HERO: Heterogeneous Embedded Research Platform for Exploring RISC-V Manycore Accelerators on FPGA

Andreas Kurth
Pirmin Vogel

kurth@iis.ee.ethz.ch
vogel@iis.ee.ethz.ch
Integrated Systems Laboratory, ETH Zurich

Alessandro Capotondi

alessandro.capotondi@unibo.it
Microelectronics Research Group, University of Bologna

Andrea Marongiu
Luca Benini
marongiu@iis.ee.ethz.ch
benini@iis.ee.ethz.ch
Integrated Systems Laboratory, ETH Zurich
Microelectronics Research Group, University of Bologna

ABSTRACT

Heterogeneous embedded systems on chip (HESoCs) co-integrate a standard host processor with programmable manycore accelerators (PMCAs) to combine general-purpose computing with domain-specific, efficient processing capabilities. While leading companies successfully advance their HESoC products, research lags behind due to the challenges of building a prototyping platform that unites an industry-standard host processor with an open research PMCA architecture.

In this work we introduce HERO, an FPGA-based research platform that combines a PMCA composed of clusters of RISC-V cores, implemented as soft cores on an FPGA fabric, with a hard ARM Cortex-A multicore host processor. The PMCA architecture mapped on the FPGA is silicon-proven, scalable, configurable, and fully modifiable. HERO includes a complete software stack that consists of a heterogeneous cross-compilation toolchain with support for OpenMP accelerator programming, a Linux driver, and runtime libraries for both host and PMCA. HERO is designed to facilitate rapid exploration on all software and hardware layers: run-time behavior can be accurately analyzed by tracing events, and modifications can be validated through fully automated hardware and software builds and executed tests. We demonstrate the usefulness of HERO by means of case studies from our research.

CCS CONCEPTS

• **Computer systems organization** → *Multicore architectures*; *Heterogeneous (hybrid) systems*; *System on a chip*; *Embedded software*;

KEYWORDS

Heterogeneous SoCs, Multicore Architectures

ACM Reference Format:
Andreas Kurth, Pirmin Vogel, Alessandro Capotondi, Andrea Marongiu, and Luca Benini. 2017. HERO: Heterogeneous Embedded Research Platform for Exploring RISC-V Manycore Accelerators on FPGA. In *Proceedings of Computer Architecture Research with RISC-V Workshop, Boston, MA, USA, October 14, 2017 (CARRV'17).*

1 INTRODUCTION

Heterogeneous embedded systems on chip (HESoCs) are used in various application domains to combine general-purpose computing with domain-specific, efficient processing capabilities. Such architectures co-integrate a general-purpose host processor with programmable manycore accelerators (PMCAs). While leading companies continue to advance their products [14, 23, 24], computer architecture research on such systems lags behind: little is known on the internals of these products, and there is no research platform available that unites an industry-standard host processor with a modifiable and extensible PMCA architecture.

An important aspect of processors is their instruction set architecture (ISA), because it is the interface between software and hardware and ultimately determines their usability and performance in the system. The RISC-V ISA [32] has recently gained considerable momentum in the community [8, 12, 33] because it is an open standard and designed in a modular way: a small set of base instructions is accompanied by standard extensions and can be further extended through custom instructions [16]. This allows computer architects to implement the extensions suitable for their target application. Moreover, the ISA is suitable for various types of processors from tiny microcontrollers [27] to high-performance superscalar out-of-order cores , because it does not specify implementation properties. Combined, these characteristics make RISC-V an interesting candidate for specialized PMCAs.

There are many different PMCA architectures, such as Kalray MPPA [11], KiloCore [3], STHORM [19], Epiphany [21], and PULP [22]. PULP is an architectural template for scalable, energy-efficient processing that combines an explicitly-managed memory hierarchy, ISA extensions and compiler support for specialized DSP instructions, and energy-efficient cores operating in parallel to meet processing performance requirements. PULP is a silicon-proven [12], open [27] architecture implementing the RISC-V ISA and can cover a wide range of performance requirements by scaling the number of cores or adding domain-specific extensions. Thus, it is ideally suited to serve as a baseline PMCA in research on HESoCs.

Research on heterogeneous systems traditionally follows a two-pronged approach: hardware accelerators are developed and evaluated in isolation [9, 15], and their impact on system-level performance is estimated through models and simulators [4, 18]. Compared to implementing accelerators in prototype heterogeneous systems, this approach has significant drawbacks, however: First, interactions between host, accelerators, the memory hierarchy, and peripherals are complex to model accurately, making simulations orders of magnitude slower than running prototypes. Second, even full system simulators such as gem5 [2] model HESoCs to a limited degree only [6]. For example, models of system-level interconnects or memory management units (MMUs), which dynamically influence the path from accelerators to different levels of the memory hierarchy, are missing. Third, simulations are based on assumptions. Contrary to results obtained through implementation, simulated results burden authors and reviewers with having to justify and validate the underlying assumptions. Working prototypes, on the other hand, enable efficient, collaborative, and accurate computer architecture research and development, which can compete with industry's pace [17]. To perform *system*-level research using standard benchmarks and real-world applications, however, the system must additionally be efficiently programmable: a heterogeneous programming model and support for shared virtual memory (SVM) between host and PMCAs are indispensable.

In this work, we present HERO, the first (to the best of our knowledge) heterogenous manycore research platform. HERO combines an ARM Cortex-A host processor with a scalable, configurable, and extensible FPGA implementation of a silicon-proven, cluster-based PMCA (§ 2.1). HERO will be released open-source and includes the following core contributions:

- A heterogeneous software stack (§ 2.2) that supports OpenMP 4.5 and SVM for transparent accelerator programming, which tremendously simplifies porting of standard benchmarks and real-world applications and enables system-level research.
- Profiling and automated verification solutions that enable efficient hardware and software R&D on all layers (§ 2.3).

With up to 64 RISC-V cores running at more than 30 MHz on a single FPGA (§ 3.1), HERO's PMCA implementation is nominally capable of executing more than 1.9 GIPS and outperforms cycle-accurate simulation by orders of magnitude. We demonstrate HERO's capabilities by means of case studies from our research (§ 3.2 to § 3.4).

This work was partially funded by the EU's H2020 projects HERCULES (No. 688860) and OPRECOMP (No. 732631).
CARRV'17, October 14, 2017, Boston, MA, USA.

2 PLATFORM

In this section, we present first the hardware and then the software infrastructure of our research platform.

2.1 Hardware

HERO can be implemented on different hardware platforms consisting of a hard-IP, ARM Cortex-A host CPU and an FPGA fabric used to implement the PMCA. Fig. 1 gives an overview of the implementation of HERO on the Juno ARM Development Platform (Juno ADP), which will be discussed in detail in § 3.1. On all platforms, logic instantiated in the FPGA can access the shared main dynamic random access memory (DRAM) through a low-latency AXI interface coherently with the caches of the host. This qualifies the platforms for development and prototyping of tightly-integrated accelerators and the associated software infrastructure.

Programmable manycore accelerator (PMCA). As PMCA, HERO uses the latest version of the PULP platform [22]. PULP has been employed in multiple research application specific integrated circuits (ASICs) designed for parallel ultra-low power processing. To overcome scalability limitations, it uses a multi-cluster design and relies on multi-banked, software-managed scratchpad memories (SPMs) and lightweight, multi-channel direct memory access (DMA) engines instead of data caches. The 32b RISC-V processing elements (PEs) [12] within a cluster primarily operate on data present in the shared L1 SPM to which they connect through a low-latency, logarithmic interconnect. The PEs use the cluster-internal DMA engine to copy data between the local L1 SPM and remote SPMs or shared main memory. Transactions to main memory pass through the remapping address block (RAB) [28], which performs virtual-to-physical address translation based on the entries of an internal table, similar to the MMUs of the host CPU cores. This lightweight hardware block is managed in software directly on the PMCA [30]. The host and the PMCA can thus efficiently share virtual address pointers. As such, SVM substantially eases overall system programmability and enables efficient sharing of linked data structures in the first place.

Since HERO uses FPGA logic to implement the PMCA, it cannot reach the high clock frequency and energy efficiency of an ASIC implementation. Nevertheless, the performance of a fully integrated HESoC can be accurately determined: One option is to proportionally scale down the clock frequency of host and DRAM. Even more accurately, the provided tracing infrastructure (§ 2.3.1) can be used to monitor the interfaces of the PMCA, from which the effect of clock frequency ratio differences between PMCA, host, and DRAM can be calculated. The platform is perfectly suited for studying the system-level integration of a PMCA, developing heterogeneous software, and exploring architectural variations of the PMCA including, e.g., cluster-internal auxiliary processing units (APUs) and application-specific accelerators, hardware-managed caches and coherency protocols, interconnect topologies, and scalable system MMUs. In essence, the PMCA is composed of exchangeable and modifiable blocks and interfaces, and different architectures can be derived from our implementation to match individual research interests.

The PMCA is highly configurable. Tab. 1 gives an overview of the different configurability options currently supported. Besides the number of clusters and the number of PEs and SPM banks per cluster, the 32b RISC-V PEs themselves can be configured to trade off hardware resources and computing performance. The single-precision floating-point unit (FPU) can be private, moved to the APU to be shared among multiple PEs within a cluster or completely disabled. Similarly, the integer DSP extension unit, the divider, and the multiplier can be private or shared in the APU. In addition, different designs for the shared instruction caches (single- or multi-ported) and top-level interconnects (bus or network on chip) can be selected. Also the design of the RAB is configurable: The number of translation lookaside buffer (TLB) entries and levels as well as the architecture of the second-level TLB can be adjusted.

Table 1: Configuration options for HERO's PMCA

Component	Options
# Clusters	1, 2, 4, **8**
System-level interconnect	**Bus** or network on chip
# PEs per cluster	2, 4, **8**
FPU	Private, shared (APU), **off**
Integer DSP unit, divider, multiplier	**Private**, shared (APU)
L1 SPMs # banks	4, 8, **16**
L1 SPMs size [KiB]	32, 64, 128, **256**
L2 SPM size [KiB]	32, 64, 128, **256**
Instruction cache design	**Single-** or multi-ported
Instruction cache size [KiB]	2, 4, **8**
Instruction cache # banks	2, 4, **8**
RAB L1 TLB size	4, 8, 16, **32**, 64
RAB L2 TLB size	0, 256, 512, **1024**, 2048
RAB L2 TLB associativity	16, **32**, 64
RAB L2 TLB # banks	1, 2, **4**, 8

Bold and underlined values refer to implementations discussed in § 3.1.

2.2 Software

In this section, we describe the various components of HERO's software stack. Fig. 2 shows how the different software layers and components of host and PMCA interact. These components seamlessly integrate the PMCA into the host system and allow for transparent accelerator programming using OpenMP 4.5. The application developer can write and compile a single application source. Application kernels suitable for offloading to the PMCA can simply be encapsulated using the OpenMP `target` directive. The actual offload is then taken care of by the OpenMP runtime environment (RTE).

2.2.1 Heterogeneous OpenMP programming. To allow the host OpenMP RTE to actually perform an offload to the PMCA, a custom plugin that contains the PMCA-specific implementations of the generic application programming interfaces (APIs) is required [7]. For example, this plugin defines how the input and output variables specified in the `target` construct are passed between host and PMCA. HERO currently supports both copy-based and zero-copy offload semantics. The latter exploits the SVM capabilities of the platform to just pass virtual address pointers, thereby avoiding costly data copies to a physically contiguous, uncached section in main memory and enabling efficient sharing of linked data structures.

2.2.2 Heterogeneous cross compilation. Generating both host and PMCA binaries from a single application source requires a set of compiler extensions to build a heterogeneous cross-compilation toolchain based on GCC 5.2 [7]. When the compiler expands a `target` construct in the front end, a new function is outlined that is ultimately also compiled for the PMCA. This is achieved by first streaming the functions to be offloaded into a link-time optimization (LTO) object, which is then fed to a custom offload tool at link time. This tool first recompiles the functions for the target PMCA using the RISC-V back-end compiler and links all PMCA runtimes and libraries. It then creates the hooks for the offloadable functions required by the host OpenMP runtime. Finally, the tool packs everything inside a dedicated section in the host binary.

An additional compiler pass is used to instrument all load and stores of the PMCA to variables in SVM within the `target` constructs with calls to low-overhead macros [29]. These macros protect the PMCA from using invalid responses returned by the hardware in case of TLB misses in the RAB and interface with the virtual memory management (VMM) library [30].

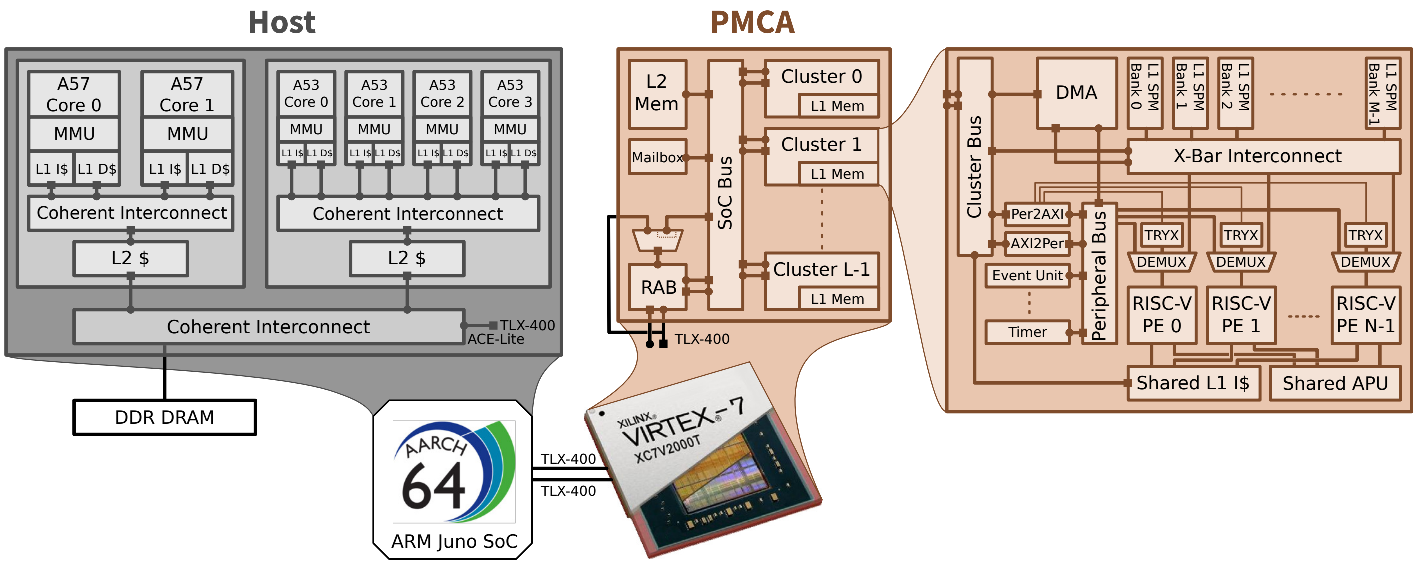


Figure 1: HERO's hardware, as implemented on the Juno ADP.

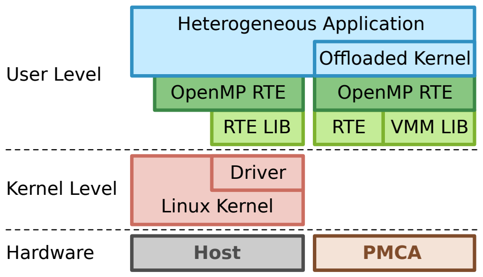


Figure 2: HERO's software stack.

2.2.3 Host runtime library and Linux driver. The host RTE library interfaces the host-side OpenMP runtime with the Linux driver. In addition, it is used to reserve all virtual addresses overlapping with the physical address map of the PMCA. This is required as any access of the PMCA to a shared variable located at such an address would not be routed to SVM but instead to its internal SPMs or memory-mapped registers. The driver handles low-level tasks such as interrupt handling, synchronization between PMCA and host, host cache maintenance, operation of the system-level DMA engine (e.g. to offload the PMCA binary), operating the profiling hardware, and initially setting up the RAB to give the PMCA access to the page table of the heterogeneous user-space application.

2.2.4 PMCA virtual memory management (VMM) library. Having access to the page table of the heterogeneous user-space application, the PMCA can operate its virtual memory hardware autonomously. A VMM library [30] on the PMCA abstracts away differences between host architectures and RAB configurations and provides a uniform API to explicitly map pages and handle RAB misses. When a core accesses virtual memory through the RAB, the corresponding address translation may not be configured. In this case, the core that caused the miss goes to sleep and the miss is enqueued in the RAB. To handle a miss, the VMM library dequeues it, translates its virtual address to a physical one by walking the page table of the host user-space process, selects a RAB table entry to replace and configures it accordingly, and wakes up the core that caused the miss. The VMM library is compatible with any host architecture supported by the Linux kernel.

2.3 Tools for Hardware and Software R&D

2.3.1 Event tracing and analysis. Fine-grained information on the run-time behavior of the PMCA in the HESoC is crucial for both hardware and software engineers to evaluate their designs and implementations. While simulations can provide first estimates, they do not accurately reproduce run-time behavior of HESoCs, as stated in the introduction. Instead, this information can be extracted by *tracing events* in the running HESoC prototype, which poses the following challenges: First, the tracer must not interfere with program execution; in particular, inserting instructions (e.g., to write memory) is not an option. Second, the tracer must be cycle-accurate to allow analysis of rapid, consecutive cause-effect events yet be able to handle measurements spanning millions of events to cover complex applications. Third, the tracer should use FPGA resources economically to not hamper the evaluation of complex hardware. Fourth, the tracer should not require modifications of applications, but should allow application-specific analyses.

HERO's event tracing solution is a hybrid design composed of lightweight tracer hardware blocks, which can be inserted anywhere on the FPGA, and a driver on the host. The customizable tracer hardware blocks are attached to signals and record their values as timestamped event when user-specified activation conditions are met. They store events in dedicated, local buffers implemented with block RAMs (BRAMs). When a buffer is full, the tracer hardware stops the PMCA by disabling its clock and raises an interrupt to delegate control to a driver on the host. The driver then reads out all events from the buffers to main memory, clears the buffers, and re-enables the clock of the PMCA. This process is entirely transparent to the PMCA, whose state is frozen during the transfer. The timestamps of all loggers are synchronized because they are driven from a common clock, which is disabled with the PMCA clock.

After an application terminated, all traced events are available in main memory for analysis. HERO's event analysis software processes the data in three layers. The first layer is generic: it reads the binary data from memory and creates a time-sorted list of events, which contain generic meta data and the ID of the tracer, and a collection of properties of the platform on which they were recorded. The second layer is measurement- and platform-specific. For example, if a logger traced memory accesses by cores, each of its generic events is converted to a read/write access to a memory address by one core; if a logger traced synchronization events, each is converted to a set of involved cores. The third layer is application-specific and optional: by linking run-time information such as memory accesses and synchronization events with knowledge about algorithms and data structures, questions about bottlenecks and how hardware and software design choices affect them can be answered very precisely.

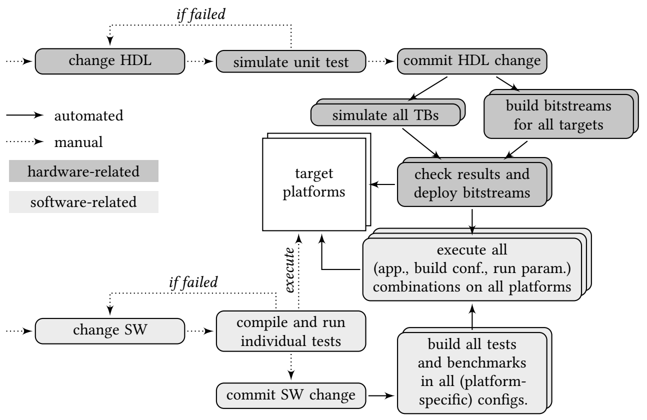

if failed
change HDL
simulate unit test
commit HDL change
automated
manual
hardware-related
software-related
simulate all TBs
build bitstreams for all targets
target platforms
check results and deploy bitstreams
execute all (app., build conf., run param.) combinations on all platforms
execute
if failed
change SW
compile and run individual tests
build all tests and benchmarks in all (platform-specific) configs.
commit SW change

Figure 3: HERO's automated HW/SW build and test flow.

2.3.2 Automated Implementation and Validation. Automated full-system builds and tests are a prerequisite for many effective development paradigms. In our case, the system consists of an entire heterogeneous hardware and software stack. As shown in Fig. 3, different parts of the stack have to be built and tested depending on the change: When modifying hardware, for example, we change code, evaluate the change with a unit test in the simulator, and commit the change once it passes that test. Then, an integration server simulates all testbenches and builds bitstreams for all target platforms (in parallel) with the commit applied. If, for a given target platform, all testbenches pass and the bitstream builds, the bitstream is deployed to the target platform. Finally, the integration server starts all target platforms with updated bitstreams, runs all test and benchmark applications in all configurations (more on this below) on all platforms, collects the results, and reports them to the developer.

The automated hardware builds are relatively simple: the PMCA hardware description contains synthesis conditionals and parameters, whose values are defined in a platform-specific configuration file. With the hardware description, this configuration file, and a uniform build script, Vivado generates platform-specific bitstreams.

The automated software builds and test runs are more complex: To maximally optimize every PMCA kernel for streamlined execution, the PMCA runtime is co-compiled with each PMCA application and statically linked into a single binary with the same *build parameters*. There are platform-specific build parameters, which are defined in one configuration file per platform, and each application comes with its own set of build parameters and *run arguments*, which are defined in an application-specific configuration file. Each application must specify on which target platform which builds can be executed with which run arguments. As the number of combinations grows exponentially with the number of parameters and arguments, listing all combinations manually would be redundant, error-prone work. Instead, the combinations are specified in a compact, graph-based notation. By flattening the graph, the integration server obtains the list of platform-application-parameter combinations, which are then built and executed automatically.

3 EVALUATION

In this section, we describe the currently supported platforms and how their FPGA resources are used (§ 3.1), demonstrate exploration of parallel execution and memory hierarchy usage (§ 3.2), show the positive impact of SVM on the total PMCA run time (§ 3.3), and give examples how event tracing and analysis can be used to efficiently and accurately validate and characterize run-time behavior (§ 3.4).

3.1 Supported Platforms

HERO is currently implemented on two different development platforms, and we are working on an implementation on the next-generation Xilinx Zynq UltraScale+ MPSoC. Porting HERO to a new Xilinx platform is an effort of approximately two man weeks.

Juno ARM Development Platform (Juno ADP). The Juno ADP features an ARMv8-based, multi-cluster host CPU (two A57 and four A53 cores), a Mali-T624 graphics processing unit (GPU), and 8 GiB of DDR3L DRAM. In addition, the system on chip (SoC) offers a low-latency AXI chip-to-chip interface (TLX-400) connecting to a Xilinx Virtex-7 FPGA, through which 4 to 8 PMCA clusters on the FPGA can access the shared DRAM coherently with the caches of the host. The ARMv8 host CPU runs 64b Linaro Linux 4.5 with a 64b root filesystem (both `aarch64-linux-gnu`) generated using the OpenEmbedded build system. We have configured the root filesystem to have multilib support, such that the host can also execute 32b binaries (`arm-linux-gnueabihf`) in ARMv7 mode, which guarantees compatibility of data and pointer types between the host and the 32b PMCA architecture in heterogeneous applications.[1]

Xilinx Zynq ZC706 Evaluation Kit (ZC706). The Xilinx Zynq-7045 SoC found on the ZC706 combines an ARMv7, dual-core A9 host CPU with a Kintex-7 FPGA on a single chip. The two subsystems are connected through a set of low-latency AXI interfaces and share 1 GiB of DDR3 DRAM. Using the Accelerator Coherency Port (ACP), the single PMCA cluster instantiated in the FPGA can also coherently access data from the data caches of the host. The main advantages of the ZC706 is higher availability and better affordability compared to the Juno ADP. The 32b ARMv7 host CPU runs Xilinx Linux 3.18 with a root filesystem generated using Buildroot.

FPGA resource utilization. Tab. 2 shows the FPGA resource utilization of the PMCA on the two development platforms in terms of lookup table (LUT) slices, flip-flops (FFs), DSP slices, and BRAM cells. The table lists both the absolute and the relative usage of the clusters and the top-level module containing also the host interfaces. The configuration parameters selected for implementation are highlighted as bold and underlined text Tab. 1 for the Juno ADP and the ZC706, respectively. The clusters dominate resource usage: 8 clusters on the Juno ADP and 1 cluster on the ZC706 account for more than 90% and 80% of the total resource usage, respectively. While LUT and DSP slices scale linearly from the single cluster on the ZC706 to the 8 clusters on the Juno ADP, BRAMs and FFs behave differently due to different instruction cache designs: the larger, single-ported cache on the Juno ADP uses more FFs and less BRAMs per cluster than the multi-ported cache on the ZC706. Neither configuration includes FPUs, and the integer data path alone uses relatively little DSP slices, even though it supports multiplication and division. The top-level configuration is identical for both platforms, with the exception of the different interfaces to the host and the number of clusters, which enlarges the registered SoC bus. On both platforms, the available LUTs and BRAMs are the limiting factors. The PMCA can be clocked at 31 MHz and 57 MHz on the Juno ADP and the ZC706, respectively. The difference is due to the denser utilization of the Juno ADP and the fact that the Virtex-7 FPGA of the Juno ADP consists of multiple dies connected through stacked silicon interconnects.

3.2 Case Study: Parallel Speedup Analysis

To demonstrate the parallel execution and data transfer capabilities of the PMCA, we use a matrix-matrix multiplication benchmark. The computations for calculating the product of two matrices, $C = AB$, are distributed over the clusters by tiling A and C row-wise. Each cluster iterates over

[1]This compatibility could also be achieved by running the application binary in ILP32 mode, which would allow the host to use ARMv8-specific CPU features. However, the support for ILP32 is still experimental in Linaro.

Table 2: PMCA FPGA resource utilization

		Juno ADP		ZC706	
All Clusters	LUT	936 k	76%	128 k	59%
	FF	450 k	18%	43 k	10%
	DSP	384	18%	48	5%
	BRAM	1152	89%	384	70%
Top Level and Host Interface	LUT	70 k	6%	24 k	11%
	FF	61 k	2%	26 k	6%
	DSP	0	0%	0	0%
	BRAM	75	6%	71	13%

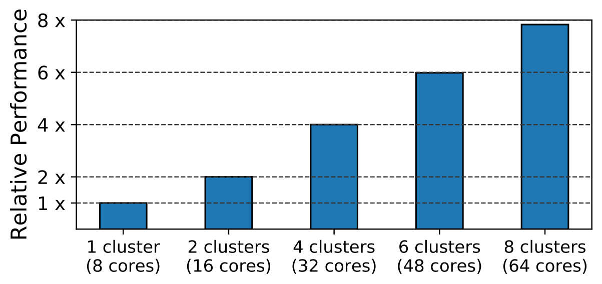


Figure 4: Overall execution speedup by parallelizing matrix-matrix multiplication.

its rows and parallelizes each row block-wise over its cores: it transfers a row of A and a column of B from the DRAM to its local L1 SPM banks, computes the resulting row of C into its L1 SPM, and transfers the resulting row to the DRAM.

Fig. 4 shows the speedup achieved when parallelizing the workload over multiple clusters. In the baseline (leftmost bar), a single cluster performs the work. The bars to the right of the baseline are for two, four, six, and eight clusters. Parallelizing execution over two, four, and six clusters leads to ideal speedups compared to the baseline. For eight clusters, the interconnect between the clusters becomes the bottleneck to data transfers and limits the speedup to ca. 2% below the ideal value. In the evaluated implementation, the interconnect is a bus, which provides low latency but no scalable bandwidth. A network on chip, which is the other option for the interconnect between the clusters, scales in bandwidth and can thus, depending on the target workload, reduce the overall execution time by supporting parallel data transfers for even more PEs.

3.3 Case Study: Virtual Memory Performance Analysis

SVM support in the PMCA is essential for efficient data sharing between host and the PMCA: Without SVM, data must be copied to and from a dedicated, physically-contiguous, uncached memory section before and after accelerator execution, respectively. This copy operation depends on the data structure and may be very complex; e.g., the values of all pointers in a linked data structure must be changed. With SVM, offloading simply means passing a pointer.

Fig. 5 shows the run time of different benchmarks executed on the PMCA with (orange, right bar in a pair) and without SVM (blue, left bar in a pair). The run time is broken down into offload time, i.e., the time it takes the host to offload the computation and prepare the data for the PMCA, and the actual kernel execution time on the PMCA. All times are normalized to the total run time without SVM. **PageRank (a)** is a well-known algorithm for analyzing the connectivity of graphs and is used, e.g., for ranking web sites. It is based on a linked data structure (LDS),

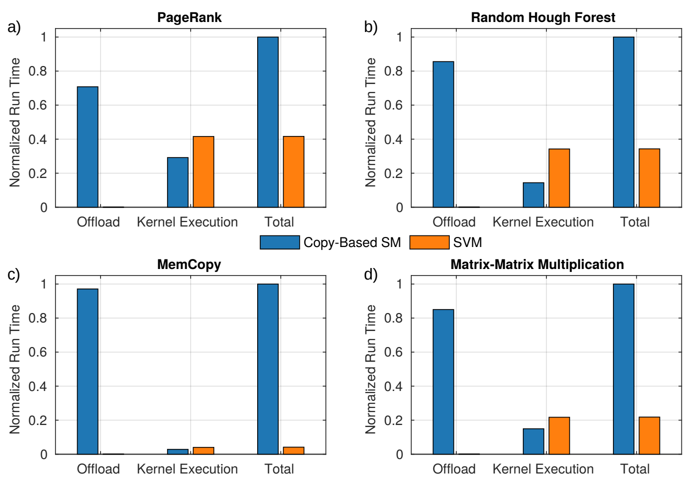
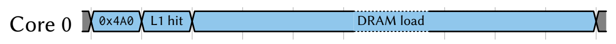
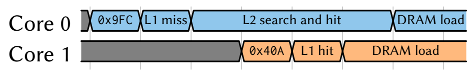


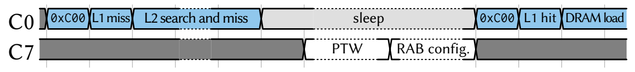
Figure 5: Offload and kernel execution time for different benchmarks with and without SVM support.



(a) RAB L1 TLB behavior and DRAM access latency.



(b) RAB L1 hit-under-miss behavior and L2 TLB behavior.



(c) RAB miss and miss handling (PTW and RAB reconfiguration).

Figure 6: Memory access events by different cores at the RAB. For space efficiency, only the LSBs of each VA are shown and events that take many clock cycles to complete have been compressed, as indicated by dots.

which makes copy-based offloading expensive because the host must modify many pointers. With SVM, virtual addresses must be translated at run time. This causes a run time overhead if translations are not in the TLB of the RAB. Nonetheless, the offload time of copy-based SM dominates, and SVM reduces the run time by nearly 60%. **Random Hough Forests (b)** consist of multiple binary decision trees and are used, e.g., for image classification. The trees have a very large memory footprint, but only a part of them is accessed, depending on the input data. With SVM, the PMCA can readily access the entire trees by performing the necessary address translations at run time. With copy-based SM, the trees must be made available to the PMCA in their entirety before classification can start. This leads to a lot of data being copied by the host that is never accessed by the PMCA. SVM reduces the run time by more than 60%. **MemCopy (c)** simply copies a large array into the PMCA and back to memory. This benchmark is representative of streaming applications that require the PMCA to perform only little work. With copy-based SM, letting the host copy data to and from the physically contiguous, uncached memory to prepare the offload clearly dominates the run time. In contrast, the PMCA benefits from high-bandwidth DMA transfers. SVM removes the need for data copying by the host, reducing the total run time by more than 95%. The **matrix-matrix multiplication benchmark (d)** involves three matrices stored in arrays, thus shows the same basic behavior as MemCopy. However, as the PMCA performs computations while traversing the data, the copy-based offload becomes a lesser part of the total run time. In this case, SVM reduces the total run time by nearly 80%.

3.4 Case Study: Advanced Event Tracing

To evaluate different aspects of the interaction between PMCA and host, we inserted event tracers on the AXI read and write request and response channels of the RAB as well as on its configuration port. We then executed different short programs on the PMCA to stimulate different behaviors of the memory subsystem. The AXI tracers recorded raw memory access requests and responses and identified the related core through the AXI ID. Finally, we used the event analysis tool to convert that data into series of events per core for each program, as shown in Fig. 6.

In the first program (a), a single core loads data from the DRAM through a virtual address (VA) that is in the L1 TLB of the RAB. After a single-cycle address translation, the load is passed to the DRAM. With the PMCA running at much lower clock frequencies than it would in a silicon implementation, this load takes fewer cycles (an average of 7.8 at 20 MHz) than it would in a real HESoC, which would distort performance evaluations. By tracing all memory accesses in the execution of a benchmark, however, performance can be accurately determined by multiplying all access latencies with the implementation-to-emulation clock ratio.

In the second program (b), one core accesses a VA that misses in the L1 TLB, which triggers a multi-cycle search in the L2 TLB. While the L2 TLB is being searched, a second core accesses a VA that is in the L1 TLB and is indeed translated within a single clock cycle. To verify that this hit-under-miss behavior is always maintained, the analyzer supports definable assertions. Additionally, the number of clock cycles taken to find a VA in the L2 TLB can be used to evaluate different placement strategies in the set-associative L2 TLB.

In the third program (c), a core accesses a VA that misses in both the L1 and the L2 TLB, upon which it reports the miss to another core and goes to sleep. The other core handles the miss through the VMM library by walking the page table and configuring a L1 TLB entry to translate that VA. It then wakes the first core, which retries the memory access and proceeds with the load. We used this to evaluate our VMM implementation on the PMCA in [30].

4 RELATED WORK

HERO extends the principle of prototyping computer architectures on FPGAs to HESoCs. In the FPGA Architecture Model Execution (FAME) taxonomy [25], HERO is a Direct FAME system, meaning it implements the target architecture with a one-to-one correspondence in clock cycles on an FPGA. More sophisticated FAME levels decouple timing and functionality, exchange structural equivalence for modeling abstractions, and share FPGA resources in time between components of the target architecture to increase model flexibility and emulation throughput. An example of a sophisticated FAME system is RAMP Gold [26], which is designed for the rapid early-design-space exploration of manycore systems. It is cycle-accurate and comparable in throughput to HERO, but requires the development of a behavioral model that is not directly used in the silicon implementation. In contrast to highly sophisticated FAME systems, HERO is not designed for early-stage design explorations but for the evaluation, advancement, and extension of a proven PMCA template and for studying the integration of PMCAs in a HESoC. By staying as close to the silicon implementation as possible, co-development and maintenance of separate models are avoided. Commercial Direct FAME systems, such as Cadence Palladium and MentorGraphics Veloce, are targeted at the verification of entire ICs. To reach the required capacity, they employ custom logic simulation engines and highly intrusive tracing systems in addition to FPGAs. They come with proprietary tools and protective licenses at very high costs, which bars the vast majority of the research community from using them.

The Flexible Architecture Research Machine (FARM) [20] is a system for prototyping custom hardware implemented on an FPGA that is connected to an AMD multiprocessor. Both FARM and HERO provide a cache-coherent link to the host processor and data transfer (or DMA) engines. While FARM leaves the task of implementing an accelerator from scratch and integrating it with the system to the researcher, HERO comes with a RISC-V manycore implementation, a heterogeneous toolchain, and tools to allow efficient hardware and software research using standard benchmarks and real-world applications.

Intel uses FPGAs to prototype heterogeneous—in their definition two sets of cores of the same ISA but different power-performance design points—architectures [10, 31]. They combine a Xeon [31] and an Atom [10] CPU with an FPGA on which they implement up to four "very old" [10] Pentium 4 cores. While an evaluation platform with a Xeon and an Atom CPU (both hardwired) was shared with selected academic partners, the reconfigurable, FPGA-based prototypes remain restricted to Intel [10]. HERO, on the other hand, is openly available, implements a modern RISC-V manycore on an FPGA, and uses the extended concept of heterogeneity with different ISAs.

HERO is more than a PMCA implemented on an FPGA, but its PMCA implementation is nonetheless related to the following recent works: OpenPiton [1] is the first open-source, multithreaded manycore processor and is available in FPGA implementations for prototyping. Our PMCA implementation on the FPGA differs from that of OpenPiton in two ways: First, our PMCA implements the RISC-V ISA, which has recently gained a lot of momentum. Second, it allows evaluation on the FPGA with more cores: we currently support up to 64 cores compared to OpenPiton's maximum of 4 cores (both on a Xilinx Virtex-7, albeit of different size). GRVI Phalanx [13] is an array of clusters of RISC-V cores interconnected by a network on chip (NoC). Cores, clusters, and the NoC are optimized for FPGAs and utilize FPGA blocks very efficiently, allowing to implement hundreds of RV32I cores on a mid-range FPGA. While FPGAs are *the* design target of GRVI Phalanx, HERO uses FPGAs as a prototyping target to support a wide range of implementation targets and architectural exploration. Moreover, GRVI Phalanx is programmed bare-metal, whereas the PMCA on HERO comes with a runtime that supports well-established programming paradigms such as OpenMP including seamless accelerator integration. lowRISC [5] is a work-in-progress open-source SoC implementing the RISC-V ISA. Its goal is to lower the barrier of entry to producing custom silicon by establishing an ecosystem of IP blocks around RISC-V cores. In contrast, HERO aims to facilitate exploration on all layers of software and hardware in HESoCs by implementing a modifiable, working full-stack prototype accompanied by tools for validation and evaluation of novel concepts.

5 CONCLUSION

We presented HERO, the first heterogeneous manycore research platform, which unites an ARM Cortex-A host processor with a fully modifiable RISC-V manycore implemented on an FPGA. Our heterogeneous software stack, which supports SVM and OpenMP 4.5, tremendously simplifies porting of standard benchmarks and real-world applications, thereby enabling system-level research. Our profiling and automated verification solutions enable efficient hardware and software research on all layers. We have been successfully using HERO in our research over the last years and will continue its development. To further advance the research community, we are currently working towards releasing HERO under an open-source license on `pulp-platform.org/hero`.

REFERENCES

[1] J. Balkind *et al.* 2016. OpenPiton: An Open Source Manycore Research Framework. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16).*

[2] N. Binkert *et al.* 2011. The gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2, 1–7. https://doi.org/10.1145/2024716.2024718

[3] B. Bohnenstiehl *et al.* 2017. KiloCore: A 32-nm 1000-Processor Computational Array. *IEEE Journal of Solid-State Circuits (JSSC)* 52, 4, 891–902.

[4] D. Bortolotti *et al.* 2016. VirtualSoC: A Research Tool for Modern MPSoCs. *ACM Trans. Embed. Comput. Syst.* 16, 1, 27 pages. https://doi.org/10.1145/2930665

[5] A. Bradbury *et al.* 2014. Tagged Memory and Minion Cores in the lowRISC SoC. http://www.lowrisc.org/downloads/lowRISC-memo-2014-001.pdf

[6] A. Butko *et al.* 2016. Full-System Simulation of big.LITTLE Multicore Architecture for Performance and Energy Exploration. In *IEEE MCSOC.* 201–208. https://doi.org/10.1109/MCSoC.2016.20

[7] A. Capotondi and A. Marongiu. 2017. Enabling Zero-copy OpenMP Offloading on the PULP Many-core Accelerator. In *Proceedings of the 20th International Workshop on Software and Compilers for Embedded Systems (SCOPES '17).* ACM, New York, NY, USA, 68–71. https://doi.org/10.1145/3078659.3079071

[8] C. Celio *et al.* 2015. The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-167.html

[9] T. Chen *et al.* 2015. A High-Throughput Neural Network Accelerator. *IEEE Micro* 35, 3, 24–32. https://doi.org/10.1109/MM.2015.41

[10] N. Chitlur *et al.* 2012. QuickIA: Exploring Heterogeneous Architectures on Real Prototypes. In *IEEE HPCA.* 1–8. https://doi.org/10.1109/HPCA.2012.6169046

[11] B. D. de Dinechin *et al.* 2013. A Clustered Manycore Processor Architecture for Embedded and Accelerated Applications. In *IEEE HPEC.* 1–6. https://doi.org/10.1109/HPEC.2013.6670342

[12] M. Gautschi *et al.* 2017. Near-Threshold RISC-V Core With DSP Extensions for Scalable IoT Endpoint Devices. *IEEE TVLSI* PP, 99, 1–14. https://doi.org/10.1109/TVLSI.2017.2654506

[13] J. Gray. 2016. GRVI Phalanx: A Massively Parallel RISC-V FPGA Accelerator Accelerator. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM).* 17–20. https://doi.org/10.1109/FCCM.2016.12

[14] J. Ho. 2016. NVIDIA's Tegra Parker SoC. *AnandTech.* http://www.anandtech.com/show/10596

[15] D. Johnson *et al.* 2011. Rigel: A 1,024-Core Single-Chip Accelerator Architecture. *IEEE Micro* 31, 4, 30–41. https://doi.org/10.1109/MM.2011.40

[16] D. Kanter. 2016. RISC-V Offers Simple, Modular ISA: New CPU Instruction Set Is Open and Extensible. https://riscv.org/2016/04/risc-v-offers-simple-modular-isa

[17] Y. Lee *et al.* 2016. An Agile Approach to Building RISC-V Microprocessors. *IEEE Micro* 36, 2, 8–20. https://doi.org/10.1109/MM.2016.11

[18] S. Li *et al.* 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *IEEE/ACM MICRO.* 469–480.

[19] D. Melpignano *et al.* 2012. Platform 2012, a Many-core Computing Accelerator for Embedded SoCs: Performance Evaluation of Visual Analytics Applications. In *Proceedings of the 49th Annual Design Automation Conference (DAC '12).* ACM, New York, NY, USA, 1137–1142. https://doi.org/10.1145/2228360.2228568

[20] T. Oguntebi *et al.* 2010. FARM: A Prototyping Environment for Tightly-Coupled, Heterogeneous Architectures. In *IEEE FCCM.* 221–228. https://doi.org/10.1109/FCCM.2010.41

[21] A. Olofsson. 2016. Epiphany-V: A 1024 processor 64-bit RISC System-On-Chip. *CoRR* abs/1610.01832. http://arxiv.org/abs/1610.01832

[22] D. Rossi *et al.* 2014. Energy efficient parallel computing on the PULP platform with support for OpenMP. In *IEEEI.* 1–5. https://doi.org/10.1109/EEEI.2014.7005803

[23] R. Smith. 2017. Apple's A10X SoC. *AnandTech.* http://www.anandtech.com/show/11596

[24] R. Smith. 2017. Samsung's Exynos 8895 SoC. *AnandTech.* http://www.anandtech.com/show/11149

[25] Z. Tan *et al.* 2010. A Case for FAME: FPGA Architecture Model Execution. In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10).* ACM, New York, NY, USA, 290–301. https://doi.org/10.1145/1815961.1815999

[26] Z. Tan *et al.* 2010. RAMP Gold: An FPGA-based Architecture Simulator for Multiprocessors. In *Proceedings of the 47th Design Automation Conference (DAC '10).* ACM, New York, NY, USA, 463–468. https://doi.org/10.1145/1837274.1837390

[27] A. Traber *et al.* 2016. PULPino: A small single-core RISC-V SoC. https://riscv.org/wp-content/uploads/2016/01/Wed1315-PULP-riscv3_noanim.pdf

[28] P. Vogel *et al.* 2015. Lightweight Virtual Memory Support for Many-core Accelerators in Heterogeneous Embedded SoCs. 45–54. http://dl.acm.org/citation.cfm?id=2830840.2830846

[29] P. Vogel *et al.* 2016. Lightwight Virtual Memory Support for Zero-Copy Sharing of Pointer-Rich Data Structures in Heterogeneous Embedded SoCs. *IEEE TPDS* 28, 7, 1947–1959.

[30] P. Vogel *et al.* 2017. Efficient Virtual Memory Sharing via On-Accelerator Page Table Walking in Heterogeneous Embedded SoCs. *ACM TECS* PP, 99, 1–19.

[31] Q. Wang *et al.* 2010. An FPGA Based Hybrid Processor Emulation Platform. In *FPL.* 25–30. https://doi.org/10.1109/FPL.2010.16

[32] A. Waterman. 2016. Design of the RISC-V Instruction Set Architecture. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-1.html

[33] B. Zimmer *et al.* 2016. A RISC-V Vector Processor With Simultaneous-Switching Switched-Capacitor DC-DC Converters in 28 nm FDSOI. *IEEE JSSC* 51, 4, 930–942. https://doi.org/10.1109/JSSC.2016.2519386