Co-Designing Clusters of Lightweight Manycores and Asymmetric Operating System Kernels

Pedro Henrique Penna, Lucas Maciel, Joao Vicente Souto, Davidson Francis Lima, Marcio Castro, François Broquedis, Henrique Freitas, Jean-François Méhaut, Pedro Henrique Penna

Co-Designing Clusters of Lightweight Manycores and Asymmetric Operating System Kernels

Pedro Henrique Penna*†, Lucas Maciel†, João Vicente Souto‡, Davidson Francis Lima†,
Márcio Castro‡, François Broquedis*, Henrique Freitas† and Jean-François Méhaut*

*Université Grenoble Alpes – Grenoble, France
†Pontifícia Universidade Católica de Minas Gerais – Belo Horizonte, Brazil
‡Universidade Federal de Santa Catarina – Florianópolis, Brazil

***Abstract*—Multikernel Operating Systems (OSs) were introduced to cope with challenges in software development and deployment in lightweight manycores. Among the possible structures for a multikernel OS, we focus on designs based on asymmetric kernels. This design delivers better performance isolation, but it suffers from an overhead in energy efficiency. In this work, we overcome this issue with a co-design solution between the cluster of a lightweight manycore and an asymmetric kernel. We designed a 4-core heterogeneous cluster with one core tuned for the OS kernel and we patched the OS kernel to better match the characteristics of this core. Our experiments unveiled that our solution consumes 14.1% less power than the baseline and also improves the OS kernel performance by up to 6.5%.**

I. Introduction

Lightweight manycores were introduced to cope with performance and energy efficiency demands of applications [1]. To deliver performance, these processors feature a clustered layout, a distributed memory architecture and a rich on-chip interconnect [2]. To achieve energy efficiency, they are built with simple and low-power cores; have a Scratchpad Memory (SPM) system with small local memories [3]; do not feature a global cache-coherent domain [4]; and exploit heterogeneity [5]. Some examples of lightweight manycores are the Kalray MPPA-256 [4] and the Sunway SW26010 [2].

Although lightweight manycores stand out in performance and energy efficiency, they currently face challenges in software development and deployment [6]. To address these issues, multikernel Operating Systems (OSs) were introduced [7]. In this approach, the OS is structured as a distributed system: a set of independent OS kernels is deployed in the processor; these kernels communicate via message-passing; and they implement OS subsystems in a distributed fashion.

Multiple structures for a multikernel OS are possible [8], [9]. Nevertheless, we are interested in designs based on asymmetric kernels, due to their outstanding compromise between kernel-level and user-level performance [10]. In an asymmetric design, cores within the same cluster share the same OS kernel instance. The OS kernel runs on a dedicated core of the cluster, named K-Core. The remainder cores are left for user threads.

We thank CNRS, CNPq and FAPEMIG for supporting this research. This study was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES)* - Finance Code 001 and by the CAPES-PrInt Program (grant number 88881.310783/2018-01).

Unfortunately, having a dedicated core for the OS kernel not only reduces the parallelism available for user threads but also the energy efficiency of the system when the OS kernel is underused [6]. However, increasing the number of cores of the cluster to improve parallelism and scalability may not be possible due to the limited amount of on-chip memory [2], [4]. In this paper, we investigate a co-designed solution between the cluster of a lightweight manycore and an asymmetric kernel to mitigate its energy consumption overhead. This work delivers the following contributions to the state of the art on hardware support of lightweight manycores for OSs:

- A co-designed cluster for lightweight manycores that is energy-optimized for asymmetric kernels. This cluster has a heterogeneous configuration in which the K-Core is narrowed for executing kernel routines whereas the reminder cores deliver high performance for user threads;
- An optimized asymmetric kernel for our co-designed cluster. We specifically engineered our OS kernel so that it matches the hardware characteristics of the K-Core.

Furthermore, we present an open-source implementation of our co-designed solution. We built our solution using OpTiMSoC [11] and a patched version of the Nanvix kernel [6]. We prototyped our solution on a Field Programmable Gate Array (FPGA) platform and assessed its energy efficiency using representative benchmarks. In the next sections, we discuss about asymmetric multikernel OSs (Section II), present our methodology (Section III) and contributions (Section IV). Then, we evaluate our solution (Section V), discuss related works (Section VI) and draw our conclusions (Section VII).

II. Asymmetric Multikernel OSs

Multiple structures for a multikernel OS are possible [8], [9]. OS kernel instances may or may not feature the same architecture and/or provide the same set of functionalities (homogeneous vs. heterogeneous architecture). Furthermore, they may run on all cores of the processor or on a selected set of them (symmetric vs. asymmetric design) [6]. In this spectrum of possibilities, in this work we are interested on asymmetric multikernel OSs, due to their outstanding performance isolation between kernel and user spaces [10].

Figure 1 presents a snapshot of an asymmetric multikernel OS running in a lightweight manycore. Cores within the same cluster share an OS kernel instance. The OS kernel runs on a

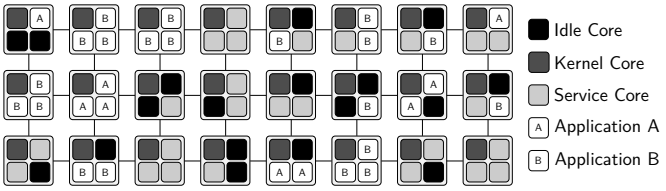

Fig. 1. Snapshot of a multikernel OS running on a lightweight manycore.

dedicated core of the cluster (K-Core); whereas user threads run in the remainder cores of the cluster (U-Cores). In this design, performance isolation is delivered because hardware structures of a core are not time-shared between two execution flows (i.e. kernel and user) and there is no contention in structures of the OS kernel.

Unfortunately, asymmetric multikernel OSs have an intrinsic energy-consumption overhead, since one core is restricted to the OS kernel and cannot run user threads. Thus, the energy efficiency of the system is reduced if the OS kernel is underused [6]. For instance, up to 25% of energy efficiency may be compromised in the example given in Figure 1. While this problem may be mitigated when several cores are bundled in a single cluster [2], [4], from a system-level perspective the problem is not solved, but forwarded [6]. The reason for this is two-fold. If more cores are available in a cluster, either: (i) multiple single-threaded applications are deployed in the same cluster, which is a challenge due to the small size of local memories; or (ii) user applications are highly parallel and scale up linearly in a cluster, which is not a valid assumption in a general-purpose scenario.

III. Methodology

In this paper, we investigate a co-design solution between the cluster of a lightweight manycore and an asymmetric OS kernel to mitigate the energy consumption overhead discussed in Section II. We relied on the following design decisions:

(i) start from established projects in lightweight manycores and OSs, to contribute to ongoing research efforts;
(ii) study structural changes on hardware-side (i.e. cache associativity and arithmetic capabilities), to avoid stepping into customization of hardware units which leads to a space of design that cannot be thoroughly explored; and
(iii) explore optimizations in OS-side that impact multiple workloads, to deliver a solution that effectively extends to use-cases that are not covered in our analysis.

Based on these decisions, we employed a two-step process to build a co-designed solution. First, we carried out a Design Space Exploration (DSE) for the K-Core. Next, we deployed the OS kernel in this core and studied OS-level optimizations.

We used four representative benchmarks to evaluate our co-designed solution [6], which assess important aspects of an asymmetric kernel:

- *R-Kcall*: it evaluates the upper-bound performance for serving kernel calls by issuing multiple low-latency calls, which are handled sequentially by the kernel.
- *Fork-Join*: it exercises important kernel structures, such as the table of threads, by spawning multiple threads and waiting for all of them to terminate.
- *Buffer*: it assesses thread synchronization facilities exposed by the kernel by launching multiple pairs of threads that perform buffered transfers. Data is stored in a ring buffer in user space, and synchronization is achieved by synchronization primitives (i.e. semaphores and mutexes).
- *Server*: it models the functioning of a server. Multiple requests are launched and, for each one of them, a thread is created to serve it. The benchmark makes intensive use of OS kernel calls to create/terminate threads and to synchronize the access to the buffer of requests.

To build the software stack, we employed GCC 9.1.0 and GNU Binutils 2.32.51 with `-O3` optimization. We used an Xilinx Artix-7 FPGA and Vivado 2019.2 for hardware prototyping. We relied on reports output by Vivado to retrieve information about resource utilization and total power consumption (i.e. static and dynamic). Furthermore, we employed performance counters to gather high-precision execution statistics. We carried out 30 replications of each experimental configuration. All comparisons have a 95% confidence threshold.

IV. Co-Designed Cluster for Asymmetric Kernel

In this section, we detail how we applied our two-step methodology to develop our co-designed solution.

A. Hardware Design Space Exploration

To design the cluster, we relied on OpTiMSoC, a framework for prototyping OpenRISC-based lightweight manycores in FPGAs [11]. We started from an initial configuration with four cores, each of which with a 32-bit, 6-stage OpenRISC pipeline and an 8 kB Instruction Cache (I-Cache). The pipeline featured a simple branch predictor; a barrel shifter; a serial divider; a three-stage multiplier; and no FPU. On the other hand, the I-Cache presented a 2-way, 4k-set associativity. We refer to this initial configuration as Initial-Core (I-Core). Next, we considered structural changes in the I-Core, one at a time, to find out the best configuration for the K-Core. Table I details all the hardware parameters that are possible to be changed in OpTiMSoC. Overall, we explored all the 192 possible hardware configurations. The parameters highlighted in the table refer to the initial configuration (I-Core). As a final remark, it is important to note that we applied OS kernel optimizations on the best hardware configuration found.

TABLE I
Design space for kernel core (K-Core).

Hardware Unit	Parameters
Branch Predictor	**Simple**, SatCounter and GShare
Shift Unit	Serial and **Barrel**
Divider	None and **Serial**
Multiplier	None and **Three-Stage**
FPU	**None**, IEEE-754
I-Cache	1-way 8k-set, **2-way 4k-set**, 4-way 2k-set, 8-way 1k-set

B. OS Kernel Optimizations

We considered the asymmetric microkernel of Nanvix [6]: a multikernel OS for lightweight manycores. This kernel

is structured in three layers. On the bottom, the *Hardware Abstraction Layer (HAL)* enables the portability of the kernel across multiple processors. In the middle, the *Modules Layer* hosts the implementation of the functionalities of the micro-kernel, such as thread management, memory management and inter-cluster communication. On the top, the *Kernel Call Layer* exposes the functionalities of each module to user space. This layer performs security checking, controls execution flow, and handles the asymmetric characteristic of the kernel.

We focused on optimizing the *Modules Layer* and *Kernel Call Layer*, since they implement most of the abstractions and facilities exposed to user space and any changes to these layers impact multiple OS workloads. We considered three optimizations: (i) software emulation of hardware division and multiplication operations; (ii) macro-inlining of performance–critical functions; and (iii) branch condition hinting.

The rationale behind these optimizations is three-fold. First, we explored software emulation for division and multiplication operations because they are not extensively used by the Nanvix kernel, thus some specific hardware could be removed. Second, macro-inlining was exploited to improve spatial locality of performance-critical functions at compile-time, thus enabling us to extract performance from a simple instruction cache (i.e. direct-mapped cache). Noteworthy, the size of inlined code was small enough to fit in the I-Cache. Finally, we leveraged condition branch hinting to enable a simpler branch condition unit to be employed, while delivering low performance penalties due to miss-predicted branches.

V. Experimental Results

In this section, we first show the results of the hardware design. Then, we discuss the OS kernel optimizations.

A. Hardware Design

Table II details the resource utilization for designs of a core, such as LUTs (Look-up Tables), Registers (Regs), F7 Multiplexers (M) and Block RAMs (B). We depict statistics for the designs that targeted: (i) the branch predictor; (ii) I-Cache; and (iii) shift unit. Moreover, we present these statistics for the selected configurations to build up our co-designed cluster, which has 1 K-Core and 3 U-Cores (in bold).

When evaluating the possible designs for the branch predictor, we observed that the *Simple* and *SatCounter* predictors yield similar resource utilization and power consumption. In contrast, we noted that the GShare predictor consumes about $1.31\times$ more power, due to an increase on the number of LUTs, registers and F7 multiplexers. On the other hand, concerning the layouts for the I-Cache, we did not spot a significant variance in terms of power consumption. Each design minimized the utilization of a particular resource in the experimental FPGA. Similarly, we noted little difference in power consumption in both designs for the shift unit. However, this result unveiled a small overhead for having a more complex shift unit (i.e. barrel shifter).

TABLE II
Design space exploration (DSE) for cores of the cluster.

Design	LUTs	Regs	M	B	P (mW)
I-Core (I)	2987	2042	57	8	28
(I) + Branch SatCounter	2988	2046	57	8	29
(I) + Branch GShare	5968	4112	434	8	38
(I) + I-Cache 1 Way, 8k Set	2874	1947	57	9.5	28
(I) + I-Cache 4 Way, 2k Set	3175	2159	57	9	30
(I) + I-Cache 8 Way, 1k Set	3701	2666	55	7	29
(I) + Shift Serial	2830	2080	56	8	26
K-Core (proposed)	**2500**	**1829**	**56**	**9.5**	**23**
U-Core (proposed)	**7564**	**4853**	**414**	**12**	**53**

Fig. 2. Static instruction class analysis of the kernel.

Finally, for deciding whether or not floating point operations as well as integer divisions and multiplications should be supported by the hardware in K-Core, we relied on a static analysis of the binary file of the OS kernel. If these operations are heavily used, having hardware support would be desirable. Notwithstanding, as Figure 2 pictures, this analysis unveiled that these operations are barely used by the kernel. Floating point operations were not used at all, and multiplications/divisions were employed in initialization code.

Our DSE unveiled important aspects. First, the branch predictor has an important impact on the overall power consumption of the core. Hence, runtime benchmarking is required to identify the configuration that delivers the best compromise between energy consumption and performance. Second, the layout for the I-Cache does not result in significant differences in power consumption. Thus, we can fine-tune the I-Cache for better supporting execution flows of the kernel. Third, the impact of the shift unit in power consumption is very small, so a more complex hardware (i.e. barrel shifter) can be used to better support execution flows that heavily rely on it. Finally, floating point operations, multiplications and divisions are barely used by the kernel, thus hardware support for them could be removed to further reduce power consumption.

B. OS Kernel Optimizations

Table II also details the resource utilization of the two types of cores that we proposed and used to build up our cluster:

- 1 K-Core with the following configuration: simple branch predictor, barrel shifter, no divider, no multiplier, no FPU and 8 kB 1-way 8k-set I-Cache.

TABLE III
BENCHMARK RESULT FOR CO-DESIGNED CORE.

Metric	Design	R-Kcall	Fork-Join	Buffer	Server
Energy (µJ)	U-Core	84.06	1994	27640	32903
	K-Core	41.07	1002	12750	14983
	K-Core+Opt Kernel	39.65	859	11215	13372
Time (ms)	U-Core	1.58	37.63	521.51	620.82
	K-Core	1.78	43.56	554.37	651.43
	K-Core+Opt Kernel	1.72	37.34	487.64	581.41

- 3 U-Cores with the following configuration: GShare branch predictor, barrel shifter, serial divider, three-stage multiplier, IEEE 754 FPU and 8 kB 2-way 4k-set I-Cache.

Overall, the K-Core features the best configuration found in our analysis. The configuration of the U-Core was built with the best general-purpose variant for each hardware unit.

Table III presents an assessment of these two core designs. It unveils how efficient it is to run kernel-intensive workloads in a general-purpose core, like it would happen in a homogeneous cluster design. Results obtained with the K-Core show the gains on energy consumption when running benchmarks in a more specialized core. Finally, the assessment of the K-Core running on an optimized version of the kernel illustrates the upper-bound energy efficiency that may be achieved. The energy consumed by the OS kernel drops about 50% when we move from the U-Core to the K-Core. Additionally, when running an optimized version of the kernel in the K-Core (see Section IV-B), we further reduced energy consumption in 10% and improved performance by up to 6.5% (Fork-Join benchmark). The rationale for this behavior is two-fold. First, the U-Core inherently consumes more power, once it features more complex units. Second, this extra complexity does not necessarily imply on better support for the kernel execution flow. For instance, our co-design unveiled that a simple direct-mapped cache better handles the kernel workload than set-associative ones. As a concluding remark, we observed that a heterogeneous cluster composed of one K-Core and three U-Cores (our solution) consumes 14.1% less power than a cluster with four U-Cores (baseline). In multi-cluster architectures we expect that these outcomes scale up linearly with the number of clusters in the processor.

VI. RELATED WORK

Lightweight manycores are being shipped with an extra core in their clusters for local resource management. For instance, clusters of Kalray MPPA-256 [4] have a firmware core that handles on-chip communication; and clusters of Sunway SW26010 [2] have a management processing element that features a general-purpose design and is intended for running a full-weight OS on top of it. Similarly, a fabric controller is included in the cluster to orchestrate the application in PULP [3], [12]. Alternative approaches look for integrating a special hardware units for driving a cluster from a remote OS kernel [10], [13]. In contrast, we targeted a co-design solution that comprises the cluster of a lightweight manycore and the OS kernel to improve the energy efficiency of asymmetric multikernel OSs.

VII. CONCLUSIONS

Multikernel OSs were introduced to cope with challenges in software development and deployment in lightweight many-cores. Multiple structures for multikernel OSs are possible, but in this work we focused on designs based on asymmetric kernels. This design delivers better performance isolation between user and kernel execution flows, but it has a worse energy efficiency. In this work, we aimed at mitigating this drawback with a co-design solution between the cluster of a lightweight manycore and an asymmetric OS kernel. We designed a heterogeneous cluster with one core tuned for OS kernel execution and we patched the OS kernel to better match the characteristics of the hardware. Overall, our experiments unveiled that our co-designed cluster consumes 14.1% less power than the baseline and also improves the OS kernel performance by up to 6.5%. As future work, we intend to explore changes in OpTiMSoC to enable cores in the cluster to switch to low power modes dynamically, thus improving even further the energy efficiency. Furthermore, we highlight future investigations may also exploit software-level optimizations in the HAL of Nanvix.

REFERENCES

[1] E. Francesquini *et al.*, "On the Energy Efficiency and Performance of Irregular Application Executions on Multicore, NUMA and Manycore Platforms," *Journal of Parallel and Distributed Computing (JPDC)*, vol. 76, no. C, pp. 32–48, february 2015.

[2] F. Zheng *et al.*, "Cooperative Computing Techniques for a Deeply Fused and Heterogeneous Many-Core Processor Architecture," *Journal of Computer Science and Technology*, vol. 30, no. 1, pp. 145–162, jan 2015.

[3] D. Rossi *et al.*, "Energy-Efficient Near-Threshold Parallel Computing: The PULPv2 Cluster," *IEEE Micro*, vol. 37, no. 5, pp. 20–31, sep 2017.

[4] B. D. de Dinechin *et al.*, "A Clustered Manycore Processor Architecture for Embedded and Accelerated Applications," in *IEEE High Performance Extreme Computing Conf.*, ser. HPEC '13, sep 2013, pp. 1–6.

[5] S. Davidson *et al.*, "The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips," *IEEE Micro*, vol. 38, no. 2, pp. 30–41, mar 2018.

[6] P. H. Penna *et al.*, "On the Performance and Isolation of Asymmetric Microkernel Design for Lightweight Manycores," in *Brazilian Symp. on Computing Systems Engineering*, ser. SBESC '19, november 2019, pp. 1–8.

[7] A. Baumann *et al.*, "The Multikernel: A New OS Architecture for Scalable Multicore Systems," in *ACM SIGOPS Symp. on Operating Systems Principles*, ser. SOSP '09, oct 2009, pp. 29–44.

[8] F. Kluge *et al.*, "An Operating System for Safety-Critical Applications on Manycore Processors," in *Int. Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing*, ser. ISORC '14, sep 2014, pp. 238–245.

[9] A. Barbalace *et al.*, "Popcorn: Bridging the Programmability Gap in Heterogeneous-ISA Platforms," in *Int. European Conf. on Computer Systems (EuroSys)*, 2015, pp. 1–16.

[10] E. B. Nightingale *et al.*, "Helios: Heterogeneous Multiprocessing with Satellite Kernels," in *ACM SIGOPS Symp. on Operating Systems Principles*, ser. SOSP '09, oct 2009, pp. 221–234.

[11] S. Wallentowitz *et al.*, "A Framework for Open Tiled Manycore System-On-Chip," in *Int. Conf. on Field Programmable Logic and Applications*, ser. FPL '2012, aug 2012, pp. 535–538.

[12] M. Payami *et al.*, "A Hybrid Instruction Prefetching Mechanism for Ultra Low-Power Multicore Clusters," *IEEE Embedded Systems Letters*, vol. 9, no. 4, pp. 125–128, 2017.

[13] N. Asmussen *et al.*, "M3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores," in *Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16, mar 2016, pp. 189–203.