

# Memory Sandbox: A Versatile Tool for Analyzing and Optimizing HBM Performance in FPGA

Elias Perdomo

Barcelona Supercomputing Center (BSC)  
Universitat Politècnica de Catalunya (UPC)

Barcelona, Spain  
elias.perdomo@bsc.es

Xavier Martorell

BSC  
UPC

Barcelona, Spain  
xavier.martorell@bsc.es

Teresa Cervero

BSC

Barcelona, Spain  
teresa.cervero@bsc.es

Behzad Salami

BSC

Barcelona, Spain  
behzad.salami@bsc.es

**Abstract**—Main memory access has become an increasing performance bottleneck for traditional and High-Performance Computing (HPC) applications. High Bandwidth Memory (HBM) has emerged as an alternative to conventional DRAMs, offering higher bandwidth, lower power consumption, and greater integration capabilities to meet the escalating demands of contemporary applications. The transition to HBM of the most advanced Field Programmable Gate Arrays (FPGAs) marks a paradigm shift. However, users face substantial challenges due to the scarce technical documentation and tools supporting HBM on FPGAs.

This paper introduces the *Memory Sandbox*, an open-source tool integrated into our FPGA-Shell<sup>1</sup>, designed to address the complexity of utilizing HBM in FPGAs. It allows developers and students to explore roofline performances while gaining insights to improve their designs. The *Memory Sandbox* enables users to configure various parameters such as the number of processing elements accessing memory, the access pattern (sequential, pseudo-random, or sparse-wise), and memory configurations, emulating multiple processor threads in diverse heterogeneous scenarios. It provides detailed analyses of memory access impacts in terms of latency and throughput for scenarios with accesses *within* and *across* HBM pseudo-channels; as well as HBM performance under *concurrent* access scenarios.

Our results show that HBM achieves 99.99% of its nominal peak bandwidth with long sequential accesses but drops to 0.17% with random data access patterns. The tool also highlights the impact of multiple AXI ports targeting the same pseudo-channel, revealing that the aggregated throughput remains constant regardless of the pseudo-channel count. Furthermore, we validate the *Memory Sandbox*'s capabilities by effectively profiling complex access patterns like Sparse Matrix-Vector (SpMV), demonstrating its effectiveness in providing accurate performance insights.

**Index Terms**—HBM, DDR, performance, throughput, FPGA, pseudo-channel, micro-switches

## I. INTRODUCTION

The use of heterogeneous computing systems has significantly increased in recent years. Modern computing architectures, ranging from massive High-Performance Computing (HPC) platforms to compact Internet of Things (IoT) devices, increasingly integrate co-processors and accelerators to improve the overall system performance and reduce power

The MEEP Project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 946002. The JU receives support from the European Union's Horizon 2020 research and innovation program and Spain, Croatia, and Turkey

<sup>1</sup>[https://github.com/MEEPproject/fpga\\_shell](https://github.com/MEEPproject/fpga_shell)

consumption. This hardware diversity enables the execution of various specific or non-specific workloads usually faster and more efficiently than traditional general-purpose systems [1]. However, when optimizing performance in heterogeneous computing, designers must consider that certain computing resources may be suitable for a problem and data combination, while others may be suitable for a different combination [2].

Custom and general hardware have improved considerably in recent decades. However, while commercial microprocessors speeds have increased annually by 70%, the performance improvement for commodity DRAMs has been just 50% over the entire past decade [3]. This disparity increases the performance gap between processing elements and memory, leading to significant performance bottlenecks in numerous applications [3].

To bridge this gap, designers are exploring new architectures based on the utilization of both general and custom hardware. FPGAs have emerged as a promising solution for custom hardware acceleration, offering unique advantages over other commonly used computing devices in modern electronics like Graphics Processing Units (GPUs) [4], [5], Application-Specific Integrated Circuits (ASICs) [6], [7] and emerging technologies like Google's Tensor Flow Processing Units [8], [9]. FPGAs' architectural differences, performance traits, power efficiency, adaptability, and scalability illustrate why they play a significant role in computing nowadays [10], [11]. However, FPGAs' limited resources, particularly in memory, with an absence of a default cache system, make it crucial to explore ways to optimize memory usage to address these applications' limitations.

Traditionally, boosting performance in critical memory-bound systems involved adding more computational capabilities and bringing more memory on-chip, especially in scenarios processing large volumes of data locally or regionally. However, such an approach no longer scales since we are reaching the boundaries of the Von Neumann, Moore's Law, and Dennard scaling trifecta. To overcome these limitations while improving performance, recent research has focused on bringing processing elements closer to, or even into memory.

Memory semiconductor companies, such as Samsung, have proposed memory technologies such as Hybrid Memory Cube (HMC) and High Bandwidth Memory (HBM) [12], [13] to

TABLE I  
HBM STANDARDS EVOLUTION

HBM Specification	HBM	HBM2	HBM2e	HBM3	HBM3e*
JEDEC Standard	Oct 2013	Jan 2016	Dec 2018	Jan 2023	next
Die Density	2Gb	8Gb	16Gb	16Gb	24Gb
Max dies per stack	4Hi	4Hi/8Hi	4Hi/8Hi/12Hi	8Hi/12Hi/16Hi	8Hi/12Hi/16Hi
Channels to SoC	8 independent. 2x 128-bit per die		8x 128-bit channels	16x 64-bits channels	16x 64-bits channels
Total HBM width			1024 bits ( for 8-Hi stack)		
Interface to SoC			Interposer or direct stack		
Max Pin Transfer Rate	1 Gb/s	2.4 Gb/s	3.6 Gb/s	6.4 Gb/s	9.8 Gb/s
Max Capacity per stack	1GB	4GB/8GB	8GB/16GB/24GB	16GB/24GB	36 GB
Max Bandwidth per stack	128 GB/s	307 GB/s	460 GB/s	819 GB/s	1.2 TB/s

address the need for greater bandwidth and a better performance. Companies like NVIDIA and AMD have incorporated HBM in their latest devices. The NVIDIA Tesla V100, with thousands of computing cores, ensures a memory bandwidth of 1134 GB/s using 32 GB of HBM2, becoming the first GPU to break the 100 TFLOPs barrier for Deep Learning [14]. Similarly, FPGA companies like AMD are incorporating HBM in their latest commercial FPGAs to provide an order of magnitude gain in memory bandwidth compared with the same GPU generation. Table I shows the evolution of the Alveo family [15]–[19] and AMD’s the undeniable transition to HBM for their Alveo FPGAs targeting HPC systems.

The HPC field targeted by these boards can be hugely impacted by HBM [16]. HPC is a growing market in size and importance, valued at USD 27.28 billion in 2017 and projected to reach USD 52.73 billion by 2027 [20]. Current HPC systems are based on heterogeneous architectures with different accelerators having diverse access patterns each. Many of these High-Performance Data Analytics (HPDA) and HPC applications are memory-bound due to their sparse memory-access patterns, like the well-known Sparse Matrix-vector Multiplication (SpMV) [21] and High-Performance Conjugate Gradients (HPCG) benchmarks [22]. Even advanced systems, like Fugaku, performs around 5% of the peak for HPCG, whereas for dense benchmarks its speedup is 80% [22], [23].

Despite its potential, leveraging HBM efficiently in FPGA-based solutions is challenging due to limited information on its behavior. To leverage HBM efficiently within FPGA-based solutions, it is crucial to understand its benefits, limitations, and optimal utilization strategies. Thus, we focus on addressing HBM in FPGAs from an HPC perspective. Our efforts are oriented towards developing a holistic Sandbox Design Exploration Tool for analyzing diverse designs from memory, network, and processor perspectives. In this paper, we address the imperative of maximizing HBM benefits.

This work makes the following contributions towards an efficient use of HBM on FPGAs:

- We examine historical Heterogeneous Computing and CPU-bound and memory-bound applications trends while exploring FPGA characteristics, usage progression, and barriers to their utilization in these scenarios.

- We study HBM architectural features, standards, and its addition to Xilinx Alveo HPC-oriented boards.
- We introduce the Memory Sandbox tool to analyze HBM performance in FPGAs while comparing these results to DDR.
- Our Memory Sandbox tool empowers users to configure the number of processing elements accessing memory, the access pattern (sequential, pseudo-random, or sparse), and various memory configurations to emulate multiple behaviors of processing threads in diverse heterogeneous scenarios.
- Using our tool’s capabilities, we provide insights into the impact of memory accesses in terms of latency, bandwidth, and power consumption. We perform analyses within and across HBM pseudo-channels and evaluate HBM performance in concurrent access scenarios.
- The tool allows both educators and hardware designers to validate state-of-the-art results such as those presented by Shuhai [24], [25] or Holzinger [26] while shedding light on memory access due to its scalable, and versatile nature.
- The tool also facilitates the exploration of new custom scenarios. As a demonstrative use case, we present an implemented trace-based exploration of a SpMV operation to study memory access behavior and optimize HBM performance.

**Outline:** In Section 2, a study of the HBM features, standards, and their growing importance in current computing systems is presented. In Section 3, a detailed description of the Memory Sandbox tool is presented from a co-design point of view detailing its front and end pieces. Then, in Section 4, using the Memory Sandbox tool we perform a microbenchmark-based design analysis exploration in multiple scenarios with DDR and HBM considering different address mapping policies, within and across HBM pseudo-channels access, concurrent access scenarios, and sequential and sparse access patterns. Notably in Section 5, a real-wise application representative of a real sparse memory access pattern is explored with our tool and a comparison of the obtained throughput with the ideal cases presented in the previous section is presented. Section 6 focuses on the related work. Finally, Section 7 summarizes the key points of this work and gives some examples of next-generation ideas.

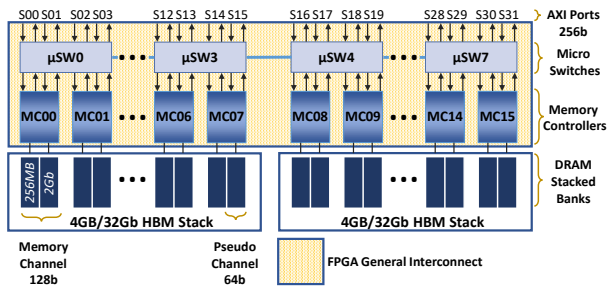


Fig. 1. Alveo U280 HBM topology.

## II. HBM FEATURES AND STANDARDS

HBM is a type of memory interface used in 3D-stacked Dynamic Random Access Memory (DRAM) oriented for GPUs, HPC, data center servers, and networking and client spaces. HBM technology works by vertically stacking memory chips to reduce data movements while allowing for smaller form factors. Additionally, with two 128-bit channels per die, the HBM bus is wider than other types of DRAM memory buses. Stacked memory chips adopt state-of-the-art Integrated Circuits (IC) packaging technologies to ensure faster memory accesses using Through Silicon Via (TSVs), micro-bumps and connection via an interposer, rather than on-chip [27]–[31]. It is therefore not surprising that HBM can overcome all DRAM challenges as an enabler of architectures for high-performance and/or low-power computing, while its low speed/pin consumption also improves power efficiency [32], [33].

Several versions of HBM standard and future updates are shown in Table I [34]–[38]. The latest version HBM3 offers a 1.8x speed increase over HBM2e and HBM3e is expected to offer 2.6x. Nevertheless, as no FPGA boards includes it yet [39], HBM2 has been the JEDEC standard since December 2018 with specifications updated in 2020. [12].

Our experiments were performed on a Xilinx Alveo U280 [40], the only Xilinx HPC-oriented board having DDR and HBM, ensuring fair comparisons within the same test environment [41], [42]. Moreover, all the Alveo U280 HBM insights directly apply to the Alveo U55C [43], because their HBM topology is equivalent. The HBM controller in the Alveo U280 has two 4GB stacks (Fig. 1). Each stack has 8 channels further divided into 16 pseudo-channels of 256MB. There is a 256bits AXI Port working up to 450MHz to access each of the pseudo-channels. According to Xilinx, HBM has a bandwidth of 460.8GB/s from the equation  $32 * 256bits * 450MHz$ . Every 4 AXI Ports enters a fully implemented micro-switch providing the same behavior for any access inside the 4 pseudo-channels comprised. The two stacks involve 8 of these micro-switches connected adjacently, allowing access to any pseudo-channel in any micro-switch. Performing “domain changes” among micro-switches needs the global addressing mode, imposing Lateral AXI Switch Access Throughput losses [44].

The Alveo U280 is also equipped with two 16 GB DDR4 RDIMMs with a port width twice that of the HBM but can only run up to a frequency of 300MHz. Therefore, Xilinx indicates a bandwidth for this Alveo DDR4 subsystem of 38.4GB/s from

the equation  $2 * 512bits * 300MHz$  [45]. Having these precepts, we profile HBM performance in the Alveo U280 doing a characterization analysis based on latency and throughput, and compared it with the DDR using our *Memory Sandbox tool*.

## III. MEMORY SANDBOX: ARCHITECTURE

To shed some light on the intrinsic details of HBM, we developed the *Memory Sandbox tool* providing higher configurability, more control over measurements, and further insights (i.e. clock cycles of each memory transaction) than the current HBM monitor offered by Xilinx. Our configurable environment is structured in two main pieces: 1) a *front-end piece* as a user interface for setting up the experiments to be executed, and 2) a *back-end piece* composed by a set of hardware IPs to run the experiments in the FPGA, according to the data introduced in the *front-end*. Thus, the most relevant IP we have developed is a highly *Configurable Pattern Generator*, which mimics processor threads data requests with sequential and pseudo-random memory access patterns.

An initial analysis of typical memory access patterns allows us to implement benchmarks to reveal the subjacent characteristics of HBM and DDR in FPGAs. For this purpose, we emulate the Repetitive Sequential Traversal (*RST*) a typical sequential access pattern widely used in FPGA programming and sparse accesses with pseudo-random accesses. HBM and DDR4 performance in FPGAs are then compared since it is the most common memory used in computer applications.

### A. Back-end piece

RTL implementations are the best way to fully stress any memory efficiently. Therefore, we designed our own *Configurable Pattern Generator (CPG)* controlling memory access pattern generation and simplifying debugging tasks. By replicating the *CPG*, we can also emulate processor threads concurrently requesting data from memory. Different experiments were conducted, either *RST* or *pseudo-random* for different configurations, to guarantee a fair comparison between DDR and HBM, in terms of throughput and latency. These dissimilar access patterns can also be concurrent, closely emulating the scenario of heterogeneous HCP systems nowadays.

Our base design maintains simplicity ensuring that memory access through any AXI Port is the main behavior. Therefore, it

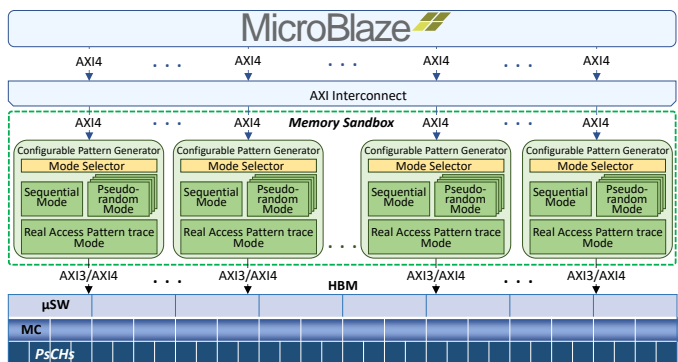


Fig. 2. Hardware architecture for our profiling.

only comprises a *MicroBlaze processor* and  $N$  of our custom *Configurable Pattern Generator* accessing one AXI Port of HBM or DDR bank each. This initial comparison helps us to determine HBM capabilities and to establish a baseline. *MicroBlaze CPU* is used to provide flexibility, being responsible for interpreting the configuration sent from the *front-end* and orchestrating the *CPGs*. For the desired benchmark,  $N$  can be 1 or 2 for DDR and up to 32 for HBM (Fig. 2).

*Configurable Pattern Generator (CPG)*: IP written in VHDL that instantiates an AXI configurable interface in design time to handle the AXI3 and AXI4 interfaces for HBM and DDR respectively. The AXI interface, regardless of the version, comprises the following channels: Write Address (*WA*), Write Data (*WD*), Write Response (*WR*), Read Address (*RA*) and Read Data (*RD*) [46], [47]. We implemented a state machine to handle the write-related channels (*WA*, *WD* and *WR*) and another to handle the read-related channels (*RA* and *RD*).

The behavior of the state machines varies depending on the design parameters (Table II). These parameters enable the emulation of different processor threads with different access patterns concurrently. As an example, the user can have a *CPG* doing sequential accesses inside pseudo-channel  $X$ , another *CPG* doing pseudo-random accesses across and inside pseudo-channels, while a different *CPG* always performs pseudo-random accesses inside pseudo-channel  $Y$ .

Upon receiving a transaction trigger signal our *CPG* always attempts to saturate the read or write related channels while validating appropriate transaction success. To maximize achievable throughput, outstanding transactions are always requested. According to our measurements, HBM admits up to 22 of these outstanding transaction requests before the channel saturates. We measure throughput and latency for Read and Write processes by looking at the actual signals inside the FPGA with an ILA for each *CPG*.

Each instantiated *Configurable Pattern Generator* directly interfaces an AXI Port to avoid delays or arbitration introduced by IPs such as the AXI Interconnect. Moreover, each *CPG* uses the same clock frequency of the memory it is interfacing (450MHz for HBM and 300MHz for DDR4). This configuration allows maximum stress on the associated memory AXI Port and ensures the same clock region, avoiding unnecessary

TABLE II  
SUMMARY OF DESIGN PARAMETERS

Parameter	Definition
<i>Is_DDR</i>	AXI interface for DDR(AXI4) or HBM(AXI3)
<i>Rand_PSCH</i>	Whether randomize AXI pseudo-channels
<i>Rand_Whole_Addrr</i>	Whether randomize the remaining <b>27 address bits</b>
<i>Rand_Bank_Group</i>	If <i>Rand_Whole_Addrr</i> = '0'. Whether Randomize the <b>bank group</b> bits
<i>Rand_Bank</i>	If <i>Rand_Whole_Addrr</i> = '0'. Whether Randomize the <b>bank</b> bits
<i>Rand_Col</i>	If <i>Rand_Whole_Addrr</i> = '0'. Whether Randomize the <b>column</b> bits
<i>Rand_Row</i>	If <i>Rand_Whole_Addrr</i> = '0'. Whether Randomize the <b>row</b> bits

TABLE III  
HBM LATENCY FOR DIFFERENT PSEUDCHANNELS

PSCH	0-3	0-4	8-11	12-15	16-19	20-23	24-27	28-31
Write	14	16	18	20	31	33	35	37
Read	48	50	52	54	67	69	71	73

TABLE IV  
SUMMARY OF RUN-TIME PARAMETERS

Parameter	Definition
<i>Burst_Size</i>	Beats of the memory read/write transactions
<i>Num_trans</i>	Number of memory read/write transactions (beats) to perform
<i>PSCH_Addrr_Init</i>	Starting pseudo-channel/channel for the randomization
<i>PSCH_Addrr_End</i>	Final pseudo-channel/channel for the randomization

FIFOs and additional latencies.

*Latency* is measured in clock cycles from *AWVALID* to *RREADY* signal rises in Table III. HBM memory latency is higher than the measured 5 and 24 cycles for *DDR4 Write* and *Read processes* respectively and it gets worse with the lateral losses. Although we can establish that HBM presents evident disadvantages when running latency-sensitive applications on FPGA, this paper primarily focuses on *Throughput* because the state of the art is considered the most critical factor.

*Throughput* follows the next equation, where we measure the transactions' clock cycle duration and establish the number of transactions:

$$Throughput = \frac{Num\_trans * Bytes\_per\_transfer * Frequency}{Number\_of\_cyclenesspent\_by\_transfer} \quad (1)$$

### B. Front-end piece

The *front-end piece*'s main function is to provide flexibility in terms of run-time parameters. These run-time parameters reduce how many times FPGAs are reconfigured when performing memory analysis. With the *back-end piece*, it is desirable to guarantee performance as a main factor. Therefore, it should be as "transparent" as possible in terms of throughput and minimum achievable latency, so that the data obtained is relative only to the memory, whether it is HBM or DDR.

The *front-end piece* aims to provide user-friendly run-time reconfiguration of the *Memory Sandbox*. Not having to regenerate the FPGA bitstream or, even recoding the design, saves a considerable amount of time and allows obtaining relevant performance characteristics in different scenarios with hardly any changes from the user's point of view. With this purpose, we defined the run-time parameters in (Table IV).

This setup allows us to control if the generated accesses are inside or across different pseudo-channels, the burst size, and the number of transactions. The number of experiments for the same bitstream can be calculated as follows:

$$\begin{aligned} Number\ of\ experiments &= 2^4 * 2^{33} * (2^{33(any)}) = 2^{70} \\ &= Burst\_size * Num\_trans * (PsCH\_End - PsCH\_init) \end{aligned} \quad (2)$$

TABLE V  
ADDRESS MAPPING POLICIES BITS DISTRIBUTION FOR HBM AND DDR4.  
*Default Policies are marked in bold*

Policy	HBM ( <i>app addr</i> [27:5])[21]	DDR4 ( <i>app addr</i> [33:6])[22]
<i>RCB</i>	14R-5C-2G-2B	<b>17R-7C-2B-2G</b>
<i>RCBI</i>	N/A	17R-6C-2B-1C-2G
<i>BRC</i>	2BG-2B-14R-5C	2G-2B-17R-7C
<i>BRGCG</i>	2B-14R-1G-5C-1G	N/A
<i>RBC</i>	14R-2G-2B-5C	17R-2G-2B-7C
<i>RGBCG</i>	<b>14R-1G-2B-5C-1G</b>	N/A
<i>RBC true</i>	<b>RGBCG</b> with reordering	N/A
<i>RBC false</i>	<b>RBC</b> with reordering	N/A

#### IV. MEMORY SANDBOX: HBM MICROBENCHMARKS

Microbenchmarks are useful for evaluating specific performance metrics (i.e. bandwidth, latency, throughput) characteristics of a system while reducing simulation time. Usually, microbenchmarks are specifically hand-crafted to stress well-identified regions of a design to study its behavior and identify potential performance improvement opportunities. In this work, our microbenchmarks target HBM and are manually generated by manipulating specific bits, in the HBM Controller, to configure different scenarios. Each experiment was executed five times, and the average result is presented. As the results are highly consistent for the same conditions, no standard deviation analysis was performed.

##### A. Baseline throughput and address mapping policies impact

The first scenarios intend to stress HBM and DDR to measure how close they are to the theoretical throughput peak (bandwidth) when using our *Memory Sandbox*. To achieve this, each *Configurable Pattern Generator* is set to perform sequential accesses (*RST*) in its vertically attached pseudo-channel or bank. Moreover, outstanding transactions are enabled and burst size set to the maximum (16 and 256 beats, respectively).

Table V summarizes the different address mapping policies that we have analyzed, exploring their impact on performance for HBM and DDR4, Fig. 3(b) and Fig. 3(a) respectively. These policies correspond to the address mapping options provided by the Xilinx memory IPs (HBM and DDR) where each capital letter corresponds to specific bits. Thus, *R* corresponds to the row, *C* to the column, *B* to the bank address, *G* to the bank group bits, and *I* means interleaved. In addition to that, *true* and *false* statements correspond to the status of the reordering option and the ID features for the default policies in both memories (*RGBCG*, *RBC*). The lowest order address bits are related to the *payload width*. They are set to '0' to ensure 32 bytes and 64 bytes are accessed by the HBM and the DDR respectively. Previous works perform similar tasks without showing DDR trends, thus no comparison against HBM was reported [24], [25], [48], [49].

To measure exactly the maximum throughput supported by the memory, we effectively access all pseudo-channels and banks in parallel by using multiple *CPGs* and adding the measured per-pseudo-channel throughput to obtain the overall throughput. In contrast, other endeavors get this metric by

taking the throughput measured in only one pseudo-channel and multiplying that by the total number of pseudo-channels.

Address mapping policies microbenchmarks results are shown in Fig. 3, and from the experiments, we observe that:

- DDR achieve their best performance for sequential accesses with the *RCBI* address mapping policy; a variation of the *RCB* in the Xilinx IP that interleaved column bits taking advantage of the protocol access times to increase throughput.
- HBM Xilinx IP has 2 default and 5 custom policies. The default options use "*ID Feature*"; a mechanism that enables transaction reordering to improve performance. More specifically, it executes an out-of-order execution of transactions, which is possible by generating an identifier for each transaction and reordering those based on that identifier. Note that *RBC* and *RBC false* have the same mapping bits (*14R-2BG-2B-5C*), and throughput increment is only due to the use of the "*ID Feature*". *RBC true* and *RGBCG* also have the same mapping bits (*14R-1BG-2B-5C-1BG*), interleaving the address bits related to the Bank Group in addition to the reordering. This takes advantage of the fact that protocol access times between Banks, located in different Bank Groups, are lower than when accessing Banks within the same Bank Group.
- The best throughput for the sequential configuration is always achieved by the default policies.
- For HBM, we measure the throughput for 4 pseudo-channels and verify it is 4 times that of a single pseudo-channel throughput. Thus, we demonstrate that the HBM micro-switches are fully implemented 4x4 crossbars where the 4 pseudo-channels behavior remains consistent irrespective of the address. Moreover, the memory controller (MC) exhibits the capability to manage 2 pseudo-channels without any performance loss.
- Read and write transactions follow the same trend regardless of the policy being used. Read performance is always slightly better than write performance, as expected.
- DDR4's highest measured throughput is 6.18% lower than the maximum achievable, while for HBM is only 0.01% lower.
- DDR4's worst measured throughput is 92.02% lower than the maximum achievable, while for HBM is 56.39% lower.
- At maximum performance, each DDR4 bank delivers 19.2GB/s. In contrast, despite the HBM port is half the width of a DDR one, each pseudo-channel delivers 14.4GB/s because of their higher operation frequencies.
- When operating at the same frequency, HBM needs 4 pseudo-channels working in parallel to outperform the DDR4 throughput. From there, by adding more pseudo-channels in the HBM, the difference increases.

##### B. Exploring micro-switches cross domain

Most modern computer applications, including traditional and new HPC ones, require large amounts of memory and/or access many times to memory regions. In HBM, as each pseudo-channel has a size of 256MB, multiple pseudo-channels will likely be accessed by most applications. Xilinx warns that "*domain changes among micro-switches lead to Lateral AXI Switch Access Throughput Losses*", but it does not provide more information in this regard. To clarify the

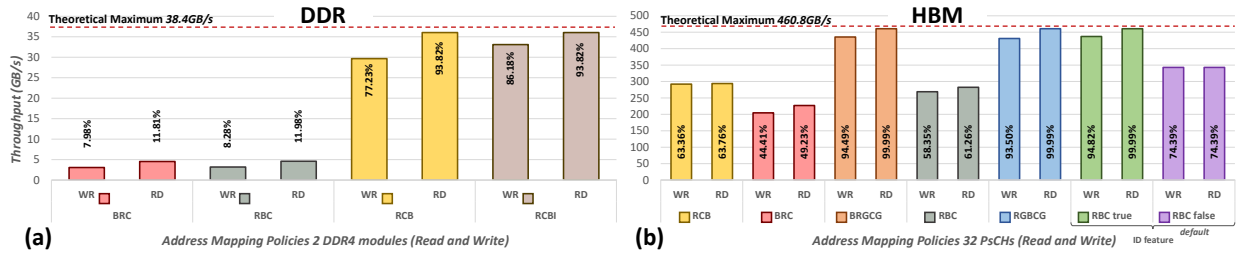


Fig. 3. Throughput Results for (a)DDR4 and (b) HBM Address Mapping Policies.

criticality of this phenomenon, we run experiments in this direction as part of the HBM exploration.

From the previous section, we know that the performance of a single pseudo-channel is the result of any address mapping policy in Fig. 3(b) divided by the total amount of pseudo-channels (32). This helps to set a baseline reference for comparison for the next experiments, where a single AXI Port within or across pseudo-channels is analyzed.

Fig. 4 illustrates the results of all the executed experiments. Each is represented with a letter (**B**, **C**, **D**, and **E**), where its explanation is detailed in the subsection equally named.

Fig. 4B shows the results of accessing different HBM pseudo-channels emulating a single-threaded processing element connected to AXI Port 0. These experiments are performed with a sequential access pattern (*RST*), a burst size of 16, which is the maximum for the HBM AXI3 version, and *RBC true* as address mapping policy, which offers the best performance for this type of access pattern according to our experiments. Two main conclusions can be drawn from these experiments shown in Fig. 4B:

- Pseudo-channels on the same micro-switch show the same performance regardless of the AXI port accessing them. Demonstrating that each micro-switch is a fully implemented 4x4 crossbar with no losses within.
- Throughput experiences an average degradation of 50% if the processing element performs memory accesses outside the pseudo-channel to which it is directly connected and, as a consequence uses another micro-switch to access another pseudo-channel's memory. This performance loss is the same for the adjacent micro-switch or the furthest one. There is no linear degradation. The performance is either the same for the 4 pseudo-channels within the same micro-switch or 50% in the other 28 pseudo-channels.

The importance of keeping HBM accesses within the same micro-switch is highlighted. Whether more memory is needed, data should be split among the pseudo-channels and accessed in parallel from another AXI port to get the maximum possible throughput. Otherwise, performance will be severely affected.

### C. Burst impact on HBM throughput

Performing an analysis based on pseudo-random accesses implies that valid data is not located in consecutive addresses, therefore the burst concept does not apply directly. Consequently, to have a fair sequential baseline for comparison against pseudo-random accesses over the next sections, we

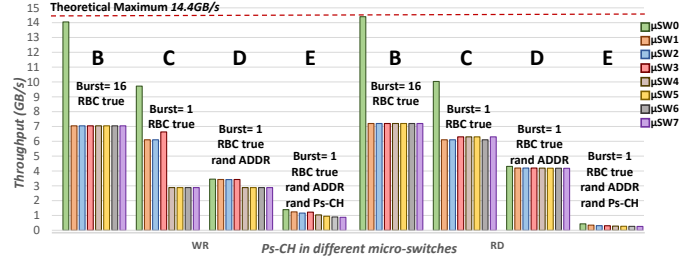


Fig. 4. Throughput Results for HBM accessing different micro-switches.

performed sequential accesses with a *burst size = 1beat = 256bits*. In Fig. 4C we illustrate these experiments results:

- No change in throughput was noticed by changing the burst size between 2 and 16. Nevertheless, using no burst at all causes a throughput decrease of 30% when the AXI Port is accessing its vertical pseudo-channel, which is the already proven best-case scenario.
- As previously, the configuration in which the AXI Port directly interfaces all pseudo-channels of a micro-switch, delivers the best throughput. The remaining pseudo-channels in each stack behave alike with an overall lower throughput of 15% compared to the experiments using burst size in Fig. 4B.
- There is a significant throughput difference between the two stacks (only for the write transactions) that is around 60% less compared to the experiments with burst size in Fig. 4B.

### D. Sparse accesses emulation: Randomizing within a pseudo-channel

HPC-domain applications include both sequential and sparse accesses. Previous sections analyzed sequential accesses. Now, the following sections explore processing threads performing sparse accesses. We emulate sparse accesses by performing pseudo-random accesses with our *Configurable Pattern Generators (CPG)*.

Our first approach, when emulating processors or accelerators with sparse patterns, was to randomize accesses *within* the same pseudo-channel. We keep the address bits related to the pseudo-channel and randomize the application address bits related to rows, columns, bank address, and bank groups; which in the case of HBM are [27:5]. Our *Memory Sandbox* eases this by just changing one parameter *Rand\_Whole\_Addr*. With this configuration, Fig. 4D shows the impact of pseudo-random accesses due to the opening and closing of bank groups and banks when accessing different columns and rows of the pseudo-channel. In this experiment, we notice that:

- For the first time, there are almost no differences between the pseudo-channels in the micro-switch to which the AXI Port is connected, and the rest within the same stack (less than  $0.5GB/s$ ). For the read transaction, there are even fewer differences and every pseudo-channel behaves almost the same (less than  $0.25GB/s$  difference).
- Moreover, for the write transactions in the first stack all the AXI Ports suffer a throughput degradation of  $44\%$  while the vertically attached one reduces its throughput by  $65\%$ . The ones in the furthest stack, on the other hand, behave very similarly to the experiments in Fig. 4C.
- Meanwhile for the read transactions, only the interfaced micro-switch behaves somehow differently and the rest have the same behavior as before.

#### E. Emulating sparse accesses: Randomizing across different pseudo-channels

In this section, we wanted to emulate processors or accelerators with sparse patterns by randomizing the address *within* the pseudo-channel as before, but also randomizing accesses *across* pseudo-channels. This is done by setting the parameters *Rand\_Whole\_Addr* and *Rand\_PSCH*.

With this configuration, we analyze the significant impact that arises when changing the pseudo-channel during pseudo-random accesses, altering every bit in the address. The idea was to emulate a single thread processor that performs accesses starting in its vertical pseudo-channel and add micro-switches to randomize for each new experiment e.g., the first experiment randomizes among 4 pseudo-channels, the second one among 8 and up to the 32 pseudo-channels.

Fig. 4E performance is the worst measured until now. It decreases to  $0.61\%$  of the best-case scenario measured for the write transactions and  $0.17\%$  for the read ones.

We are aware that some of these experiments are quite extensive and that there are only slight probabilities that, for example, an accelerator always accesses different pseudo-channels. Nevertheless, the goal is to generate baselines for future comparison and for other developers to have a starting point to compare with. In fact, our *Memory Sandbox* is provided as part of the *FPGA Shell* [50] to provide a software development and experimentation platform to enable software readiness for new hardware.

#### F. Emulating simultaneous accesses to a pseudo-channel

In previous experiments, the emulated architecture is based on a *single thread processing element*. Nevertheless, in modern accelerators, based on *multi-core heterogeneous architectures*, several processing elements access memory in parallel and simultaneously. The information presented in previous sections applies to any core or accelerator in this heterogeneous system. However, this section explores what happens when several of those processing elements target the same memory region.

With this purpose, we designed a set of experiments in which multiple *Configurable Pattern Generators* emulate different processing elements. The impact on throughput is analyzed while adding more *CPGs* interfacing different AXI

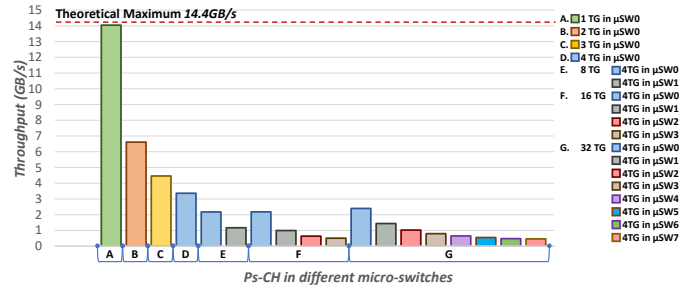


Fig. 5. Throughput Results for several AXI Ports accessing PSCHO.

Ports, but always addressing the same pseudo-channel 0. As described before, the global addressing mode is activated to enable this behavior. For this experiment, we increased the probability of creating collisions in memory by using sequential access to the same memory address. Fig. 5 shows, with different letters, the huge impact that simultaneous accesses might have in HBM by performing several experiments with different amounts of pseudo-channels:

- All processing elements connected to any AXI Port in the same micro-switch behave similarly, as before.
- Nonetheless, within the same micro-switch, when going from one to two processing elements, the throughput for each of those two is reduced by  $50\%$  (Fig. 5B). When going from two to three, all of them get a throughput reduction of  $32\%$  (Fig. 5C) and from 3 to 4, the 4 of them get a throughput reduction of  $25\%$  (Fig. 5D). According to our measurements, the aggregate throughput of all the pseudo-channels in each experiment is the same as that of only one processing element.
- The three rightmost sets of bars are for 8 (Fig. 5E), 16 (Fig. 5F), and 32 (Fig. 5G) processing elements in different micro-switches. Across these three experiments, we observe that no matter the number of processing elements, those in the same micro-switch behave similarly i.e AXI Ports in the first micro-switch have the same throughput for 8, 16, and 32. This is explained by understanding that the AXI Ports inputs in micro-switches form an already proven  $4 \times 4$  crossbar with priority over those from a lateral micro-switch. According to our measurements, each *CPG* throughput decreases with any additional processing element added while the aggregate throughput (total sum of the throughput per pseudo-channel) remains the same, equal to the throughput obtained with one processing element.

## V. MEMORY SANDBOX: REAL-TIME APPLICATIONS (MIMIC MEMORY ACCESS PATTERNS)

To our *Memory Sandbox* functionality and validate our results, we have added the capability of emulating real-wise applications, by mimicking their memory access patterns and measuring the potential throughput when they execute. As a representative of a real sparse memory access pattern, we have selected the Sparse Matrix-Vector (SpMV) [21] multiplication kernel of the High-Performance Conjugate Gradient (HPCG) benchmark [51]. The experiment follows the memory access pattern exhibited by SpMV and compares the obtained throughput with the resultant from the ideal cases (previ-

ous sections). Adding this feature to the Memory Sandbox moves further from traditional microbenchmarking, to enable exploration of memory access models to predict the resultant performance for a selected kernel, benchmark, or application.

The High-Performance Conjugate Gradient (HPCG) benchmark [51] is a tool for ranking computer systems, similar and complementary in its purpose to High-Performance LINPACK (HPL) benchmark [52], currently used to rank systems as part of the TOP500 project benchmark [53]. While computational and data access patterns of HPL are still relevant for many scalable applications, they do not cover the entire spectrum. To bridge this gap, HPCG has been introduced, since its computational and data access patterns reflect a different and diverse range of critical applications characterized by sparse memory access behaviors.

In its operation, HPCG generates a structured sparse linear system, mathematically close to a finite element, finite volume, or finite difference discretization of a three-dimensional equation. The problem is tackled using domain decomposition [4], employing a conjugate gradient method with an additive Schwarz preconditioner. Subsequently, each subdomain undergoes further preconditioning via a symmetric Gauss-Seidel sweep. HPCG reference implementation is a comprehensive and self-contained code developed in C++ with MPI and OpenMP support. Implemented through a multigrid preconditioned conjugate gradient algorithm, exercising key kernels on nested sets of coarse grids. The unified framework comprises the following applications [54]:

- **Sparse Matrix-Vector multiplication.**
- Vector updates.
- Global dot products.
- Local symmetric Gauss-Seidel smoother.
- Sparse triangular solve (Gauss-Seidel smoother part).

Sparse Matrix-Vector multiplication (SpMV) is a fundamental kernel in the HPCG benchmark due to its critical role in a diverse array of applications. It is employed in scientific computing, such as linear algebra solvers [55], artificial intelligence applications like sparse neural networks [56], and graph processing [57]. Its significance is demonstrated by its inclusion among the seven dwarfs of parallel computing research, specifically in the domain of Sparse Linear Algebra.

SpMV presents a memory access pattern characterized by random accesses; one of the main reasons why traditional architectures have a limiting performance. This poses a significant efficiency obstacle, particularly for FPGAs that do not have a default memory hierarchy with caches to mitigate the latency of main memory. Achieving high-performance metrics when running SpMV on FPGAs is notably intricate. However, FPGAs' capacity to customize the memory hierarchy offers an attractive advantage over conventional architectures, especially for large matrices where caching effectiveness might be limited without additional cache optimizations.

Mathematically, SpMV is expressed as the multiplication of a 2D sparse matrix, characterized by having a majority of zero elements, by a dense vector with non-zero elements. When working with sparse matrices, substantial reductions in

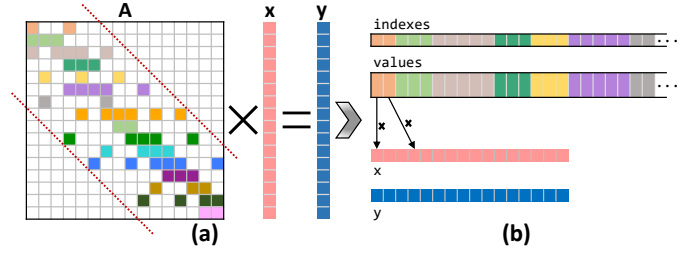


Fig. 6. (a) SpMV algebra conceptualization and (b) CSR data reorganization.

memory requirements can be achieved by storing only the non-zero entries. Depending on the quantity and distribution of non-zero entries, various data structures can be employed, resulting in significant memory savings compared to the basic approach [58]. However, the counterpart is that accessing each element becomes more complex, requiring additional structures to recover the original matrix. These formats can be broadly categorized into two groups:

- Formats that facilitate efficient modification: DOK (Dictionary of Keys), LIL (List of Lists), or COO (Coordinate List).
- Formats with efficient access and matrix operations support: Bitmap, CSB (Compressed Sparse Blocks), CSC (Compressed Sparse Column), or **CSR** (Compressed Sparse Row).

Efforts have been dedicated to improving SpMV performance by modifying these encoding existing formats, proposing alternative formats, or with new architectures. For instance, in [59], a task-based approach is applied to enhance their b8c format performance. Other initiatives, such as [60], integrate compute logic near memory banks to exploit bank-level bandwidth. Some endeavors have even migrated to different memory architectures, like HMCSPP [61], leveraging the Micron Hybrid Memory Cube (HMC) atomic operations with processing in memory (PIM) capabilities to mitigate the memory transaction latency of SpMV.

In Figure 6, we illustrate a matrix sub-block of size  $16 \times 16$ , which serves as an example of a SpMV and its representation in CSR encoding. This encoding corresponds to the application from which we obtained the access pattern under evaluation. In the CSR format, the 2D sparse matrix is stored in two dense vectors. One of them contains only the '*indexes*' and the other the '*values*' of the non-zero elements. The dense vector '*x*' to be multiplied is then accessed in a sparse form and the multiplication result is stored in a dense vector '*y*'.

In simple terms, as depicted in Fig. 6(b), there is a dense address access stream for '*indexes*', forming the basis for the irregular address access stream for '*x*'. While these two sets of accesses are ongoing, a third regular stream for '*values*' is also accessed. After completing all accesses, the benchmark executes a single write operation to store the result of the operation in '*y*' and proceeds to the next set of data accesses.

We aim to assess the SpMV access pattern performance. Given the complexity of implementing an actual solution in FPGAs, we instrument the SpMV HPCG benchmark to capture a real access pattern (memory addresses, and execution order). This instrumented version is compiled with support for OpenMP parallelism. The output is a trace with the list

of memory accesses and their execution order. Overall, the algorithm generates a data flow structure, as depicted in Fig. 7.

The size and the non-zero distribution of the matrix elements determine the size of the trace. In most cases, resource limitations in FPGAs make it impossible to reproduce the experiment while holding the data on chip. As a solution, we opted to inject the trace information as an algorithm able to accurately represent the kernel within our *Memory Sandbox*.

In practice this meant adapting our *Configurable Generator Pattern* to handle four pseudo-channels in parallel, assigning each one of the Fig. 7 address streams (*'indexes'*, *'values'*, *'x'* and *'y'*) to a different Finite State Machine (FSM). FSMs handle HBM pseudo-channels and synchronize those to be compliant with the kernel trace. Our implementation accurately generates 3,118,454 addresses out of the 3,237,104 original trace total when modeling SpMV, achieving a model fidelity of 96.3%. Under these conditions, we ran different experiments modifying the number of *CPGs* executing the SpMV trace. We measured the SpMV trace execution and obtained a **3.6GB/s** throughput for a single *CPGs* and for up to eight *CPGs* the performance scales up to **28.8GB/s**. Based on these experiments, we formulate the following conjectures:

- The performance observed with 8 *Configurable Pattern Generators* experiment is eight times that of the single generator experiment, as we previously optimized access distribution to eliminate lateral losses and prevent concurrent access to the same memory regions. This highlights the critical need for meticulous management of concurrent accesses and strategic data placement when using HBM.
- Figure 7 illustrates that, due to the duration of the sparse *'x'* FSM, the behavior of both *'indexes'* and *'values'* FSMs becomes "hidden". Consequently, the entire SpMV behaves as the sparse *'x'* FSM.
- Performance results for this SpMV application validate the ones obtained during the HBM microbenchmarking. Referring to Figure 4, SpMV access pattern profile corresponds to the case *C* of the *RD* side. In the ideal case a bandwidth of 4.2GB/s is expected. However, the achieved 3.6GB/s by our instrumented SpMV HPCG benchmark corresponds to **85.7%** of the maximum achievable throughput. Synchronization among the 4 FSMs and the requirement of waiting for the write operation before continuing with the next chunk explains this performance drop. These insights help designers understand that advertised high performances, such as the 460.8GB/s, might not be achievable because it needs to use all 32 pseudo-channel access. Similarly, the maximum performance of 14.3GB/s for 1 pseudo-channel will not be achieved because

it requires dense access that can also utilize bursts.

- We demonstrate that designers, by profiling their applications and understanding their access patterns based on sequential or random behavior, burst possibilities, and other considerations like staying in the same micro-switch or incurring in lateral losses due to movement to another memory region, along with the other parameters described in the previous section, can refer to Figures 4 and 5 to obtain a close-to-real throughput prediction for their application before design it.
- If none of the behaviors displayed in Figures 4 and 5 describe the user's application; our *Memory Sandbox* configurable pattern generators, can be tailored using the multiple parameters we previously described to closely resemble diverse designs. This allows obtaining real achievable throughput predictions for their application before designing it.

## VI. RELATED WORK

To the best of our knowledge, our work is the first one that profiles HBM on FPGAs intending to create a comprehensive baseline for HPC developers including accesses across pseudo-channels, concurrent access and a real application validation. Table VI lists the related works and their characteristics compared to our *Memory Sandbox*. To understand the novelty of our paper, we broadly classify the related work from the recent literature into four categories:

- DDR-based application acceleration on FPGAs.
- Benchmarking DDRs on FPGAs.
- HBM-based application acceleration on FPGAs.
- Benchmarking HBM on FPGAs.

Efforts using FPGAs to improve a plethora of different applications can be easily found in the bibliography [64]–[71] as FPGAs have become an attractive alternative to GPUs when it comes to optimizing Heterogeneous Computing systems. On the other hand, our work goal is to provide a baseline and a platform for those developing applications to set the expectations for HBM in FPGAs.

Works related to benchmarking commonly used DDRs on FPGAs such as [72]–[77] concentrating their efforts on HLS. In contrast, we also use FPGAs, but to evaluate the recently released HBM. In addition, for this purpose, we use RTL to get closer to the actual hardware signals and have more control.

Endeavors like the ones presented in [78]–[81], do not profile HBM. Instead, they use it to enhance their applications' performance. Some also used different architectures than our targeted one (Xilinx), the Intel Knights Landing.

Endeavors like [48], [49], [62], [63] also perform an interesting profile of HBM but with a different approach which is based on High-Level Synthesis (HLS) language and taking HBM as a whole while exploring the impact of features different to ours such as the different Super Logic Regions (SLR), design space exploration and power consumption.

In [24], [25] they develop a platform to deeply benchmark a single HBM pseudo-channel on FPGAs. Moreover, they focus on sequential accesses with stride because they target common applications. In contrast, our paper evaluates the impact on HBM performance of both sequential and pseudo-random

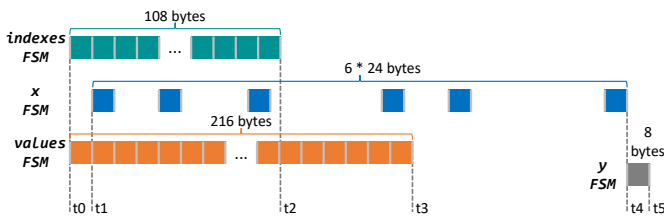


Fig. 7. Data requests and flow while accessing HBM with the SpMV product.

TABLE VI  
COMPARISON OF OUR *Memory Sandbox* WITH THE STATE OF THE ART SOLUTIONS

	HLS based	RTL based	DDR+FPGA acceleration	DDR+FPGA profiling	HBM+FPGA acceleration	HBM+FPGA profiling						
						Latency	Within PSCHs	Across PSCHs	Concurrent PSCHs	RST	Pseudo-random	Real access pattern
[24], [25]	✓	□	□	□	□	✓	✓	□	□	✓	□	□
[48], [49], [62], [63]	✓	□	□	□	□	✓	□	□	□	✓	□	□
[64]–[71]	✓	✓	✓	□	□	□	□	□	□	□	□	□
[72]–[77]	✓	□	□	✓	□	✓	□	□	□	✓	✓	□
[78]–[81]	✓	✓	□	□	✓	□	□	□	□	□	□	□
[26]	✓	□	□	□	□	✓	✓	✓	□	✓	✓	□
Memory Sandbox	□	✓	□	✓	□	✓	✓	✓	✓	✓	✓	✓

accesses using our *Memory Sandbox* tool. Furthermore, our work is not only focused on a single HBM pseudo-channel but mainly on what happens across pseudo-channels and micro-switches. Moreover, we perform a novel analysis of simultaneous accesses to the same memory region.

The closest endeavors in comparison to ours is [26] endeavor where they take like us a thorough approach to profile HBM. Nevertheless, our solution, being RTL-based, gives more control over signals, shows concurrent access behavior, and enables real access pattern evaluation.

## VII. CONCLUSIONS

When FPGAs are used to address HPC applications, but their resources are not properly configured, the resulting implementations can underperform quite significantly. This is especially important regarding memories, as FPGAs do not have a preconfigured and tested cache hierarchy like microprocessors or GPUs. HBM appears as a solution being integrated into FPGAs to face the memory wall issue and large companies are already committed to its use. As expected, the throughput performance was more than 12 times better when using all 32 pseudo-channels in the HBM in parallel than when using the 2 memory banks in the DDR. The different address mapping policies, the burst size, accesses within a micro-switch or external ones, and the randomization of the address can have a huge impact on the HBM throughput.

Our *Memory Sandbox* enables software readiness for new hardware and can emulate a diverse set of architectures with different processing threads and access patterns. The analysis presented makes an important contribution to the state of the art regarding the impact on the HBM performance of pseudo-random accesses across and within the pseudo-channels and the concurrent access to the same memory region. More importantly, our *Memory Sandbox* tool is not only valid or restricted to the provided test scenarios, designers and researchers can infer combinations from our scenarios but also create their own. Giving a powerful test architecture for HPC on FPGAs developers who need these patterns in a daily basis.

In future work, we plan to remove MicroBlaze to make it easier to use and to improve memory hierarchies used on RISC-V processors synthesized on these FPGAs with the information learned from our experiments. Also, we are already running these experiments in our FPGA cluster.

## REFERENCES

[1] Mohamed Zahran, “Different players,” in *Heterogeneous Computing: Hardware and Software Perspectives*. New York, NY, USA:

Association for Computing Machinery, 2019, vol. 23. [Online]. Available: <https://doi.org/10.1145/3281649.3281652>

[2] E. S. Chung, P. A. Milder, J. C. Hoe, and K. Mai, “Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?” in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, Dec. 2010, pp. 225–236, iSSN: 2379-3155. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/5695539>

[3] Todd Carl Mowry, “Tolerating latency through software-controlled data prefetching,” PhD Thesis, Stanford University, Mar. 1994.

[4] B. Betkaoui, D. B. Thomas, and W. Luk, “Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing,” in *2010 International Conference on Field-Programmable Technology*, Dec. 2010, pp. 94–101. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/5681761>

[5] J. Cong, Z. Fang, M. Lo, H. Wang, J. Xu, and S. Zhang, “Understanding Performance Differences of FPGAs and GPUs,” in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Apr. 2018, pp. 93–96, iSSN: 2576-2621. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8457638>

[6] I. Kuon and J. Rose, “Measuring the gap between FPGAs and ASICs,” in *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, ser. FPGA ’06. New York, NY, USA: Association for Computing Machinery, Feb. 2006, pp. 21–30. [Online]. Available: <https://dl.acm.org/doi/10.1145/1117201.1117205>

[7] M. Véstias and H. Neto, “Trends of CPU, GPU and FPGA for high-performance computing,” in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2014, pp. 1–6, iSSN: 1946-1488. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/6927483>

[8] Google, “Tensor Processing Units (TPUs).” [Online]. Available: <https://cloud.google.com/tpu>

[9] A. Shahid and M. Mushtaq, “A Survey Comparing Specialized Hardware And Evolution In TPUs For Neural Networks,” in *2020 IEEE 23rd International Multitopic Conference (INMIC)*, Nov. 2020, pp. 1–6, iSSN: 2049-3630. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9318136>

[10] T. Sueyoshi, “Basic Knowledge to Understand FPGAs,” in *Principles and Structures of FPGAs*, H. Amano, Ed. Singapore: Springer, 2018, pp. 1–22. [Online]. Available: [https://doi.org/10.1007/978-981-13-0824-6\\_1](https://doi.org/10.1007/978-981-13-0824-6_1)

[11] A. Boutros and V. Betz, “Field-Programmable Gate Array Architecture,” in *Handbook of Computer Architecture*, A. Chattopadhyay, Ed. Singapore: Springer Nature, 2022, pp. 1–47. [Online]. Available: [https://doi.org/10.1007/978-981-15-6401-7\\_49-1](https://doi.org/10.1007/978-981-15-6401-7_49-1)

[12] Copyright © 2022 Samsung ALL rights reserved, “Next-level performance,” Samsung HBM, Tech. Rep., 2022. [Online]. Available: <https://www.samsung.com/semiconductor/dram/hbm/>

[13] M. Ujaldón, “HPC Accelerators with 3D Memory,” *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*, pp. 320–328, Aug. 2016, doi: 10.1109/CSE-EUC-DCABES.2016.203.

[14] Copyright © 2022 NVIDIA Corporation, “NVIDIA® V100 Tensor Core GPU,” NVIDIA Corporation, Tech. Rep., 2022. [Online]. Available: <https://www.nvidia.com/en-us/data-center/v100/>

[15] Xilinx Inc., “Xilinx Extends Data Center Leadership with New Alveo U280 HBM2 Accelerator Card,” Xilinx Inc., Tech. Rep., Nov. 2018. [Online]. Available: <https://www.xilinx.com/news/press/2018/xilinx-ext>

- ends-data-center-leadership-with-new-alveo-u280-hbm2-accelerator-card-dell-emc-first-to-qualify-alveo-u200.html
- [16] Xilinx Inc., "Xilinx Launches Alveo U55C, Its Most Powerful Accelerator Card Ever, Purpose-Built for HPC and Big Data Workloads," Xilinx Inc., Tech. Rep., Nov. 2021. [Online]. Available: <https://www.xilinx.com/news/press/2021/xilinx-launches-alveo-u55c-its-most-powerful-accelerator-card-ever-purpose-built-for-hpc-and-big-data-workloads.html>
- [17] AMD Xilinx, "Alveo V80 Data Center Accelerator Card Data Sheet," Tech. Rep. DS1013, May 2024. [Online]. Available: <https://docs.amd.com/r/en-US/ds1013-v80/Summary>
- [18] Ehab Mohsen, "Unleashing Computational Power with the New AMD Alveo™ V80 Compute Accelerator," May 2024, section: Adaptive Computing. [Online]. Available: <https://community.amd.com/t5/adaptive-e-computing/unleashing-computational-power-with-the-new-amd-alveo-v80/ba-p/683882>
- [19] Xilinx Inc., "ALVEO™ Portfolio Product Selection Guide by Xilinx Inc. XMP451 (v2)," May 2024. [Online]. Available: <https://docs.amd.com/v/en-US/alveo-product-selection-guide>
- [20] Absolute Reports, "Global High-performance Computing (HPC) Market 2023 Size and Emerging Trends that Value Expected to Reach USD 52731.44 Mn | Growing at CAGR of 8.19% | Forecast to 2027," Absolute Reports, Tech. Rep., Dec. 2022. [Online]. Available: [https://www.theexpresswire.com/pressrelease/Global-High-performance-Computing-HPC-Market-2023-Size-and-Emerging-Trends-that-Value-Expected-to-Rreach-USD-5273144-Mn-Growing-at-CAGR-of-8-19-Forecast-to-2027\\_18527995](https://www.theexpresswire.com/pressrelease/Global-High-performance-Computing-HPC-Market-2023-Size-and-Emerging-Trends-that-Value-Expected-to-Rreach-USD-5273144-Mn-Growing-at-CAGR-of-8-19-Forecast-to-2027_18527995)
- [21] P. Zardoshti, F. Khunjush, and H. Sarbazi-Azad, "Chapter 14 - Adaptive sparse matrix representation for efficient matrix-vector multiplication," in *Advances in GPU Research and Practice*, ser. Emerging Trends in Computer Science and Applied Computing, H. Sarbazi-Azad, Ed. Boston: Morgan Kaufmann, Jan. 2017, pp. 349–369. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/B9780128037386000148>
- [22] J. Dongarra, M. A. Heroux, and P. Luszczyk, "High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems," *The International Journal of High Performance Computing Applications*, vol. 30, no. 1, pp. 3–10, 2016, doi: 10.1177/1094342015593158.
- [23] J. Dongarra and P. Luszczyk, "TOP500," in *Encyclopedia of Parallel Computing*, David Padua, Ed. Boston, MA: Springer US, 2011, pp. 2055–2057, doi: 10.1007/978-0-387-09766-4\_157.
- [24] Z. Wang, H. Huang, J. Zhang, and G. Alonso, "Shuhai: Benchmarking High Bandwidth Memory On FPGAs," in *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2020, pp. 111–119, doi: 10.1109/FCCM48280.2020.00024.
- [25] H. Huang, Z. Wang, J. Zhang, Z. He, C. Wu, J. Xiao, and G. Alonso, "Shuhai: A Tool for Benchmarking High Bandwidth Memory on FPGAs," *IEEE Transactions on Computers*, vol. 71, no. 5, pp. 1133–1144, May 2022, doi: 10.1109/TC.2021.3075765.
- [26] P. Holzinger, D. Reiser, T. Hahn, and M. Reichenbach, "Fast HBM Access with FPGAs: Analysis, Architectures, and Applications," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Jun. 2021, pp. 152–159. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9460576>
- [27] C. Kim, H.-W. Lee, and J. Song, *High-Bandwidth Memory Interface*, 1st ed., ser. SpringerBriefs in Electrical and Computer Engineering. Springer Science & Business Media, 2013, doi: 10.1007/978-3-319-02381-6.
- [28] J. Macri, "AMD's next generation GPU and high bandwidth memory architecture: FURY," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, 2015, pp. 1–26, doi: 10.1109/HOTCHIPS.2015.7477461.
- [29] K. Cho, H. Lee, H. Kim, S. Choi, Y. Kim, J. Lim, J. Kim, H. Kim, Y. Kim, and Y. Kim, "Design optimization of high bandwidth memory (HBM) interposer considering signal integrity," in *2015 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS)*, 2015, pp. 15–18, doi: 10.1109/EDAPS.2015.7383697.
- [30] H. Jun, C. Jinhee, L. Kangseol, H.-Y. Son, K. Kwiwook, J. Hanho, and K. Kim, "HBM (High Bandwidth Memory) DRAM Technology and Architecture," in *2017 IEEE International Memory Workshop (IMW)*, 2017, pp. 1–4, doi: 10.1109/IMW.2017.7939084.
- [31] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 41–54.
- [32] S. Li, D. Reddy, and B. Jacob, "A performance & power comparison of modern high-speed dram architectures," in *Proceedings of the International Symposium on Memory Systems*, 2018, pp. 341–353.
- [33] J. Kim and Y. Kim, "Hbm: Memory solution for bandwidth-hungry processors," in *2014 IEEE Hot Chips 26 Symposium (HCS)*, 2014, pp. 1–24.
- [34] JEDEC, "JEDEC Updates Groundbreaking High Bandwidth Memory (HBM) Standard | JEDEC," Dec. 2018. [Online]. Available: <https://www.jedec.org/news/pressreleases/jedec-updates-groundbreaking-high-bandwidth-memory-hbm-standard-0>
- [35] —, "HIGH BANDWIDTH MEMORY (HBM) DRAM | JEDEC," Mar. 2021. [Online]. Available: <https://www.jedec.org/standards-documents/docs/jesd235a>
- [36] —, "JEDEC Publishes HBM3 Update to High Bandwidth Memory (HBM) Standard | JEDEC," Jan. 2022. [Online]. Available: <https://www.jedec.org/news/pressreleases/jedec-publishes-hbm3-update-high-bandwidth-memory-hbm-standard>
- [37] —, "HIGH BANDWIDTH MEMORY (HBM3) DRAM | JEDEC," Jan. 2023. [Online]. Available: <https://www.jedec.org/standards-documents/docs/jesd238a>
- [38] R. Smith, "Samsung Announces 'Shinebolt' HBM3E Memory: HBM Hits 36GB Stacks at 9.8 Gbps," Oct. 2023. [Online]. Available: <https://www.anandtech.com/show/21104/samsung-announces-shinebolt-hbm3e-memory-hbm-hits-36gb-stacks-at-9-8-gbps>
- [39] T. Kim, C. Jo, and S. Moon, "Signal Integrity(SI) aware HBM2e/3 interposer design approach considering y-axis offset between logic and HBM die for HPC/AI/Network applications," in *2021 IEEE 71st Electronic Components and Technology Conference (ECTC)*, Jun. 2021, pp. 1270–1275, doi: 10.1109/ECTC32696.2021.00206.
- [40] AMD Xilinx Inc., "Alveo U280 Data Center Accelerator Card Data Sheet," Xilinx Inc., Tech. Rep. DS963, 2022. [Online]. Available: [https://www.xilinx.com/content/dam/xilinx/support/documents/data\\_sheets/ds963-u280.pdf](https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds963-u280.pdf)
- [41] Xilinx Inc., "ALVEO™ Product Selection Guide Datasheet," Xilinx Inc., Tech. Rep. XMP451 (v1.7), 2021. [Online]. Available: <https://www.xilinx.com/support/documentation/selection-guides/alveo-product-selection-guide.pdf>
- [42] AMD Xilinx Inc., "UltraScale Architecture-Based FPGAs Memory IP v1.4 LogiCORE IP Product Guide," Xilinx Inc., Tech. Rep. PG150, Apr. 2022. [Online]. Available: <https://docs.xilinx.com/v/en-US/pg150-ultrascale-memory-ip>
- [43] AMD Xilinx Inc., "Alveo U55C Data Center Accelerator Cards Data Sheet," Xilinx Inc., Tech. Rep. DS978, 2022. [Online]. Available: [https://www.xilinx.com/content/dam/xilinx/support/documents/data\\_sheets/ds978-u55c.pdf](https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds978-u55c.pdf)
- [44] Xilinx Inc., "AXI High Bandwidth Memory Controller v1.0 LogiCORE IP Product Guide," Xilinx Inc., Tech. Rep. PG276 (v1.0), Aug. 2021. [Online]. Available: [https://www.xilinx.com/support/documentation/ip\\_documentation/hbm/v1\\_0/pg276-axi-hbm.pdf](https://www.xilinx.com/support/documentation/ip_documentation/hbm/v1_0/pg276-axi-hbm.pdf)
- [45] Micron Technology Inc., "DDR4 SDRAM RDIMM MTA18ASF2G72PZ- 16GB (x72, ECC, SR) 288-Pin Features Rev D 8/16," Micron Technology Inc., Tech. Rep., 2015. [Online]. Available: <https://www.micron-semiconductor.com/datasheet/7c-MTA18ASF2G72PZ-2G9E1.pdf>
- [46] ARM Corporation, "AMBA AXI and ACE Protocol Specification AXI3, AXI4, and AXI4-Lite ACE and ACE-Lite," ARM Corporation, Tech. Rep. ID102711, 2011.
- [47] Xilinx Inc., "Vivado Design Suite: AXI Reference Guide," Xilinx Inc., Tech. Rep. UG1037, Jul. 2017. [Online]. Available: <https://docs.xilinx.com/v/en-US/ug1037-vivado-axi-reference-guide>
- [48] Y.-K. Choi, Y. Chi, W. Qiao, N. Samardzic, and J. Cong, "HBM Connect: High-Performance HLS Interconnect for FPGA HBM," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '21. New York, NY, USA: Association for Computing Machinery, Feb. 2021, pp. 116–126, event-place: Virtual Event, USA, doi: 10.1145/3431920.3439301.
- [49] R. Shi, K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, and G. Alonso, "Exploiting HBM on FPGAs for data processing," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 4, pp. 1–27, Dec. 2022, publisher: ACM New York, NY, US, doi: 10.1145/3491238.

- [50] E. Perdomo, A. Kropotov, F. K. Cano Ladino, S. Zafar, T. Cervero, X. M. Bofill, and B. Salami, "Makinote: An fpga-based hw/sw platform for pre-silicon emulation of risc-v designs," in *Proceedings of the 16th Workshop on Rapid Simulation and Performance Evaluation for Design*, ser. RAPIDO '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 29–34. [Online]. Available: <https://doi.org/10.1145/3642921.3642928>
- [51] J. Dongarra, M. A. Heroux, and P. Luszczek, "High-performance conjugate gradient benchmark: A new metric for ranking high-performance computing systems," *The Intl Journal of High Performance Computing Applications*, vol. 30, no. 1, pp. 3–10, 2016, doi: 10.1177/1094342015593158.
- [52] J. Dongarra, P. Luszczek, and A. Petitet, "The linpack benchmark: Past, present, and future," *Concurrency: Practice and Experience*, vol. 15, pp. 803–820, 2008.
- [53] E. Strohmaier, J. Dongarra, H. Simon, M. Meuer, and H. Meuer, "The top500 lists." [Online]. Available: <https://www.top500.org>
- [54] "HPCG Benchmark." [Online]. Available: <https://www.hpcg-benchmark.org/>
- [55] Y. Saad, "Iterative methods for sparse linear systems," *Society for Industrial and Applied Mathematics*, Jan. 2003.
- [56] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Proceedings of the 28th International Conference on Neural Information Processing Systems*, vol. 1, pp. pp. 1135–1143, Dec. 2015.
- [57] J. Kepner, P. Aaltonen, D. Bader, A. Buluç, F. Franchetti, J. Gilbert, D. Hutchison, M. Kumar, A. Lumsdaine, H. Meyerhenke, S. McMillan, C. Yang, J. D. Owens, M. Zalewski, T. Mattson, and J. Moreira, "Mathematical foundations of the GraphBLAS," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2016, pp. 1–9. [Online]. Available: <https://ieeexplore.ieee.org/document/7761646>
- [58] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, *Templates for the solution of algebraic eigenvalue problems: a practical guide*. Society for Industrial and Applied Mathematics, Jan. 2000, vol. 11.
- [59] J. Oliver, C. Álvarez, T. Cervero, X. Martorell, J. D. Davis, and E. Ayguadé, "Accelerating SpMV on FPGAs Through Block-Row Compress: A Task-Based Approach," in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. Gothenburg, Sweden: IEEE, Sep. 2023, pp. 151–158. [Online]. Available: <https://ieeexplore.ieee.org/document/102963571>
- [60] X. Xie, Z. Liang, P. Gu, A. Basak, L. Deng, L. Liang, X. Hu, and Y. Xie, "SpaceA: Sparse Matrix Vector Multiplication on Processing-in-Memory Accelerator," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2021, pp. 570–583, iSSN: 2378-203X. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/9407163>
- [61] C. Qian, B. Childers, L. Huang, Q. Yu, and Z. Wang, "HMCSP: Reducing Transaction Latency of CSR-based SPMV in Hybrid Memory Cube," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2018, pp. 114–116. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/8366942>
- [62] A. Olgun, M. Osseiran, A. G. Yağlıkcı, Y. C. Tuğrul, H. Luo, S. Rhyner, B. Salami, J. G. Luna, and O. Mutlu, "Read disturbance in high bandwidth memory: A detailed experimental study on hbm2 dram chips," in *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2024, pp. 75–89.
- [63] S. S. N. Larimi, B. Salami, O. S. Unsal, A. C. Kestelman, H. Sarbazi-Azad, and O. Mutlu, "Understanding power consumption and reliability of high-bandwidth memory with voltage undervolting," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 517–522.
- [64] J. Fowers, K. Ovcharov, K. Strauss, E. S. Chung, and G. Stitt, "A High Memory Bandwidth FPGA Accelerator for Sparse Matrix-Vector Multiplication," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2014, pp. 36–43, doi: 10.1109/FCCM.2014.23.
- [65] D. Gschwend, "ZynqNet: An FPGA-Accelerated Embedded Convolutional Neural Network," *ArXiv preprint arXiv:2005.06892*, May 2020, doi: 10.48550/arXiv.2005.06892.
- [66] H. Zeng and V. Prasanna, "GraphACT: Accelerating GCN Training on CPU-FPGA Heterogeneous Platforms," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 255–265, event-place: Seaside, CA, USA, doi: 10.1145/3373087.3375312.
- [67] H. Chen, S. Madaminov, M. Ferdman, and P. Milder, "FPGA-Accelerated Samplesort for Large Data Sets," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 222–232, event-place: Seaside, CA, USA, doi: 10.1145/3373087.3375304.
- [68] J. Li, Y. Chi, and J. Cong, "HeteroHalide: From Image Processing DSL to Efficient FPGA Acceleration," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 51–57, event-place: Seaside, CA, USA, doi: 10.1145/3373087.3375320.
- [69] J. Hoozemans, J. Peltenburg, F. Nonnemacher, A. Hadnagy, Z. Al-Ars, and H. P. Hofstee, "FPGA Acceleration for Big Data Analytics: Challenges and Opportunities," *IEEE Circuits and Systems Magazine*, vol. 21, no. 2, pp. 30–47, 2021, doi: 10.1109/MCAS.2021.3071608.
- [70] A. Olgun, J. G. Luna, K. Kanellopoulos, B. Salami, H. Hassan, O. Ergin, and O. Mutlu, "PiDRAM: A Holistic End-to-end FPGA-based Framework for Processing-in-DRAM," *ACM Transactions on Architecture and Code Optimization*, vol. 20, no. 1, pp. 1–31, Mar. 2023. [Online]. Available: <https://dl.acm.org/doi/10.1145/3563697>
- [71] F. Minervini, O. Palomar, O. Unsal, E. Reggiani, J. Quiroga, J. Marimon, C. Rojas, R. Figueras, A. Ruiz, A. Gonzalez, J. Mendoza, I. Vargas, C. Hernandez, J. Cabre, L. Khoirunissa, M. Bouhali, J. Pavon, F. Moll, M. Olivieri, M. Kovac, M. Kovac, L. Dragic, M. Valero, and A. Cristal, "Vitruvius+: An Area-Efficient RISC-V Decoupled Vector Coprocessor for High Performance Computing Applications," *ACM Transactions on Architecture and Code Optimization*, vol. 20, no. 2, pp. 1–25, Jun. 2023. [Online]. Available: <https://dl.acm.org/doi/10.1145/3575861>
- [72] Altera Corporation, "Guidance for Accurately Benchmarking FPGAs, ver. 1.2," Altera Corporation, Tech. Rep. WP-01040-1.2, Dec. 2007.
- [73] Q. Gautier, A. Althoff, P. Meng, and R. Kastner, "Spector: An OpenCL FPGA benchmark suite," in *2016 International Conference on Field-Programmable Technology (FPT)*, Dec. 2016, pp. 141–148, doi: 10.1109/FPT.2016.7929519.
- [74] H. R. Zohouri and S. Matsuoka, "The Memory Controller Wall: Benchmarking the Intel FPGA SDK for OpenCL Memory Interface," in *2019 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, Nov. 2019, pp. 11–18, doi: 10.1109/H2RC49586.2019.00007.
- [75] Y.-K. Choi, Y. Chi, J. Wang, L. Guo, and J. Cong, "When HLS Meets FPGA HBM: Benchmarking and Bandwidth Optimization," *ArXiv preprint arXiv:2010.06075*, Oct. 2020, doi: <http://arxiv.org/abs/2010.06075>.
- [76] M. Meyer, T. Kenter, and C. Plessl, "Evaluating FPGA Accelerator Performance with a Parameterized OpenCL Adaptation of Selected Benchmarks of the HPCChallenge Benchmark Suite," in *2020 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, Jun. 2020, pp. 10–18, doi: 10.1109/H2RC51942.2020.00007.
- [77] J. Zhang, N. Beckwith, and J. J. Li, "GORDON: Benchmarking Optane DC Persistent Memory Modules on FPGAs," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2021, pp. 97–105, doi: 10.1109/FCCM51124.2021.00019.
- [78] Y. You, A. Buluç, and J. Demmel, "Scaling Deep Learning on GPU and Knights Landing clusters," in *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2017, pp. 1–12, doi: 10.1145/3126908.3126912.
- [79] Y. Hu, Y. Du, E. Ustun, and Z. Zhang, "GraphLily: Accelerating Graph Linear Algebra on HBM-Equipped FPGAs," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Nov. 2021, pp. 1–9, doi: 10.1109/ICCAD51958.2021.9643582.
- [80] S. K. Prakash, H. Patel, and N. Kapre, "Managing HBM Bandwidth on Multi-Die FPGAs with FPGA Overlay NoCs," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2022, pp. 1–9, doi: 10.1109/FCCM53951.2022.9786203.
- [81] B. Bramas, "Fast sorting algorithms using AVX-512 on Intel Knights Landing," *ArXiv preprint arXiv:1704.08579*, vol. 305, p. 315, 2017.