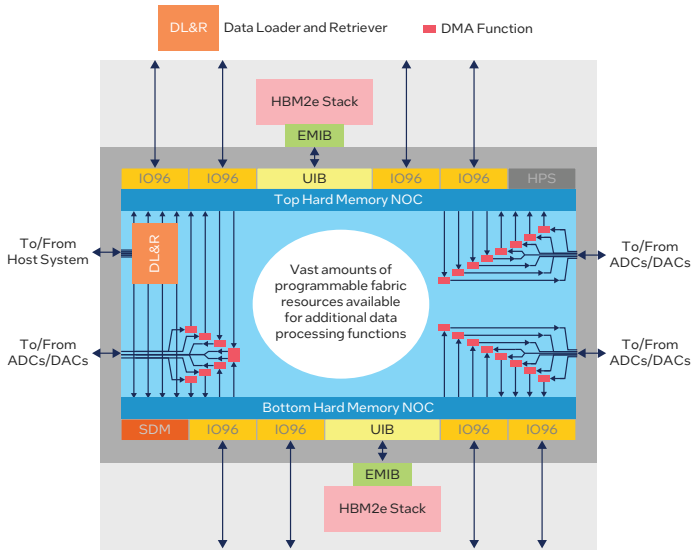

Figure 6. Vast amounts of programmable fabric resources remain available to implement additional functions such as data reduction logic.

The 12 DAC and 12 ADC channels each operate at up to 4 Gsps (giga samples per second). With word lengths of 32-bits per complex sample (16I + 16Q), this equates to a bandwidth of 15.625 GBps per channel, resulting in an I/O bandwidth of 24 x 15.625 = 375 GBps for the tiles interfacing with the DACs and ADCs, and a total I/O bandwidth of 375 + 64 = 439 GBps when the PCIe 5.0-configured tile communicating with the host system is taken into account.

At this sampling rate, there's too much data to be stored in the on-chip M20K memory blocks; instead, it has to be stored in the in-package HBM2e stacks. Inside the FPGA, the waveforms from the host system are loaded into the HBM2e stacks via the NoCs. Playout data from the HBM2e stacks is

[3] Use cases developed by Intel engineering

fed via the NoCs to direct memory access (DMA) functions implemented in the programmable fabric. Using optimized routing for the NoC's vertical networks guarantees Fmax closure and frees-up the FPGA's regular logic resources for other tasks.

In addition to communicating the data to the DACs driving the DUT, the DMA functions also buffer the data because DRAM access is not 100% regular due to activities such as auto-refresh. Similarly, capture data from the ADCs driven by the DUT is communicated to DMA functions implemented in the programmable fabric. These functions pass the data to NoCs, which -- in turn -- pass it to the HBM2e stacks.

Vast amounts of programmable fabric resources remain available to implement data reduction logic, such as a massive fast Fourier transform (FFT) engine that can access and process data from the top and bottom hard memory NoCs simultaneously.

For this application to use the HBM2e efficiently, the data is handled as parallel data streams based on pseudo channels (PCs). Four PCs on the top and bottom HBM2e stacks are reserved for the PCIe 5.0 x16, which can read four PCs and write four PCs simultaneously (double buffering is used when transferring waveforms from the Host System to the HBM2e). In each half of the device, six PCs are reserved for playout data and six PCs are reserved for capture data. In this design, each PC is used as only a data source or a data sink to eliminate contention.

The NoCs featured in M-Series devices convey many advantages, including the fact that the association between data source and destination can be changed dynamically -- each NoC can be configured as a 16 x 16 switch, so any DAC or ADC can talk to any PC. The NoCs provide the ability to read from different PCs simultaneously. If the PCIe is loading new data into the FPGA while capture is in progress, the NoC enables the playout engine to read the new waveforms from different PCs to the ones used for the previous set of waveforms, which means the application is not stuck with 1:1 associations between DACs and PCs. If required, it's also possible to perform full duplex read and write by connecting user-defined logic in the programmable fabric to the initiators in the NoCs.

The Shallow Water Model

The Shallow Water Model (SWM) fits into a class of HPC algorithms called stencil problems that simulate physical phenomena by partitioning the space into discrete cells. To model the behavior through time, each cell is recalculated iteratively according to a set of equations that depend on the current values of a set of variables in each cell, as well as the values of variables in a set of neighboring cells called a "halo."

The stencil is a structure that includes the current cell and its neighboring cells. Using a stencil, we can parallelize the computation so that large numbers of cells can be recomputed at the same time.

This SWM example can be a proxy for a wide range of applications and problem domains. Thermodynamic problems, for example, use stencil-based computations to simulate how heat moves through a semiconductor package. Other applications such as antenna simulation and computational fluid dynamics use similar approaches, where we discretize space and time and apply a set of equations at each time step.

Weather forecasting models use complex stencil-based methods to model pressure, temperature, wind velocity, and other variables in creating and updating forecasts. Our example will be a proxy to weather modeling (which is a 3D problem and has more than 10 variables to compute at each point). We will use SWM code that is part of the SPECfp2000 benchmark suite to illustrate the performance benefits of HBM2e and DDR5 in stencil-based computations. SWM simplifies the weather model to a 2D problem with three variables: p (pressure), v (vertical component of velocity), and u (horizontal component of velocity).

In weather forecasting, cells are updated many times at discrete time steps to predict the weather one or two days into the future. SWM maintains current and prior variable values for each cell in memory, updates them at each time step according to a set of update equations, and then writes the new values back to memory. This is performed until the entire space is computed.

When using FPGAs to accelerate compute-intensive workloads such as this, most designers start from the perspective of maximizing datapath parallelism. This is natural, as FPGAs, with their copious DSP/arithmetic resources, offer the potential for massive acceleration of algorithms manipulating large arrays of data.

However, because M-Series devices are specifically designed for cases where memory and/or I/O are the computational bottlenecks, a better approach may be to explore the solution from the perspective of using all available memory bandwidth, and then determining the amount of data parallelism required to support the specified bandwidths.

Data Flows

Our calculation employs three variables as follows:

u – Horizontal component of velocity

v – Vertical component of velocity

p – Pressure

Prior to optimization for FPGA acceleration, the calculation has the following phases:

1. Use p, (u, v) to calculate a set of intermediate variables (U, V), z, h to save computational effort.
2. Exchange freshly calculated intermediate variable values by applying cyclic boundary conditions to (U, V), z, h.
3. Use the (U, V), z, h intermediate values to calculate new p, (u, v) values.
4. Apply cyclic boundary conditions to p, (u, v).
5. Time-smooth the p, (u, v) by keeping old versions of the data (we actually keep two copies of the historical data to facilitate sequential read/write).

Our implementation utilizes oneAPI to facilitate seamless development for heterogeneous targets. It combines phases 1 and 2 and coalesces phases 3, 4, and 5 to improve memory efficiency, resulting in an algorithm with two computational stages separated by boundary data exchanges (Figure 7).

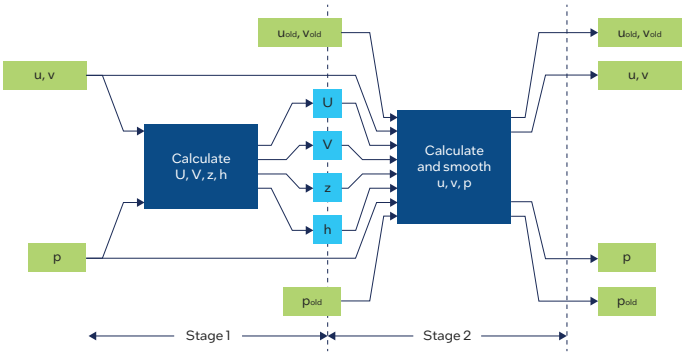

Figure 7. Data flows within an iteration.

Stage 1: Calculates the intermediate values U, V, z, and h from u, v, and p. At the boundary between Stage 1 and Stage 2, there is an update of the boundary conditions where freshly calculated intermediate values are copied into halo cells along the edges of the 2D grid.

Stage 2: Read intermediate values and the current values to calculate new u, v, and p values. The time-smoothing computation uses current, new, and previous variable values to ensure numerical stability. For time-smoothing, we also need to read old values.

Memory Requirements

Since this is a memory-bound computation, we will minimize the number of arrays stored in HBM2e and DDR5 to improve the performance by utilizing M20Ks to store some variables.

In our scenario M20Ks are used to store u, v, p, U, V, z, and h array values. 896 M20Ks are used for storing SWM's 3 variables + 4 intermediate variables for a 64 x 64 patch size. This reduces inter-stage latency between Stage 2 and Stage 1 of the next iteration and reduces the idle time in computation. "Old" p, u, v array values used in time-smoothing are stored in DDR5/HBM2e. Only Stage 2 uses these "old" p, u, v values, so the DDR/HBM access latency is less important–DRAM output from Stage 2 can be committed to memory while Stage 1 of the next iteration is in progress.

For maximum throughput, we can use all the available 32 pseudo-channels of HBM2e (16 on the top and 16 on the bottom) and also use 8x32 GB DDR5 memories (4x32 GB on the top, and 4x32 GB on the bottom).

An efficient formulation of Stage 2 uses two copies of historical data. It reads uold, vold, and pold from half the memory channels, performs the calculations, and then writes new versions of these arrays to the other half of the memory channels. This purely sequential access pattern enables each memory channel to operate at 22.4 GB/s. Therefore, we can attain a device aggregate of 716GB/s (HBM2e only) or 896GB/s (HBM2e + DDR5) (for a -2 speed-grade FPGA).

In double-precision floating-point, the memory bandwidth is matched by data path parallelism. We should aim for data parallelism of 40, if using all 40 memory channels (32 HBM2e + 8 DDR5). This means that the fabric will read 40 elements of each row simultaneously feeding 40 computational pipelines. We need to arrange data so that all the memories are constantly busy with sequential reads and writes, thereby ensuring that DRAM bandwidth is fully utilized.

Implementation

To maintain sequential access patterns, we must create two buffers in which to store the "old" values of our three variables. We will call these (pold, vold, uold) top, (pold, vold, uold) top', and This allows us to read from one pseudo-channel while writing to another. We split each copy of the array into two halves, one half stored in the top HBM2e stack, and one stored in the bottom. The 40 computational pipelines are similarly distributed between the top and bottom halves of the device fabric, each pipeline working on data from the nearest memory (Figure 8).

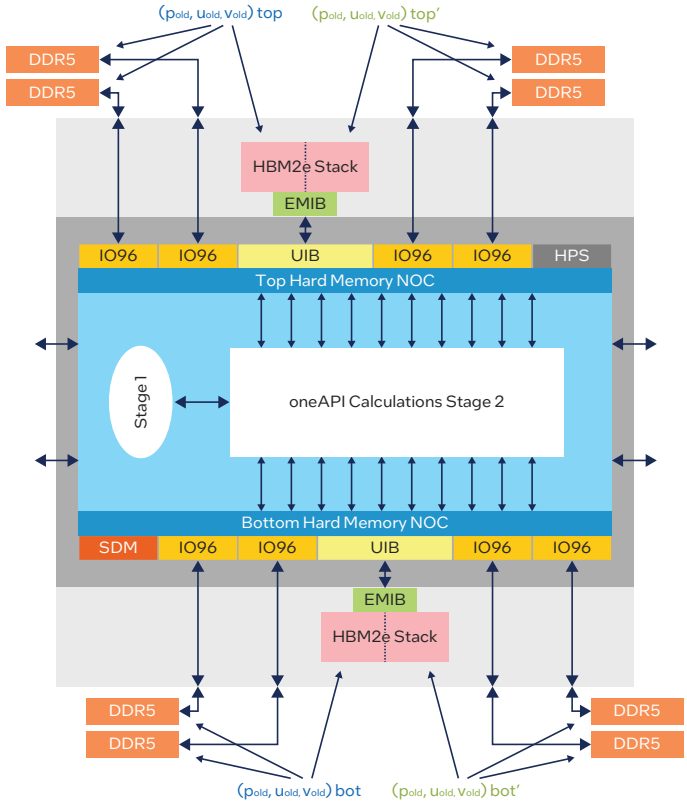

Figure 8. In one iteration, we read from the top channel(s) [HBM2e+DDR5] and write to the top' channel(s). In the next iteration, we read from top' and write to top. Similarly for the bottom.

This analysis is based on a 64x64 computational patch size. Its surface to volume ratio -- and hence ratio of communication to computation -- is representative of many parallel algorithms. Our array is 64x64. Each element in the array contains three double-precision floating-point values. We need to stripe the array across multiple pseudo-channels, putting the first 64 bytes of the array into the first pseudo-channel, then the next 64 bytes into the next pseudo-channel, and so on, in order to achieve a good logical-to-physical address mapping. This consumes all 16 pseudo-channels plus four DDR5 channels on each of the top and bottom HBM2e arrays. We access these 20 channels on each side using 10 initiators of the NoC. Since our initiators are full-duplex, they can read and write to a pair of pseudo-channels at the same time. This allows us to effectively use up the memory bandwidth without running out of initiators.

Each iteration of the algorithm performs 65 floating point operations per cell: 25 multiplications, 39 additions, and one division. Implemented in double precision, 40 pipelines consume 3,360 of the M-Series device's12,500 DSP blocks. Since the two computational stages are not simultaneously active, we can project application performance of up to 650 double-precision GFlops based on a fabric frequency of 500 MHz.

Conclusion

Today's computational workloads are larger, more complex, and more diverse than ever before. Some applications require the streaming of vast quantities of data, while others are characterized by large quantities of short random bursts. Similarly, some algorithms may demand minimal latency when accessing memory, while others may be more tolerant.

To handle these high-demand applications, Intel has created the M-Series device — the first Intel Agilex FPGAs implemented on the Intel 7 process technology that feature in-package HBM2e memory. M-Series devices also include hardened controllers for other state-of-the-art memory technologies such as DDR5 and LPDDR5. Hard memory NoC functions provide the FPGA fabric with high-bandwidth, resource-efficient access to both in-package HBM2e and out-of-package (on-board) memory resources.

The biggest challenges in networking, data center, and edge require the combination of high compute resources coupled with high memory and high I/O bandwidth. M-Series devices offer the most INT8 TOPS and FP32 TFLOPs of any HBM-enabled FPGA. They also offer the highest aggregate memory bandwidth -- over 1 TBps using both HBM2e stacks and all eight DDR5 interfaces. Furthermore, M-Series devices deliver over 2.65 Tbps aggregate serial transceiver bandwidth in each direction (over 5.3 Tbps full-duplex) for next generation 800G/1.6T networking and network functions virtualization infrastructure (NFVI) applications.

The power of M-Series devices is available to all developers -- hardware design engineers can use the Intel® Quartus® Prime Software design tool, while software developers can employ oneAPI (a core set of tools and libraries for developing high-performance, data-centric applications across diverse architectures).

WP-01313-1.2