Evaluating IOMMU-Based Shared Virtual Addressing for RISC-V Embedded Heterogeneous SoCs

Cyril Koenig
Integrated Systems Laboratory
ETH Zurich
cykoenig@iis.ee.ethz.ch

Enrico Zelioli
Integrated Systems Laboratory
ETH Zurich
ezelioli@iis.ee.ethz.ch

Luca Benini
ETH Zurich
Zurich, Switzerland
Università di Bologna
Bologna, Italy
lbenini@iis.ee.ethz.ch

***Abstract*—Embedded heterogeneous systems-on-chip (SoCs) rely on domain-specific hardware accelerators to improve performance and energy efficiency. In particular, programmable multi-core accelerators feature a cluster of processing elements and tightly coupled scratchpad memories to balance performance, energy efficiency, and flexibility. In embedded systems running a general-purpose OS, accelerators access data via dedicated, physically addressed memory regions. This negatively impacts memory utilization and performance by requiring a copy from the virtual host address to the physical accelerator address space. Input-Output Memory Management Units (IOMMUs) overcome this limitation by allowing devices and hosts to use a shared virtual paged address space. However, resolving IO virtual addresses can be particularly costly on high-latency memory systems as it requires up to three sequential memory accesses on IOTLB miss.**

In this work, we present a quantitative evaluation of shared virtual addressing in RISC-V heterogeneous embedded systems. We integrate an IOMMU in an open source heterogeneous RISC-V SoC consisting of a 64-bit host with a 32-bit accelerator cluster. We evaluated the system performance by emulating the design on FPGA and implementing compute kernels from the RajaPERF benchmark suite using heterogeneous OpenMP programming. We measure the transfers and computation time on the host and accelerators for systems with different DRAM access latencies. We first show that IO virtual address translation can account for 4.2% up to 17.6% of the accelerator's runtime for `gemm` (General Matrix Multiplication) at low and high memory bandwidth. Then, we show that in systems containing a last-level cache, this IO address translation cost falls to 0.4% and 0.7% under the same conditions, making shared-virtual addressing and zero-copy offloading suitable for such RISC-V heterogeneous SoCs.

***Index Terms*—RISC-V, Hardware accelerators, Heterogeneous computing, IOMMU, Shared virtual addressing**

I. Introduction

Heterogeneous SoCs employ domain-specific accelerators to improve performance and energy efficiency. Vendors such as Nvidia and AMD have developed platforms coupling integrated graphical processing unit (GPU) in so-called accelerated processing unit (APU), offering high performance from the edge - with the Nvidia Tegra series [1] - to datacenters - with the AMD EPYC processors [2]. In these platforms, the host and accelerator share the same memory controller, making them particularly suited to also share a unified virtual address space to simplify data sharing in heterogeneous applications. Traditionally, such systems enable shared virtual memory using a hardware IO-memory management unit (MMU) located between the highest GPU cache and the memory fabric [1].

With the democratization of the open source RISC-V instruction set architecture (ISA), a new era of heterogeneous platforms opens, combining RISC-V hosts and RISC-V accelerator IPs implementing custom extensions. Similarly to APUs, these new platforms can also leverage shared virtual addressing and zero-copy offloading using the recently ratified RISC-V IO memory management Unit (IOMMU) specification 1.0 [3].

To this day, many of the proposed RISC-V parallel accelerator architectures [4] [5] [6] follow the programmable manycore accelerator (PMCA) design pattern. These chips consist of multiple processing elements (PEs) assembled in clusters around a fast-access scratchpad memory (SPM) refilled using explicit direct memory access (DMA) calls.

In this work, we integrate the open source RISC-V IOMMU implementation from [7] [8] into a heterogeneous open source SoC combining a RISC-V Linux capable host and a floating-point optimized RISC-V PMCA. By emulating this platform on FPGA, we evaluate the accelerator's performance on four heterogeneous compute benchmarks using heterogeneous OpenMP-offloading. We show a significant increase in data transfer time when enabling shared virtual memory support, causing up to 17.6% performance degradation on `gemm` at high memory latency. However, we show that after integrating a shared last-level cache (LLC) before the memory controller, degradations caused by IO translation lookaside buffer (IOTLB) misses get down to 0.7% without requiring any further IOMMU optimization or extensions [9] [10]. We then confirm these results under synthetic concurrent memory traffic on the shared cache from the host.

Overall, we present the following contributions:

- The integration of the IOMMU from [7] [8] in a RISC-V heterogeneous SoC featuring an eight-cores scratchpad-

978-3-9826741-0-0/DATE25/© 2025 EDAA

based PMCA. The hardware is available in open source[1].

- The implementation and benchmarking of heterogeneous userspace applications on an FPGA demonstrator under different memory latencies.
- The evaluation of shared virtual addressing benefits on this demonstrator without any custom IOMMU architectural optimization when coupled to an LLC.

II. Background

This study builds upon open source architectural building blocks and integrates several hardware and software components to propose an evaluation of shared virtual addressing on heterogeneous RISC-V platforms.

Cheshire [11] is a fully open source and an in-depth host SoC built around the 64-bit, application-class CVA6 RISC-V processor [12]. Alongside the CVA6 core, Cheshire offers multiple peripherals for IO communication and an LLC that can be partitioned between cache and static random access memory (SRAM) via run-time configurations. The system components are connected via an advanced extensible interface (AXI) fully connected crossbar [13]. The Cheshire SoC supports the seamless integration of domain-specific accelerators (DSAs), which can be connected to the system's AXI crossbar via dedicated master and slave ports.

The Snitch cluster [14] is an open source, RISC-V, multi-core accelerator targeting energy-efficient execution of floating-point workloads. The cluster comprises eight `rv32imafd` PEs, which are small Snitch integer cores coupled to FPUs. A ninth core controls the cluster's DMA engine. The cluster contains an L1 tightly-coupled data memory (TCDM) split into parallel accessible banks. In this work, the Snitch cluster is referred to as *device* or *accelerator* interchangeably.

The RISC-V IOMMU specification v1.0 [3] defines the hardware and software standards to build IO virtual address (IOVA) support for devices and DMA engines. This specification was recently implemented in an open source IP [7] available at [8]. This module can enable memory protection, virtualization, and zero-copy heterogeneous applications.

OpenMP [15] is a standardized application programming interface (API) for parallel C/C++ and Fortran programming. It also supports heterogeneous programming via its *target* API. In this work, we use this API to build userspace applications accelerated by a Snitch cluster DSA.

III. Methods

A. Hardware and software architecture

Based on the key building blocks described above, we designed a prototype heterogeneous platform to explore shared virtual addressing on RISC-V. The proposed platform is presented in Figure 1, the Cheshire SoC is configured with 128 KiB of LLC/SPM and a single-core 64-bit CVA6 with 32 KiB of write-through data-cache. The IOMMU is located between the SoC crossbar interconnect and the Snitch cluster.

[1] https://github.com/pulp-platform/carfield/tree/date_iommu_evaluation

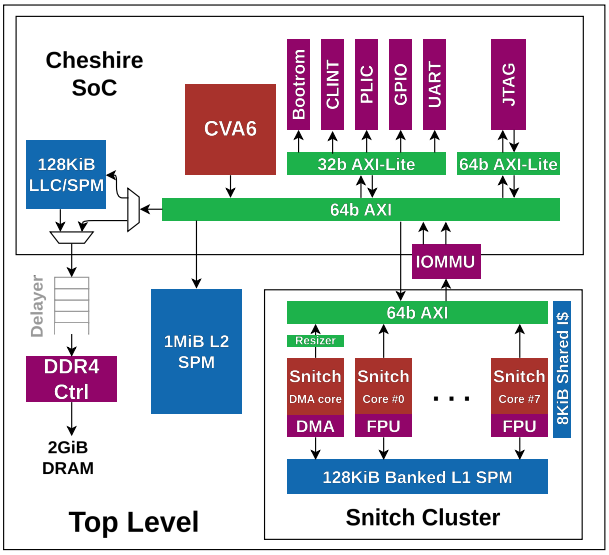

Fig. 1: Block diagram of the prototype platform. Note that the DRAM delayer is added on emulation to provide more accurate performance evaluation.

It is configured with four IOTLB entries and one device directory entry, caching the location of the root page table for one (device, process) pair. The IOMMU connects to the main crossbar via two AXI ports, one for the device transactions and one for page table walk (PTW) transactions. The platform also contains 1 MiB of non-cached and physically addressed scratchpad memory (one-to-one translation). This L2 SPM stores device binaries and shared data structures such as software mailboxes for synchronization. Finally, the DDR controller is connected to 2 GiB of off-chip dynamic random access memory (DRAM), the lower half is allocated to Linux, the upper half is reserved for allocating physically contiguous DMA buffers for the accelerator in case the application does not use shared virtual memory but explicit data copies. The reserved DRAM space is uncached by the LLC via muxes visible on the left of the system crossbar on Figure 1.

The platform is emulated on a Xilinx Ultrascale+ VCU128 field-programmable gate array (FPGA) development board, the Snitch cluster domain is clocked at 20 MHz while the rest of the platform, including the IOMMU, belongs to the host domain clocked at 50 MHz. Note that multiple clock domains are common in heterogeneous SoCs; for instance, in the Nvidia Xavier chips, the central processing unit (CPU) frequency can be up to 1.7× the GPU frequency [1]. However, due to the low operating frequency of the emulated platform compared to the DRAM chip, the main memory latency appears consequently smaller on FPGA than on silicon. At 50 MHz, CVA6 has a read latency in DRAM of only about 35 clock cycles. We add a parametrizable AXI delayer before the DDR controller to address this limitation. This module is implemented with FIFO macroblocks and delays `b` and `r` channels for a configurable number of cycles. This allows us to quantitatively evaluate the effects of main memory latency on the system's performance,

TABLE I: Total runtime in cycles for each kernel at variable memory latency.

Kernel	Input size	Description
`gemm`	128×128	Generic matrix-matrix multiplication.
`gesummv`	512×512	Generic matrix-vector multiplication.
`heat3d`	$64 \times 64 \times 64$	3D heat propagation equation.
`axpy`	32768	Generic vector-vector addition.
`merge sort`	65536	Merge sort algorithm.

especially when IOMMU-based address translation is enabled.

B. Software stack and benchmarks

The CVA6 core runs a patched Linux 6.9.0 with support for the RISC-V IOMMU Architecture Specification Version 1.0 [16]. We implement a simple device driver to attach our accelerator to IOMMU domain and create IO-virtual-physical mappings from the host. This driver also enables CVA6 to access the accelerator's register space, L2, and a reserved portion of the DRAM. The driver can be accessed via a user-space library API, which is then used by the heterogeneous OpenMP runtime to trigger kernel computation on the device.

In order to benchmark the performance of the accelerated system under different memory requirements, we implement a subset of linear and non-linear benchmarks from the RajaPERF suite [17] presented in Table I. For linear kernels, we select operations from the *basic* and *polybench* group with increasing arithmetic intensity: `axpy`, `heat3d`, `gesummv`, and `gemm`. For nonlinear kernels, we implement `sort` from the *algorithm* group (using a parallel merge sort). All our benchmarks are run for single-precision floating-point data using input tiling and double-buffering to make the best use of the L1-SPM and the cluster's DMA engine. In order to measure the IOMMU effects for input and output data only, we offload each computation twice and measure performance with a hot instruction cache. In SoCs based on this evaluation platform, the cluster's instruction cache can bypass page translation with a different device identifier pointing to a bypassed device directory entry [3]. For comparison, we implement the same compute kernels on the single-threaded host core.

IV. Results

Using the platform and benchmarks described above, we present the performance impact of shared virtual addressing in this section. First, we present the benefits of shared addressing thanks to zero-copy offloading at the application scale. Then, we focus on its impact on the accelerator runtime at the compute kernel scale.

A. Zero-copy offloading

To show the benefits of shared virtual addressing on heterogeneous applications, we compare the execution time of a single precision `axpy` operation. We consider three cases: CVA6 executes the kernel; CVA6 copies the data to shared DRAM and the Snitch cluster executes the kernel; CVA6 creates IOVA mapping and Snitch cluster executes the kernel in a zero-copy fashion. We select a relatively small problem size of 32.768

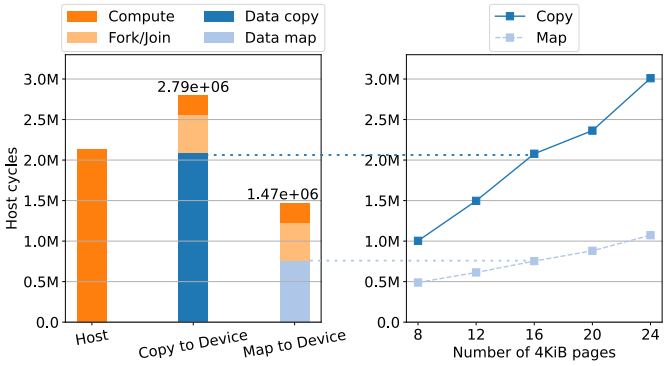

Fig. 2: (Left) $\texttt{axpy}_{32.768}$ breakdown for three scenarios. Host-only execution; Data copy and device execution; Data mapping and device execution. (Right) Time spent copying or mapping data of different input sizes.

elements per vector (corresponding to 16 input pages) in order to present the different overheads accurately and prevent the compute time from dominating the whole execution. We split the accelerated application into three regions: the time required to copy or map data to the cluster, the overhead caused by triggering execution and synchronizing using the OpenMP target fork/join model, and the computation on the device.

In the left half of Figure 2, we show the runtime breakdown for all three cases. First, the computation part is always faster on the cluster since it contains eight PEs and does not need to handle kernel-related hardware interrupts. Then, we note that offloading can be more expensive with explicit data copy than a regular host execution. This is particularly true for an `axpy` kernel that is heavily memory-bound. However, we also note that although page mapping requires at most 24 bytes (three-page table entries) per every $4\,\mathrm{KiB}$ of input, a consequent overhead of a few hundred thousand cycles remains in the third case. This is partly due to the communication with the Linux kernel module via `ioctl`. Overall, we measure in Figure 2 that zero-copy offloading is 47% faster than copy-based offloading. On the right side of Figure 2, we present the explicit copy and zero-copy runtime with increasing problem sizes. As expected, zero-copy offloading also scales better with problem size than an explicit copy.

However, data copy overhead does not only depend on the input size. In Figure 3, we compare copying and mapping for increasing DRAM latencies. We note that copying 16 pages ($64\,\mathrm{KiB}$) of data to the physically contiguous portion of the DRAM is $3.4\times$ slower when the memory latency raises from 200 cycles to 1000 cycles. However, for the same variation in latency, the time required to map the IO page entries time is multiplied by $2.1\times$. This is due to IO-mapping accessing data structures likely to be present in the CVA6 data cache.

B. Device execution and IOMMU overhead

The previous section shows that shared virtual addressing drastically reduces offloading overhead in heterogeneous applications. Nevertheless, using IO virtual addresses requires

TABLE II: Total runtime in cycles for each kernel at variable memory latency.

Kernel	$gemm_{128}$			$gesummv_{512}$			$heat3d_{64}$			$mergesort_{65536}$		
DRAM Latency	200	600	**1000**	200	600	**1000**	200	600	**1000**	200	600	**1000**
Baseline	2.03e6	2.24e6	2.45e6	4.93e5	6.38e5	9.16e5	2.00e6	4.60e6	7.21e6	6.94e6	7.98e6	9.05e6
% DMA	7.3%	16.0%	**23.2%**	1.4%	23.5%	**46.3%**	36.3%	71.9%	**80.8%**	17.7%	29.2%	**38.3%**
IOMMU	2.12e6	2.50e6	2.89e6	5.20e5	1.08e6	1.70e6	2.84e6	7.09e6	1.13e7	7.67e6	1.08e7	1.44e7
	11.1%	24.6%	**34.5%**	6%	54%	**70.4%**	54.9%	78.9%	**84.8%**	27%	63.4%	**82.6%**
IOMMU+LLC	2.04e6	2.25e6	2.47e6	4.95e5	6.45e5	9.29e5	2.05e6	4.68e6	7.30e6	6.96e3	8.00e6	9.07e6
	7.7%	16.4%	**23.7%**	1.5%	24.1%	**46.9%**	37.8%	72.2%	**81.0%**	22.4%	29.5%	**38.6%**

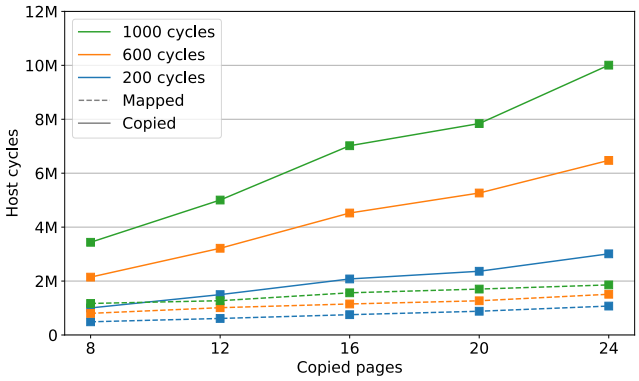

Fig. 3: Data copying and mapping time with input size and different DRAM latencies.

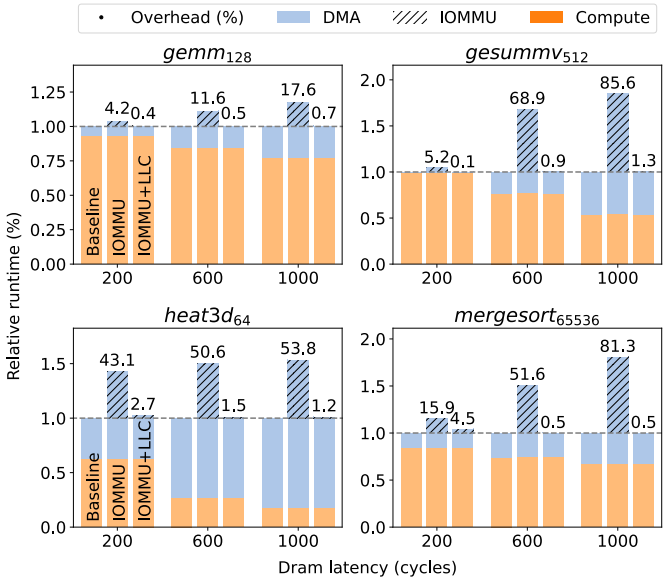

Fig. 4: Kernel execution for different DRAM latencies with three configurations. With IOMMU disabled, with IOMMU enabled and LLC disabled, and with IOMMU enabled and LLC enabled.

the accelerator to translate each access to DRAM. This IO page table walking is a sequential operation that may significantly increase the time spent on the accelerator, especially under high memory latency.

In this section, we measure the runtime on the accelerator for the four selected kernels under different DRAM latencies.

```
a = malloc(n_bytes)
prepare_input(a)
flush_l1()
flush_last_level_cache()
a_iova = create_iommu_mapping(a, n_bytes)
flush_l1()
#pragma omp target device(1) map(to: a_iova)
  device_kernel(a_iova + LLC_BYPASS_OFFSET)
```

Listing 1: Self invalidation based coherency.

We do not measure offloading or synchronization time with the host, and we propose a breakdown into two regions. The DMA region measures the cycles where the cores are busy waiting for data transfers. The compute region measures the rest of the cycles to complete the kernel on the accelerator. Note that, thanks to double-buffering and a dedicated DMA-engine, when the kernel is compute bound, the DMA region tends to zero even if megabytes of data are transferred. We take the total runtime without IOMMU as the baseline, and we plot the relative runtime for all configurations in Figure 4. For IOMMU-enabled configurations, we add the percentage overhead on the plot, and we display for all experiments the absolute measurements in Table II. We first note that for all kernels, the time spent waiting for DMA increases with memory latency, even without IOMMU. Naturally, this increase is smaller for kernels with high arithmetic intensity such as `gemm` (up to $23,2\%$) than for more memory intensive kernels such as `heat3d` (up to 80.8%). When using IO virtual addresses, the latency of a single DMA transfer may increase by 300% since an IOTLB miss requires three new additional memory accesses. In case a DMA transfer is larger than a page, it will be split into multiple bursts (according to the AXI specifications), and every burst causing IOTLB misses may reduce the effective memory bandwidth for the DMA-engine. In Figure 4, we show that this reduced effective bandwidth creates an overhead of 81.3% compared to baseline for the `heat3d` kernel with 1000 cycles of DRAM latency. On the other hand, thanks to high data reuse, this translation overhead is of 17.6% in the `gemm` kernel under the same conditions.

To face the significant overhead caused by IO virtual address resolution, we support the idea that a shared last-level cache is necessary for heterogeneous architectures using shared virtual addressing. Usually, scratchpad-based accelerators rely on DMA engines to amortize long latency accesses to DRAM. The presence of an LLC may lower the accelerator's bandwidth by reducing long bursts to the length of a cache line. However, due to the presence of the IOMMU, the accelerator now also needs to access page table entries at a low granularity

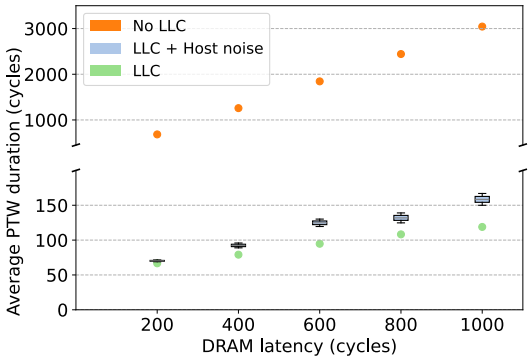

Fig. 5: Average IOMMU page table walk time with and without LLC and host interference for increasing DRAM latencies.

with low latency. To facilitate such accesses, we propose to leverage a shared LLC, caching only host and IOMMU PTW memory transactions. We show in Figure 4 and Table II the effect of this LLC on the performance. For all selected kernels and at all memory latency, the IOMMU overhead is now lower than 2% of the total runtime.

To ensure that the LLC is only used by the host subsystem, the IOMMU, and not the DMA engine, we use a pair of demux and mux visible in Figure 1. These architectural blocks allow to remap the same DRAM addresses to two bus addresses separated by a fixed offset. With this offset, the DMA engine can access input data in bursts larger than the cache line and write output data without invalidating unrelated host data. Similar bypass regions can be implemented on network-on-chip (NoC) platforms [18]. However, using uncached DMA transactions requires the host to self-invalidate data from the LLC when the device needs to access it as shown in Listing 1. Note that by flushing the LLC before mapping the IO virtual addresses, we maximize the LLC hits for the device IOMMU.

C. Page table walking time and LLC impact

In this part, we confirm the benefits of sharing a last-level cache for the IOMMU and host. Similarly to the previous section, we focus on runtimes and overhead on the device. We measure the average IO PTW time when running the `axpy` kernel with the shared LLC when it is concurrently stressed by host memory traffic. We create this interference by issuing a synthetic random memory from the host during the accelerator's execution. Figure 5 shows the average IO page table walk time under varying DRAM latencies. We observe that using a shared LLC between the accelerator and host domains significantly improves the IOMMU performance by reducing the PTW time by $15\times$ on average. Indeed, thanks to the LLC, the average page table walk time does not exceed 200 clock cycles, even for 1000 cycles of DRAM latency. The page table entries have a high chance of being cached in the LLC as the mapping operation is performed by the host core right before offload, following the offload model shown in Listing 1. This reduces the probability of compute and energy expensive LLC misses.

However, Figure 5 shows that the activity of the host may also affect the IOMMU address translation performance due to interference on the system bus and concurrent evictions in the LLC. We measure in average a $20\,\%$ slowdown of the PTW time when CVA6 frequently accesses memory. Although such interference must be taken into consideration when evaluating the feasibility of implementing IOMMU-based shared virtual memory heterogeneous systems, the LLC still provide considerable performance improvements without requiring any custom support in the IOMMU.

V. Related works

Heterogeneous SoCs, often referred to as APUs, early adopted shared virtual addressing via IOMMU hardware units to enhance programmability and memory usage. Several works evaluated the key performance bottlenecks [19] [20] induced by the additional IO page walking required to translate IO virtual addresses to physical addresses. For example, GPUs typically rely on deep cache-hierarchy and latency tolerant single-instruction-multiple-threads (SIMT) to maximize compute utilization. Upon L2-cache misses, GPUs access DRAM via a shared LLC with the CPU. Conversely, multiple-instructions-multiple-data (MIMD) accelerator clusters typically rely on scratchpad memories and DMA-engines to refill contiguous data chunks from DRAM [21]. In this scenario, different IOMMU architectures have been proposed over the years, such as multi-level IOTLBs [9], enabling sharing of commonly accessed page table entries (PTEs) among different accelerators, significantly increasing performance on shared data access patterns.

On the other end of the spectrum, previous works also explored IOMMU-less solutions to enable shared virtual addressing to improve flexibility and reduce area and power consumption. A former study from Kurth et al. [22] proposes a hybrid architecture for heterogeneous systems-on-chip based on programmable accelerators, in which page table walking and prefetching are executed by software threads running on the accelerator cluster itself. This solution enables high versatility by tuning the proportion of page walker threads per compute thread for each kernel. Nevertheless, on systems as the one described in section III, software-based page table walking could waste precious computational resources such as the double-precision floating-point unit (FPU) coupled to each PE. To reduce energy consumption, [23] proposes a hardware-managed active forwarding from the host data cache to the accelerator's SPM. This solution is reported to reduce energy utilization significantly for simple dataflow accelerators. However, active forwarding is limited to problem sizes that can entirely fit within the (typically limited) on-chip accelerator scratchpad memory. While these limitations in terms of flexibility can be tolerated on certain DSAs with highly predictable execution patterns, accelerators based on programmable PEs justify the use of dedicated hardware IOMMU between the PMCA and the system bus.

Within the context of IOMMU-based approaches, Ben-Yehuda et al. [24] identified two types of performance overheads: CPU utilization and memory bandwidth. In the case of low memory bandwidths, hardware solutions have been proposed to limit the IOTLB wall. TLB coalescing [25] can be used in the host MMU to reduce the number of PTWs without the need of superpages. This method can be similarly applied to IOMMUs. In [10], the authors propose to exploit the locality of IO page table entries in cache lines for efficient coalescing. To adress the problem of CPU overhead. In this case, software solutions can be used to reduce the cost of mapping and un-mapping DMA buffers. In [26], the authors propose a DMA allocator specially designed to reuse IOMMU mappings.

In this work, we focus on the RISC-V IOMMU proposed in [7] coupled with a scratchpad-based programmable accelerator cluster with DMA engine. We benchmark the page translation overhead on the accelerator under different memory latencies to exhibit the high cost of shared virtual addressing. We also show that by integrating a last-level cache shared by the IOMMU and the host, page walking overhead is highly reduced, making shared virtual addressing suitable without custom IOTLB coalescing or prefetching. However, to avoid lowering the accelerator's bandwidth, the shared LLC must bypass DMA transactions. This bypass is implemented in platforms such as ESP [18] to enable high bandwidth within network-on-chips. Our work shows that such memory hierarchy is also required to implement shared virtual addressing on heterogeneous platforms efficiently.

VI. Conclusion

In this work, we evaluate the performance overhead of shared virtual addressing on an open source RISC-V heterogeneous SoC. Our prototype platform contains a 64-bit CVA6 core running Linux and an 8-cores Snitch cluster. We implement several Linux userspace heterogeneous kernels using OpenMP and benchmark them under variable off-chip memory latencies. We show that IOMMU-based IOVA translation can significantly reduce the effective memory bandwidth for the accelerator and contribute to up to 17.6% of the accelerator's execution time for a `gemm` kernel. However, we also show that this issue can be mitigated without any changes to the IOMMU by integrating a last-level cache shared by the host and the IO page table walker. While classical programmable manycore accelerator architectures rely on direct DRAM access via DMA and do not need an LLC, we show that this hardware unit reduces address translation overhead to less than 2% for all kernels benchmarked, even under high memory latency. We also propose to bypass the shared cache for the device DMA as it may lower the effective DMA bandwidth under large bursts. In these conditions, and using the open source RISC-V IOMMU from [7], we show that shared virtual addressing and zero-copy offloading is suitable for heterogeneous RISC-V systems-on-chip.

VII. Acknowledgment

This work has been supported in part by the EPI SGA2 project which received funding from the European High-Performance Computing Joint Undertaking (JU) (Specific Grant Agreement No 101036168), and the TRISTAN project which received funding from the HORIZON KDT-JU program (Grant Agreement No 101095947).

References

[1] Nvidia Corporation, "Introducing Jetson Xavier NX, the World's Smallest AI Supercomputer." Available: https://developer.nvidia.com/blog/jetson-xavier-nx-the-worlds-smallest-ai-supercomputer/.

[2] G. H. e. A. Loh, "A Research Retrospective on AMD's Exascale Computing Journey," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, (New York, NY, USA), Association for Computing Machinery, 2023.

[3] RISC-V, "RISC-V IOMMU Architecture Specification." Available: https://github.com/riscv-non-isa/riscv-iommu/releases/tag/v1.0.0.

[4] D. R. Ditzel and the Esperanto team, "Accelerating ML Recommendation With Over 1,000 RISC-V/Tensor Processors on Esperanto's ET-SoC-1 Chip," *IEEE Micro*, vol. 42, no. 3, pp. 31–38, 2022.

[5] M. Thüning, "Attention in SRAM on Tenstorrent Grayskull," 2024.

[6] G. Paulin, P. Scheffler, T. Benz, M. Cavalcante, T. Fischer, M. Eggimann, Y. Zhang, N. Wistoff, L. Bertaccini, L. Colagrande, G. Ottavi, F. K. Gürkaynak, D. Rossi, and L. Benini, "Occamy: A 432-Core 28.1 DP-GFLOP/s/W 83

[7] B. S. Manuel Rodríguez, Francisco Costa and S. Pinto, "Open-source RISC-V Input/Output Memory Management Unit (IOMMU) IP," in *RISC-V Summit Europe, Barcelona, 5-9th June 2023*, 2023.

[8] Zero Day Labs, "RISC-V IOMMU IP." Available: https://github.com/zero-day-labs/riscv-iommu.

[9] Y. Hao, Z. Fang, G. Reinman, and J. Cong, "Supporting Address Translation for Accelerator-Centric Architectures," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 37–48, 2017.

[10] S. Shin, M. LeBeane, Y. Solihin, and A. Basu, "Neighborhood-Aware Address Translation for Irregular GPU Applications," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 352–363, 2018.

[11] A. Ottaviano, T. Benz, P. Scheffler, and L. Benini, "Cheshire: A Lightweight, Linux-Capable RISC-V Host Platform for Domain-Specific Accelerator Plug-In," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 10, pp. 3777–3781, 2023.

[12] F. Zaruba and L. Benini, "The Cost of Application-Class Processing: Energy and Performance Analysis of a Linux-Ready 1.7-GHz 64-Bit RISC-V Core in 22-nm FDSOI Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, 2019.

[13] A. Kurth, W. Rönninger, T. Benz, M. Cavalcante, F. Schuiki, F. Zaruba, and L. Benini, "An Open-Source Platform for High-Performance Non-Coherent On-Chip Communication," *IEEE TC*, 2022.

[14] F. Zaruba, F. Schuiki, T. Hoefler, and L. Benini, "Snitch: A tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads," *IEEE Transactions on Computers*, 2020.

[15] OpenMP Architecture Review Board, "OpenMP Application Program Interface Version 5.0," 2018.

[16] Tomasz Jeznach, "Linux RISC-V IOMMU Support."

[17] D. A. Beckingsale, J. Burmark, R. Hornung, H. Jones, W. Killian, A. J. Kunen, O. Pearce, P. Robinson, B. S. Ryujin, and T. R. Scogland, "RAJA: Portable Performance for Large-Scale Scientific Applications," 2019.

[18] J. Zuckerman, P. Mantovani, D. Giri, and L. P. Carloni, "Enabling Heterogeneous, Multicore SoC Research with RISC-V and ESP," *ArXiv*, vol. abs/2206.01901, 2022.

[19] J. Veselý, A. Basu, M. Oskin, G. H. Loh, and A. Bhattacharjee, "Observations and Opportunities in Architecting Shared Virtual Memory for Heterogeneous Systems," *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 161–171, 2016.

[20] S. Agarwal, R. Agarwal, B. Montazeri, M. Moshref, K. Elmeleegy, L. Rizzo, M. A. de Kruijf, G. Kumar, S. Ratnasamy, D. Culler, and A. Vahdat, "Understanding Host Interconnect Congestion," in *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*, HotNets '22, (New York, NY, USA), p. 198–204, Association for Computing Machinery, 2022.

[21] A. Kurth, A. Capotondi, P. Vogel, L. Benini, and A. Marongiu, "HERO: an Open-Source Research Platform for HW/SW Exploration of Heterogeneous Manycore Systems," in *Proceedings of the 2nd Workshop on AutotuniNg and ADaptivity AppRoaches for Energy Efficient HPC Systems*, ANDARE '18, (New York, NY, USA), Association for Computing Machinery, 2018.

[22] A. Kurth, P. Vogel, A. Marongiu, and L. Benini, "Scalable and Efficient Virtual Memory Sharing in Heterogeneous SoCs with TLB Prefetching and MMU-Aware DMA Engine."

[23] H.-C. Fu, P.-H. Wang, and C.-L. Yang, "Active Forwarding: Eliminate IOMMU Address Translation for Accelerator-rich Architectures," in *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, ACM.

[24] M. Ben-Yehuda, J. Xenidis, M. Ostrowski, K. Rister, A. Bruemmer, L. van Doorn, and D. Amd, "The Price of Safety: Evaluating IOMMU Performance," *Ottawa Linux Symposium (OLS)*, 01 2007.

[25] B. Pham, V. Vaidyanathan, A. Jaleel, and A. Bhattacharjee, "CoLT: Coalesced Large-Reach TLBs," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 258–269, 2012.

[26] A. Markuze, I. Smolyar, A. Morrison, and D. Tsafrir, "DAMN: Overhead-Free IOMMU Protection for Networking," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, (New York, NY, USA), p. 301–315, Association for Computing Machinery, 2018.