ARCHIVIO ISTITUZIONALE DELLA RICERCA

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

The Quest for Energy-Efficient I$ Design in Ultra-Low-Power Clustered Many-Cores

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:
Loi, I., Capotondi, A., Rossi, D., Marongiu, A., Benini, L. (2018). The Quest for Energy-Efficient I$ Design in Ultra-Low-Power Clustered Many-Cores. IEEE TRANSACTIONS ON MULTI-SCALE COMPUTING SYSTEMS, 4(2), 99-112 [10.1109/TMSCS.2017.2769046].

Availability:
This version is available at: https://hdl.handle.net/11585/613504 since: 2018-09-12

Published:

DOI: http://doi.org/10.1109/TMSCS.2017.2769046

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (https://cris.unibo.it/).
When citing, please refer to the published version.

(Article begins on next page)

06 October 2026

This is the post peer-review accepted manuscript of:

I. Loi, A. Capotondi, D. Rossi, A. Marongiu and L. Benini, "The Quest for Energy-Efficient I$ Design in Ultra-Low-Power Clustered Many-Cores," in IEEE Transactions on Multi-Scale Computing Systems, vol. 4, no. 2, pp. 99-112, 1 April-June 2018. doi: 10.1109/TMSCS.2017.2769046

The published version is available online at: https://ieeexplore.ieee.org/document/8094020/

© 2017 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

The Quest for Energy-Efficient I$ Design in Ultra-Low-Power Clustered Many-Cores

Igor Loi, Alessandro Capotondi, *Member, IEEE,* Davide Rossi, Andrea Marongiu, *Member, IEEE,* and Luca Benini, *Fellow, IEEE*

Abstract—High performance and extreme energy efficiency are strong requirements for a fast-growing number of edge-node Internet of Things (IoT) applications. While traditional Ultra-Low-Power designs rely on single-core micro-controllers (MCU), a new generation of architectures leveraging fully programmable tightly-coupled clusters of near-threshold processors is emerging, joining the performance gain of parallel execution over multiple cores with the energy efficiency of low-voltage operation. In this work we tackle one of the most critical energy-efficiency bottlenecks for these architectures: instruction memory hierarchy. Exploiting the instruction locality typical of data-parallel applications, we explore two different shared instruction cache architectures, based on energy-efficient latch-based memory banks: one leveraging a crossbar between processors and single-port banks (SP), and one leveraging banks with multiple read ports (MP). We evaluate the proposed architectures on a set of signal processing applications with different executable sizes and working-sets. The results show that the shared cache architectures are able to efficiently execute a much wider set of applications (including those featuring large memory footprint and irregular access patterns) with a much smaller area and with much better energy efficiency with respect to the private cache. The multi-port cache is suitable for sizes up to a few kB, improving performance by up to 40%, energy efficiency by up to 20%, and energy $\times$ area efficiency by up to 30% with respect to the private cache. The single-port solution is more suitable for larger cache sizes (up to 16 kB), providing up to 20% better energy $\times$ area efficiency than the multi-port, and up to 30% better energy efficiency than private cache.

Index Terms—Instruction Cache, Parallel Architectures, Tightly Coupled Cluster, Near-Threshold Computing, Computer Architecture, Energy Efficiency.

1 Introduction

The Internet of Things (IoT) [1] is becoming pervasive in our everyday life and it is expected to have an increasingly higher impact in the coming decades. Several near-sensor processing applications [2] such as security, video surveillance and e-health rely on deeply embedded systems (end-nodes) that wirelessly communicate data collected from several high-bandwidth sensors including vital sign monitors [3], low-power imagers [4], microphone arrays [5]. These smart, connected devices share the need for high performance and extreme energy efficiency for operation within power envelopes of only a few mW. In this context, a promising approach to achieve major energy efficiency improvements is Near-Threshold (NT) computing [6] on parallel architectures [7] [8] [9].

A major challenge in NT operation is the increased sensitivity of devices to *Process*, *Voltage* and *Temperature* (PVT) variations, which leads to poorly controlled performance levels and causes unreliable memory operation [10]. While PVT variations can be effectively managed in the digital cores by exploiting robust standard cell libraries or post-fabrication compensation techniques, the supply voltage of standard 6T SRAMs has to be kept at a higher value with respect to the logic causing on-chip memory to form a major bottleneck for energy efficiency of Ultra-Low-Power (ULP) designs [10]. For this reason, SRAM often dominates the overall SoC power consumption, especially in platforms leveraging the tiny processors (≈30kGE) typical of ULP computing platforms [11] [12] [8]. Several memory architectures optimized for NT have been demonstrated to operate at low voltage [13] [14] [15], overcoming the voltage scaling bottleneck typical of 6T SRAMs. However, these approaches significantly increase bit cell circuit complexity, peripheral circuit complexity and bit-lines capacitance, therfore increasing access energy for a given supply voltage [16]. A promising approach to deal with performance degradation of memories at low voltage is to rely on latch-based Standard Cell Memories (SCMs) [16] [17]. One of the primary advantages that SCMs have over traditional SRAM is their robustness, especially at low-voltage, since they are constructed exclusively with standard cells. This comes at a cost in area, as the basic SCM storage latch is much larger than both standard 6T SRAM bit-cell and SRAMs optimized for low-voltage. On the other hand, due to its architecture and the capability to operate without bit-lines pre-charge and sense amplifiers, SCM provides up to four times smaller access energy [16] [17].

Focusing on the architecture of ULP multi-processor systems-on-chip (MPSoC), it is common to design data memories as explicitly-managed scratch-pads, rather than HW-controlled caches. A shared design is often employed, which eliminates the need for maintaining data consistency [8]. Regarding instruction memory, caches are typically preferred to scratchpad memories (SPM), as their explicit management is cumbersome [18] and their read-only nature inherently limits the complexity of hardware implementation. While in high-end MPSoCs, where the area constraint is weak, big private instruction caches (I$) are usually preferred, in the ULP domain cores are much smaller and the area of replicated private caches (one per core) becomes a major concern. Indeed, during the execution

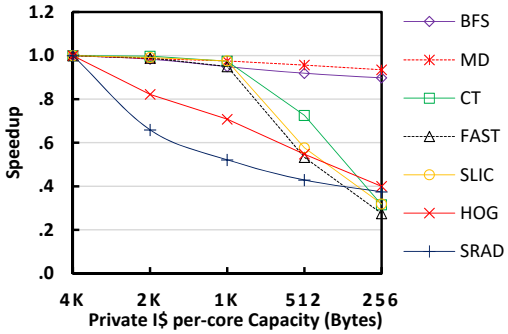


Fig. 1: Private cache configuration speedup (slowdown) down-scaling the instruction cache (I$) capacity for the seven applications used as benchmarks.

of parallel code, a large fraction of the cached instructions is replicated in every private cache, leading to inefficient usage of program cache capacity. This situation is exacerbated if we consider a parallel program that relies on a parallel programming model such as OpenMP [19]. Dedicated language constructs or compiler directives for parallelism are ultimately translated into function calls within *runtime libraries* that implement the required parallel semantics (e.g., parallel loop iteration scheduling). This may silently break the instruction locality typically found in computation-intensive signal processing algorithms, thus increasing the instruction cache miss rate during execution. The explicit use of external or auxiliary libraries adds to the same problem.

Figure 1 depicts the performance degradation suffered by a set of parallel applications[1] executing on a private-cache, 8-core ULP processor when the cache size is reduced (x-axis). The applications can be grouped in three categories: i) those that heavily rely on *library* calls (HOG, SRAD); ii) those that contain a *long* instruction block within the parallel loop(s) (FAST, SLIC, CT); iii) those that contain a *short* instruction block within the parallel loop(s) (BFS, MD). The figure shows that only *short*-instruction-block applications are weakly sensitive to instruction cache size reduction (as their execution kernels fits in cache), while the rest of the applications suffer dramatic performance drops. In this scenario, a shared cache can offer a lower miss rate and better memory utilization (no copies) at the cost of increased hardware complexity.

In this work we propose an extensive evaluation of shared instruction cache architectures for ultra-low-power, tightly coupled multi-processor clusters targeting end-node IoT devices. The exploration considers a private cache architecture as baseline, [20], and a third novel architecture based on a multi-ported SCM instruction cache. The proposed cache architectures are then evaluated carrying out both physical implementations (timing, area, and power) and an architectural evaluations based on a set of real-life OpenMP applications featuring different program access patterns. The results are based on implementations of the Parallel Ultra-Low-Power (PULP) [8] [9] in 28nm Ultra Thin Body and Box Fully Depleted Silicon On Insulator (UTBB FD-SOI) technology, while an extensive architectural exploration is carried out with PULP prototypes in the Xilinx Zynq FPGA.

1. More details on the applications are provided in Section 4.1.

Results show that the proposed multi-port cache architecture can improve the performance with respect to the private cache by up to 40% in throughput, 20% in energy efficiency, and 30% in $energy \times area$ efficiency for sizes of instruction caches of few kB (typical of the low-power microcontrollers used in end-node IoT devices). While the multi-port is most suitable for small cache configurations, the shared cache architecture based on single-port SCM banks provides a better scalability in terms of cache size and number of cores, featuring 20% better $energy \times area$ efficiency than the multi-port, and up to 30% better energy efficiency than private caches.

The rest of the paper is organized as follow: Section 2 presents a review of the state of the art. Section 3 introduces the system architecture and the architecture of the three instruction caches explored in this paper. Section 4 shows the implementation results and the performance of real-life applications running on the three architectures. Finally, Section 5 provides some concluding remarks.

2 Related Work

2.1 Instruction Memory Hierarchy of ULP SoCs

Optimization of the instruction memory hierarchy is one of the main concerns in programmable ULP architectures because it can consume more than 50% of the overall system energy [8]. A common way to reduce energy consumption of the instruction memory hierarchy in ULP microcontrollers is to couple a few lines of register- or latch- based instruction cache with a bigger (up to few kB) instruction memory integrated in a dedicated power domain at higher voltage (if 6T SRAMs are used) or implemented as dedicated memory cuts optimized for near- or sub- threshold operation. Myers et al. [21] presented a Cortex M0+ that fetches instructions from a 4kB 10T SRAM optimized for sub-threshold operation. Sleepwalker [11] features a 64-bytes direct-mapped instruction cache implemented with registers operating at 0.4V, which refills on an 8kB 6T SRAM integrated into a 1.0V power domain. Almost 50% of the overall system power is reported to be consumed in the instruction memory hierarchy. REISC [12] features a more optimized instruction memory hierarchy, consisting of a 128-bytes direct-mapped instruction cache in the core power domain at 0.6V, that refills on an 8T SRAM featuring 0.6V periphery and 0.4V array. Still, the cost of the cache and memory on the overall power consumption is relevant: 10% and 23%, respectively.

When moving to ULP multi-core platforms, instruction memory hierarchy is confirmed to be a huge bottleneck for energy efficiency. PULP [8] features 4 Open-RISC cores with one kB of instruction cache, each implemented as 6T SRAMs and instantiated in a different power domain at higher voltage (+100 mV with respect to the logic), that refills on a 64kB program memory. The instruction memory hierarchy consumes more than 50% of the overall SoC power. A significant improvement has been achieved in PULPv2 [9], where the introduction of SCMs instead of SRAMs in the instruction cache increased the SoC energy efficiency by 38%. SCMs present extremely interesting features for low-voltage and energy-efficient designs, since: i) they are able to operate at a very low-voltage supply, even lower than 10T SRAMs optimized for low voltage [16]; and ii) their

energy per access is significantly smaller (from 2× to 4×) than both 8T [22] or 10T SRAMs optimized for low voltage, and standard 6T SRAMs [17]. On the other hand, although controlled placement of standard cells to build memory cuts demonstrated the ability to reduce area overhead from 4× to 2.7×, the overhead remains huge. For example, in PULPv2 the area of the 1kB private instruction cache is nearly the same as the area of the core [9].

2.2 Improving Energy Efficiency of Instruction Fetch Subsystem

Many specialized instruction cache structures have been studied in the past to reduce energy requirements while keeping the miss-rate as low as possible to preserve performance. Such structures include filter caches [23], loop caches [24], L-caches [25], Zero-Overhead Loop Buffers [26] and deterministic early miss detection [27]. These architectures reduce the pressure on the instruction cache, relying on small, energy-efficient buffers exploiting instruction locality. Hence, they are effective to improve energy in applications where the processor executes small critical kernels for most of the time. On the other hand, when the instruction locality is not fully exposed by the applications, and the instruction cache size is on the order of hundreds of bytes per core (typical of ultra-low-power computing platforms) [11] [12] [28], these intermediate buffers become redundant, adding a third level of memory hierarchy and inefficient because small caches cause a high miss rate. Although few tens of bytes of instruction cache can be sufficient for single-core micro-controllers (MCU) executing compute-intensive, bare-metal code, this might be insufficient for multi-core computing platforms executing OpenMP programs on top of a run-time library [19], because jumps to the run-time library break the instruction locality of kernels causing a huge miss rate and a drop of performance and energy efficiency [9].

In this work we propose an approach to join the benefits of SCMs in terms of access energy and voltage scalability, while reducing their area overhead by sharing the instruction cache among all the cores within a cluster. The idea of sharing instruction memory is not new and has been mainly exploited in high-end computing platforms such as General Purpose Graphic Processing Units (GP-GPU). In GP-GPUs all the compute units in each multiprocessor execute their threads in lock-step according to the order of instructions issued by the instruction dispatcher which is shared among all of them [29]. The same approach has been proposed to leverage the intrinsic SIMD nature of several signal processing applications [7]. The presented architecture combines a tightly coupled cluster of processors operating in near-threshold with a shared instruction memory with broadcast mechanism. This allows forwarding the same instruction to all the processors within the cluster whenever they request the same address on the same cycle, reducing instruction fetch energy. Moreover, a hardware synchronization mechanism dynamically manages the lockstep execution of cores during data-dependent program flows [7]. Although this approach achieves 60% energy reduction, its applicability is restricted to data-parallel code sequences, and it is intrusive from the software viewpoint, as it requires explicitly activating and deactivating lock-step execution. Moreover, it employs a single level of memory hierarchy (not an instruction cache) and all the program code has to be stored in L1 instruction memory implemented with SRAMs, leading to bigger requirements and higher power with respect to the hierarchical and heterogeneous memory architecture that we present (i.e. SCM-based instruction cache that refills from an SRAM-based L2 memory).

2.3 Exploiting Shared Instruction Cache in Tightly-Coupled Clusters

An in-depth study of the private and shared cache architectures based on a SystemC platform running micro-benchmarks and few parallel applications [18] shows the benefits of I$ sharing on a system architecture typical of a high-end many-core platform (i.e. 16-32 cores per cluster, 50 to 150 cycles of refill latency typical of DDR memories). Although this is the typical configuration for high-end clustered many-core architectures [30] [31] and GP-GPUs [29], a large number of processors per cluster causes the power of the interconnect between the processors and the shared memory banks to dominate [32], significantly reducing the energy efficiency of the cluster. In this work, we focus on the architectural exploration of an energy-efficient and low-cost (area), tightly coupled cluster composed of 2 to 8 processors, taking advantage of parallelism and near-threshold operation to improve energy efficiency rather than performance. This cluster configuration is representative of most state-of-the-art near-threshold clustered many-core architectures [33] [8] [9]. Moreover, in [18] the exploration was conducted on a SystemC model, hence not reflecting all the interactions among architecture, micro-architecture and physical implementation on a 4 processor PULP cluster. Loi et al. [20] explored SCM-based shared cache, demonstrating the benefits of the shared cache architecture on a 4 processors PULP cluster. The main limits of the presented architecture are the high timing pressure caused by the presence of an interconnect between the cores and the shared memory banks which causes significant power overhead with respect to a private cache architecture, and the contention on the shared memory banks that can cause a performance drop if not properly managed.

In this work, we further extend this exploration by i) introducing a novel multi-ported cache architecture that exploits the benefits of the previously introduced shared cache exposing a large memory map to each core while eliminating contention and timing pressure of the previous solution; ii) analyzing the performance, area and power implications of the three architectures (private, shared, shared + multi-port) on a 28nm UTBB FD-SOI technology in near threshold; iii) proposing an extensive evaluation of the three solutions based on a wide set of real-life parallel applications implemented in OpenMP running on a cycle-accurate FPGA emulator of the tightly-coupled multi-core cluster; iv) extending the results to a wide range of configurations typical of energy-efficient tightly coupled processor clusters, both in terms of cache capacity (1kB, 2kB, 4kB, 8kB) and number of processors within a cluster (2, 4, 8).

3 Architecture

This section presents the system architecture of the target clustered many-core platform and the architecture of the PULP tightly-coupled cluster [8] where the shared program caches are deployed. Moreover, it provides a detailed description of the instruction cache's micro-architecture and their integration in the multi-core cluster.

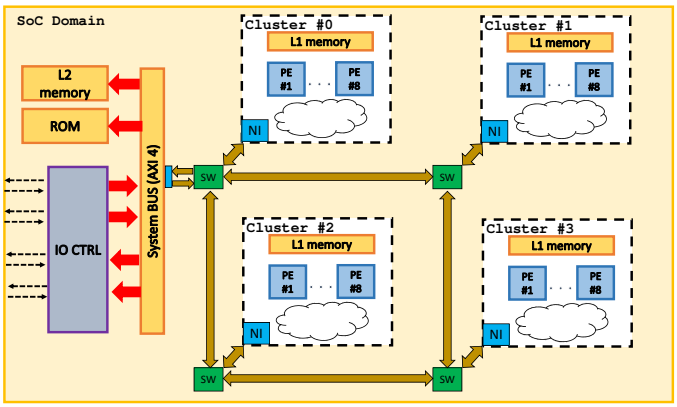


Fig. 2: SoC architecture with 4 PULP clusters. Inter-cluster communication infrastructure is based on regular NoC.

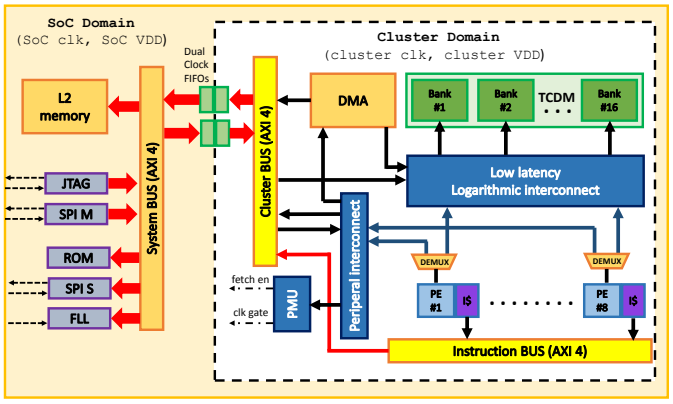


Fig. 3: Overall SoC (mono cluster depicted for the sake of simplicity), where each processing element embeds a private instruction cache.

3.1 SoC Architecture

PULP is a scalable clustered many-core system that defines at design-time its configuration with as many computing clusters as the applications require. As shown in Figure 2 the PULP fabric is integrated in a SoC featuring L2 memory, SoC peripherals and IOs shared among all clusters through a fabric interconnect. In this work, we focus on a single cluster PULP architecture [8], as shown in Figure 3.

The PULP cluster (see Figure 3), that we consider as a baseline for the proposed exploration, includes a configurable number of processing elements (PEs), based on OpenRISC OR10N cores [34] [35], each featuring a private instruction cache. The OR10N cores are based on an in-order, single-issue, four stage pipeline micro-architecture without branch prediction, improved with extensions for higher throughput and energy efficiency in parallel signal processing workloads [34]. No data caches are present, therefore avoiding memory coherency overhead and additional area penalties [30]. PEs share a ultra-low-latency L1 multi-banked Tightly Coupled Data Memory (TCDM) acting as a shared data SPM and a DMA tightly coupled to the TCDM [36]. In the context of this work, the total amount of TCDM we consider is 128kB [37], and a banking factor of 2 (i.e. the number of TCDM banks is twice the number of cores). The refill ports of the instruction caches converge on a common cluster instruction master port through an AXI4 instruction bus. SoC and cluster domains are clocked with two different Frequency-Locked Loops (FLLs) and can operate with different supply voltages. Hence, while the supply voltage of the SoC is limited to 0.8V by the presence of high-density memory banks used to implement the L2 memory, the supply voltage of the cluster, implemented with SCMs and low-voltage SRAMs can scale down to near-threshold, improving its energy efficiency. Both data and instructions are routed in and out of the cluster through an AXI4 cluster bus featuring dual clock FIFOs for clock domain crossing, causing an overhead of approximately 3 clock cycles of latency. On the SoC Domain, data/instructions are routed to the L2 memory through a System BUS (AXI). In the following, unless explicitly specified, we refer to a cluster composed of 8 PEs and 128kB of TCDM.

3.2 Private Instruction Cache

Our baseline cluster features private instruction caches, with a cache line of 32 bytes (8 Words), set-associativity of 4, and a pseudo-random (PRAND) replacement policy. To define the set of parameters we made a preliminary exploration on private caches, analyzing the impact of the set-associativity and replacement policy on different applications. Figure 4 shows the impact on the execution time of different set-associativity values analyzed, normalized with respect to the 4-way set-associative configuration. We can note that increasing the associativity to more than 4 improves performance by only up to 1%, while causing significant power overhead. On the other hand, the direct mapped and the two-way set-associative cache show a significant degradation of performance. Even if caches with smaller associativity consume less power, energy-wise the degradation of performance is more significant, since this cause the power of the whole cluster (i.e. not only of the cache) to be integrated over a longer period. The chosen PRAND replacement policy, which is simplest from the hardware implementation point of view, doesn't generate compulsory misses during the execution of our benchmark set once the cache is warmed up. These considerations led to the choice of the described baseline parameters for exploration.

The DATA and TAG array, and the decoders are implemented using conventional standard cells: the memory elements are based on latches [17], while the periphery logic is based on common cells gates. The processor fetch stage has been deeply optimized to reduce the timing pressure and dependency between the cache interface and the core pipeline. The fetch interface is based on a *request-grant* handshake protocol that relies on two channels: the *request* and the *response channel*. Through the *request channel*, the core issues a fetch request by asserting both the *request* and the

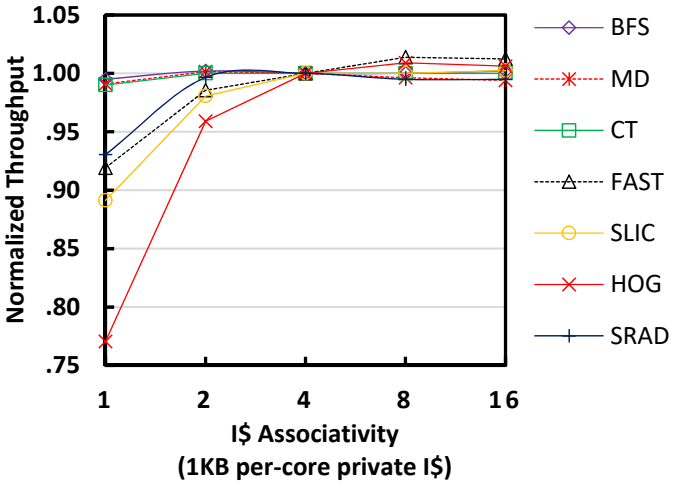


Fig. 4: Normalized throughput (to the 4-way set associative configuration) for increasing cache associativity.

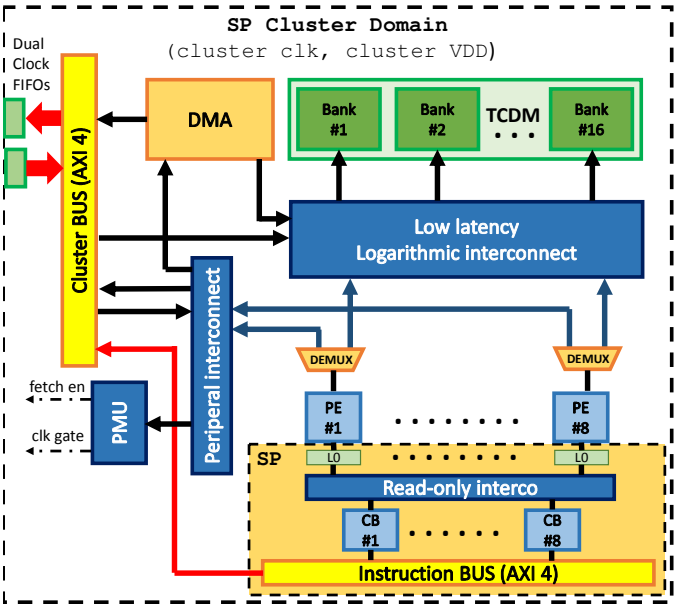


Fig. 5: Cluster architecture for Single-Ported Shared instruction cache. Cache banks are shared through a fast read-only interconnect, and a Cache filter (L0) is used to reduce the pressure on the cache bank side.

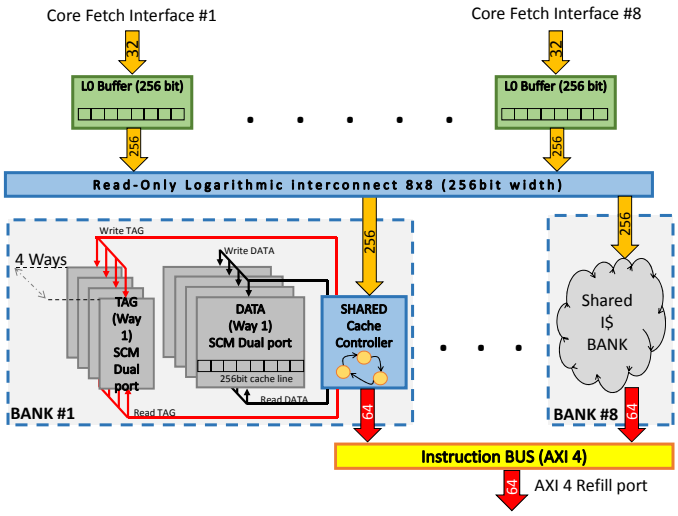


Fig. 6: Shared I$ with single read-port SCM

fetch address, and the *grant* is used to regulate the flow of requests. Once the instruction is available in the cache, the core is notified through the *response channel*, and a 32 bits instruction is moved into the core. In case of a hit, the fetch response comes on the following cycle. In case of a miss, the private cache controller blocks the core pipeline and creates an AXI read request to the L2 to retrieve the missing cache line (256 bits split into 4 burst data chunks since the AXI infrastructure is 64 bits wide). The transaction first crosses the AXI instruction bus, then the cluster bus and finally crosses the cluster and reaches the L2 memory subsystem (SoC Domain). Then it comes back as a 4-beat, burst-read transaction. In case of no contention on the AXI L2 interface, the request that generated the miss is concluded in 14 cycles (assuming same clock cycle in the cluster domain and SoC domain, and measured from fetch request to fetch response).

Data replication is the major drawback for private instruction cache multi-core systems, which leads to a degraded energy and area efficiency. In such systems (e.g., in [30]) processors work in parallel and execute the same kernel on different data-sets. Hence several cores fetch the same code segment, that is replicated in each private cache. Therefore, multiple refill requests are asserted at the same address, creating intense traffic to a remote L2 memory subsystem.

3.3 Shared Instruction Cache

Instruction cache sharing is an attractive option to increase the effective capacity of the instruction cache for a given area budget, at the cost of a higher complexity. Hence, the main challenges for shared instruction cache design successfully addressed in this work are:

- do not introduce additional cycles of latency in the processor fetch interface with respect to private caches to keep the bandwidth as high as possible (single-cycle latency).
- do not introduce additional timing pressure on the processor fetch interface to operate at the same frequency as the private cache architecture (100 MHz).
- do not introduce any additional cycle of latency in the path between the instruction cache refill and the L2 memory (14 cycles).
- minimize the area overhead.

Figure 5 illustrates the shared instruction cache micro architecture, where private caches are removed from cores and replaced by shared cache banks (CB) through a very thin and fast crossbar interconnect based on logarithmic trees [32]. This cache architecture is based on [20], and is briefly described in the following. Since CBs are shared among the cores, data replication is avoided, but two or more cores may compete to fetch instructions stored on the same cache line, or on the same cache bank. In such a condition, a round-robin arbitration policy ensures that only one core can access the CB while the others are stalled.

Hence, statistically every core has the same probability to get access to the CB. To better spread access among multiple CBs, the interconnect maps the CBs with an interleaved address scheme (implemented by the Read-only interconnect) at the granularity of a cache line. To reduce the pressure (and contention) on the shared CBs we added an instruction buffer (L0) which is capable of holding a cache line (256 bit or 8 instructions). To avoid any IPC penalty, this buffer checks the local availability of the instruction, and in case of a miss forwards in the same cycle the request to the target.

Figure 6 illustrates the internal architecture of the shared instruction cache. DATA and TAG (SMC based) are identical to the private cache, so every way has one read port and one write port controlled by its own CB cache controller. Since CBs are shared, they have been designed to support multiple outstanding transactions and non-blocking behavior in case of miss. If a miss happens, the CB asserts the respective refill request, and then, while waiting for the refill response from L2, processes incoming fetches for the other cores. The shared cache controller is also able to track more than one pending refill. This is translated in additional storage resources to track pending misses and to restore the right order from response coming from the AXI infrastructure.

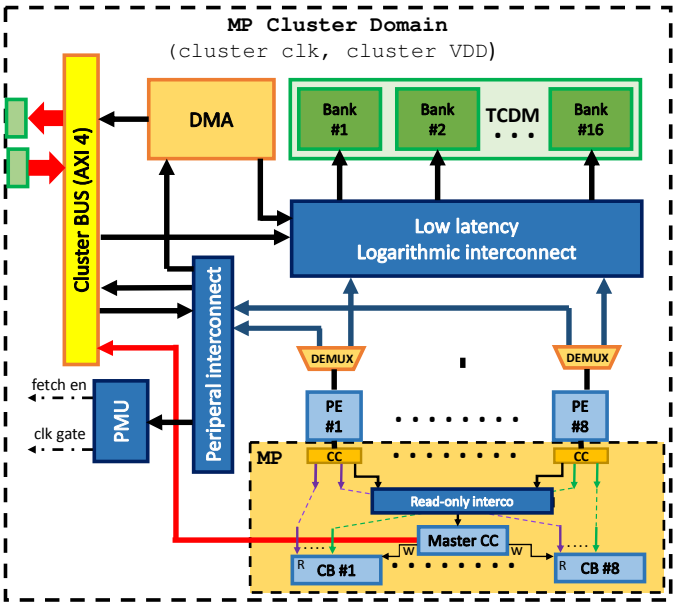


Fig. 7: Cluster architecture for multi-ported shared instruction cache.

3.4 Multi-port Instruction Cache

Another option to build a shared instruction cache is to share only TAG and DATA memories, while keeping the cache controllers private and close to the core fetch interfaces. Figure 7 illustrates the cluster architecture with the multi-port shared instruction cache, while Figure 8 shows the detailed view of this cache. To reduce memory complexity, DATA and TAG are split into several banks, and each bank is composed of 4 ways (TAG+DATA). Each of these arrays has a single write port (used to populate the cache after a miss), and 8 read ports, one for each private cache controller (8 cores) and used for normal cache access (hit/miss). By doing so, each cache controller has a direct and private path (zero contention) to read TAG and DATA memories, therefore this cache is similar to the private one, with significant differences on the way the misses are handled.

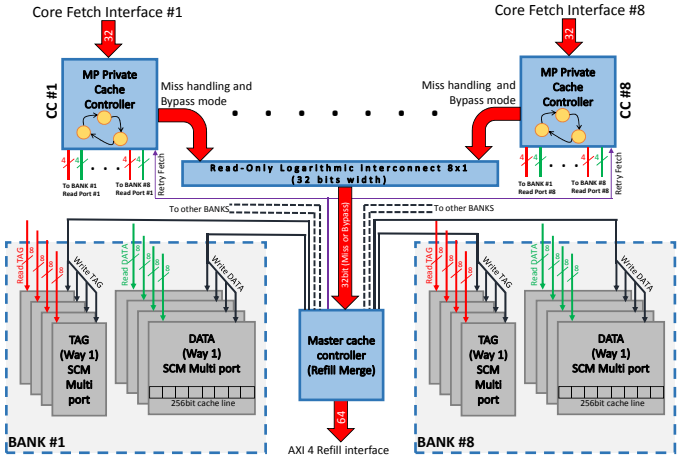


Fig. 8: Shared cache based on multi read-port SCM.

In case of a miss, the private cache directly sends the refill request through the AXI instruction BUS. If all the cores are trying to fetch the same address, then 8 refills are generated. In the shared multi-ported cache, refills are handled by a dedicated master cache controller, which is in charge of offering global services (e.g. flush, enable, bypass), of taking care of refill requests, and of dispatching to the TAG and DATA arrays. In Figure 8 and Figure 9, we can observe that after the cache controller issues a miss, the address is pushed to the master cache controller, through the read only interconnect (multiplexed with other possible miss requests). Once it arrives in the master cache controller, the miss address is stored in a CAM (Content Addressable Memory), and the AXI refill request is generated and dispatched to the L2 memory.

Once the refill response comes back, the master cache controller first clears the cache line valid bit (invalidate) while it updates progressively the cache line in the DATA memory. This operation is performed using the dedicated WRITE DATA and WRITE TAG channels. Finally, when the last chunk of data arrives (each cache line is served with a 4-beat burst-read transaction), then the master cache controller sets the cache line to valid, and clears the relative miss entry in the CAM. Finally, the master cache controller notifies the private cache controllers that the pending miss is concluded, and then the private cache controllers retry the fetch.

In case of multiple misses belonging to the same cache line, a merging logic takes care of packing the AXI transactions to L2. As shown in Figure 9, the address is pushed in a special CAM that generates a unique AXI ID (entry ID), and allows to push using the address as key. If that address is stored in the CAM, no refill requests are generated, and only the CORE ID field is updated (meaning that when refill response comes back, the cores listed in the CORE ID field will be notified). If the address is not listed, then the first empty CAM entry is allocated and the refill request is asserted to L2.

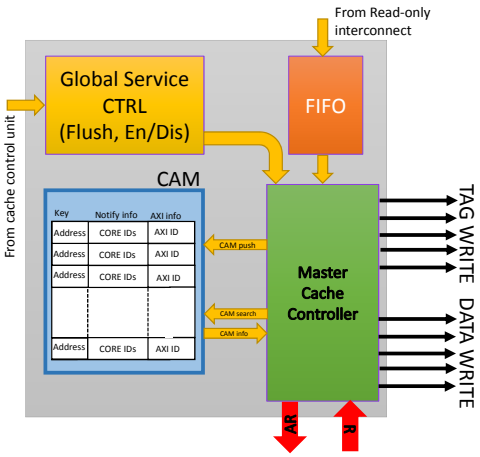


Fig. 9: Mergin refill logic and DATA-TAG updating logic following a refill resposnse and Global services support

4 Results

This section presents the comparison of the presented cache architectures. The results include both physical implementation metrics such as area, power timing and an extensive benchmarking based on the execution of parallel signal processing applications on the cluster.

4.1 Experimental Setup

To analyze the trade-offs between the discussed cache architectures in terms of performance, energy and area, we consider a single cluster with a configurable number of processors (2 to 8), and a configurable size of the instruction cache (1kB to 8kB), while the size of the TCDM is fixed to 128kB. Table 1 summarizes all the architectures used in our experiments, where a 4-way set associative configuration with a cache line width of 32 bytes (8 words) is used. To obtain physical implementation results (i.e. frequency, area, power) the design was synthesized and implemented on a 28nm CMOS UTBB FD-SOI technology library. The design was synthesized with Synopsys Design Compiler J-2014.09-SP4, while the place and route was performed with Synopsys IC compiler H-2013.03-SP4. To characterize the power consumption of the proposed instruction cache architectures, we assumed that the cluster domain operates at the supply voltage of 0.6V, while the SoC domain operates at 0.8V. To extract the average power consumption of the cluster we simulated its post place-and-route netlist when running a synthetic program able to achieve an IPC close to 1. Then, we back-annotated the switching activity traces in VCD (Value Change Dump) format, and we passed it, together with the parasitic file (in SPEF format), to Synopsys PrimePower H-2013.06-SP2 to achieve an accurate power estimation.

To analyze in depth the behavior of each architecture we used seven benchmarks based on full-fledged applications, each featuring a different behavior in terms of access patterns to the instruction memory subsystem, as well as diversified memory footprint and execution time. All the applications are parallelized using an optimized OpenMP implementation [40] and compiled with GCC 5.1. Table 2 shows in detail the characteristics of each application, their

Mnemonic	TCCC	Kind	Description
8K-PR	8kB	*Private*	1024B I$ bank per core
4K-PR	4kB	*Private*	512B I$ bank per core
2K-PR	2kB	*Private*	256B I$ bank per core
1K-PR	1kB	*Private*	128B I$ bank per core
8K-SP	8kB	*Shared*	8x 1024B I$ banks, single-port
4K-SP	4kB	*Shared*	8x 512B I$ banks, single-port
2K-SP	2kB	*Shared*	8x 256B I$ banks, single-port
1K-SP	1kB	*Shared*	8x 128B I$ banks, single-port
8K-MP	8kB	*Shared*	8x 1024B I$ banks, 8-port
4K-MP	4kB	*Shared*	8x 512B I$ banks, 8-port
2K-MP	2kB	*Shared*	8x 256B I$ banks, 8-port
1K-MP	1kB	*Shared*	8x 128B I$ banks, 8-port

TABLE 1: Architectural configurations details

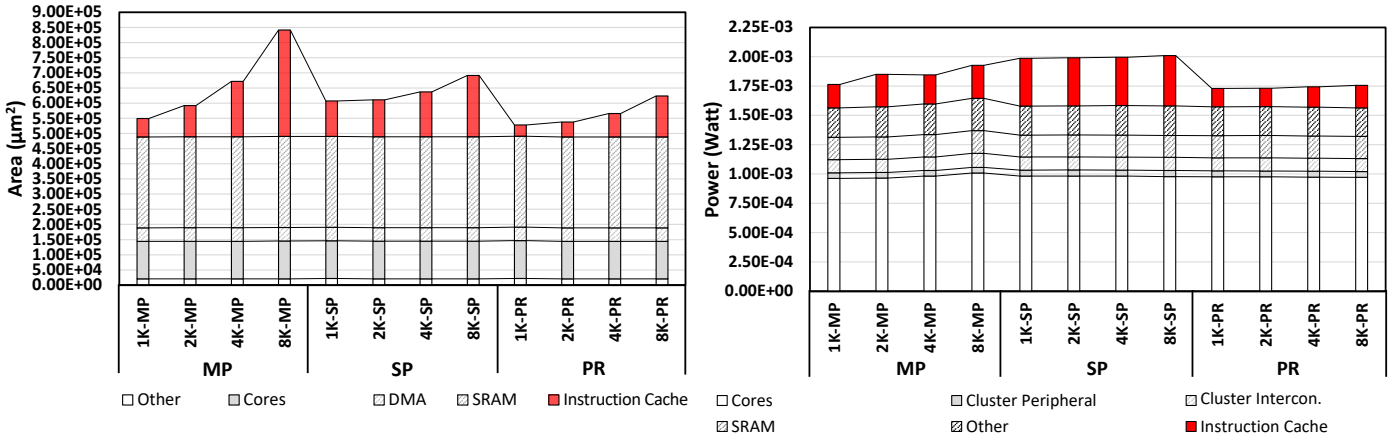
Mnem	Size	#Instr.	Class	Description
BFS	1.8KB	0.2M	*Short-Jump*	Breadth-First Search [38]
MD	5.0KB	0.9M	*Short-Jump*	Motion Detection
CT	2.9KB	7.9M	*Long-Jump*	Color Tracking
FAST	2.7KB	2.7M	*Long-Jump*	Machine-generated corner detection algorithm [39]
SLIC	26.1KB	41.5M	*Long-Jump*	Simple Linear Iterative Clustering for image segmentation
HOG	31.1KB	103.9M	*Library*	Histogram of Oriented Gradients
SRAD	30.2KB	1.5M	*Library*	Speckle Reducing Anisotropic Diffusion [38]

TABLE 2: Benchmark set details

code section sizes, expressed in kB, and the number of RISC 32-bit instructions needed for each execution. Since cache performance and code locality are strongly influenced by the control flow of each application[2], we classified the seven applications in three groups. *Short-Jump* class includes BFS and MD, which are loop-based applications with most of loop bodies smaller than two lines of cache. The second class of applications, called *Long-Jump*, groups all the loop-based applications with loop bodies greater than two lines of cache, or based on extensive use of control flow instructions. This is the case of CT, FAST, and SLIC. Finally, the *Library* class includes all the benchmarks that extensively use external libraries. In our benchmark suite, HOG and SRAD use libraries to manage non-native data types, such as float, and fixed point arithmetic, generating a big stress on caches. The classification is coherent with the performance degradation showed in Figure 1. To perform a fair comparison, avoiding spurious advantages of shared configuration in case of cold caches, our experimental results focus on the third execution run of each application.

All these applications are fairly complex and long-lived (millions of instructions in most cases), which makes their use impractical for architectural exploration based on RTL simulation. For this reason, the performance analysis is based on measures coming from RTL-equivalent, fully cycle accurate FPGA implementations, mapped on Xilinx Zynq ZC720 FPGA using Vivado 2015.1. This approach allows to

2. not only frequency of conditional statements and displacement in memory of their branches' code, but also frequency of function calls, size of these functions, etc.



(a) Cluster Area breakdown for the different cache architectures and configurations. Instruction cache contribution is placed on top.

(b) Cluster Power breakdown for the different cache architectures and configurations. Instruction cache contribution is placed on top.

Fig. 10: Area and Power implementation results for the different architectural configurations.

execute at up to 80 MHz, enabling near-to-real-life execution time. The performance analysis is based on statistics collected by hardware counters implemented inside the cluster. Energy figures are extracted combining results coming both from the FPGA emulation (performance) and post place-and-route analysis (power extraction) in different workload conditions (e.g. 1-core active, 2-cores active, etc.). All the power models (coming from an accurate power characterization) and the statistics collected in the FPGA emulation have been combined together in Matlab to estimate the total energy requirements of the different applications.

4.2 Implementation results

To characterize the performance of the various instruction cache architectures, we first performed a wide physical exploration in terms of timing, area, power. For comparison, we have implemented several variants of a baseline cluster with 8 cores, considering the three I$ designs and the size configurations listed in Table 1. For each *Total Cluster Cache Capacity* (*TCCC*) we consider the overall number of instruction cache bits within the cluster. In other words, in case of a shared cache architecture the *TCCC* represents the real cache capacity seen by the processors, while in case of private cache architectures the actual cache capacity seen by each core is $1/8$ of the *TCCC*. For example, if we consider a *TCCC* of 1kB we have 1kB of shared cache, or 8x128B of private cache, assuming an 8-processors cluster. In the following, we will refer to the private, shared and multi-ported shared cache designs with the following acronyms: PR, SP, MP.

From a timing perspective, the cache interface is usually one of the most critical blocks in embedded processors due to the complexity of the instruction fetch stage. Tightly coupled clusters of processors [30] feature a shared scratchpad memory (TCDMs) directly plugged to the load store unit of the processors through a single-cycle-latency interconnect, which is often critical as well. In case of a shared instruction cache, the additional delay of the interconnect increases the (already critical) delay of the instruction fetch stage, making the paths toward cache banks even more critical. For those devices in which high frequency operation is a requirement (e.g., ARM Cortex A-class processors), this large delay is typically managed by pipelining the paths from the instruction fetch stage of the processors to the instruction cache, and by adding a large branch prediction table to the core to minimize the probability of mis-predicted branches (and the related stalls).

In this work we focus on IoT end-nodes optimized for energy efficiency (lower operating frequency, high IPC, e.g., Cortex M-class processor). For this type of design, we can employ tiny cores without branch prediction table, which enables single-cycle response latency for the shared I$ design and maximizes the IPC. To balance the delays on the data and instruction paths of the processors, the proposed caches have been optimized in a close loop fashion with the core fetch interface to reduce the pressure timing-wise. Despite the latency from core to instruction cache being close to the one across the load store unit, the optimizations adopted allow the cluster to achieve the target frequency of 100 MHz at 0.6V for all the explored configurations (the most critical path being the one between the cores' LSU and the TCDM). This is an important milestone in our exploration since any improvement in the application execution time and power consumption with respect to the private cache configuration directly translates into a better system energy efficiency.

Figure 10a illustrates the silicon area costs for both private and shared configurations. For every *TCCC* the private cache configurations are smaller than the shared cache configurations, thanks to their simpler architecture. It is important to remember that for the private configurations the actual capacity seen by each core is $1/8$ of the *TCCC*, hence the shared caches are "virtually" 8 times bigger. The equi-*TCCC* analysis presented in Figure 10a also shows that for small cache sizes (e.g. 1kB) the SP memory is bigger than the equivalent MP, since in this condition the area of the SP is dominated by the interconnect and the master cache controller which form a significantly higher offset with respect to the 8 read ports of the MP. On the other

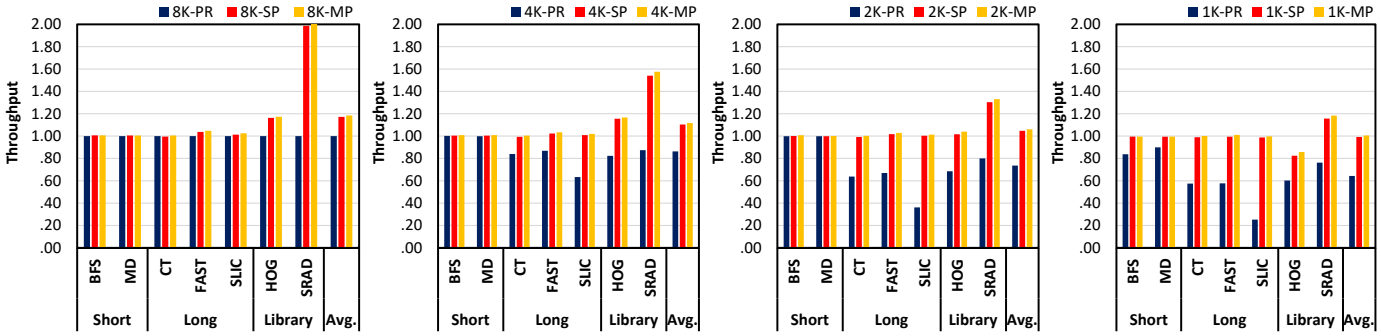


Fig. 11: Throughput of each applications normalized to 8K-PR configuration. Charts group equi-capacity architectures.

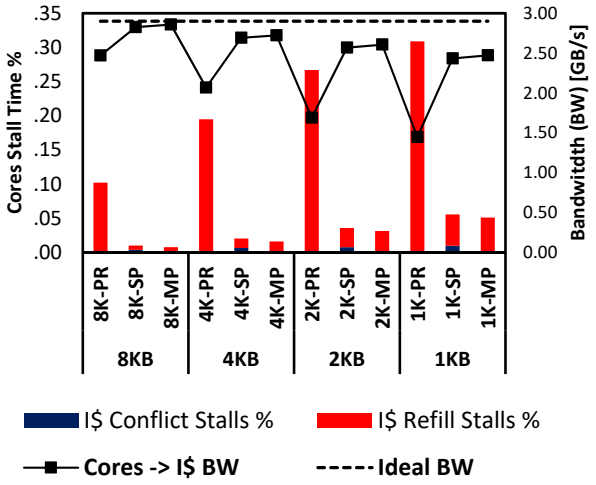


Fig. 12: Efficiency loss due to I$ inefficiencies. The y-axis on the left shows the percentage related to the total execution of core stalls due to: *I$ Refill Stalls* (red bars) and *I$ Conflict Stalls* (blue bars) on the SP architectures. The y-axis on the right shows the measured bandwidth in GB/s generated by the different I$ configurations compared to the bandwidth achievable by an ideal instruction cache.

hand, increasing the *TCCC* (e.g. see 8kB) causes the area of the MP configuration to increase, mainly due to congestion problems when dealing with big arrays coupled with a large number of ports. Following these considerations, the first intuition is that the MP provides better area results for smaller *TCCC*, while the SP provides better area results for large *TCCC*.

Figure 10b illustrates the power numbers for clusters with both private and shared configurations, when running a synthetic program at the operating frequency of 100MHz and supply voltage of 0.6V. The first observation is that, as opposed to SRAM-based instruction cache, the power cost of the SCM design is almost insensitive to their size, due to the careful design of the latch-based memory banks, which are always clock gated unless a write operation (refill) is being performed [17]. Hence, in case of read operation (typical of a fetch from processor) only the read address decoder and the output multiplexer consume power. This can be seen by looking at the flat behavior of the SP and PR configurations when increasing the size. Even for power, as expected, the lowest figures are achieved for the private configurations, since the design complexity is smaller than that of the shared cache architectures. Between SP and MP, we note that the MP power is smaller than SP, and despite the 4K-MP and 8K-MP are bigger (area) than SP, the MP benefits from lower switching activity (no interconnect between cores and cache) and clock-gating control (private cache controllers, simpler and smaller). On the other hand, the power exploration confirms the suitability of the MP for small *TCCC*, consuming 11% less power than the SP for the 1kB configuration, while for large *TCCC* (e.g. 8kB) the power advantage of the MP over the SP is only 4%. This is caused by the previously highlighted congestion problems of the MP solution when scaling up the size of the array, which also slightly increases the power of other components of the cluster (i.e. processors).

4.3 Benchmarking

This section evaluates the main metrics relevant for low-cost, low-power processor clusters, namely performance, energy, and energy-area, which is the most significant evaluation parameter for the presented instruction cache architectures.

Figure 11 resumes the throughput of applications measured on the different architectural configurations, normalized over the 8K-PR configuration, that we consider as a baseline. If we consider equi-*TCCC*, shared cache architectures always show better performance when compared to the related private configurations. Clearly, for big cache capacities, the benefit of having a shared cache is only visible on applications not completely fitting in the private cache size (i.e. $TCCC/8$) such as SRAD. On the other hand, when moving to the small side of *TCCC* (i.e. 1kB), the benefit of employing shared instruction cache architectures is significant also for applications featuring frequent jumps, leading to more than 40% better performance on average. If we look at the graphs globally, the best performance is achieved by bigger instruction caches. The only application showing a degradation of performance with respect to the baseline (i.e. 8K private) when implemented with shared cache is HOG, due to the bigger execution time among the selected benchmarks and highly divergent threads which run concurrently on different cores and cause pollution of cache lines in the shared cache, requiring some extra refills, especially for small configurations (e.g. 1kB). However, it is interesting to note that globally, shared instruction caches

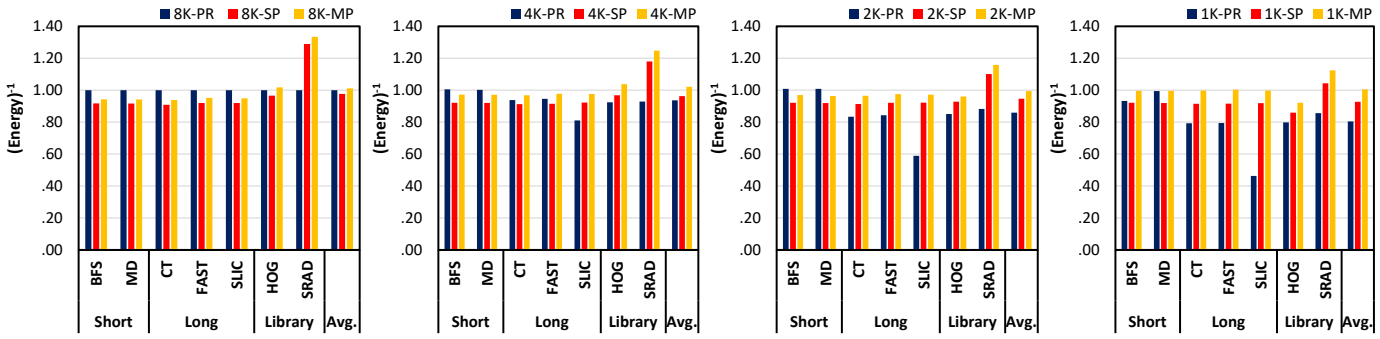


Fig. 13: Inverse of Energy consumption of each applications normalized to 8K-PR configuration. Charts group equi-*TCCC* architectures.

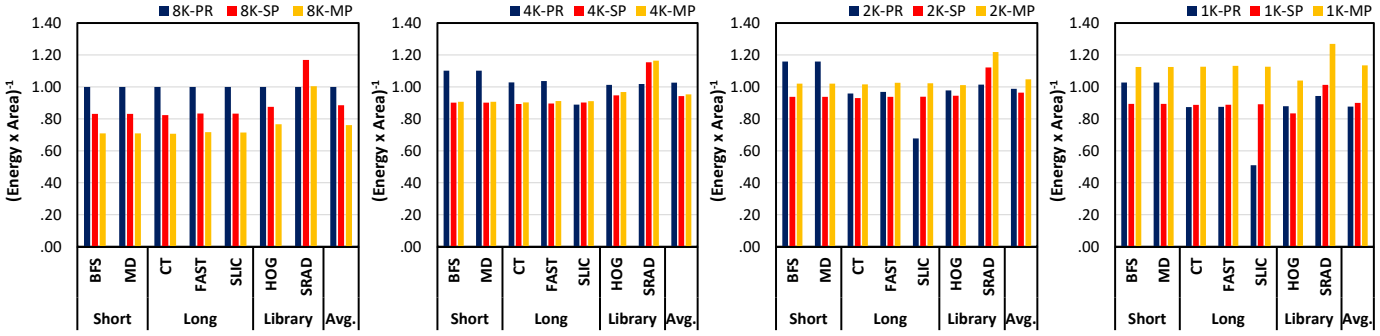


Fig. 14: Inverse of Energy-Area product of each applications normalized to 8K-PR configuration. Charts group equi-*TCCC* architectures.

provide significantly better robustness with respect to private instruction caches, when considering all the different classes of applications proposed for benchmarking.

Figure 12 shows the average bandwidth requested by the processors on the whole benchmark set (ideal BW) and the bandwidth delivered by the instruction cache for the different architectural configurations. Ideal BW is equivalent to the bandwidth delivered by an "ideal" instruction cache featuring one cycle of latency and infinite size. The non-idealities of the instruction caches cause a reduction of bandwidth degrading the performance and energy efficiency of the cluster. Figure 12 also shows the normalized amount of stalls caused by the non-idealities of the instruction caches: refills and contention (only for the SP). For each *TCCC* shared caches show up to 10× less refill stalls due to the smaller number of refills due to capacity miss. Moreover, in this scenario all the private caches within the cluster likely issue a refill at the same time, creating congestion on the AXI4 bus and L2 memory. This effect can be noted by looking at the 8K-PR and 1K-MP configurations. In these two cases each core has the visibility of the same cache capacity (1kB), but the 1K-MP configuration features almost 50% fewer stalls just by avoiding congestion on the AXI bus and L2 memory. Finally, if we compare the SP and MP configurations, we can note that effect of congestion on the shared memory banks in the SP cache, leading to a degradation of performance by less than 2% on average.

Figure 13 shows the energy efficiency (Energy^{-1}) for the different applications normalized to the 8K-PR configuration. The graphs show the same trend as the performance, due to the high correlation between the execution time of applications and the energy spent at cluster level to execute the application. The main difference here is related to the slightly better energy efficiency of the MP cache with respect to the SP, caused by the higher power consumption of the latter due to the presence of the interconnect between the cores and the memory banks. On the other hand, the energy efficiency of both SP and MP solutions is better than the one of the private cache for every *TCCC*, while if we look at the graphs globally the MP provides the best average energy efficiency (4K-MP), surpassing by 2% the baseline configuration (8K-PR). Globally, shared cache architectures, in particular the MP, have a comparable energy consumption among all the configurations, even for small caches, which is not the case for the private architecture. This is a tremendous drawback of the private cache architecture when it targets highly constrained domains such as parallel ULP architectures.

Given the described proportionality of performance and energy efficiency with the size (area), in case of private caches, and the importance of silicon area (cost) in the domain of ULP tightly-coupled clusters of processors - especially when speculating on area-expansive latch-based instruction caches to get an energy return - we propose an evaluation metric that takes into account performance, power and area. Based on the area implementation results shown in the previous section and the energy results presented in the latest paragraph, the charts in Figure 14 show the inverse of Energy-Area product ($(energy \times area)^{-1}$), normalized to the baseline configuration (8K-PR). When we

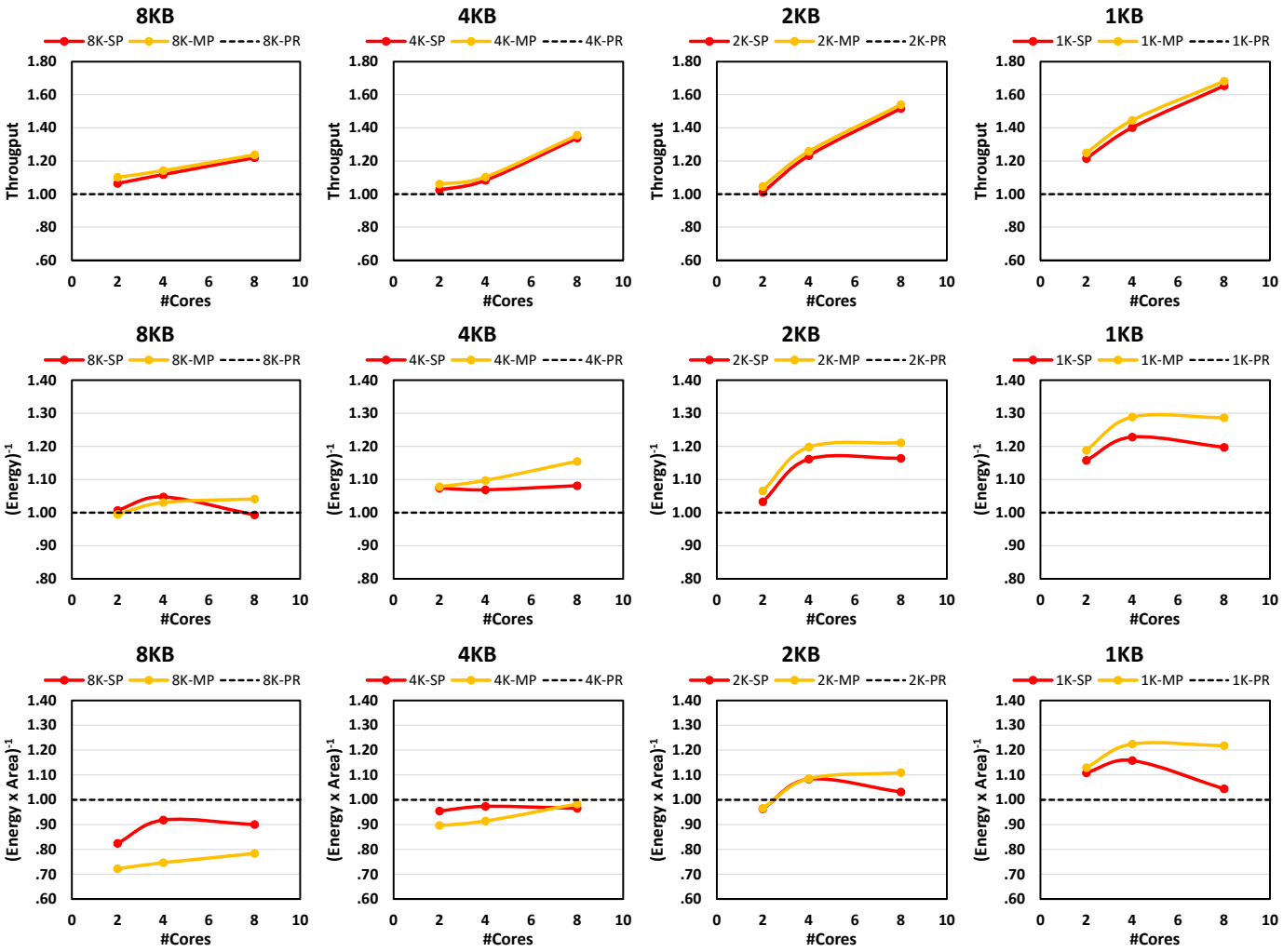


Fig. 15: Scalability analysis of the cache architectures increasing the number of core inside the cluster.

introduce this metric, for large *TCCC* (i.e. 4K-* and 8K-*) we can note that the private cache performs better on average, since for these configurations the shared caches are oversized for the chosen applications, while paying significant area overhead with respect to the private cache. On the other hand, when moving to small *TCCC* (i.e. 2K-* and 1K-*) the applications start to suffer from capacity misses, and the overhead of the MP with respect to the PR memory is much smaller. As a consequence, the 1K-MP configuration provides the best absolute $energy \times area$ metric, surpassing by 15% all the other configurations. It is interesting to note that, comparing the SP with the MP solution, the former provides a better $energy \times area$ metric for large configurations, while the latter provides a better metric for small configurations. Hence, for large, library based applications that do not fit even the largest instruction cache explored (e.g. SRAD), the best absolute instruction cache architecture is represented by the SP.

Figure 15 illustrates the performance, energy and $energy \times area$ metrics of the cache architecture when scaling down the number of processors within the cluster from 8 to 2. The data in each plot is normalized to the related private cache configuration (number of cores, *TCCC*). Hence, in this figure, each graph has to be co considered independently from the others. If we consider, like in the previous plots, an equi-*TCCC* analysis, it is possible to see that the benefits of the shared cache architectures in terms of throughput scales down with the number of cores. This is caused by the fact that the cache is shared among a smaller number of cores. In the MP configurations, the actual cache capacity seen by each core increases with smaller clusters. More precisely, in a 4-core cluster PR configuration, the cache capacity seen by the core is $TCCC/4$, while it is $TCCC/2$ for a 2-core cluster. Moreover, the replication of code within the private caches, causes less overheads, since having fewer cores implies less contention upon refills. As highlighted in previous experiments, the benefits of the shared cache configurations increases when decreasing the *TCCC*. When we consider the energy and $energy \times area$ metrics, for small *TCCC* configurations (e.g. 1KB-*) the MP provides better performance in all cases, thanks to the smaller overhead in terms of area and power with respect to the SP. When we increase the *TCCC* beyond 4KB, the SP configuration provides better results for any number of cores, confirming the better suitability of the SP configuration for large cache sizes and number of cores.

Until this point, we always considered the different

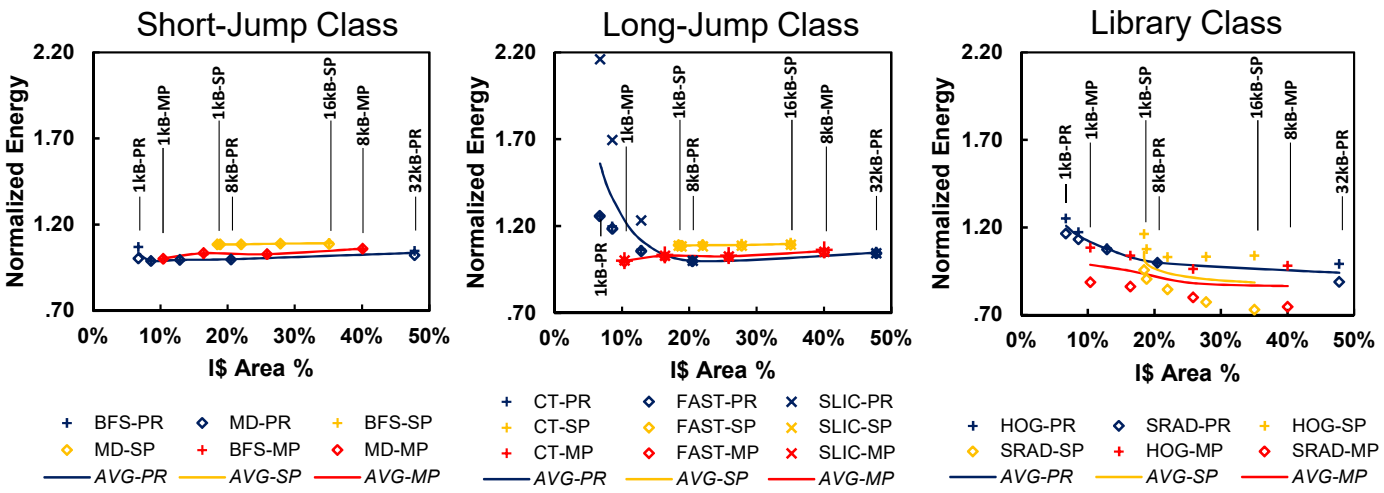


Fig. 16: Energy-Area Pareto charts. Each chart groups the measured Energy-Area coordinates for the benchmarks within the same class of application, while the curves show the trend for each architectural configuration. To provide a complete coverage of I$ Area% span, additional architectural configurations were considered too: 32K-PR and 16K-SP[3].

instruction cache architectures in terms of throughput, area, or energy in a constant benchmark set scenario. If we now take the point of view of hardware architects, who are usually less concerned about benchmarks and more focused on designing systems with fixed and strict area constraints and where - sometimes - the target application domain is not well-defined, flexibility and architectural robustness to various types of applications is crucial. With this idea in mind, Figure 16 shows three Pareto charts, one for each class of application. The x-axis shows the percentage of cluster area dedicated to the instruction cache. The different markers show on the y-axis the normalized energy consumption (over the 8K-PR instance) for each application, while the curves group the average energy consumption trends for the three instruction cache architectures, differentiated by colors: blue for PR, yellow for SP, and red for the MP.

The plots show that while private instruction caches are highly sensitive to area constraints, shared architectures show a robust responsiveness to locality variations. PR cache can be a good solution in case of applications with high locality, but this is not the case when more complex applications are taken into account, or when strict area constraints are in place. For parallel workloads, shared instruction caches improve hit rate by avoiding data replication. This becomes key when the cache size is small (10% of the cluster area) and applications rely on library calls (shared caches can host a larger portion of such applications compared to private caches, which partition the available space). Focusing on shared cache architectures, we can conclude that MP architectures always dominate SP when the area constraint is strict. Small cache instances increase the probability of incurring a bank conflict on SP architectures, generating stalls and performance loss compared to a *conflict-free* architecture such as MP. The opposite is true when the area constraint is less severe (bigger than 30%). In such case, SP can provide better capacity scaling by hosting nearly twice as much the cache lines in the same silicon area, and consequently, be more tolerant to application variability. This is clearly visible for the *Library* class application SRAD, which provides the best energy results for larger cache sizes. Table 3 summarizes these findings, showing the dominant architecture configuration in term of $energy \times area$ for each area budget constraint for the three different application class. Considering an application class with high-locality, such as the *Short-Jump* class of benchmarks, small private caches are able to achieve the best trade-off in area and energy due the fact that a large instruction cache capacity is not a requirement for this class of applications. Thus, a really simple and small private program cache is sufficient to tolerate that kind of program flow requirement, without introducing any bottlenecks. As soon as the scenario becomes more complex - this is the case of *Long-Jump* class of benchmarks - when a highly constrained area budget is considered (10%), the 1K-MP provides the most efficient solution. Compared to the private cache configuration, it enables better hit-rate and it avoids replications, while compared to the SP solution, it is not affected by conflicts and requires less power. Obviously, if the area constraints are looser, big private caches are effective also for *Long-Jump* class of benchmarks. Finally, when *Library* based applications are considered, shared caches are always the best solution for all the configuration evaluated. Considering that for this class of applications the instruction cache can be a real bottleneck, and that the capacity of the caches has major impact on the performance, the SP is able to provide the best $energy \times area$ trade-off when the I$ area portion inside the cluster is bigger than 20%.

Application Class / **Tolerance**	**I$ Area Budget (% of total cluster area)** 10%	20%	30%	40%
Short-Jump	**2K-PR**	**2K-PR**	**2K-PR**	**2K-PR**
Long-Jump	**1K-MP**	**8K-PR**	**8K-PR**	**8K-PR**
Library	**1K-MP**	**2K-MP**	**8K-SP**	**16K-SP**

TABLE 3: Dominant architecture configurations given an increasing instruction cache area budget.

3. Respectively, 4kB private I$ per core configuration (32K-PR), and 8x 2048B single port shared I$ banks configuration (16K-SP)

5 Conclusion

In this work, we explored instruction cache architectures for energy efficient and cost effective tightly coupled clusters of processors for end-node IoT devices. Although a large private instruction cache based on latches provides good performance and energy efficiency, it is not an effective solution for low-cost, area-constrained designs, due to the large area overhead of latch-based memories with respect to SRAMs. We thus studied two different cluster architectures based on shared instruction caches: one featuring a crossbar between the processors and the memory banks, and one exploiting memory banks with multiple ports. We conducted an exploration running several signal processing applications featuring diverse instruction memory access patterns on the same cluster configured with different instruction cache architectures. The results of this exploration show that the shared cache configurations can execute in a more energy efficient way for a much wider class of applications with much less silicon area when compared to private caches. The multi-port cache architecture improves the performance with respect to the private cache by up to 40% in throughput, 20% in energy efficiency, and 30% in energy $\times$ area efficiency for sizes of few kB (typical of ultra-low-power architectures). While the multi-port solution is optimal for small sizes, the one based on shared single-port banks is more suitable for large cache configurations, providing better scalability with the size and number of cores, especially in area, providing 20% better energy $\times$ area efficiency than the multi-port, and up to 30% better energy efficiency than private caches.

Acknowledgments

This work was supported by the EU H2020 Project OPRECOMP (g.a. 732631) and EU ERC MULTITHERMAN (g.a. 291125).

References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. [Online]. Available: http://doi.acm.org/10.1145/2342509.2342513

[2] F. Conti, R. Schilling, P. D. Schiavone, A. Pullini, D. Rossi, F. K. Grkaynak, M. Muehlberghuber, M. Gautschi, I. Loi, G. Haugou, S. Mangard, and L. Benini, "An IoT Endpoint System-on-Chip for Secure and Energy-Efficient Near-Sensor Analytics," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. PP, no. 99, pp. 1–14, 2017.

[3] M. Konijnenburg, S. Stanzione, L. Yan, D. W. Jee, J. Pettine, R. van Wegberg, H. Kim, C. van Liempd, R. Fish, J. Schluessler, H. de Groot, C. van Hoof, R. F. Yazicioglu, and N. van Helleputte, "28.4 A battery-powered efficient multi-sensor acquisition system with simultaneous ECG, BIO-Z, GSR, and PPG," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, pp. 480–481.

[4] M. Rusci, D. Rossi, M. Lecca, M. Gottardi, E. Farella, and L. Benini, "An Event-Driven Ultra-Low-Power Smart Visual Sensor," *IEEE Sensors Journal*, vol. 16, no. 13, pp. 5344–5353, July 2016.

[5] G. Ottoy, B. Thoen, and L. D. Strycker, "A low-power MEMS microphone array for wireless acoustic sensors," in *2016 IEEE Sensors Applications Symposium (SAS)*, April 2016, pp. 1–6.

[6] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, Feb 2010.

[7] R. Braojos, A. Dogan, I. Beretta, G. Ansaloni, and D. Atienza, "Hardware/software approach for code synchronization in low-power multi-core sensor nodes," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2014, pp. 1–6.

[8] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Grkaynak, A. Bartolini, P. Flatresse, and L. Benini, "A 60 GOPS/W, -1.8 v to 0.9 v body bias ULP cluster in 28 nm UTBB FD-SOI technology," *Solid-State Electronics*, vol. 117, pp. 170–184, 2016. [Online]. Available: www.scopus.com

[9] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Grkaynak, A. Teman, J. Constantin, A. Burg, I. Miro-Panades, E. Beign, F. Clermidy, P. Flatresse, and L. Benini, "Energy-Efficient Near-Threshold Parallel Computing: The PULPv2 Cluster," *IEEE Micro*, vol. 37, no. 5, pp. 20–31, September 2017.

[10] M. Alioto, "Ultra-Low Power VLSI Circuit Design Demystified and Explained: A Tutorial," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 1, pp. 3–29, Jan 2012.

[11] D. Bol, J. D. Vos, C. Hocquet, F. Botman, F. Durvaux, S. Boyd, D. Flandre, and J. D. Legat, "A 25MHz 7uW/MHz ultra-low-voltage microcontroller SoC in 65nm LP/GP CMOS for low-carbon wireless sensor nodes," in *2012 IEEE International Solid-State Circuits Conference*, Feb 2012, pp. 490–492.

[12] N. Ickes, Y. Sinangil, F. Pappalardo, E. Guidetti, and A. P. Chandrakasan, "A 10 pJ/cycle ultra-low-voltage 32-bit microprocessor system-on-chip," in *ESSCIRC (ESSCIRC), 2011 Proceedings of the*, Sept 2011, pp. 159–162.

[13] A. Teman, L. Pergament, O. Cohen, and A. Fish, "A 250 mV 8 kb 40 nm Ultra-Low Power 9T Supply Feedback SRAM (SF-SRAM)," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 11, pp. 2713–2726, Nov 2011.

[14] B. H. Calhoun and A. P. Chandrakasan, "A 256-kb 65-nm Subthreshold SRAM Design for Ultra-Low-Voltage Operation," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 3, pp. 680–688, March 2007.

[15] N. Verma and A. P. Chandrakasan, "A 256 kb 65 nm 8T Subthreshold SRAM Employing Sense-Amplifier Redundancy," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 141–149, Jan 2008.

[16] P. A. Meinerzhagen, S. M. Y. Sherazi, A. P. Burg, and J. N. Rodrigues, "Benchmarking of standard-cell based memories in the sub-VT domain in 65-nm CMOS technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 2, pp. 173–182, 2011.

[17] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Power, Area, and Performance Optimization of Standard Cell Memory Arrays Through Controlled Placement," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 4, pp. 59:1–59:25, May 2016. [Online]. Available: http://doi.acm.org/10.1145/2890498

[18] D. Bortolotti, F. Paterna, C. Pinto, A. Marongiu, M. Ruggiero, and L. Benini, "Exploring instruction caching strategies for tightly-coupled shared-memory clusters," in *System on Chip (SoC), 2011 International Symposium on*, Oct 2011, pp. 34–41.

[19] A. Marongiu and L. Benini, "An OpenMP Compiler for Efficient Use of Distributed Scratchpad Memory in MPSoCs," *IEEE Trans. Comput.*, vol. 61, no. 2, pp. 222–236, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1109/TC.2010.199

[20] I. Loi, D. Rossi, G. Haugou, M. Gautschi, and L. Benini, "Exploring Multi-banked shared-L1 Program Cache on Ultra-low Power, Tightly Coupled Processor Clusters," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ser. CF '15. New York, NY, USA: ACM, 2015, pp. 64:1–64:8. [Online]. Available: http://doi.acm.org/10.1145/2742854.2747288

[21] J. Myers, A. Savanth, R. Gaddh, D. Howard, P. Prabhat, and D. Flynn, "A Subthreshold ARM Cortex-M0+ Subsystem in 65 nm CMOS for WSN Applications with 14 Power Domains, 10T SRAM, and Integrated Voltage Regulator," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 31–44, Jan 2016.

[22] J. S. Wang, Y. C. Chien, F. Z. Liu, and P. Y. Chang, "A Calibration-Free PVTD-Variation-Tolerant Sensing Scheme for Footless-8T SRAM Designs," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 1, no. 3, pp. 159–167, July 2015.

[23] J. Kin, M. Gupta, and W. H. Mangione-Smith, "Filtering memory references to increase energy efficiency," *IEEE Transactions on Computers*, vol. 49, no. 1, pp. 1–15, Jan 2000.

[24] L. H. Lee, B. Moyer, and J. Arends, "Instruction fetch energy reduction using loop caches for embedded applications with small tight loops," in *Low Power Electronics and Design, 1999. Proceedings. 1999 International Symposium on*, Aug 1999, pp. 267–269.

[25] N. E. Bellas, I. N. Hajj, and C. D. Polychronopoulos, "Using dynamic cache management techniques to reduce energy in general purpose processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 6, pp. 693–708, Dec 2000.

[26] J. Eyre and J. Bier, "DSP processors hit the mainstream," *Computer*, vol. 31, no. 8, pp. 51–59, Aug 1998.

[27] O. D. Olorode and M. Nourani, "Improving Cache Power and Performance Using Deterministic Naps and Early Miss Detection," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 1, no. 3, pp. 150–158, July 2015.

[28] F. Botman, J. de Vos, S. Bernard, F. Stas, J. D. Legat, and D. Bol, "Bellevue: A 50MHz variable-width SIMD 32bit microcontroller at 0.37V for processing-intensive wireless sensor nodes," in *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, June 2014, pp. 1207–1210.

[29] H. Wong, M. M. Papadopoulou, M. Sadooghi-Alvandi, and A. Moshovos, "Demystifying GPU microarchitecture through microbenchmarking," in *Performance Analysis of Systems Software (ISPASS), 2010 IEEE International Symposium on*, March 2010, pp. 235–246.

[30] L. Benini, E. Flamand, D. Fuin, and D. Melpignano, "P2012: Building an ecosystem for a scalable, modular and high-efficiency embedded computing accelerator," in *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2012, pp. 983–987.

[31] B. D. de Dinechin, "Kalray MPPA #xAE;: Massively parallel processor array: Revisiting DSP acceleration with the Kalray MPPA Manycore processor," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, Aug 2015, pp. 1–27.

[32] A. Rahimi, I. Loi, M. R. Kakoee, and L. Benini, "A fully-synthesizable single-cycle interconnection network for Shared-L1 processor clusters," in *2011 Design, Automation Test in Europe*, March 2011, pp. 1–6.

[33] D. Fick, R. G. Dreslinski, B. Giridhar, G. Kim, S. Seo, M. Fojtik, S. Satpathy, Y. Lee, D. Kim, N. Liu, M. Wieckowski, G. Chen, T. Mudge, D. Sylvester, and D. Blaauw, "Centip3De: A 3930DMIP-S/W configurable near-threshold 3D stacked system with 64 ARM Cortex-M3 cores," in *2012 IEEE International Solid-State Circuits Conference*, Feb 2012, pp. 190–192.

[34] M. Gautschi, A. Traber, A. Pullini, L. Benini, M. Scandale, A. D. Federico, M. Beretta, and G. Agosta, "Tailoring instruction-set extensions for an ultra-low power tightly-coupled cluster of OpenRISC cores," in *2015 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, Oct 2015, pp. 25–30.

[35] M. Gautschi, D. Rossi, and L. Benini, "Customizing an Open Source Processor to Fit in an Ultra-low Power Cluster with a Shared L1 Memory," in *Proceedings of the 24th Edition of the Great Lakes Symposium on VLSI*, ser. GLSVLSI '14. New York, NY, USA: ACM, 2014, pp. 87–88. [Online]. Available: http://doi.acm.org/10.1145/2591513.2591569

[36] D. Rossi, I. Loi, G. Haugou, and L. Benini, "Ultra-low-latency Lightweight DMA for Tightly Coupled Multi-core Clusters," in *Proceedings of the 11th ACM Conference on Computing Frontiers*, ser. CF '14. New York, NY, USA: ACM, 2014, pp. 15:1–15:10. [Online]. Available: http://doi.acm.org/10.1145/2597917.2597922

[37] STMicroelectronics, "Discovery kit with STM32F407VG MCU," in *http://www.st.com/en/evaluation-tools/stm32f4discovery.html*, 2016.

[38] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. IEEE, 2009, pp. 44–54.

[39] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*. Springer, 2006, pp. 430–443.

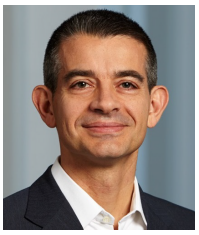
[40] A. Marongiu, A. Capotondi, G. Tagliavini, and L. Benini, "Simplifying Many-Core-Based Heterogeneous SoC Programming With Offload Directives," *IEEE Transactions on Industrial Informatics*, vol. 11, no. 4, pp. 957–967, Aug 2015.

Igor Loi received the PhD from the University of Bologna, Italy, in 2010. He has been a post doc researcher in the Department of Electrical, Electronic and Information Engineering Guglielmo Marconi at the University of Bologna since 2006, where he currently holds an assistant professor (RTD-A) position. His research activities are currently focused on ultra-low power multi-core systems, memory systems evolution, and ultra low-latency interconnects. In this field he has published more than 40 paper in international peer-reviewed conferences and journals. He has collaborated with several international research institutes (Stanford University and IMEC) and companies (e.g. STMicroelectronics, Toshiba and Samsung).

Alessandro Capotondi received the PhD degree in Electronics Engineering, Telecommunications, and Information Technology from the University of Bologna, Italy, in 2016. He is IEEE member and he currently is a post doc at The Center for Industrial Research on Information and Communication Technologies (CIRI ICT) of the University of Bologna. His research interests concern parallel programming models and code optimization for heterogeneous many-cores architectures.

Davide Rossi received the PhD from the University of Bologna, Italy, in 2012. He has been a post doc researcher in the Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi" at the University of Bologna since 2015, where he currently holds an assistant professor position. His research interests focus on energy efficient digital architectures in the domain of heterogeneous and reconfigurable multi and many-core systems on a chip. In this fields he has published more than 60 paper in international peer-reviewed conferences and journals.

Andrea Marongiu received the MSc degree in electronic engineering from the University of Cagliari, Italy, in 2006 and the PhD degree in electronic engineering from the University of Bologna, Italy, in 2010. Since 2013 he has been a Research Fellow at ETH Zurich. He currently is an Associate Professor at the University of Bologna. His research interests concern parallel programming model and architecture design in the single-chip multiprocessors domain, with special emphasis on compilation for heterogeneous architectures, efficient usage of on-chip memory hierarchies and SoC virtualization. He has published more than 80 papers in peer reviewed international journals and conferences. He is a member of the IEEE.



Luca Benini holds the chair of digital Circuits and systems at ETHZ and is Full Professor at the Universita di Bologna. His research interests are in energy-efficient system design for embedded and high-performance computing. He is also active in the area of energy-efficient smart sensors and ultra-low power VLSI design. He has published more than 800 papers, five books and several book chapters. He is a Fellow of ACM and a member of the Academia Europaea. He is the recipient of the 2016 IEEE CAS Mac Van Valkenburg award.