HERO: an Open-Source Research Platform for HW/SW Exploration of Heterogeneous Manycore Systems

Author(s):
Kurth, Andreas; Capotondi, Alessandro; Vogel, Pirmin; Benini, Luca; Marongiu, Andrea

Publication date:
2018-11-04

Permanent link:
https://doi.org/10.3929/ethz-b-000314220

Originally published in:
https://doi.org/10.1145/3295816.3295821

HERO: an Open-Source Research Platform for HW/SW Exploration of Heterogeneous Manycore Systems

Andreas Kurth
IIS, ETH Zürich
8092 Zürich, Switzerland
akurth@iis.ee.ethz.ch

Alessandro Capotondi
DEI, University of Bologna
40136 Bologna, Italy
alessandro.capotondi@unibo.it

Pirmin Vogel
IIS, ETH Zürich
8092 Zürich, Switzerland
vogelpi@iis.ee.ethz.ch

Luca Benini
IIS, ETH Zürich
8092 Zürich, Switzerland
benini@iis.ee.ethz.ch

Andrea Marongiu
DISI, University of Bologna
Bologna, Italy
a.marongiu@unibo.it

ABSTRACT

Heterogeneous systems on chip (HeSoCs) co-integrate a high-performance multicore host processor with programmable manycore accelerators (PMCAs) to combine "standard platform" software support (e.g. the Linux OS) with energy-efficient, domain-specific, highly parallel processing capabilities.

In this work, we present HERO, a HeSoC platform that tackles this challenge in a novel way. HERO's host processor is an industry-standard ARM Cortex-A multicore complex, while its PMCA is a scalable, silicon-proven, open-source many-core processing engine, based on the extensible, open RISC-V ISA.

We evaluate a prototype implementation of HERO, where the PMCA implemented on an FPGA fabric is coupled with a hard ARM Cortex-A host processor, and show that the run time overhead compared to manually written PMCA code operating on private physical memory is lower than 10 % for pivotal benchmarks and operating conditions.

CCS CONCEPTS

• **Computer systems organization → Parallel architectures**; **Heterogeneous (hybrid) systems**; *System on a chip*; • **Hardware → Simulation and emulation**; *Reconfigurable logic and FPGAs*; • **Software and its engineering → Parallel programming languages**.

KEYWORDS

Heterogeneous SoCs, Multi- and Many-core Architectures, Shared Virtual Memory

ACM Reference Format:
Andreas Kurth, Alessandro Capotondi, Pirmin Vogel, Luca Benini, and Andrea Marongiu. 2018. HERO: an Open-Source Research Platform for HW/SW Exploration of Heterogeneous Manycore Systems. In *2nd Workshop on AutotuniNg and aDaptivity AppRoaches for Energy efficient HPC Systems (ANDARE'18), November 4, 2018, Limassol, Cyprus.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3295816.3295821

ANDARE'18, November 4, 2018, Limassol, Cyprus

ACM ISBN 978-1-4503-6591-8/18/11…$15.00
https://doi.org/10.1145/3295816.3295821

1 INTRODUCTION

The continous negotiation between power consumption, performance, programmability, and portability drives all computing industry designs, in particular in the embedded systems domain. Heterogeneous systems on chip (HeSoCs) are nowadays widely adopted to address such challenges combining on the same device general-purpose computing with domain-specific, efficient processing capabilities. HeSoCs typically co-integrate a host multiprocessor processor, optimized for average performance of general, single-threaded and weakly multi-threaded workloads, with programmable manycore accelerators (PMCAs), each optimized for domain-specific, highly parallel workload kernels.

The first-class integration of domain-specific architectures (DSAs) is expected to enable scaling computing performance in face of the energy wall [11, Ch. 7]. However, there are two predominant system-level challenges that need to be solved to put this into practice. First, the programmabilty: how to write software that is portable over different HeSoC designs and that can be optimized by the toolchain to exploit the hardware architecture of a specific design? Second, the computational synergy between PMCAs and host: how can processors with vastly different memory architectures—caches and virtual memory in general-purpose processors contrast with physically-addressed, software-managed scratchpad memories (SPMs) typical for PMCAs—efficiently share data?

Knowing the custom instruction set architecture (ISA) of each PMCA allows acceleration libraries and compilers to turn generic code into very efficient machine instructions. However, the cost of doing so is prohibitively high if each ISA is completely custom. The free and open RISC-V ISA [32] alleviates this problem by specifying a minimal base instruction set, standard extensions (e.g., floating-point, bit manipulations, and vector operations), and plenty of opcode space reserved for domain-specific custom instructions. Promising RISC-V processor cores for different application domains [10, 34] have already been developed. Thus, RISC-V is a suitable candidate ISA for PMCAs in a HeSoC.

As for the programming framework, OpenMP [6] has established itself as the de-facto standard programming framework for *homogeneous* shared memory parallel programming. It is very effective at expressing single program, multiple data (SPMD) loop-level parallelism through compiler directives, but was originally not designed for *heterogeneous* computing. Extensions to the OpenMP standard were proposed by academia [8, 18] and by the industry [22]. The synthesis of these works has been incarnated on last OpenMP specifications. Since version 4.0 OpenMP allows to offload work from a host processor to a PMCA through the `target` directive [24], which is already being used to program GPUs in high performance computing [20]. This makes OpenMP a suitable candidate for a unified programming interface of HeSoCs.

Efficient data sharing between host and PMCAs is crucial in HeSoC. However, OpenMP just knows copy-based offloading, where the host copies data from virtually-addressed, cached host memory to physically-addressed SPMs in a PMCA. This puts a daunting task on application developers: they need to deal with cache flushes, virtual-to-physical address translation, and direct memory access (DMA) transfers of properly-sized data tiles. Beyond violating programmability requirements, those operations are very costly, make PMCAs completely dependent on the host, and often kill performance [4]. In contrast, coherent shared virtual memory (SVM) enables to share data simply by passing a pointer from the host to a PMCA. This works when PMCAs can access the shared main memory coherently with the caches of the host (e.g., through an Accelerator Coherency Port (ACP) [33]) and have a memory management unit (MMU) to translate virtual addresses at run time. MMUs that support the large number of parallel memory access bursts common for high-performance PMCAs were long available only at prohibitive hardware costs, but recently developed *hybrid MMUs* [15, 31] have changed this.

In this paper, we present the hardware and software that form HERO, the first RISC-V-based (to the best of our knowledge) open and extensible HeSoC architecture that solves the described system-level challenges. HERO's hardware architecture (§ 2) combines an ARM Cortex-A host processor with a cluster-based, silicon-proven, open, and customizable RISC-V-based PMCA. HERO's software stack (§ 3) supports OpenMP 4.5-based kernel offloading combined with SVM for transparent parallel programming and compiles a single source code to different target ISAs. We evaluate a prototype implementation of HERO, where the PMCA is implemented on FPGA fabric, and show that the run time overhead compared to manually written PMCA code operating on private physical memory is lower than 10 % for pivotal benchmarks and operating conditions (§ 4).

2 HARDWARE ARCHITECTURE

HERO's SoC architecture combines an ARM Cortex-A host processor with one or multiple domain-specific, RISC-V-based PMCAs. Fig. 1 gives an overview of the components and their connections. The host processor consists of ARM Cortex-A cores attached to a coherent interconnect. Each core includes private hardware-managed caches and an MMU. The host shares main memory with the PMCAs through the system interconnect, which is coherent to the

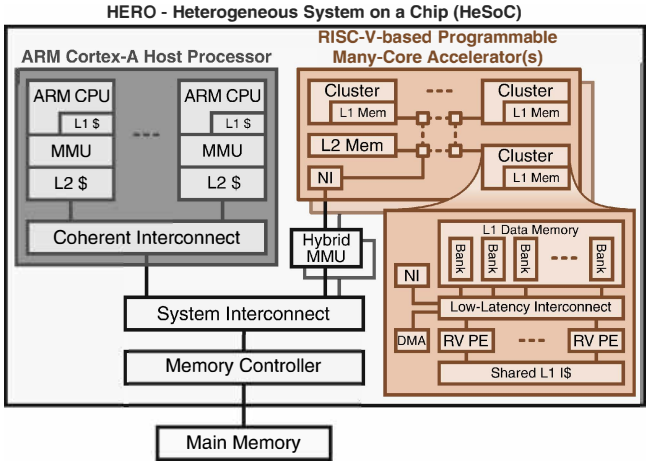

Figure 1: HERO's hardware architecture.

caches of the host. Instead of caches, a PMCA employs software-managed SPMs in its internal memory hierarchy, together with DMA engines to move data. To access virtual memory, each PMCA is attached to a hybrid MMU, which is managed by the PMCA.

As a concrete implementation of a RISC-V-based PMCA, we use the latest version of the Parallel Ultra Low Power (PULP) platform [29]. PULP has been employed in research and commercial application specific integrated circuits (ASICs) designed for parallel ultra-low power processing. To overcome scalability limitations, it uses a multi-cluster design [21] and relies on multi-banked, software-managed SPMs and lightweight, multi-channel DMA engines [28] instead of data caches. The 32b RISC-V processing elements (PEs) [10] within a cluster primarily operate on data present in the shared L1 SPM to which they connect through a low-latency, logarithmic interconnect [16]. 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 hybrid MMU [30], which performs virtual-to-physical address translation based on the entries of an internal table, similar to the MMUs of the host cores. This lightweight hardware block is managed in software directly on the PMCA [15, 31]. 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.

The PMCA is highly configurable. 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 auxiliary processing unit (APU) to be shared among multiple PEs within a cluster, or completely disabled. Similarly, the integer digital signal processing (DSP) extension unit, the divider, and the multiplier can be private or shared in the APU. Domain-specific instructions can be added to the PEs and exposed as extensions to the basic RISC-V ISA. If the target domain does not map well to the SPMD execution model of the RISC-V PEs, PEs can be replaced or complemented by domain-specific acceleration engines. Acceleration engines can be implemented as custom hardware [5] or as reconfigurable circuit [7] and, like PEs, are attached to the low-latency interconnect.

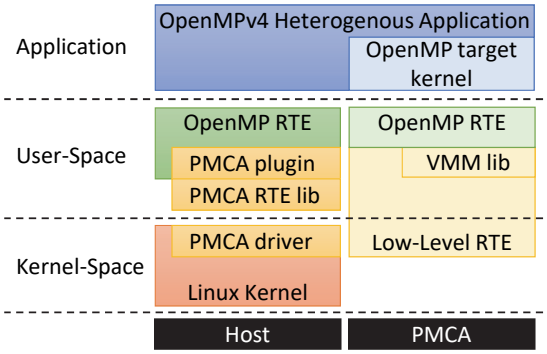

Figure 2: HERO's software stack.

3 SOFTWARE ARCHITECTURE

HERO includes a complete end-to-end software-stack enabling fast and easy-to-use heterogeneous programming. 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 the OpenMP programming interface and SVM hardware capability. The application developer—writing a single application source code—can simply offload computation to the PMCA by encapsulating suitable application kernels in an OpenMP `target` region. The actual offload from the host side is then taken care of by the OpenMP runtime environment (RTE) and afterwards by the lower levels of the software stack that is deployed to the PMCA.

3.1 Heterogeneous Cross Compilation

To allow the host OpenMP RTE to perform an offload to the PMCA, the `target` code region must be outlined by the host compiler, compiled by the PMCA-specific target compiler, linked against PMCA-specific libraries, and embedded into the final host *fat binary*. HERO's toolchain is based on the GNU GCC-7 compiler, which already supports the outlining of OpenMP `target` regions for HSA, Nvidia PXT, and Intel Xeon Phi devices. At the end of interprocedural analysis (IPA), the GCC intermediate representation (IR) of all offloaded functions is streamed out into an link-time optimizer (LTO) object. When the linker is executed, its LTO Wrapper enables link-time recompilation if at least one object file contains LTO sections. HERO's GNU GCC extends the offloading capability of the compiler to RISC-V based accelerators. To enable such new devices, we extended the LTO Wrapper to execute a new *mkoffload* for HERO that (i) invokes the specific ISA back-end for the PMCA (in this case RISC-V), (ii) statically links the target-specific libraries (including the OpenMP runtime library), (iii) fills the Offload Table, which stores all target hooks for the outlined functions in the host binary, and (iv) packs everything into an Offload Image that is loaded to the PMCA at run time.

3.2 Heterogeneous OpenMP Runtime

The GNU OpenMP library, *libgomp*, supports plugins to bind target-specific implementations of runtime functions to its generic application programming interface (API). HERO's customized GNU GCC includes two *libgomp* plugins for the PMCA to implement two different offloading schemes: copy-based and zero-copy with SVM. In copy-based offloads, all shared data is copied to a contiguous, unpaged, uncached memory region and the PMCA is given the physical address into that region. In SVM-based offloads, the PMCA accesses host virtual addresses through the hybrid MMU and instrumented load/store operations (§ 3.3).

The first time a `target` code region is going to be executed, the host OpenMP runtime loads the dynamic shared object (DSO) containing the binary of all offloaded functions together with the accelerator-side OpenMP runtime onto the target PMCA. For every `target` code region with copy-based offload semantics, the host OpenMP runtime copies the shared data to contiguous memory, passes a physical address to the PMCA, and copies data back after the PMCA has finished executing. For every `target` code region with SVM-based offload semantics, the host OpenMP runtime simply passes virtual addresses to the shared data to the PMCA. All these host-to-PMCA interactions go through the host runtime library (§ 3.4).

3.3 Compiler Support for Hybrid-MMU-Based SVM

With a hybrid MMU, loads and stores to SVM by PEs in the PMCA can fail if the accessed virtual address misses in the translation lookaside buffer (TLB). As this semantic deviates from standard load and store, the compiler instruments [30] accesses to SVM with an additional read of a register, through which a PE is informed whether its last SVM access was successful. The compiler transforms memory accesses inside a `target` code region based on the data sharing context: During the OpenMP expansion pass, the compiler annotates all shared variables as candidates for instrumentation. In a static single assignment (SSA) pass, it traverses *use-def* chains to determine which uses of the annotated variables need to be instrumented. Scalar variables can then directly be instrumented, while pointer variables require an additional *escape analysis* to determine when and how a pointer dereference is propagated to instrument accesses through the propagated value.

3.4 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), and initially setting up the hybrid MMU to give the PMCA access to the page table of the heterogeneous user-space application. The PMCA is accessed by the RTE library as a memory-mapped device which allows for low-latency host-to-PMCA communication.

3.5 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 [31] on the PMCA abstracts away differences between host architectures and MMU configurations and provides a uniform API to explicitly map pages and handle TLB misses. When a core accesses virtual memory through the hybrid MMU, the corresponding address translation may be missing in the TLB. In this case, the core that caused the miss goes to sleep and the miss is added to a queue in the L1 SPM. 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 TLB 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.

4 EVALUATION

We evaluated a prototype implementation of HERO (§ 4.1) with two benchmarks (§ 4.2) to show that acceleration speed-ups in the range of 5x to 10x are achievable with low programming overhead and that the run-time overhead compared to manually written PMCA code operating on physical memory is lower than 10 %.

4.1 Evaluation Platform

Our prototype implementation of HERO is based on the ZC706 implementation of the Heterogeneous Research Platform (HERO) [14]. The Xilinx ZC706 board contains a Zynq-7045 SoC, which 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 Advanced eXtensible Interface (AXI) interfaces and share 1 GiB of DDR3 DRAM. Using the ACP, the PULP PMCA cluster instantiated in the FPGA can access the shared main memory coherently with the caches of the ARM host CPU. The two ARM cores have separate L1 instruction and data caches with a size of 32 KiB each, and they share 512 KiB of unified L2 instruction and data cache. The 8 PEs within the PULP cluster share 8 KiB of multi-banked L1 instruction cache and 256 KiB of multi-banked L1 SPM. The ARM is clocked at 800 MHz, the PULP PMCA at 50 MHz. The hybrid MMU features an L1 TLB with 32 entries and an L2 TLB with 1024 entries, and user pages are mapped exclusively in the L2 TLB.

We use the HERO SDK v1.0.1 [26] and bigPULP v1.0.0 [25] for the PMCA implementation. The host CPU runs Linux 4.9.0 with swapping and LPAE disabled. The PMCA runs hero-v1.0.0 of the PULP SDK. The entire compiler toolchain is based on GCC 7.1.1, and benchmarks are compiled with `-O3`.

4.2 Benchmark Results

We evaluated HERO with two benchmarks extracted from the core of two application domains frequently demanding acceleration: matrix-matrix multiplication for signal processing and the Advanced Encryption Standard (AES) block cipher for cryptography. Both benchmarks are written in a single source file, in which we annotate the benchmark kernel with different OpenMP directives to show the performance difference of each variant. We measure the run time of each execution variant on the host using `clock_gettime(CLOCK_MONOTONIC_RAW)`. The measurements contain *all* parts of an offload, including synchronization and transfer of data and parameters between host and PMCA. Static data on host and PMCA

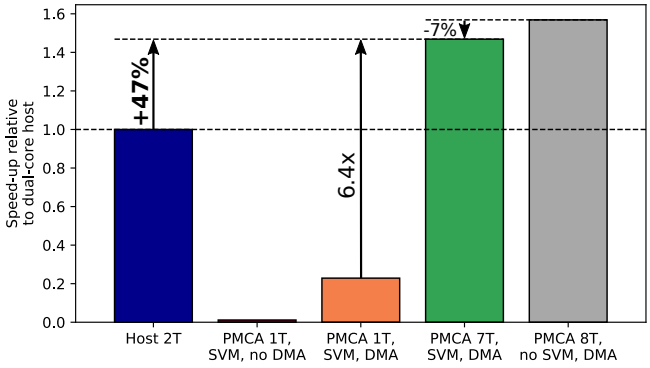

Figure 3: Performance of different MMM implementations compared to the baseline on the dual-core ARM host CPU.

are initialized at the start of the benchmark application, before the first measurement.

Matrix-Matrix Multiplication (MMM). In this benchmark, two square matrices `a` and `b` are multiplied into square matrix `c`. Each matrix has 128 by 128 32-bit elements and thus takes 64 KiB of memory. The C code for the application kernel is

```
for (unsigned i = 0; i < 128; ++i) {
  for (unsigned j = 0; j < 128; ++j) {
    unsigned sum = 0;
    for (unsigned k = 0; k < 128; ++k)
      sum += a[i*width+k] * b[k*width+j];
    c[i*width+j] = sum;
  }
}
```

Fig. 3 shows the performance of different implementations, relative to the baseline (leftmost, blue bar) in which the dual-core ARM CPU runs the kernel code annotated with

```
#pragma omp parallel for \
    firstprivate(a, b, c) collapse(2)
```

That is, the multiplication is parallelized over the rows of `a`. Offloading the kernel to the PMCA is as simple as annotating the kernel code with

```
#pragma omp target map(to: a, b) map(from: c)
```

The PMCA then directly accesses each word of any matrix through pointers to shared virtual memory. However, as the PMCA does not feature caches, performance drops drastically (second, red bar). Since the PMCA is designed to operate on its local SPM, the first optimization is to allocate buffers in SPM before the loop and to insert DMA transfers into the loop. With this, the performance improves to what a single PE can handle (third, orange bar). The kernel is now compute-bound and can be parallelized with the same annotation used to parallelize the ARM code. With this, 7 PEs process the multiplication in parallel—when using SVM, one PE is statically allocated to manage the hybrid MMU in the current version of HERO—leading to a net performance improvement of 6.4x. Compared to a bare-metal, hand-tuned C implementation, where 8 PEs operate on buffers in SPM, and DMA transfers run between SPMs and physically-addressed shared memory (rightmost, gray bar), the run-time overhead is just 7 %. Compared to the dual-core ARM implementation, on the other hand, the speed-up is 47 %

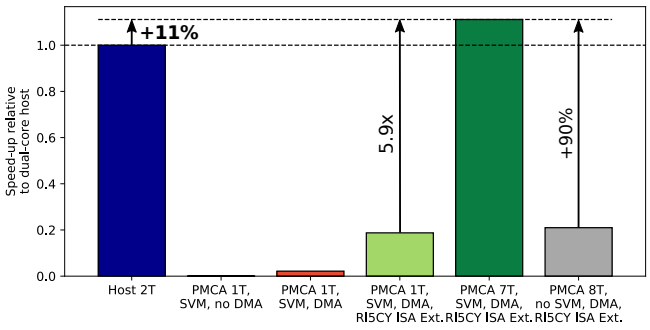

Figure 4: Performance of different AES implementations compared to the baseline on the dual-core ARM host CPU.

even though the PMCA on the FPGA runs at only 50 MHz. For a production HeSoC ASIC, frequencies of 2 GHz and 800 MHz for host and PMCA, respectively, would be more realistic. In that case, this kernel would not saturate the memory bandwidth (even of the Z-7045, which is around 300 MB/s), and offloading it to the PMCA could bring a speed-up of ca. 9x.

AES Block Cipher (AES). We use a popular, small C implementation [13] of the AES block cipher to encrypt 1024 different ciphertexts, each 128 byte long, with 128-bit keys in CBC mode. The C code for the application kernel is

```
for (unsigned i = 0; i < 1024; ++i) {
  AES_CBC_encrypt_buffer(ctx[i], buf[i], 128);
}
```

where `ctx` are AES contexts containing initialization vectors and keys and `buf` are the buffers that contain the plaintext before encryption and the ciphertext after encryption.

Fig. 4 shows the performance of different implementations, relative to the baseline (leftmost, blue bar) in which the dual-core ARM CPU runs the kernel code annotated with

```
#pragma omp parallel for firstprivate(ctx, buf)
```

The second, dark red bar shows the offload to the PMCA with

```
#pragma omp target map(tofrom: ctx, buf)
```

The PMCA now operates on the shared main memory and the caches of the host instead of its SPM. Thus, the next step is to allocate buffers in SPM and control the DMA engine inside the loop to transfer data while computations are running (third, orange bar). As the AES block cipher performs mostly byte-wise operations that map much worse to the basic RISC-V (RV32IM) ISA than to the more complex ARMv7-A ISA, performance is still relatively low. Since the full ISA of the PMCA is exposed, however, we can tune the code to leverage the extensions it offers: The RI5CY PEs [10] in this PMCA implement instructions that interpret the four bytes in a 32-bit word as elements of a byte vector, instructions to pack and unpack bytes from and into words, and instructions to shuffle and rotate bytes within a word. Such operations are at the heart of a block cipher, and using them massively improves performance (fourth, light green bar). We finally use the aforementioned `omp parallel for` annotation to parallelize encryption among 7 PEs, speeding up execution by 5.9x (fifth, dark green bar). With the PMCA running at 50 MHz on the FPGA, this is a modest 11 % faster than the dual-core ARM host. In a production HeSoC ASIC, with host and PMCA running at 2 GHz and 800 MHz, respectively, the application would not saturate memory bandwidth and offloading it to the PMCA could bring a speed-up of ca. 6.8x. Even more remarkably, the SVM-based offload is 90 % faster than a bare-metal implementation where all 8 PEs compute and DMA transfers run between SPM and physically-addressed shared memory (rightmost, gray bar). The reason is that when SVM is not used, the host must gather buffers from application virtual memory into a dedicated, physically-addressed, uncached memory region before the offload and scatter the buffers back after the offload, and the host is very inefficient at doing this.

In summary, these results show (i) that HERO's heterogeneous software stack allows to effectively exploit both the standard ARM ISA and a specialized RISC-V ISA, bringing the acceleration potential of parallel, domain-specific PMCAs to bear, and (ii) that HERO's hardware and software enable host and PMCA to efficiently share data at a minimal programming effort and with a performance impact that ranges from slightly negative to significantly positive compared to copy-based memory sharing.

5 RELATED WORK

General-purpose computing acceleration through accelerator offloading started to gain more traction with the advent of programmable GPUs. Today, nearly all GPU models—ranging from embedded, mid-end IP cores [1] up to high-end, data-center acceleration boards [23]—can serve as a target for offloading application kernels from the host. However, implementing and optimizing a heterogeneous application for GPU-based systems is not a trivial task. Typically, the offloadable kernels must be implemented in separate source files using lower-level programming languages such as OpenCL or CUDA, and are compiled online before offloading. This not only requires special compilers decoupled from the host toolchain, but it also means more programming effort and prevents fine-grained kernel offloading. These problems can be somewhat alleviated by heterogeneous toolchains with open-source OpenMP [9, 19] or OpenACC [27] GPU front ends. However, such front ends can only generate intermediate code and still require to invoke proprietary GPU compilers. The ISA of the accelerator is closed and internal functions remain inaccessible for the developer. There is, e.g., no DMA engine exposed to overlap computations with data transfers. To achieve such behavior for hiding main memory latency, DMA transfers must either be explicitly emulated using regular loads and stores [9], or the kernel must inherently offer very high degrees of data-level parallelism. In addition, GPU drivers and RTE libraries are completely closed for most devices [1, 23]. In contrast, the host side of HERO is partially and its PMCA is completely open source starting from RTE libraries down to the actual hardware. This gives the developer the possibility to optimally leverage the available hardware and exploit the full potential of the PMCA platform with a fully-integrated toolchain.

Heterogeneous compilers have also been implemented by others, both in research [2, 3, 17] and commercially [12]. [2] implemented an OpenMP plugin for GCC 5 to offload to OpenRISC-based PMCAs. While the ISA in that work is also exposed to the toolchain, OpenRISC was not designed with domain-specific extensions in mind. In contrast, HERO's software stack and toolchain are capable

of fully leveraging custom extensions of the RISC-V ISA, as shown by our experimental evaluation. Intel is offering OpenMP-based programming of its Xeon Phi accelerators [12]. Although both the Intel host CPU and the Xeon Phi accelerator implement the x86 ISA, they differ in (vendor-defined) extensions. While GCC can be used to offload to Xeon Phi, the offloaded LTO itself must be generated with the proprietary Intel C compiler to make use of all vector extensions. Similar to GPUs, the accelerator ISA is thus not directly accessible to developers, barring them from exploiting all accelerator features in their libraries. In HERO, in contrast, RISC-V-based PEs can be extended with domain-specific instructions fully exposed to library developers through an end-to-end open software stack.

6 CONCLUSION

We presented HERO, the first RISC-V-based HeSoC architecture that enables efficient collaboration between host and PMCAs, both at application design time and at run time. HERO combines an industry-standard ARM Cortex-A multicore processor as host with a silicon-proven, RISC-V-based many-core processing engine as PMCA. Both ISAs are fully exposed to HERO's heterogeneous toolchain, allowing optimized compilation for both ISAs from a single source code. Beyond heterogeneous compilation, HERO's open software and hardware stack supports shared virtual memory and the toolchain implements OpenMP 4.5 for transparent accelerator programming. We evaluated a prototype implementation of HERO to show that both ISAs can be effectively exploited, bringing the acceleration potential of parallel, domain-specific PMCAs to bear, and that host and PMCA can efficiently share data at a minimal programming effort.

REFERENCES

[1] ARM Ltd. 2017. ARM Mali GPU OpenCL.

[2] A. Capotondi and A. Marongiu. 2017. Enabling Zero-copy OpenMP Offloading on the PULP Many-core Accelerator. In *SCOPES '17*. ACM, New York, NY, USA, 68–71. https://doi.org/10.1145/3078659.3079071

[3] A. Capotondi, A. Marongiu, and L. Benini. 2018. Runtime Support for Multiple Offload-Based Programming Models on Clustered Manycore Accelerators. *IEEE Transactions on Emerging Topics in Computing* 6, 3 (July 2018), 330–342. https://doi.org/10.1109/TETC.2016.2554318

[4] Young-kyu Choi, Jason Cong, Zhenman Fang, Yuchen Hao, Glenn Reinman, and Peng Wei. 2016. A quantitative analysis on microarchitectures of modern CPU-FPGA platforms. In *DAC '16*. ACM, 109.

[5] F. Conti, P. D. Schiavone, and L. Benini. 2018. XNOR Neural Engine: a Hardware Accelerator IP for 21.6 fJ/op Binary Neural Network Inference. *IEEE TCADICS* (2018), 1–1. https://doi.org/10.1109/TCAD.2018.2857019

[6] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *IEE CSE* 5, 1 (1998), 46–55.

[7] S. Das, K. J. M. Martin, P. Coussy, and D. Rossi. 2018. A Heterogeneous Cluster with Reconfigurable Accelerator for Energy Efficient Near-Sensor Data Analytics. In *ISCAS '18*. 1–5. https://doi.org/10.1109/ISCAS.2018.8351749

[8] Alejandro Duran, Eduard Ayguadé, Rosa M Badia, Jesús Labarta, Luis Martinell, Xavier Martorell, and Judit Planas. 2011. OmpSs: a proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters* 21, 02 (2011), 173–193.

[9] B. Forsberg, L. Benini, and A. Marongiu. 2018. HePREM: Enabling predictable GPU execution on heterogeneous SoC. In *DATE '18*. 539–544. https://doi.org/10.23919/DATE.2018.8342066

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

[11] John L Hennessy and David A Patterson. 2018. *Computer architecture: a quantitative approach* (6 ed.). Elsevier.

[12] Intel Corp. 2018. Migrating Offloading Software to Intel Xeon Phi Processor. white paper. https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/migrating-offloading-software-paper.pdf.

[13] kokke. 2018. tiny-AES-c: Small portable AES 128/192/256 in C. GitHub repository. https://github.com/kokke/tiny-AES-c/tree/f56dbc05ab0d795d74f43436aac9da56a7cc8e11

[14] 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. *CoRR* abs/1712.06497 (2017). arXiv:1712.06497 http://arxiv.org/abs/1712.06497

[15] A. Kurth, P. Vogel, A. Marongiu, and L. Benini. 2018. Scalable and Efficient Virtual Memory Sharing in Heterogeneous SoCs with TLB Prefetching and MMU-Aware DMA Engine. In *ICCD '18*. IEEE.

[16] I. Loi, A. Capotondi, D. Rossi, A. Marongiu, and L. Benini. 2018. The Quest for Energy-Efficient I$ Design in Ultra-Low-Power Clustered Many-Cores. *IEEE TMSCS* 4, 2 (Apr 2018), 99–112. https://doi.org/10.1109/TMSCS.2017.2769046

[17] Andrea Marongiu, Alessandro Capotondi, and Luca Benini. 2016. Controlling NUMA effects in embedded manycore applications with lightweight nested parallelism support. *Parallel Comput.* 59 (2016), 24 – 42. https://doi.org/10.1016/j.parco.2016.02.002 Theory and Practice of Irregular Applications.

[18] A. Marongiu, A. Capotondi, G. Tagliavini, and L. Benini. 2015. Simplifying Many-Core-Based Heterogeneous SoC Programming With Offload Directives. *IEEE TII* 11, 4 (Aug 2015), 957–967. https://doi.org/10.1109/TII.2015.2449994

[19] Matt Martineau, Simon McIntosh-Smith, Carlo Bertolli, Arpith C. Jacob, Samuel F. Antao, Alexandre Eichenberger, Gheorghe-Teodor Bercea, Tong Chen, Tian Jin, Kevin O'Brien, Georgios Rokos, Hyojin Sung, and Zehra Sura. 2016. Performance Analysis and Optimization of Clang's OpenMP 4.5 GPU Support. In *PMBS '16*. 54–64. https://doi.org/10.1109/PMBS.2016.11

[20] M. Martineau, S. McIntosh-Smith, and W. Gaudin. 2016. Evaluating OpenMP 4.0's Effectiveness as a Heterogeneous Parallel Programming Model. In *IPDPSW '16*. 338–347. https://doi.org/10.1109/IPDPSW.2016.70

[21] D. Melpignano et al. 2012. Platform 2012, a Many-core Computing Accelerator for Embedded SoCs: Performance Evaluation of Visual Analytics Applications. In *DAC '12*. 1137–1142. https://doi.org/10.1145/2228360.2228568

[22] Gaurav Mitra, Eric Stotzer, Ajay Jayaraj, and Alistair P Rendell. 2014. Implementation and optimization of the OpenMP accelerator model for the TI Keystone II architecture. In *International Workshop on OpenMP*. Springer, 202–214.

[23] NVIDIA Corp. 2017. NVIDIA Tesla V100 GPU Architecture. white paper.

[24] OpenMP Architecture Review Board. 2013. *OpenMP API v4.0.*

[25] PULP Platform. 2018. bigPULP: RISC-V manycore accelerator for HERO (v1.0.0). GitHub repository. https://github.com/pulp-platform/bigpulp/tree/v1.0.0

[26] PULP Platform. 2018. HERO SDK (v1.0.1). GitHub repository. https://github.com/pulp-platform/hero-sdk/tree/v1.0.1

[27] Ruymán Reyes, Iván López-Rodríguez, Juan J Fumero, and Francisco de Sande. 2012. accULL: an OpenACC implementation with CUDA and OpenCL support. In *ECPP '12*. Springer, 871–882.

[28] D. Rossi, I. Loi, G. Haugou, and L. Benini. 2014. Ultra-low-latency Lightweight DMA for Tightly Coupled Multi-core Clusters. In *CF '14*. ACM, New York, NY, USA, Article 15, 10 pages. https://doi.org/10.1145/2597917.2597922

[29] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. GÃijrkaynak, A. Teman, J. Constantin, A. Burg, I. Miro-Panades, E. BeignÃĺ, F. Clermidy, P. Flatresse, and L. Benini. 2017. Energy-Efficient Near-Threshold Parallel Computing: The PULPv2 Cluster. *IEEE Micro* 37, 5 (Sept 2017). https://doi.org/10.1109/MM.2017.3711645

[30] P. Vogel et al. 2015. Lightweight Virtual Memory Support for Many-core Accelerators in Heterogeneous Embedded SoCs. In *CODES '15*. 45–54.

[31] P. Vogel et al. 2017. Efficient Virtual Memory Sharing via On-Accelerator Page Table Walking in Heterogeneous Embedded SoCs. *ACM TECS* 16, 5s (2017), 154:1–154:19.

[32] Andrew Waterman and Krste Asanovic. 2017. *The RISC-V Instruction Set Manual, Volume I: User-Level ISA, v2.2.*

[33] Xilinx Inc. 2016. Zynq-7000 All Programmable SoC Overview. Product Specification.

[34] 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 (April 2016), 930–942. https://doi.org/10.1109/JSSC.2016.2519386