Exploring Time and Energy for Complex Accesses to a Hybrid Memory Cube

Juri Schmidt Holger Fröning Ulrich Brüning
Institute of Computer Engineering
University of Heidelberg
Mannheim, Germany
{juri.schmidt,holger.froening,ulrich.bruening}@ziti.uni-heidelberg.de

ABSTRACT

Through-Silicon Vias (TSVs) and three-dimensional die stacking technologies are enabling a combination of DRAM and CMOS die layer within a single stack, leading to stacked memory. Functionality that was previously associated with the microprocessor, e.g. memory controllers, can now be integrated into the memory cube, allowing to packetize the interface for improved performance and reduced energy consumption per bit. Complex memory networks become feasible as the logic layer can include routing functionality. The massive amount of connectivity among the different die layers by the use of TSVs in combination with the packetized interface leads to a substantial improvement of memory access bandwidth. However, leveraging this vast bandwidth increase from an application point of view is not as simple as it seems. In this paper, we point out multiple pitfalls when accessing a stacked memory, namely the Hybrid Memory Cube (HMC) in combination with the publicly available openHMC host controller. HMCs internal architecture still has many similarities with traditional DRAM chips like the page-based access, but it is internally partitioned into multiple vaults. Each vault comprises a memory controller and multiple DRAM banks. Pages are rather small and rely on a closed-page policy. Also, the ratio of read and write operations has an optimum of which the application should be aware. The built-in support for atomic operations sounds like a great opportunity for off-loading, but the impact of contention cannot be neglected. Besides exploring such performance pitfalls, we further start exploring the energy efficiency of memory accesses to stacked memory.

CCS Concepts

•**Hardware → Memory and dense storage;**

Keywords: Hybrid Memory Cube, openHMC, Processing in Memory, PIM, Stacked DRAM

DOI: 10.1145/1235

1 INTRODUCTION

The gap between the processor and traditional memory performance continues to widen. As processors are increasingly relying on a vast amount of parallelism for performance improvements, the interface to memory is limited in such regard by the mandatory off-chip interface. However, emerging die stacking technology in combination with Through-Silicon Vias (TSV) enables CMOS and DRAM die layers to co-exist within a single stack. Logic previously associated with the processor die, in particular the memory controller, can now be moved into this die stack allowing substantial interface simplifications between processor and memory. Such a packetized interface enables the use of high-speed, serialized links with longer distances compared to a traditional transactional bus interface for DRAM technologies. Hence, memory access bandwidth can be significantly increased. Furthermore, the construction of complex memory networks becomes available by including routing logic in the die stack in combination with multiple ports or interfaces.

Power and energy increasingly move into focus as technology trends dictate hard constraints on power and energy consumption. Recent work (e.g. [15]) observed that already today more energy is spent on data movement than for computation, and projections show that this trend will intensify in the future. As a result, computation overhead will be negligible in terms of energy while the overall energy footprint is dominated by data movement. In addition, energy consumption for data movement is highly dependent on the distance covered. In particular, one can observe a large increase when moving from on-chip to off-chip connectivity (basically a transition from R-C to a transmission line). As the interface to stacked memory like Hybrid Memory Cube (HMC) is greatly simplified, we find it mandatory to explore the implications towards power consumption and energy efficiency in terms of pJ per bit.

We anticipate that moving the memory controller into the die stack is only the first step towards a range of off-loading explorations. For such processing-in-memory (PIM), various early explorations have already been proposed, e.g. [4], [10], [3], [11]. However, such work in future research directions also should understand the implications of the memory interface in detail, in particular as any future architecture will have to operate within hard power and maybe energy constraints.

In a previous work, we have introduced the open-source HMC host controller openHMC [13]. openHMC is a vendor-agnostic, AXI-4 compliant HMC controller that can be pa-

rameterized to different data-widths, link configurations, and clock speeds depending on speed and area requirements. The main objective of openHMC is to provide an intellectual property that helps others exploring HMC performance without constraints regarding certain technologies. It can be easily mapped to different ASIC and FPGA technologies.

In this work, we extend our previous work by performance and energy explorations. We set up a test system that implements the openHMC controller in an FPGA which is connected to an HMC. We included comprehensive instrumentation for time and power measurements, allowing us to characterize the memory interface in detail regarding bandwidth, latency, and energy efficiency. In particular, we make the following contributions:

- Analysis of the HMC architecture as a requirement to understand the importance of access patterns on bandwidth
- Exploration and discussion on sustained memory performance for linear and complex access patterns, varying mixtures of read and write operations, and using atomic operations
- Measurement and analysis of the HMC read latency
- Discussion of performance pitfalls, recommendations for best practice, and power analysis

The remainder of this work continues by providing detailed background information on the HMC. We then move on to an introduction of our experimental set-up and the importance of access patterns on performance. Afterwards, we present and discuss results in terms of bandwidth and latency for this set-up. We continue by an exploration of atomic operations to offload computation and an energy analysis for various access patterns. This paper is concluded by a discussion about related work, some findings, and usage recommendations.

2 THE HYBRID MEMORY CUBE

The Hybrid Memory Cube stacks several layers of DRAM on top of a logic base using TSVs [17]. The logic layer handles all DRAM related management and only exposes multiple serial links to connect a processor over a packet-based protocol. The memory itself is organized in independent sections, called vaults, and each vault has its own memory controller and multiple DRAM banks.

The first generation of HMC devices was specified by the Hybrid Memory Cube Consortium (HMCC) in 01/2013 [8] and later revised with the release of the HMC specification 1.1. In 11/2014, the HMCC announced the availability of the HMC specification 2.0 with a new Very Short Reach (VSR) interface. This work focuses on the HMC specification 1.1 as there are currently no 2.0 compliant devices available.

2.1 Architecture

The basic HMC architecture is shown in Fig. 1. It is separated into 16 independent vaults where each vault connects the upper DRAM layers with a dedicated memory controller (the vault controller) using 32 TSVs [9]. Each DRAM layer

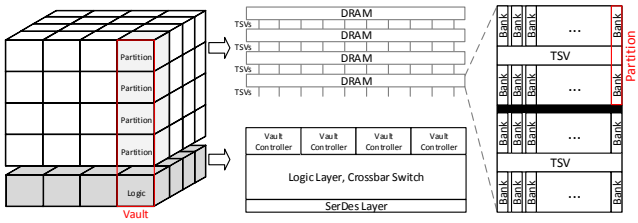

Figure 1. HMC architecture

comprises 16 partitions with 2 DRAM banks. In [9], the HMC Gen1 DRAM stack was introduced as a composition of 68mm^2 1Gbit dies manufactured in 50nm. Here, four layers are stacked for a total capacity of 512MB. Current HMC Gen2 devices [7] stack four 4Gbit DRAM dies on top of the logic base, increasing the capacity to 2GB (4 layers * 16 partitions * 2 banks = 128 banks). The capacity growth from Gen1 to Gen2 is based on denser memory arrays with a bank capacity increase from 4MB (Gen1) to 16MB (Gen2).

HMC comes with a DRAM closed-page policy as opposed to an open-page policy where a row stays active in the sense amplifiers until it times out or another row is accessed. While an open-page policy is in particular beneficial for applications with high locality (i.e. a high page $\frac{hit}{miss}$ ratio) it also increases power consumption since the sense amplifiers stay active after a memory access. Additionally, an open-page policy introduces additional delay on a page miss as pre-charge of the word-line does not take place immediately after the row has been accessed. As a result, closed-page policy theoretically performs better for random access patterns.

The DRAM row or page size in HMC has been drastically reduced from 512Byte-2KByte for DDR4 [1] and up to several kilobytes for DDR3 [16]. The HMC specification outlines a DRAM granularity of 32 Byte. We assume integrated access burst mechanisms to accommodate larger transfer sizes at best performance. In general, a smaller page size reduces the probability for a DRAM over-fetch where only a fraction of the information contained in an active page is accessed, and therefore it also reduces dynamic power consumption. The small page size also makes an open-page policy impracticable and is another reason why the HMC developers preferred a closed-page policy.

Many sources often highlight the potential link bandwidth of 240GByte/s (4 Links * 60GByte/s). The effective bandwidth, however, is limited by the number of vaults. With 32 TSVs each and a clock frequency of 1.25GHz [12] a single vault delivers 10GByte/s. Hence, the total usable bandwidth with 16 vaults is 160GByte/s. It is reasonable to provide more link bandwidth than the DRAM stack can deliver due to protocol overhead on the link. Experiments in [12] showed that the effective link bandwidth eventually flattens at 240GB/s for a given 160GByte/s TSV or DRAM bandwidth. Further increasing the link bandwidth will not provide any benefits when all four links are kept busy.

Lastly, processors or other HMCs can be connected to any of the four links exposed by the logic layer. Hence, multiple HMCs can be 'chained' together with varying routing options to increase the capacity. A single link comprises 16 bi-directional high-speed serial lanes. Every link is local to four vaults and a crossbar ensures that all links can access

all vaults and other links. The 4-Link HMC comes in a 31x31mm package while there is also a 2-Link device with equal characteristics available (19.5x16mm).

2.2 Protocol

The HMC protocol defines packet-based, request-response communication with a granularity (FLIT size) of 16 bytes. The protocol supports reading and writing data packet sizes ranging from 16 to 128 bytes along with commands for atomic operations and HMC configuration. Packets are framed by a header and a tail. Flow control in both directions is achieved using tokens (i.e. credits), where one token represents buffer space for one FLIT. The use of tokens prevents the input buffer of the respective receiver from overflowing. Consequently, tokens are returned after packets are processed by the receiver and the corresponding buffer space is free. The HMC protocol also defines a set of link integrity features. Every packet that is transmitted also carries a pointer, the so called Forward Retry Pointer (FRP). This pointer represents the position of the packet in the retry buffer of the sender. As soon as the packet has been processed at the receiver, this pointer will be returned as Return Retry Pointer (RRP). The RRP signals the original transmitter that the packet was received and that its space in the retry buffer can be reused. Additional link integrity is preserved by a CRC, packet length checks, and consecutive sequence numbers. Lastly, the HMC logic die is able to reorder packets for faster execution (e.g. if one vault is accessed more frequently) and responses may return out of order but can be identified by a tag field.

2.3 Link

A single HMC has two or four independent links, where a link consists of either 8 (half-width) or 16 (full-width) differential pairs (lanes) per direction, i.e. data to and from the HMC can flow at the same time. Available link speeds are 10, 12.5, or 15 Gbps. This results in a single-link bandwidth of 16 lanes $\times$ 15Gbps = 240Gbps = 30GByte/s per direction or 60GByte/s bi-directional. The polarity of individual lanes can be inverted, and the lane order can be reversed to simplify signal routing on a PCB. The link is complemented by a set of sideband signals: An active-low reset (PRST_N), power state signals (RXPS and TXPS), and a unidirectional, HMC-driven fatal error indicator (FERR_N). Both sides of a link, host and HMC, share a common reference clock, removing the need to transmit a dedicated clock along with the data-lanes.

2.4 Memory Addressing

The HMC logic layer provides the ability to internally remap the 34 bit memory address in the header of every request packet. The actual address is interpreted as a combination of four segments: vault, bank, DRAM, and finally byte address. The latter one remains in the four least significant bits in the address and is ignored by the HMC due to 16Byte granularity. By default HMC uses a *low interleave* remapping algorithm. Sequential accesses will be spread over individual vaults first, then banks, and finally DRAM. Furthermore the *Maximum Block Size* property can be altered to set the location of the vault and bank segments. For example, with maximum block size set to 128, linear 64Byte accesses with corresponding 64Byte address increment per request will always target a single vault two times in a row. To fix this issue, the maximum block size should be set to 64. This shifts the vault segment one bit towards the least significant bits.

2.5 HMC Retry Buffer

Understanding the HMC retry buffer is important to successfully maximize performance. At the same time (along with the token return time, see Section 2.6) its implementation poses the biggest challenge for a host controller design. It is required to ensure that packets will be retransmitted after an error has occurred in the receiver of the communication partner. Section 2.2 described how the retry buffer space is managed. In case pointers are returned too slow, a retry buffer full condition will throttle packet transmission and lower the bandwidth. The HMC specification therefore defines the maximum allowable *retry pointer loop time* for each individual lane-speed to avoid such a situation. Although not mentioned in this specification, two observations can be derived from the facts:

2.5.1 The retry buffer size for a link at 10Gbps is smaller than for 12.5Gbps and 15Gbps

The retry buffer full period for a 16x, 10Gbps configuration is defined with $tFull$ = 153.6ns. The time it takes to process a FLIT in the HMC in 16x, 10Gbps configuration is $tFlit$ = 0.8ns. $\frac{tFull}{tFlit}$ = 192 gives the retry buffer size in FLITs. Similarly, the retry buffer size at 12.5Gbps is calculated with $\frac{tFull}{tFlit} = \frac{163.84}{0.64} = 256$ FLITs. It remains 256FLITs for 15Gbps but now runs full much faster. Hence, 12.5Gbps is the best choice when experiencing issues with pointer loop times.

2.5.2 The retry buffer full period is as twice as big for half-width configurations

The HMC specification does not highlight a difference for the retry buffer size for different link-widths, i.e. half- or full-width. Experiments showed that the retry buffer size remains the same. The time it takes to fill the retry buffer therefore doubles for half-width configurations.

2.6 HMC Token-Based Flow Control

Another important factor in order to maintain highest possible bandwidth is the time it takes to consume, process, and return tokens for transmitted and received packets. Similar to the retry pointer loop time, returning tokens too slow will throttle link bandwidth. This is in particular the case when the input buffer in a host controller runs full. Since tokens may only be returned after a received packet passed the input buffer, the token return time constraint is potentially even harder to meet than the retry pointer loop time.

Designers of a host controller should always keep these two metrics in mind. Especially when targeting FPGAs with relatively low operating frequencies, processing pointers and tokens can take up a large amount of the allowable return loop times.

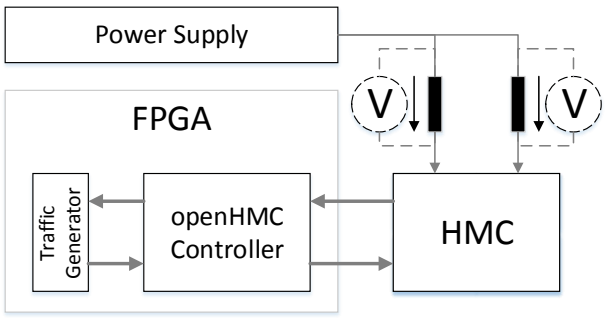

Figure 2. Test setup. FPGA connected to a 4-Link HMC. Power consumption is measured using voltage drop over high-precision resistors

3 TEST SETUP AND ACCESS PATTERN CONVENTIONS

In order to obtain reliable numbers we created a test setup as shown in Fig. 2. It consists of an FPGA that connects a 2GByte, 4 Link HMC using one full-width (16x) link at 10Gbps and 12.5Gbps. We were not able to implement a 15Gbps link since the resulting core clock frequency was too high for the given FPGA. A low-impedance, high-precision resistor per individual power rail is used to measure the power consumption via a Linear Technologies DC1613A PMBus module. We use the publicly available open-source HMC host controller openHMC [13] [5]. The HMC address mapping mode is configured to low-interleave and maximum block size is set to 128Byte. Address (re-)mapping in the HMC core logic can be a useful tool to optimize performance for given access patterns and will be discussed in Section 4. Before moving on to the actual results it is crucial to understand the impact of access patterns on bandwidth. This will also help to avoid pitfalls in a host controller design and applications.

3.1 Understanding the Impact of Access Patterns

Traditional DDR interfaces use a single transactional databus to transmit and receive data to and from the DRAM. Also, a single channel can only serve one command at a time so that execution of subsequent commands requires the previous command to complete first. In contrary, HMC comes with a bi-directional communication scheme. Requests and responses are transmitted on separate channels. This fact requires well balanced access patterns, i.e. an optimum read/write ratio in order to efficiently utilize both link directions and to maximize bandwidth. As described in Section 2.2 the HMC protocol defines a packet-based request/response communication with a FLIT granularity of 16 Bytes. Supported packet sizes range from 16Byte up to 128Byte. Every transmitted packet also requires an additional protocol overhead of 16 Bytes. Therefore, a maximum-sized write request of 128Byte requires 8FLITs payload + 1FLIT overhead (=9FLITs) to be transmitted on the request channel. A read request appears as 16Byte overhead on this channel while it returns one FLIT overhead and up to 8 FLITs payload in response direction. Due to this fact the optimum ratio to maximize HMC link bandwidth is not

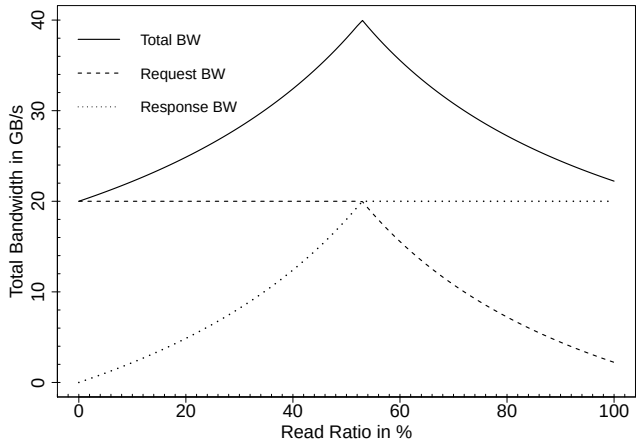

Figure 3. Impact of read/write ratio on total bandwidth at 128 byte requests

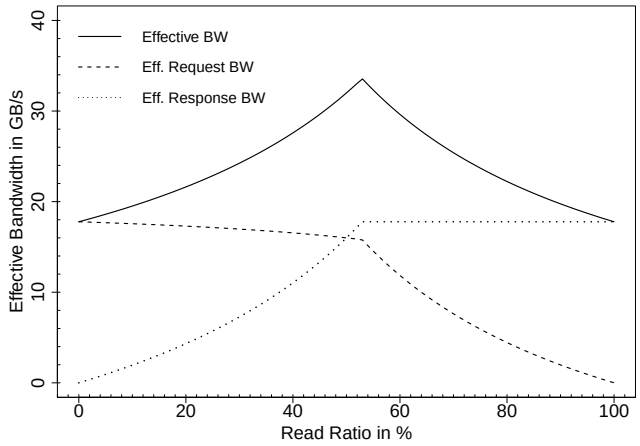

Figure 4. Impact of read/write ratio on effective bandwidth at 128 byte requests

$\frac{1read}{1write}$ as a single maximum sized read+write results in 10 FLITs on the request channel while only 9 FLITs will be returned.

Fig. 3 shows the impact of read-to-write ratio on the total request, response, and combined bandwidth for maximum sized, 128Byte read and write requests. We find that a read ratio of 53% maximizes the available total bandwidth including packet overhead. Similarly Fig. 4 presents the impact on the effective bandwidth. All figures represent the bandwidth for a full-width (16x) link at 10Gbps. The bandwidth, however, increases linearly with the lane-speed. Results for 12.5 and 15Gbps can be obtained by multiplying the bandwidth by 1.25 and 1.5 respectively. We find that the maximum effective bandwidth (i.e. excluding protocol overhead) in a 10Gbps configuration is 33.6GByte/s. Furthermore, the actual optimum ratio depends on the request size as shown in Fig. 5. It can be seen that the optimum ratio shifts towards more read requests as request sizes become smaller. Table 1 summarizes the results for each of the possible request sizes.

It is not only important to maintain the optimum ratio but also the ordering of requests is relevant. In a worst case scenario instead of alternating reads and writes in a stream of 100 × 128Byte requests with the optimum ratio of 53%,

Table 1. Optimum ratio and maximum effective bandwidth per request size

Request Size [Byte]	16	32	48	64	80	96	112	128
Optimum Ratio [% Read]	66	60	57	55	55	54	53	53
10 Gbps: Maximum Effective Bandwidth [GB/s]	14.93	22.2	26.2	28.6	30.3	31.75	32.6	33.55
12.5 Gbps Maximum Effective Bandwidth [GB/s]	18.66	27.8	32.8	35.7	37.9	39.7	40.8	41.9

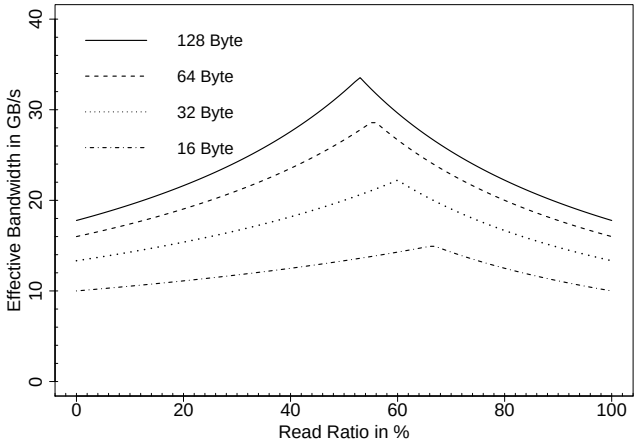

Figure 5. Impact of different request sizes on the optimum read/write ratio

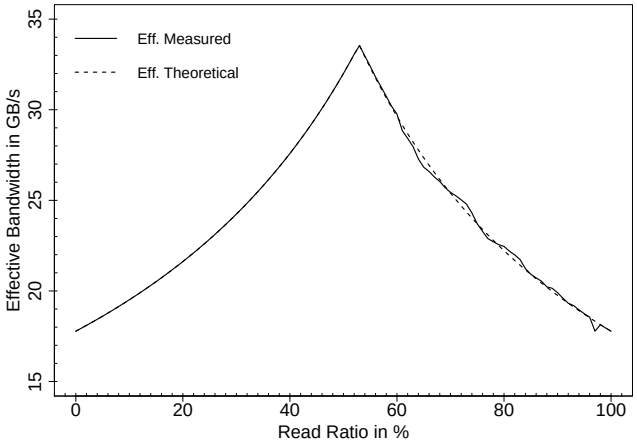

Figure 6. 128Byte request ratio sweep results: theoretical versus measured

the user issues 53 reads followed by 47 writes. We refer to this as *bad request practice*. Its impact on the overall bandwidth will be discussed in Section 4.

4 OBSERVATIONS ON THE IMPLICATIONS OF ACCESS PATTERNS ON SUSTAINED BANDWIDTH

We have identified and tested several access pattern schemes. While some of them perform best with the default address-mapping mode (low-interleave, see Section 2.4), others will benefit from a different address-mapping or maximum block size setting. This section presents the results we obtained and gives advice on how to improve performance characteristics for specific workloads.

The first measurement is drawn in Fig. 6. It compares a sweep of the read ratio for 128Byte requests between calculated and measured effective bandwidth. The results very closely match the theoretical maximum bandwidth. We observe only a slight deviation for higher read ratios most likely due to measurement errors and/or the negative impact of violating the *retry pointer loop time* (see Section 2.5) or *token return loop time* (see Section 2.6). The more reads are issued, the more responses are generated. Consequently, the retry buffer fill time decreases and the pointer loop time constraint is tightened. A different host controller may alleviate or otherwise even increase this disparity. The results for a 12.5Gbps link are very similar. We observe that the measured bandwidth matches the theoretical results.

Fig. 7 shows a plot of various access patterns for 128Byte requests and their impact on the measured effective bandwidth at 10Gbps. It can be seen that linear reading and writing delivers the theoretical maximum of 17.7GB/s. An additional experiment with strided accesses unveils a drop in bandwidth for stride=16, where stride=1 represents linear reading/writing throughout all vaults.

With a stride of 16, 128Byte request size, and low-interleave address-mapping only 1 vault is continuously stressed. The peak bandwidth for only writing a single vault is 9.8GByte/s and 9.35GB/s for reading, respectively. These results closely reflect the maximum vault bandwidth of 10GB/s, lowered by packet processing overhead. In general, increasing the stride will only affect the bandwidth when the number of accessed vaults and therefore the provided vault bandwidth is lower than the link bandwidth. For a given strided access pattern changing the address mapping mode can eliminate this limitation. In the previous case, shifting vault and bank address segments to higher address bits will improve stride=16 accesses. All other strided accesses, however, will lead to vault congestion and negatively impact performance.

The optimum read ratio of 53% gives the maximum effective bandwidth of 33.5GB/s for linear accesses and 8.9GB/s for a single vault. Random accesses do not show an impact when utilizing all available vaults while the single vault bandwidth drops to 7.58GB/s due to the increased probability of bank conflicts. Increasing the lane speed to 12.5Gbps does not improve the single vault performance as shown in Fig. 8. For all other access patterns, however, the results represent what was theoretically evaluated earlier. We introduced the term *bad request practice* in Section 3.1 to describe bad ordering of requests in a stream for a given read ratio. We measured the negative impact of this *bad request practice* to be negligible in our case. However, we were only able to manipulate the ordering of an access stream within 100 requests. Longer sequences will shift the effective bandwidth away from the optimum (Fig. 6). Hence, although the HMC is capable to internally reorder independent requests,

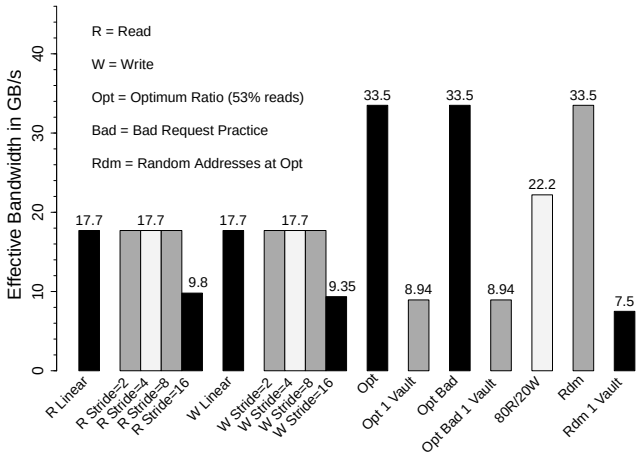

Figure 7. Effective bandwidth for different access patterns with 128Byte requests at 10Gbps

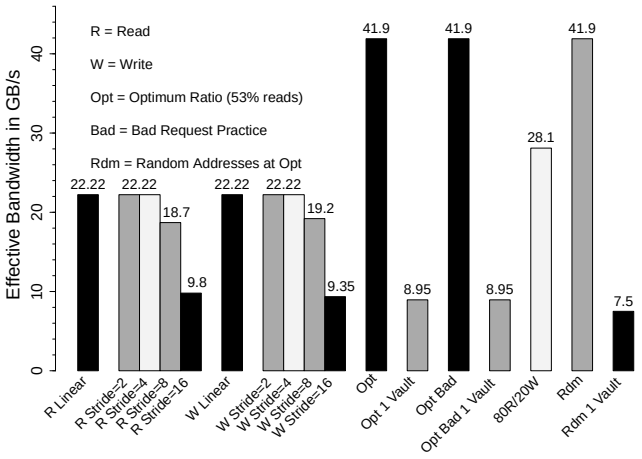

Figure 8. Effective bandwidth for different access patterns with 128Byte requests at 12.5Gbps

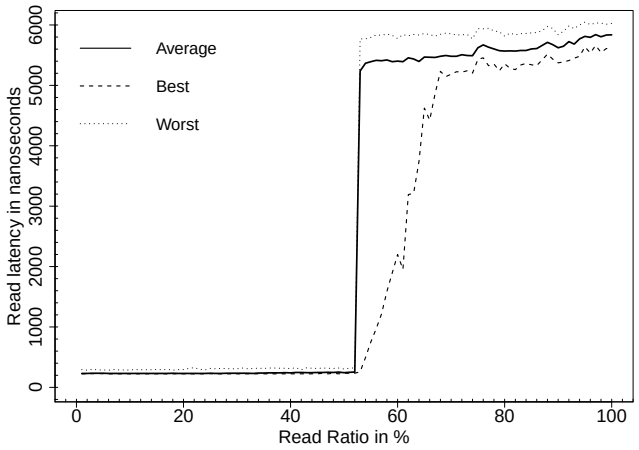

Figure 9. Host to HMC read latency at 10Gbps (t_{cycle}=3.2ns)

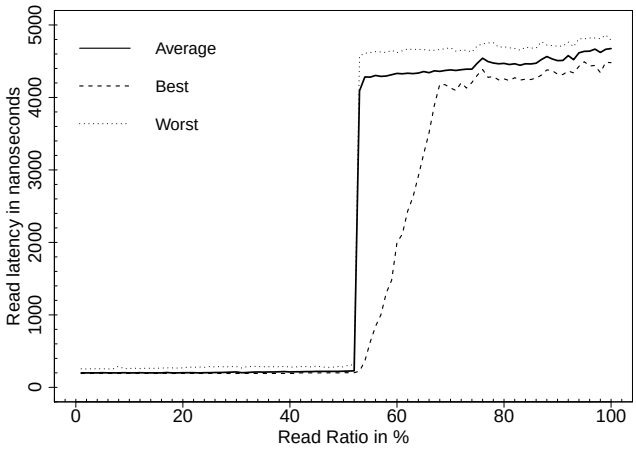

Figure 10. Host to HMC read latency at 12.5Gbps (t_{cycle}=2.56ns)

bad request ordering over a longer period of time will lead to inefficient utilization of either of the two link directions and should be avoided. If required host-sided reordering should be performed in order to maximize the link bandwidth.

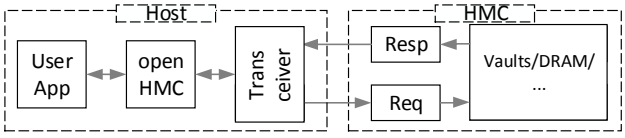

Figure 11. Host to HMC read latency contributors

5 READ LATENCY

The latency of individual requests (i.e. the single read access latency in a given request stream) for HMC is higher than for a transactional memory interface such as DDR. Several contributors to this latency can be identified as depicted in Fig. 11. The delay for the user application and the openHMC controller are well known. We ran the transceiver internal loopback mode of the FPGA to quantify the delay introduced by serialization and deserialization. It is assumed that the transmission line is not contributing significantly. If we subtract all these delays from the overall latency we can estimate the actual HMC read latency. Table 2 summarizes the results for the individual contributors in our test design for 10Gbps and 12.5Gbps. It can be seen that the overall latency can be significantly reduced by increasing the frequency of the FPGA logic. To provide a real world scenario we measured the overall read request latency starting from the point where a packet is created in the user application until the corresponding response returns there. Fig. 9 and Fig. 10 plot the latency over a read ratio sweep for 128Byte requests for a linear address sweep. We applied one ratio at a time for 20 seconds and measured the best, worst, and average latency for randomly selected individual requests in the access stream. It can be seen that the initial read latency starts with an average of 224ns (70 cycles) for 10Gbps, and 192ns (75 cycles) for 12.5Gbps respectively. The latency then remains stable until the optimum ratio threshold is reached. At this point, more reads are requested than the HMC and in particular the response link can serve.

Table 2. Host-sided read latency contributors

Type / Delay	Cycles	@ 10Gbps	@ 12.5Gbps
User App	2	6.4 ns	5.12 ns
openHMC	29	92.8 ns	74.24 ns
Transceiver	19	60.8 ns	48.64 ns
Total non-HMC	50	160 ns	128 ns

10Gbps: 312.5MHz FPGA clock (t_{cycle}=3.2ns)
12.5Gbps: 390.625MHz FPGA clock (t_{cycle}=2.56ns)

The read latency continues to grow the more reads are sent, up to several microseconds. The reason is that the HMC input buffer runs full with unanswered requests and the response link is in bandwidth saturation; preventing the host to continue. The disparity between the best and worst case latencies originates from corner cases where a corresponding read request enters the openHMC controller right at the time that traffic is throttled. The request therefore remains in buffers waiting to be transmitted, while this waiting time accounts for the overall latency.

In summary it becomes clear that the single access latency gets much worse when either of the link directions saturates. A low latency host design including transceiver, host controller, and user application in combination with well balanced access patterns are the key elements to lower the HMC access latency. In our case, however, increasing the link speed to 15Gbps (i.e. also an increase in FPGA frequency) was not an option due to implementation issues.

6 USE OF ATOMIC OPERATIONS TO OFFLOAD COMPUTATION

The HMC protocol defines packet types for atomic operations that will be performed by the HMC logic layer, eliminating the need for wasteful read-modify-write cycles on the host. The two available commands add either an 8Byte value to a 16Byte memory operand (16-byte immediate add) or two 4Byte values to two 8Byte memory operands (dual 8-byte immediate add) respectively. Such an operation is referred to as *update*. Fig. 12 summarizes the maximum updates per second when updating a single address, a single vault, and up to all available vaults, represented by the corresponding access stride, where stride=16 accesses only 1 vault in our configuration. It can be seen that the maximum number of updates per second increases proportional with the number of accessed vaults and inverse with the stride size respectively for both types of atomics. Since the actual packet throughput remains the same, dual 8-byte add immediate commands can update as twice as many values compared to single 16-byte add. Fig. 13 points out that increasing the lane speed does not improve the maximum number of atomic operations. The key observation in these plots is that increasing the number of accessed vaults has a positive impact on the total number of updates per second. This is in contrast to regular read/write requests that already saturate with less vaults. The positive effect of accessing more vaults concurrently, however, would also apply for regular reading and writing when using more than one link. Another interesting fact is that increasing the link speed has no effect on the throughput of atomic operations

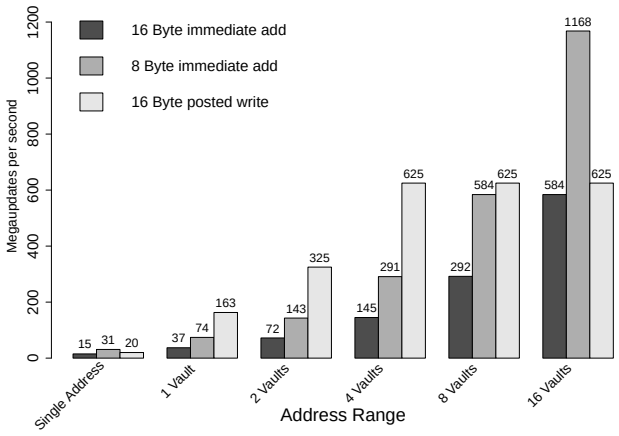

Figure 12. Megaupdates/second versus address range at 10Gbps

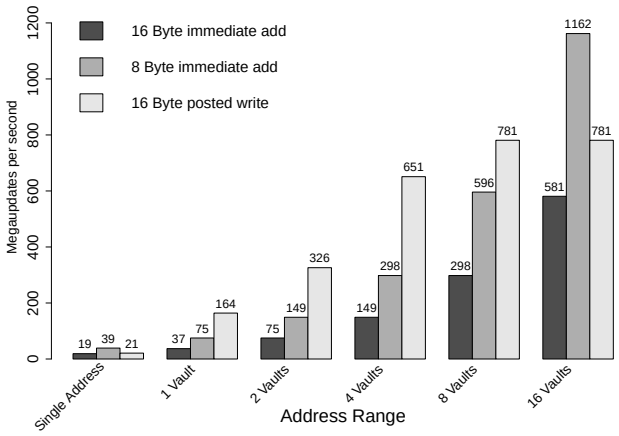

Figure 13. Megaupdates/second versus address range at 12.5Gbps

since the ALUs maintain their operating frequency.

7 ANALYSIS OF ENERGY CONSUMPTION FOR VARIOUS WORKLOADS

We measured the HMC power consumption for the workloads presented in Fig. 7 and Fig. 8 using the test setup shown in Fig. 2. We estimated the total power consumption by measuring the voltage drop of each individual power rail over high-precision resistors. The following results, however, represent our own measurements at best efforts and are furthermore subject to parasitic effects (e.g. efficiency of the power source and other components) and deviation (e.g. temperature, measurement error).

Fig. 14 and Fig. 16 plot the HMC power consumption at 10Gbps and 12.5Gbps respectively. We included results for HMC in power-on/reset and idle state as a reference. It can be seen that static power makes up a major fraction of the overall consumption. While a link in idle already consumes about 5 watts, actual traffic does not excessively contributes to the overall power footprint. One expected observation is that the more bandwidth is requested the higher the power consumption gets. The main contributors here are the sources for the DRAM and the logic core. Fig. 15 and

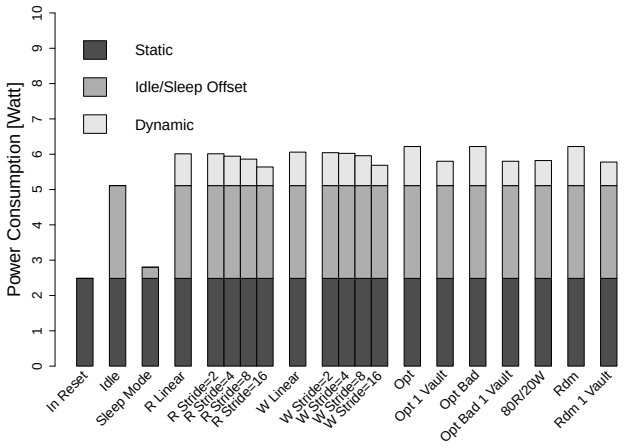

Figure 14. HMC power consumption for various workloads in pJ/bit at 10Gbps

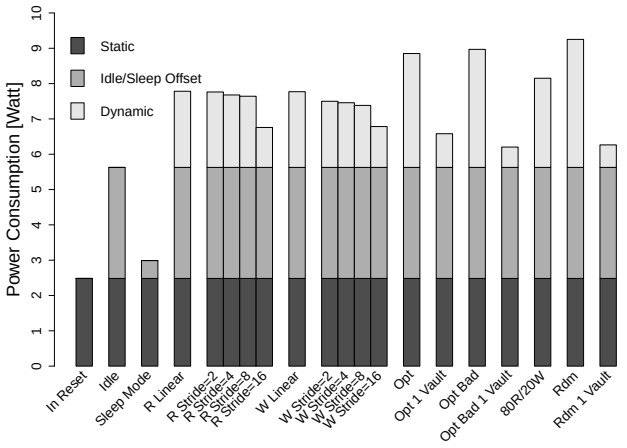

Figure 16. HMC power consumption for various workloads in pJ/bit at 12.5Gbps

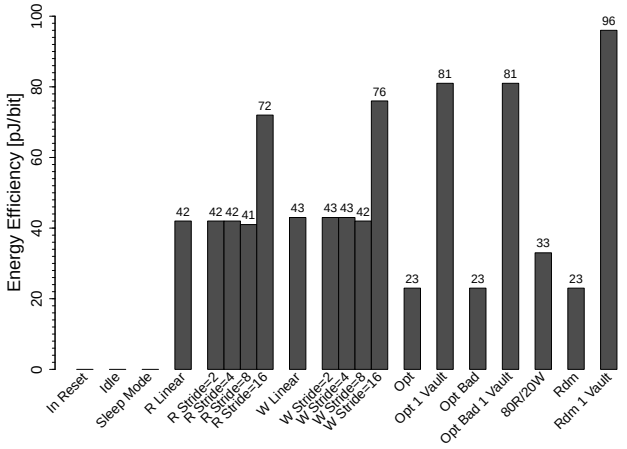

Figure 15. HMC energy efficiency for various workloads in pJ/bit at 10Gbps (lower=better)

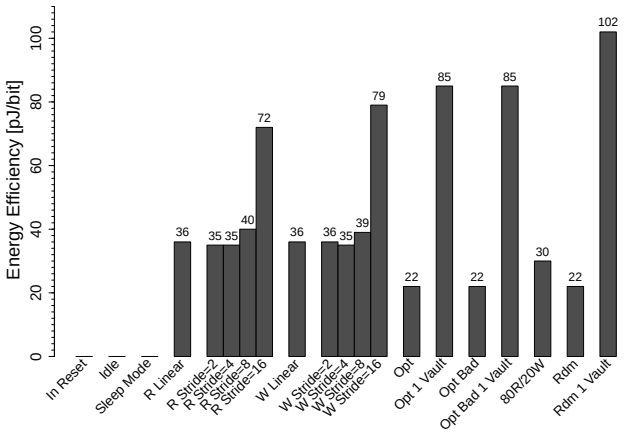

Figure 17. HMC energy efficiency for various workloads in pJ/bit at 12.5Gbps (lower=better)

Fig. 17 show the measured power efficiency in [pJ/bit] for the individual workloads at 10Gbps and 12.5Gbps. The efficiency is calculated as the power consumption in [Watt] divided by the effective bandwidth. We find an idle power consumption (i.e. after the link has trained) of 5.1Watt and the best energy efficiency with 23.2pJ/bit at the optimum read/write ratio for a 10Gbps link. Similarly, the idle power consumed for 12.5Gbps is 5.6Watt and the highest efficiency was measured with 21.7pJ/bit. All efficiencies are relative to the effective delivered bandwidth. There are no values provided for reset, idle, and sleep as there is no data transmitted at that time. We discover that accessing random addresses do not affect power consumption except when bank conflicts occur which lower the bandwidth. Furthermore we were not able to measure any difference between properly ordered request streams and the bad request practice introduced earlier. As mentioned before it would require very long streams of disordered accesses to see an effect here.

In general, both plots point out that the power efficiency improves (i.e. pJ/bit drops) when the link is kept busy. In contrast, static power consumption dominates for inefficient link utilization and the efficiency is reduced. It is expected that increasing the number of active links and their lane-speeds will have a positive effect on energy efficiency as static power is a major contributor to the overall consumption.

The HMC sleep mode can be entered to reduce power consumption when the link is idle to save about 45% of the idle power at 10Gbps and 49% at 12.5Gbps. However, it must be noted that entering and exiting sleep mode takes some time and requires an additional link initialization sequence.

8 RELATED WORK

In [18], a reconfigurable memory controller is integrated into the die stack with the processor still being located in a different chip. Some early experiences with specialized ARM cores are made in [6]. The Smart Memory Cube [2] is an extension of the HMC, and first step towards PIM by introducing an on-chip network in front of an HMC. The authors of [14] propose a memory-centric network in which memory also serves for processor-to-processor communication. At the time of writing, HMC has not moved into volume production yet and reliable performance and power numbers are rare. In [12], the author analyzed the HMC architecture in detail and provided performance results for single cubes and networks of multiple HMCs. This work, however, is based on

an HMC simulation model and does not reflect real-system measurements.

9 CONCLUSION

This paper introduced the Hybrid Memory Cube and provided real-system measurements for bandwidth, latency, computation off-loading using HMC built-in features, and energy efficiency for a single link in 16-lane, 10Gbps and 12.5Gbps configurations.

We showed that a proper understanding of the HMC architecture is mandatory in order to optimize the overall performance. This also requires a well-designed HMC host controller and properly ordered request streams. As long as requests access more vaults than actually required to saturate the link bandwidth, the limiting factor is the link itself. Link performance, however, is mainly dependent on the corresponding host controller. We pointed out that a low-latency optimized controller is crucial to avoid any flow-control drawbacks through the circulation time of packet pointers and tokens. An evaluation of sustained bandwidth when using all four links and the performance impact of chained topologies is yet outstanding.

The HMC read latency was analyzed and it became clear that the host portion of the request/response loop strongly influences the overall latencies. The results, however, reflect an FPGA host implementation. ASICs with higher clock speeds and low latency SerDes implementations would significantly reduce the numbers presented.

We presented results for off-loading computation using atomic operations. It can be a useful feature to eliminate computation and communication overhead for add operations. HMC specification 2.0 compliant devices will support more advanced logical and comparison commands. Complex operations, however, still require processor interaction.

Finally, we analyzed the HMC energy efficiency. The results prove that stacked memories and in particular HMC are capable to reduce the power-penalty for accessing memory. Additional experiments with more links and higher lane-speeds will ultimately identify its full potential.

In conclusion, it became clear that HMC indeed provides the bandwidth it claims. It furthermore contributes to meet the power and energy requirements for future systems. 3D integration of CMOS logic and processing elements will continue to gain importance and we will hopefully see some advanced offloading capabilities integrated into the memory controller in the future.

10 ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) via the DEEP-ER project under Grant Agreement no 610476.

References

[1] DDR4 SDRAM, JESD79-4 Specification. Technical report, Jedec Solid State Technology Association, Sep 2012.

[2] E. Azarkhish, D. Rossi, I. Loi, and L. Benini. High performance axi-4.0 based interconnect for extensible smart memory cubes. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 1317–1322. IEEE, 2015.

[3] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson. Near-data processing: Insights from a micro-46 workshop. *IEEE Micro*, 34(4):36–42, July 2014.

[4] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die stacking (3d) microarchitecture. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 469–479, Dec 2006.

[5] Computer Architecture Group - University of Heidelberg. openHMC - an Open-Source Hybrid Memory Cube Controller, Apr. 2015.

[6] R. G. Dreslinski, D. Fick, B. Giridhar, G. Kim, S. Seo, M. Fojtik, S. Satpathy, Y. Lee, D. Kim, N. Liu, et al. Centip3de: a many-core prototype exploring 3d integration and near-threshold computing. *Communications of the ACM*, 56(11):97–104, 2013.

[7] Hybrid Memory Cube Consortium. Hybrid Memory Cube Specification 1.1.

[8] Hybrid Memory Cube Consortium. Hybrid Memory Cube Consortium Website, Apr. 2015.

[9] J. Jeddeloh and B. Keeth. Hybrid memory cube new dram architecture increases density and performance. In *VLSI Technology (VLSIT), 2012 Symposium on*, pages 87–88. IEEE, 2012.

[10] G. H. Loh, Y. Xie, and B. Black. Processor design in 3d die-stacking technologies. *IEEE Micro*, 27(3):31–48, May 2007.

[11] L. Nai and H. Kim. Instruction offloading with hmc 2.0 standard: A case study for graph traversals. In *Proceedings of the 2015 International Symposium on Memory Systems*, MEMSYS '15, pages 258–261, New York, NY, USA, 2015. ACM.

[12] P. Rosenfeld. *Performance Exploration of the Hybrid Memory Cube*. PhD thesis, University of Maryland, 2014.

[13] J. Schmidt and U. Bruning. openhmc - a configurable open-source hybrid memory cube controller. In *ReConFigurable Computing and FPGAs (ReConFig), 2015 International Conference on*, pages 1–6, Dec 2015.

[14] A. Sethia, G. Dasika, M. Samadi, and S. Mahlke. Memory-centric system interconnect design with hybrid memory cubes. In *Parallel Architectures and Compilation Techniques (PACT), 2013 22nd International Conference on*, pages 145–156. IEEE, 2013.

[15] J. Shalf, S. Dosanjh, and J. Morrison. Exascale computing technology challenges. In *High Performance Computing for Computational Science–VECPAR 2010*, pages 1–25. Springer, 2010.

[16] J. Standard and D. S. Specification. Jesd 79-3b. *JEDEC Solid State Technology Association*, page 37, Apr. 2008.

[17] Sung Kyu Lim. 3D Circuit Design with Through-Silicon-Via: Challenges and Opportunities. Technical report, GTCAD Laboratory, 2010.

[18] J. Zhao, G. Sun, G. H. Loh, and Y. Xie. Optimizing gpu energy efficiency with 3d die-stacking graphics memory and reconfigurable memory interface. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(4):24, 2013.