Exploiting Private Local Memories to Reduce the Opportunity Cost of Accelerator Integration

Emilio G. Cota Paolo Mantovani Luca P. Carloni
Department of Computer Science, Columbia University
New York, NY, USA
{cota,paolo,luca}@cs.columbia.edu

ABSTRACT

We present ROCA, a technique to reduce the opportunity cost of integrating non-programmable, high-throughput accelerators in general-purpose architectures. ROCA exploits the insight that non-programmable accelerators are mostly made of private local memories (PLMs), which are key to the accelerators' performance and energy efficiency. ROCA transparently exposes PLMs of otherwise unused accelerators to the cache substrate, thereby allowing the system to extract utility from accelerators even when they cannot directly speed up the system's workload. ROCA adds low complexity to existing accelerator designs, requires minimal modifications to the cache substrate, and incurs a modest area overhead that is almost entirely due to additional tag storage.

We quantify the utility of ROCA by comparing the returns of investing area in either regular last-level cache banks or ROCA-enabled accelerators. Through simulation of non-accelerated multiprogrammed workloads on a 16-core system, we extend a 2MB S-NUCA baseline system to show that a 6MB ROCA-enabled last-level cache built upon typical accelerators (i.e. whose area is 66% memory) can, on average, realize 70% of the performance and 68% of the energy efficiency benefits of a same-area 8MB S-NUCA configuration, in addition to the potential orders-of-magnitude efficiency and performance improvements that the added accelerators provide to workloads suitable for acceleration.

CCS Concepts

•**Hardware** → **Hardware accelerators;** *Economics of chip design and manufacturing;* On-chip resource management; •**Computer systems organization** → *Heterogeneous (hybrid) systems;*

Keywords

Accelerator memory, non-uniform cache architectures, private local memory, opportunity cost

ICS '16, June 01-03, 2016, Istanbul, Turkey

DOI: http://dx.doi.org/10.1145/2925426.2926258

1. INTRODUCTION

Owing to their specific purpose, non-programmable, high-throughput accelerators allow designers to tailor the microarchitecture to a specific workload, thereby delivering near-optimal performance and energy efficiency [4]. Unfortunately, these come at the expense of generality; a given accelerator can only speed up a specific instance of an algorithm, with little or no flexibility to incorporate potential improvements to it. As a result, the opportunity cost of integrating accelerators in general-purpose architectures is usually prohibitive, since few accelerators are likely to apply to a workload that is unknown at design time. Thus, investing area in accelerators for these architectures frequently implies forgoing more generally applicable and therefore productive alternatives, such as larger caches and/or cores, greater core counts, or not making the investment at all.

Prior work improves average on-chip memory utilization as a way to reduce accelerator opportunity cost by building on two observations: accelerators are mostly made of private local memories (PLMs) and have low average utilization. Thus, the resulting solutions reduce the overall amount of integrated PLMs by providing storage that accelerators can allocate memory from. This storage is implemented either as an accelerator-only memory pool [24] or by allocating blocks for accelerators from the last-level cache (LLC) [8, 13]. Although these techniques reduce the total on-chip SRAM investment, they are only applicable to low-bandwidth PLMs, which are scarce in high-throughput accelerators as we discuss in Section 2.

In this work we leverage an additional observation to further reduce the opportunity cost of accelerator integration. We observe that accelerator PLMs disseminated across the chip provide a de facto non-uniform cache (NUCA), which is the optimal organization for large, multi-megabyte caches [17]. Thus, instead of providing storage external to accelerators, our goal is to *expose* accelerator PLMs to the LLC [10], thereby extracting utility from *all* PLMs and not just from low-bandwidth ones. Moreover, this simplifies the design effort over prior work, since the designer does not need to artificially break PLMs into high and low bandwidth groups.

Three difficulties make exploiting accelerator PLMs by the LLC challenging. First, exposing PLMs to the cache substrate requires logic to abstract the PLMs' diversity in size, bitwidth and number of ports. Second, the cache substrate has to dynamically handle the intermittent availability of some of its capacity. Last, the delay due to the eviction of dirty blocks from PLMs reclaimed by an accelerator can potentially degrade the accelerator's performance.

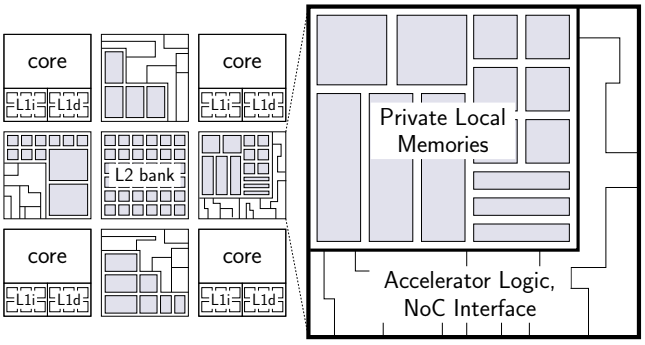


Figure 1: 4-core chip with 2-level cache and four accelerators. Most accelerator area is devoted to private local memories (PLMs) that, when otherwise inactive, are used by ROCA to extend the last-level cache. The memory blocks of the resulting LLC are shaded in gray.

We propose ROCA, a technique to exploit accelerator PLMs to *reduce the opportunity cost of integrating accelerators*. ROCA transparently exposes accelerator PLMs to the cache substrate, thereby extending LLC capacity while accelerators are not in use. Figure 1 illustrates the goal of our technique. A four-core chip shares a Network-on-Chip (NoC) interconnect with four accelerators, whose PLMs' diversity (in number of ports, size, and bitwidth) is represented by rectangles of varying dimensions. With ROCA, the LLC is formed by combining regular NUCA banks with PLMs from otherwise unused accelerators. In the figure, the resulting ROCA-enabled LLC is shown shaded in gray.

ROCA uses a combination of old and new techniques to achieve low complexity, high performance and modest area overhead. First, it adds a minimum amount of logic around PLMs to expose them through an additional port to the cache substrate. Second, it relies on the decoupling between cache tags and the corresponding data [6], keeping tag storage external to accelerators. Third, it leverages selective cache ways [1] to dynamically adapt to the intermittent availability of PLMs, flushing dirty blocks to DRAM as PLMs are reclaimed by accelerators.

In this paper we make the following contributions:

- We present ROCA, a technique that leverages accelerator PLMs to dynamically expand the last-level cache (Section 3), and show its modest area overhead, which is almost entirely due to additional tag storage (Section 4).
- We perform full-system simulation of multiprogrammed workloads running on a multi-core architecture whose LLC is mostly implemented from ROCA accelerators, evaluating ROCA's performance and energy efficiency overhead, which we show to be low, and studying ROCA's sensitivity to different accelerator reclamation frequencies for several spatial configurations. (Section 5).

In summary, ROCA shows that extending the last-level cache by exploiting accelerators PLMs when otherwise unused is an effective way of reducing accelerators' opportunity cost, since it decouples the utility of accelerators from the workload under consideration.

2. BACKGROUND

Recent work on non-programmable accelerators hinges on two main observations:

Private memories are key to accelerator performance and energy efficiency. Specialized hardware can potentially exploit all parallelism inherent in a given computational kernel. However, a necessary condition to realize this parallelism is the ability to fetch data, possibly in highly irregular patterns, at the same rate as they are processed by a specialized datapath. Attaching accelerators to existing CPU caches in order to save area and therefore energy might seem worth pursuing, but unfortunately cache structures cannot fulfill most accelerators' requirements. First, high-throughput accelerators require memories with a far greater number of ports than what caches can efficiently implement [5, 14]. Second, a fixed cache block size cannot serve well the needs of reads and writes of various widths that occur even within just a single accelerator. And third, the high associativity needed by caches to provide fast lookups imposes significant energy and area overheads, which goes against the efficiency goal of acceleration. Accelerators are thus best served by private local memories, which are memory blocks only exposed to accelerator hardware and tailored in their number of ports, banks and widths to precisely match the needs of each computational block within an accelerator [9].

Accelerators are mostly memory. Given the importance of low-latency, high-bandwidth memory accesses for accelerators' performance, private memory blocks take a substantial portion of accelerators' area. For instance, a survey of eleven publicly available accelerators reveals that "an average of 69% of accelerator area is consumed by memory" [24], and recent high-performance accelerators show significant private memory investments as well [5, 19].

A corollary to the second observation is that average accelerator memory utilization is low on many-accelerator systems, since not all accelerators are likely to run at the same time. Thus, prior work has reduced accelerator opportunity cost by moving private memories out of the accelerator: proposals range from an on-chip memory pool that accelerators can allocate from [24], to providing a substrate that can store cache blocks as well as accelerator data [13, 8].

ROCA has two advantages over these proposals. First, it applies to *all* accelerator memories, regardless of their bandwidth or access pattern (fixed or data-dependent). Second, it requires modifications on the accelerators that are simpler to implement: designers do not have to worry about predicting access patterns or what memory blocks are (or could be engineered to be) of higher/lower bandwidth. Instead, with ROCA designers just focus on optimizing their design without any additional requirements, and once the design is done, a small amount of logic around memory blocks is all that is needed to make an accelerator ROCA-compliant.

ROCA also adds logic to the cache substrate to make it view accelerator PLMs as de facto NUCA storage. The complexity of the logic needed, apart from additional tag storage, is low due to two ideas. First, we exploit the observation by Chishti et al. [6] that, since tag and data lookups happen sequentially in large caches, their placement can be decoupled. Second, we leverage selective cache ways by Albonesi [1] as the mechanism to accommodate accelerator PLMs of diverse sizes, and to rapidly and efficiently adapt to their intermittent activity and therefore availability.

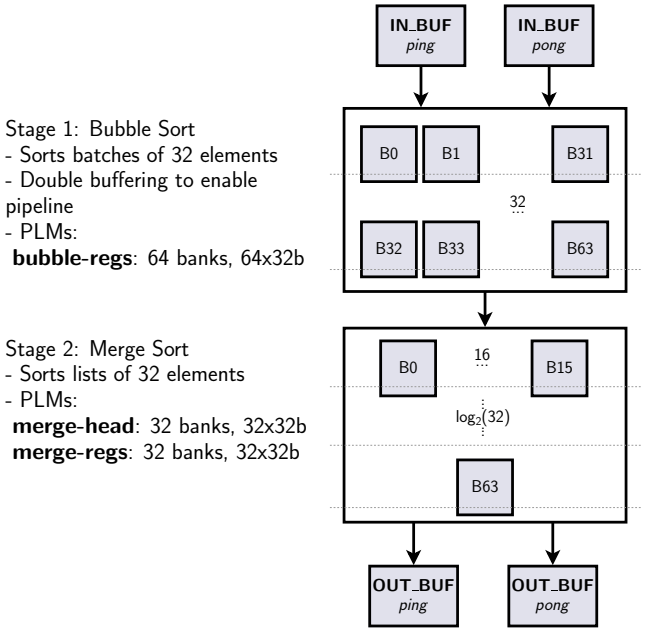


Figure 2: Structure of the Sort accelerator. Ping-pong buffering enables simultaneous processing of vectors. The dashed lines denote different pipeline stages. Banks of the five PLMs are shaded in gray.

2.1 Accelerator Example: Sort

We now illustrate the typical structure of a high-throughput accelerator through a particular example, which we will also use in the next section to describe the modifications that Roca imposes on accelerators.

Our *Sort* accelerator is designed to meet the PERFECT benchmark suite's requirements [2] by sorting batches of floating point vectors of up to 1024 elements each. The accelerator, whose structure is depicted in Figure 2, has five PLMs and computes in two stages. The first stage sorts groups of 32 elements through a parallel implementation of bubble sort: the innermost loop of the algorithm can complete in one clock cycle due to aggressive SRAM banking to feed a specialized datapath with 32 comparators. Moreover, PLMs in this stage are doubled to support the processing of groups of 32 elements in a pipeline, thus allowing reads and swap-induced writes to occur in the same clock cycle. The second stage sorts the resulting 32-element lists (with a maximum of 32 lists for a total of 1024 elements) using merge sort. PLMs are again heavily banked to maximize performance and enable pipelining.

The accelerator was optimized for performance and energy efficiency: compared to a software implementation running on an Intel Haswell processor clocked at 2.3GHz, a 1GHz silicon implementation[1] of the accelerator is up to 3.5X faster, while requiring only 0.5% of the energy; the speedup of the accelerator over a software implementation for an OpenRISC CPU, with both accelerator and CPU synthesized in an FPGA at 100MHz, is of up to 300X while consuming 3% of the energy. These significant gains are enabled by the use of high-bandwidth PLMs. Their effect is illustrated in Figure 3, which plots the PLM bandwidth over time for the two accelerator stages when sorting 4 vectors of 1024 elements. The plots show how aggressive SRAM banking enables large amounts of concurrent PLM accesses (up to 32 accesses per PLM per cycle) which are key to achieving high performance. Additional banking can also help pipelining. For instance, the *bubble-regs* PLM has twice as many banks as it would otherwise need, which provides the necessary intermediate buffering for processing different vectors in a pipeline.

[1]The accelerator logic is simple and therefore can be clocked at high frequencies. However, the achievable clock frequency of our silicon implementation is limited by the memory generators available to us.

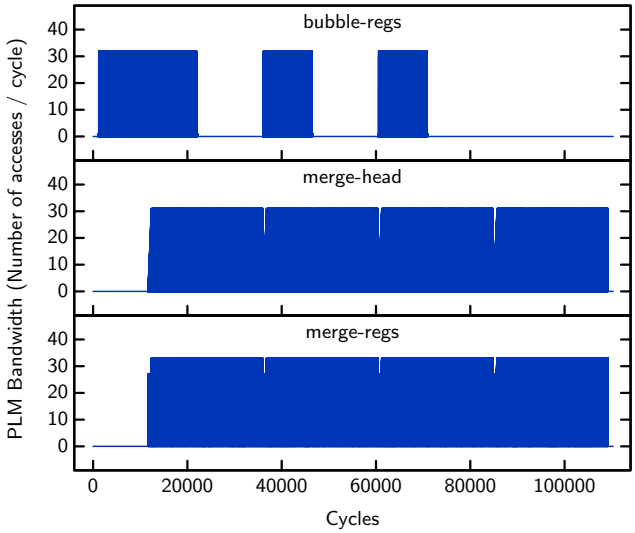


Figure 3: PLM Bandwidth of the Sort accelerator when sorting 4 vectors of 1024 elements. High PLM bandwidth is enabled by heavy SRAM banking.

PLM	Size (KB)	Banks	Peak Bandwidth
merge-head	4	32	32
merge-regs	4	32	32
IN_BUF	8	1	1
OUT_BUF	8	1	1
bubble-regs	16	64	32

Table 1: Characteristics of the PLMs in the Sort accelerator. Bandwidth is measured in number of accesses per cycle.

Applying prior approaches [8, 13, 24] to this accelerator could only be done for a subset of the PLMs and would result in a substantial impact on performance or increase in complexity. Moving memories out of the accelerator would only be possible to do for input and output buffers, since they are the only ones whose access patterns are fully predictable, i.e. do not depend on input values. As shown in Table 1 these memories (*IN_BUF* and *OUT_BUF*) amount to only 40% of the PLM total, i.e. 16 out of 40 KB. Moreover, implementing this move would be complex without significantly impacting area or performance: buffers would have to be added to hide the pipeline-induced latency of reading/writing data to a remote bank via the interconnect.

In the next section we show how Roca adds simple logic to accelerators to transparently expose all of their PLMs to the cache substrate.

ROCA Read Access Example

1. core0's L1 misses on a read from 0xf00, mapped to the L2's bank1

2. L2 bank1's tag array tracks block 0xf00 at acc2; sends request to acc2

3. acc2 returns the block to bank1

4. bank1 sends the block to core0

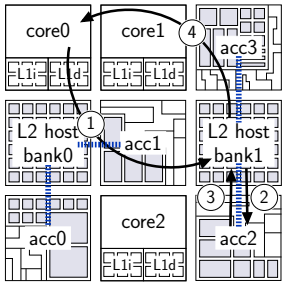


Figure 4: High-level operation example. Three cores share a LLC with blocks interleaved across two logical banks, each composed of a ROCA host bank and associated accelerators (marked with a dashed line). Requests to a logical bank are always routed to the host bank (1-2), as well as responses from ROCA accelerators with the requested data (3-4).

3. ROCA: EXPOSING ACCELERATOR MEMORY TO THE NUCA SUBSTRATE

Our implementation of ROCA combines the following hardware elements, which we describe in detail in this section:

- An enlarged tag array in the last-level cache to track blocks stored in accelerator PLMs.
- Logic and gating hardware in the tag array to enable/disable ways as accelerators reclaim their memory, as in Selective Cache Ways [1].
- Registers (one per logical bank) that accelerators or privileged software can access to enable/disable the participation of accelerators in ROCA based on the accelerators' activity rate.
- Logic on accelerators to coalesce SRAMs of different widths, sizes, and ports to expose them as a single PLM to the cache substrate via the interconnect.
- Logic on the tag array to support the flushing of dirty cache blocks in accelerator PLMs.

3.1 High-Level Operation

We explain the high-level operation of ROCA in Figure 4. For the shown 3-core chip the physical address space is split in two *logical banks*, which are regular LLC banks that are extended with ROCA-enabled accelerators. In the example, core 0 misses in its L1 on a read access to the block at address 0xf00, which is assigned to be cached at the L2's *host bank* 1. Core 0 then sends its block request to host bank 1. The bank checks its enlarged tag array (which also tracks blocks in accelerators 2 and 3) and forwards the request to accelerator 2. The accelerator sends back to host bank 1 the contents of the block, and the host bank then forwards it to core 0.

In this operation example, accelerator 2 could have directly sent the block to core 0, therefore lowering access latency. However, this modification would be ill-advised. For example, consider the scenario in which immediately after the access request for 0xf00, host bank 1 received a command to disable accelerator 2 from caching, i.e. the accelerator was reclaimed for acceleration. If there were no communication back to the host bank, the bank would not know when it would be safe to hand over the accelerator PLMs; this could result in data corruption (if the handover were too early and thus accelerator activity overwrote the not-yet-read block) or in unnecessary waiting to guarantee that the access completed.

Write accesses also require an acknowledgment message back from accelerators to their host bank. Consider two consecutive writes from different cores to the same block whose data is cached by an accelerator. Without an acknowledgment message, correctness would have to rely on an interconnect with point-to-point ordering, which is a constraint that we do not wish to impose since it limits ROCA's applicability.

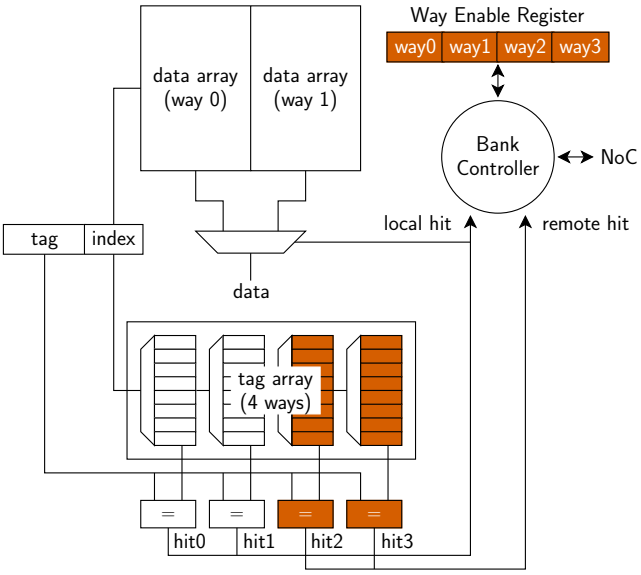


Figure 5: A 4-way ROCA host bank. Two ways are local to the bank; the other two are remote, i.e. their data arrays are accessed via the interconnect. Shaded in orange are the hardware structures necessary to convert a regular bank into a ROCA host bank.

3.2 Host Bank Organization

ROCA uses two ideas to form host banks whose capacity can be expanded to exploit accelerator PLMs. First, it leverages selective cache ways to integrate accelerators with PLMs of varying sizes, which also serves as a simple mechanism to dynamically adapt to the intermittent availability for caching of accelerators. Second, it expands the tag array in the host bank to track accelerator blocks, thereby requiring minimal changes to accelerator designs.

Figure 5 shows the organization of a 4-way ROCA host bank. The in-bank data array only contains ways 0 and 1. The tag array, however, tracks these two *local* ways as well as two *remote* ways, i.e. ways whose data blocks are stored in accelerators. Storage for tags and state of remote blocks is where the bulk of ROCA's host bank area overhead lies. In addition, logic in the bank controller is required to handle on-chip communication related to cache block transfers to/from remote ROCA banks, as well as a Way Enable Register to configure which ways to enable. Software can access this register to enable/disable ways as the corresponding accelerators change their availability for caching.

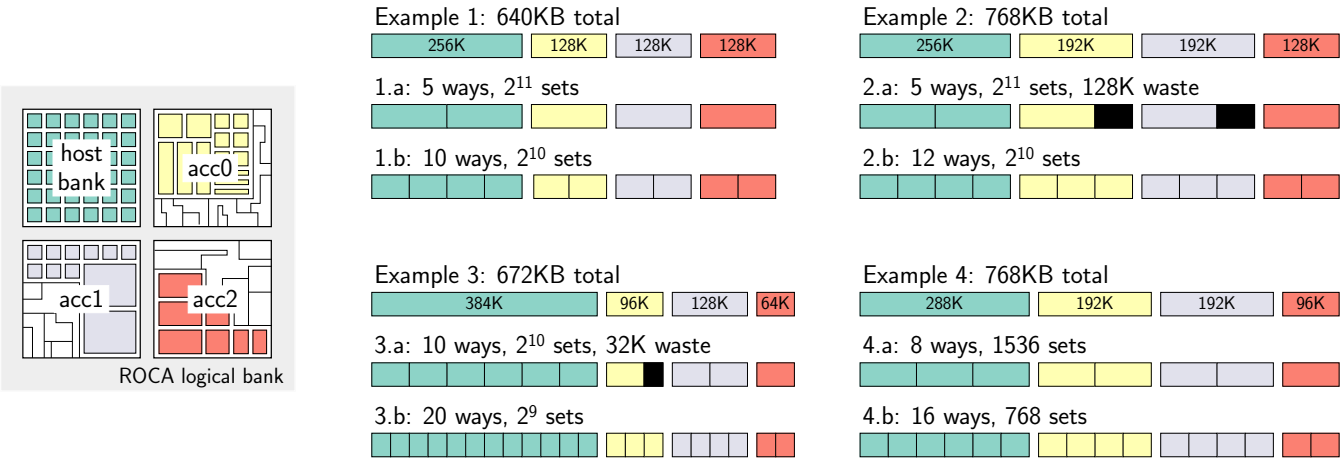


Figure 6: Way allocation examples for a ROCA logical bank with three accelerators, assuming a block size of 64 bytes.

3.3 Way Allocation in Logical Banks

Way allocation is performed at design time to coalesce accelerators with diverse PLM sizes into the same logical bank. Figure 6 illustrates way allocation through several examples of a ROCA logical bank composed of a host bank and three associated accelerators. Example 1 is the simplest; accelerators are all of the same size, which is a power of two that evenly divides the size of the memory in the host bank. Thus, in Example 1.a, the chosen number of sets is such that each accelerator can host one element per set. In other words, one way is assigned to each accelerator. The number of sets can be subsequently halved to double the number of ways, as 1.b shows.

Examples 2 and 3 are more diverse in their accelerator PLM sizes and therefore are more realistic. Both examples show that higher associativity (i.e. smaller numbers of sets) helps minimize waste due to uneven accelerator memory provisioning. In practice, an associativity typical for last-level caches (10 to 20 ways) is enough to result in negligible or no memory waste, as illustrated in examples 2.b and 3.b.

The number of sets in all three examples discussed so far is a power of two, which makes the tag/index acquisition logic a simple bit selection from the block address. Examples 4.a and 4.b show way allocations that result in non-power-of-two numbers of sets. This is in principle a plausible option in the unusual scenarios where wasting some memory is unacceptable and the necessary associativity to achieve zero waste with power-of-two numbers of sets is prohibitive. However, designers need to consider the drawbacks of this choice before committing to it. Feasible numbers of sets are limited to those whose arithmetic modulo have a fast hardware implementation, i.e. numbers of the form $2^c * (2^n - 1)$ or $2^c * (2^n + 1)$, where c and n are non-negative integers [35]. More importantly, tags need to be enlarged to include all bits in the block offset, since the modulo operation cannot be constricted to just the least significant bits of the block address.

Designers are free to assign different numbers of accelerators to logical banks, allocating ways as they see fit. This can result in varying sizes and associativity across logical banks; given enough accelerators, however, this variability can be minimized by uniformly distributing accelerators based on their PLM sizes across the chip.

3.4 Impact on Cache Coherence

The intermittent availability of accelerators and the use of way allocation have profound effects on the cache coherence protocol. For instance, maintaining a directory cache embedded in an inclusive LLC comes with substantial overhead, since the blocks cached in an accelerator about to be reclaimed would have to either be recalled from the private caches or relocated.

A simpler alternative is to give up LLC inclusion. Implementing a standalone directory cache guarantees that recalls can be made infrequent (by having enough associativity in the directory cache), and dissociates coherence from the last-level cache; logical banks are then free to be of any size and associativity, and to silently flush dirty blocks from reclaimed accelerators. The cost of having a standalone directory cache is the storage overhead of its tag array, which comes for free in an inclusive LLC. This cost, however, is modest. For example, a standalone directory cache adds 2.5% of overhead to the last-level cache when the latter is 8 times larger than the sum of the private caches, a typical ratio for inclusive designs [22]. This relative overhead grows as the shared-to-private ratio shrinks, reaching 11% when the shared cache is of the same size as the sum of the private caches. [26]

3.5 Coalescing and Exposing PLMs

ROCA exposes accelerator PLMs to the cache substrate through an additional memory port managed in the accelerator by a ROCA *controller*. To describe these two additional components, which amount to a small amount of logic in the form of multiplexers and in some cases a small lookup table (LUT), we use the Sort accelerator described in Section 2.1.

Figure 7 depicts a memory-centric view of the Sort accelerator, in which each arrow represents a memory port. We first focus on the PLM manager, shaded in gray: its role is to export and coalesce SRAM banks into multi-ported memories [30]. For example, the 64 banks of *bubble-regs* are exported as a 64-port, 32b-wide PLM that has 32 ports connected to each of the parallel bubble sort and merge sort logic blocks. Given that the number of banks in the PLM is a power of two, the PLM manager is implemented as a trivial address translation unit that via bit selection determines for each access the correct bank and offset within it.

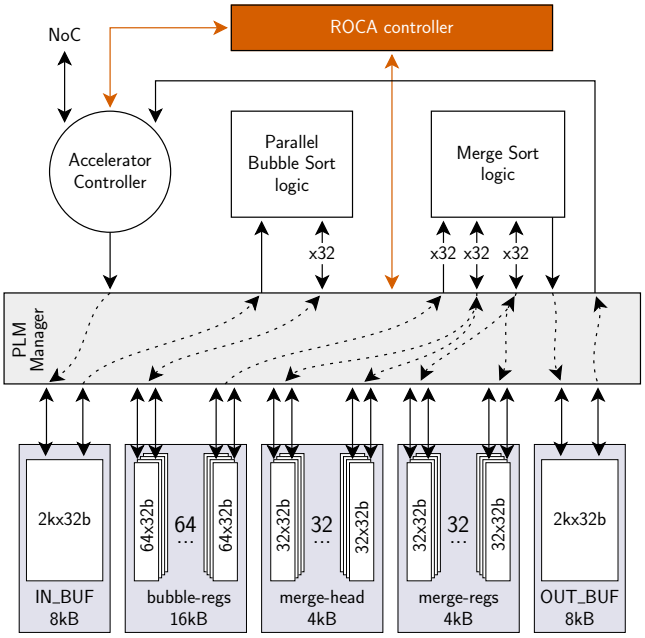


Figure 7: Memory ports in the Sort accelerator. The PLM manager aggregates SRAM banks to export them as multi-ported memories. An additional NoC-flit-wide port is exported to the ROCA controller, shaded in orange.

We now describe the ROCA port added to the PLM manager, shaded in orange in the figure. Its goal is to transfer a cache block with minimum latency, whose lower bound is imposed by the serialization in the NoC, i.e. one flit per cycle. Thus, the ROCA port aggregates all the SRAMs in the accelerator (totaling 40KB) through an interface of the same bit width as the NoC's flit length, which for this example we assume to be 128b. The attachment to the SRAMs is attained by multiplexing the SRAM control signals, since ROCA-enabled caching and acceleration do not overlap in time.

The addressing logic for the ROCA port is in general not as trivial as that for regular accelerator ports; the number of banks is rarely a power of two and the banks are not uniform in their size and bit width. The complexity of the addressing logic can be minimized by choosing an appropriate arrangement of the accelerator SRAMs; furthermore, an adequate arrangement can minimize SRAM waste and maximize bandwidth to match the target of NoC flit per cycle.

Such an arrangement for the Sort accelerator is shown in Figure 8: banks are accessed in pairs resulting in a bandwidth of 128b per cycle, with 64b/cycle coming from each bank thanks to exploiting the two ports of the dual-ported SRAMs. The SRAMs in each pair do not have to come from the same original PLM; for instance, *IN_BUF* and *OUT_BUF* as well as *merge-head* and *merge-regs* are paired together for caching purposes yet operate in separate PLMs during acceleration.

The ROCA controller converts block addresses within the total of 640 64-byte blocks (40KB) of memory into a physical offset within the appropriate pair of SRAM banks. To achieve this with minimum latency, integer division is to be avoided. Thus, we group banks of the same size as shown

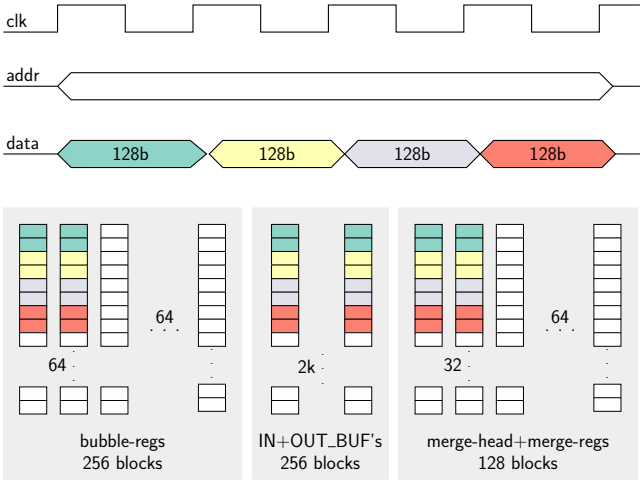


Figure 8: SRAM arrangement for the ROCA controller in the Sort accelerator. Adequate pairing of dual-ported banks brings bandwidth to one NoC flit (128b) per cycle.

in Figure 8 and then sort them by size in descending order, obtaining three groups of 256, 256 and 128 blocks. Block addresses with the 9$^{\text{th}}$ bit set go to the third group; all other addresses go to the first two groups, which are matched via bit selection since both groups are of the same power of two size. In all cases the offset within a group is also obtained via bit selection, which results in trivial logic.

Handling less uniform groups of SRAMs is also possible while avoiding integer division. A viable option is to have a small lookup table (LUT) to match the upper order bits (over that of the group sizes' greatest common factor) to the appropriate group; the offset within a group is then calculated via a shift, whose amount is also precomputed in the LUT. Another option, when the size of the banks is the same yet their number is not a power of two, is to follow the procedure presented by Seznec [35], which we mentioned when describing way allocation in logical banks (Section 3.3). A last option, when dealing with highly non-uniform cases, is to either discard some memory for caching, or pad the accelerator with additional memory (clock-gated when the accelerator is not in ROCA mode) in order to obtain more regular sizing across banks.

The same *"to pad or to waste"* decision is faced when coalescing SRAMs from PLMs of different bit widths. For example, if each bank is 8b-wide then 8 dual-ported banks need to be accessed to sustain a bandwidth of 128b per cycle. If the width is instead 7, then accessing 9 banks of which one is extended to 8 bits is an advisable option. Accelerators with diverse PLM bit widths are a common occurrence, despite what the Sort accelerator example might suggest.

So far we have assumed that the SRAM banks are dual-ported, i.e., in a single clock cycle two non-conflicting accesses to the same bank are allowed. Using single-ported SRAM banks is also possible with ROCA; the per-bank bandwidth therefore halves, requiring the number of banks (for ROCA or for acceleration) to double in order to meet the original bandwidth.

The NoC's flit length is also a variable that must be taken into consideration. In general, the larger the flit, the more aggregate bandwidth is required from the SRAMs, and therefore the larger their groups will be.

3.6 ROCA-to-Acceleration Transitions

When an accelerator is recalled from ROCA, two options are available in order to maintain the LLC in a consistent state. A first option is to relocate to other LLC banks the most frequently accessed blocks as well as the dirty ones. This option, however, is costly in hardware: timestamped block access counters such as bucketed LRU [34] are necessary to enable the ordering across blocks in the same accelerator, which holds blocks from all sets in the cache. An alternative, more practical option is to silently evict blocks, flushing to DRAM the ones that are dirty. In practice this is not much different from actively migrating frequently-accessed blocks: subsequent misses to said blocks will place them in other LLC banks, and, assuming heavy accelerator activity, these blocks will eventually be placed in LLC host banks, which are always available.

From the accelerator's point of view, the flushing of dirty blocks is simply a regular block read requested from the controller in ROCA's host bank. The host bank is not just the main serialization point for coherence as we discussed in Section 3.1; it is also a serialization point for accelerators, since an accelerator can only switch to acceleration once its ROCA host bank has completed the flushing of all the dirty blocks the accelerator was holding.

Choosing to just flush dirty blocks has an additional advantage over more complex options in that it minimizes the transition latency: its lower bound is the NoC's bandwidth, since it serializes the read requests that precede the corresponding flushes to DRAM. The importance of the transition latency is nonetheless relative to the frequency at which the accelerator switches between acceleration and caching. Thus, an accelerator that is constantly being required to accelerate is a poor target for ROCA, since the positive effect of temporarily increasing the aggregate cache size is dominated by the latency from flushing dirty blocks and subsequent cache misses. We thus let software decide when to enable/disable ROCA on accelerators, since software has complete information about the system. Our results in Section 5 quantify the accelerator invocation frequency below which ROCA should be enabled.

4. AREA OVERHEAD

ROCA's area overhead is relative to the scenario it is compared against. For instance, a system with a fixed area budget and initially no accelerators could benefit from including some ROCA-enabled accelerators, trading off part of the original cache for this purpose. The resulting cache size would depend on how much of the accelerators' area were devoted to memory; for instance, assuming accelerators were 69% memory (as discussed in Section 2), each unit of cache capacity would approximately require 44% more area when implemented in ROCA than as regular cache. In return for this area investment, however, the system would have gained the potential of drastically increasing the performance and efficiency of certain workloads by using the accelerators.

A more precise way to assess ROCA's area overhead is to consider the baseline system as one already equipped with accelerators. Expanding the existing cache with ROCA has then a modest cost, split between (1) tag storage for a standalone directory cache if it previously was embedded in the last-level cache, as discussed in Section 3.4, and (2) additional tag storage to track cache blocks in ROCA accelerators.

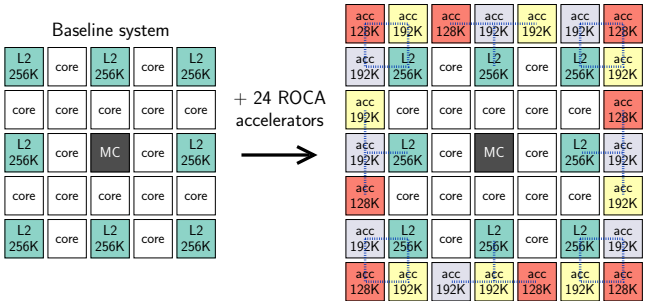


Figure 9: Topology of two of the simulated systems. The baseline system (left) is a 16-core 5x5 CMP with a 2MB S-NUCA LLC. We augment it with 24 ROCA-enabled accelerators to form a 7x7 CMP with a 6MB ROCA LLC (right). Dashed lines show the eight logical banks into which the address space is split. MC stands for memory controller.

Storage for the cache tag array is unlikely to exceed 10% of the storage needed for the data it tags. To be concrete, let us assume a 48-bit physical address space and 64-byte (2^6) cache blocks; then, in case the number of banks is not a power of two, tags need to include the entire block offset, resulting in a (48-6+2)/(64*8)=8.5% tag array area overhead relative to the data array, assuming two bits to keep valid and dirty states. In more common scenarios in which the number of sets is a power of two (and thus tags do not include the lower-order bits of the block offset) the relative overhead shrinks to, for instance, 6.6% or 5.4% assuming 2^{10} and 2^{16} sets, respectively.

ROCA requires additional logic whose area overhead is however negligible compared to that of tag storage. This logic enables Selective Cache Ways [1] in the host bank (discussed in Section 3.2), and multiplexes the control signals of accelerators' SRAM banks while appropriately translating addresses to coalesce them into a single PLM (Section 3.5).

5. ENERGY AND PERFORMANCE EVALUATION

5.1 Experimental Methodology

Modeled Systems. We conduct full-system simulation running Linux in an i386 simulator based on Qsim [16]. We initially model a 5x5 tiled CMP with 16 general-purpose cores as the baseline system, which we then augment with 24 ROCA-enabled accelerators as depicted in Figure 9, or with enlarged cache banks as described below. Our modeled cores are single-threaded and in-order, with a two-level cache hierarchy. We choose in-order cores to maximize performance sensitivity to variations in cache latency, as is commonly done when studying the impact of changes to the cache hierarchy (e.g. [34]). Energy efficiency is modeled by combining our performance models with: McPAT 1.0 [23] for core/directory/L1 power, CACTI 6.5 [28] to obtain power/latency numbers for sequential-access low-leakage L2 cache banks, DSENT [36] for NoC link/router power, and the memory power model in [11]. We approximate memory power by assigning 2.78W to background power and 51nJ per access for a single DIMM as done by Sampson and Wenisch in [33].

Modeled Last-Level Caches. We model a Shared L2 LLC in all experiments. We do not consider private caches beyond the L1's nor a L3 cache in order to maximize sen-

Cores	16 cores, i386 ISA, in-order IPC=1 except on memory accesses, 1 GHz
L1 caches	Split I/D 32KB, 4-way set-associative, 1-cycle latency, LRU replacement
L2 caches	8-cycle latency, LRU replacement S-NUCA: 16 ways, 8 banks, 2MB or 8MB total ROCA: 12 ways, 8x(256K+2x192K+128K)=6MB
Coherence	MESI protocol, 64-byte blocks, standalone directory cache
DRAM	1 Controller, 200-cycle latency, 3.5GB physical
NoC	5x5 or 7x7 mesh, 128-bit flits, 2-cycle router traversal, 1-cycle links, XY routing
Operating System	Linux v2.6.34

Table 2: Simulated systems' configuration

sitivity to L2 hit latency, which is common practice among NUCA studies [15, 17, 21]. We report all results normalized over those from the baseline system, which has a 16-way 2MB 8-bank S-NUCA LLC [17].

We augment the baseline system with 24 ROCA-enabled accelerators and convert the regular L2 banks into ROCA host banks, which renders a 7x7 system as depicted in Figure 9. We conservatively assume that on average the accelerators' reusable memory area is slightly below the typical (as discussed in Section 2) 69%; we thus model 16 accelerators with 192KB of memory and 8 with 128KB, with a way allocation as in Example 2.b in Figure 6. The resulting memory average corresponds to 66% of the total area (100% memory would mean that each accelerator has 256KB of memory, i.e. the same capacity as a regular L2 bank of the same area).

We compare the augmented system against a system that has the same total area, but instead of integrating accelerators it features larger S-NUCA banks. We conservatively assume that this system can be laid out without changing the chip's tiled structure, and therefore model a 5x5 8-bank 8MB S-NUCA configuration. Table 2 summarizes the characteristics of the three systems.

Energy Consumption Model for ROCA banks. We use CACTI to model leakage and per-access energy in the ROCA banks and to account for the energy consumption overhead of the additional tag storage. Furthermore, to account for a worst-case scenario we purposely overestimate the leakage power of accelerator-specific logic (i.e. the logic that does not participate in ROCA) as if it was induced by SRAMs instead of regular logic.

Latency Model for ROCA banks. We assume that cache blocks are accessed from both ROCA accelerators and LLC host banks in 8 cycles: 4 cycles for 128b-flits over the NoC and 4 cycles for message processing and tag lookup. This is in addition to a 2-cycle-per-router NoC latency, as shown in Table 2.

Workloads. We assess the performance impact of varying ROCA accelerator availability by simulating multiprogrammed workloads that are not amenable to acceleration. Ideally, we would simulate a mix of accelerated and non-accelerated workloads. However, we decide against this for two reasons. First, there exist few accelerator benchmarks, and those that are available (e.g. [32]) are not integrated into larger, real applications. Second, even if those accelerated applications were available, we would not be able to obtain insight with respect to accelerator availability rates beyond those present in the particular applications under study.

We therefore sweep in simulation the availability rate of ROCA accelerators, studying its impact on energy efficiency and performance when CPU cores are executing a total of 45 multiprogrammed workloads taken from SPEC06. We simulate those that when multiprogrammed can fit in the systems's 3.5GB of physical memory. The 13 SPEC06 benchmarks used are thus: astar, gobmk, gromacs, h264ref, hmmer, libquantum, namd, omnetpp, perlbench, povray, sjeng, soplex and sphinx3. The remaining 32 multiprogrammed workloads result from random combinations of those 13 benchmarks, with repetitions allowed.

We use the SPEC06 reference input sizes, launching as many benchmarks as cores. For each benchmark we fast-forward for one billion instructions to then record for 256 million instructions. Threads that reach the 256 million instruction mark earlier than others continue running, so that they still contend for shared resources. We do not pin threads to particular cores; the Linux scheduler is free to migrate threads across cores as it sees fit. All benchmarks are compiled with gcc v4.6.3 enabling -O2 optimizations.

Metrics. Energy efficiency is presented in billions of instructions per Joule (BIPJ). Given that our workloads are multiprogrammed, performance is considered equivalent to IPC throughput, i.e. $\sum_i IPC_i$, which is a consistent throughput metric [27].

Model of Accelerator Memory Availability. We make the following assumptions when simulating the availability of accelerator PLMs:

1. Accelerators are always considered to run in bursts of 100K cycles (100us given the 1GHz clock). Assuming a speedup over software of 100x, in this amount of time an accelerator can do an amount of work that is roughly the same as a CPU can do in a scheduling quantum, which is typically around 10ms. Therefore, when in our experiments we say that an accelerator is 50% active, we mean that it alternates between bursts of acceleration and ROCA caching, each 100us long. That is, it has an **activity period T**=0.2ms. Similarly, an accelerator that is 1% active runs every 10ms, i.e. T=10ms.

2. We do not consider the work done by the accelerators as part of the workload: we limit our attention to the *overhead* that accelerator activity causes on the cache substrate. Thus, when reporting energy efficiency numbers, we only count leakage and dynamic power of accelerators for the time period they serve as ROCA banks.

3. If an experiment sweeps over the accelerator use rate of n ROCA banks, throughout the simulation we consider only that same set of n banks. Further, their bursts of activity all start and terminate in unison, i.e. at the same simulated time. This is done to measure a worst-case scenario that maximizes block flushes from the accelerators as they reclaim their PLMs from ROCA.

5.2 Evaluation Under No Accelerator Activity

We first measure the energy efficiency and performance of a cache built on ROCA relative to a standalone S-NUCA, without considering any accelerator activity. This would be

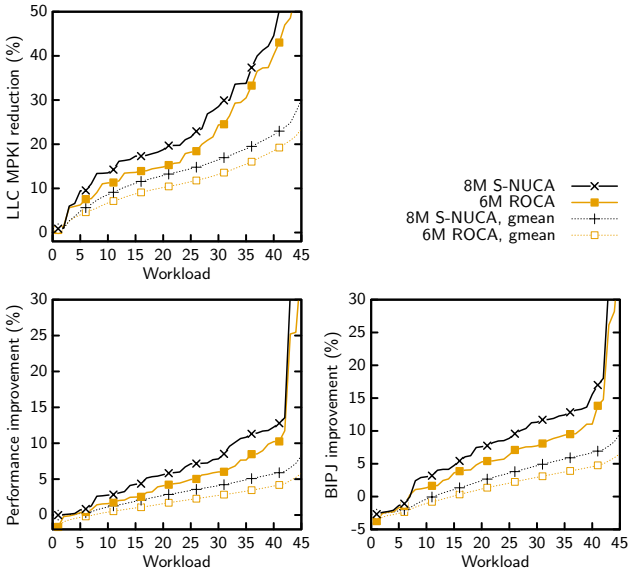
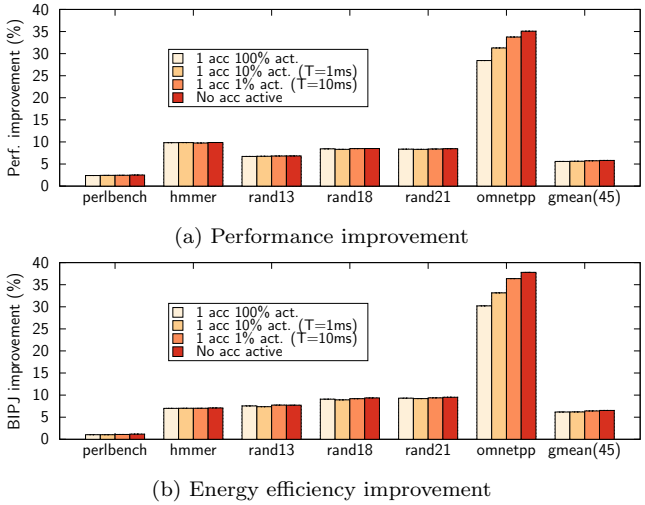


Figure 10: MPKI, performance and energy efficiency improvements over the 2MB S-NUCA baseline for all workloads for 8MB S-NUCA and 6 MB ROCA configurations. All accelerators in ROCA are inactive.



(a) Performance improvement

(b) Energy efficiency improvement

Figure 11: Performance and energy efficiency improvements over 2MB S-NUCA for 6MB ROCA with one accelerator intermittently active. Shown are improvements for some workloads, plus the gmean for all workloads.

a common scenario in general-purpose ROCA-enabled systems: since the workload is not known at design time, the integrated accelerators are highly unlikely to apply to the actual workload. However, with ROCA they can nonetheless provide value by expanding the LLC.

Figure 10 shows the throughput and energy efficiency improvements of the 6MB ROCA and 8MB S-NUCA configurations over the 2MB S-NUCA baseline. Each line plots the improvement of all workloads for each configuration, with the results sorted so that every line is monotonically increasing. An additional dashed line per configuration is also shown to represent the improvements' cumulative geometric mean.

We observe that the results' gap between the 6M ROCA and the same-area 8MB S-NUCA is commensurate with their 25% difference in capacity: ROCA realizes 78% of the MPKI reduction of the 8MB S-NUCA, while relative performance (70%) and energy efficiency (68%) improvements are slightly lower. This decrease is explained by the additional level of indirection that ROCA requires: L2 accesses that hit in an accelerator bank require NoC transfers that are not necessary in S-NUCA, where each regular bank comprehends both cache tags and data. These NoC accesses between ROCA host bank and associated result in additional overhead by consuming energy and increasing hit latency.

Some workloads show an energy efficiency degradation over the baseline for both configurations close to 5%. This is explained by the unresponsiveness of these workloads to increased last-level caching, due to either streaming access patterns or the workload already fitting in the baseline LLC. When this is the case, the investment in leakage power for a larger LLC does not pay off. We do not address the problem of opportunistically reusing accelerator PLMs to dynamically size the LLC; this is left as future work.

The low performance and energy overhead of ROCA compared to a same-area S-NUCA *in the absence of accelerator activity* is therefore established. In the remainder of our evaluation we study the impact of intermittently removing accelerator banks from ROCA, which happens when accelerators become active and reclaim their private memories.

5.3 Same-Logical-Bank Accelerator Activity

We now consider the effect on performance and energy efficiency of the intermittent activity of ROCA-enabled accelerators within the same logical bank. Such a scenario would be most likely found on a general-purpose architecture owned by a power user whose workload we assume can exploit only up to three of the integrated accelerators. We first study the case of a single active accelerator, and then consider the worst case of three accelerators being on the same logical bank; this results in the loss of 8 ways out of the 12 ways that are assigned to a logical bank.

Single Active Accelerator. Figure 11 shows the improvements in performance and energy efficiency over the baseline system for 6MB ROCA with one of the 128K accelerators switching between acceleration and caching. The activity rate of this ROCA bank shows negligible impact on performance and energy efficiency. Only the workloads with large MPKIs are sensitive to the slight L2 capacity reduction caused by the intermittent loss of a single ROCA bank.

Three Same-Logical-Bank Active Accelerators. Figure 12 shows the performance and energy efficiency improvements over the baseline for this scenario. We observe that the difference between the worst case (100% activity) and best case (no accelerators active) is small. This is due to the presence of 4 local ways in the ROCA host banks; having one eighth of the address space only cached by 4 LLC ways does not greatly impact these workloads. Frequent accelerator use (T=0.2ms) has similar impact to 100% activity in MPKI, performance and energy efficiency since in 0.1ms caching there is not enough time to make effective use of the three ROCA banks. Less frequent accelerator use (4% activity, T=2.5ms) yields more positive results across the three metrics, since the caching window is larger.

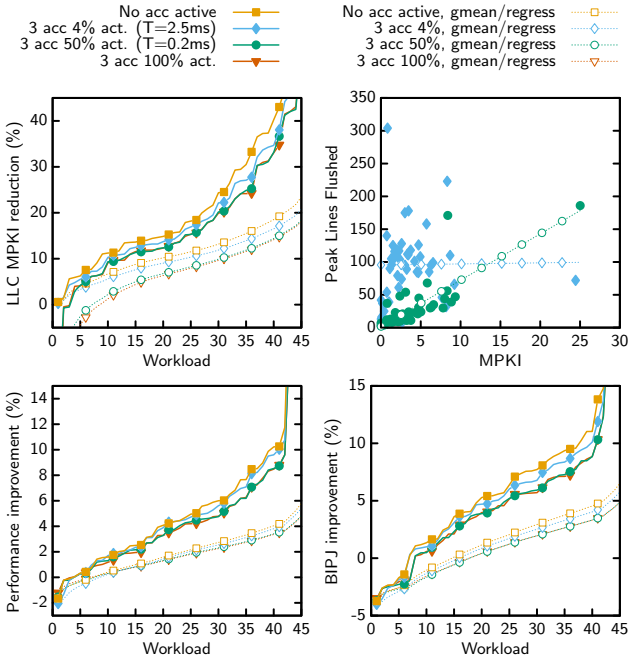


Figure 12: Improvements for 6MB ROCA over 2MB S-NUCA and characterization of peak number of blocks flushed vs MPKI (top right), for varying accelerator activity of 3 accelerators in the same logical bank.

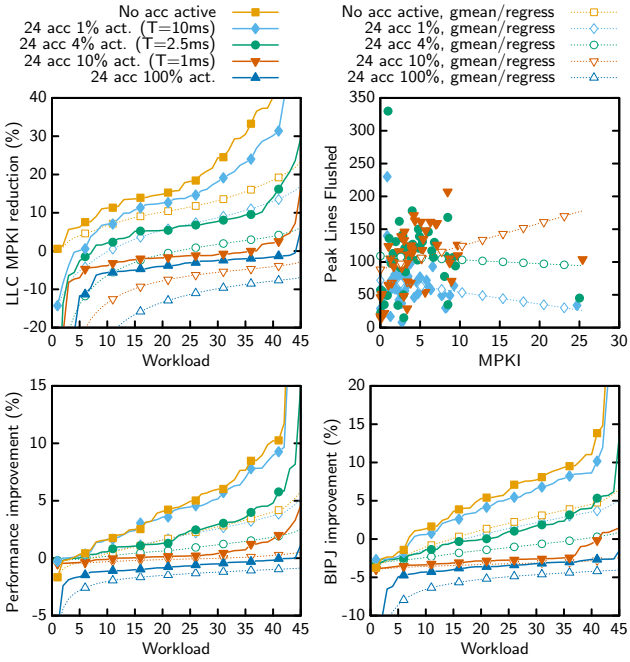


Figure 13: Improvements for 6MB ROCA over 2MB S-NUCA and characterization of peak number of blocks flushed vs MPKI (top right), for varying accelerator activity of all (24) accelerators.

A side effect of a larger caching window is that more dirty blocks must be flushed to DRAM upon accelerator reclamation. This can be seen in the top right plot, where for each run the peak number of blocks flushed by a single accelerator upon a reclaim is shown against the run's MPKI together with their computed linear regression. A larger caching window thus results in larger peak values. However, the correlation between peak flushes and MPKI is stronger as the caching window narrows; the intuition behind this is that given a large enough window, the ROCA bank will be fully populated regardless of how cache-hungry the workload may be. Short windows, on the other hand, are highly sensitive to the miss rate of the workload: only high-MPKI workloads are capable of inserting a significant amount of blocks in the ROCA bank.

5.4 Chip-Wide Accelerator Activity

We complete our performance and energy efficiency evaluation by studying the impact on caching of intense accelerator activity across the chip. In this scenario, representative of a high-performance embedded system, the 24 accelerators switch in unison between caching and acceleration.

Figure 13 shows the MPKI, performance and energy efficiency improvements for this configuration. The worst case for caching (100% activity for the 24 accelerators) remains on average close to the baseline: 0.8% and 4% average performance and efficiency degradation, respectively. This difference is explained by the even larger (close to 10% on average) MPKI drop caused by the low associativity (4 ways) of the ROCA host banks, which for cache-sensitive workloads are outperformed by the equally-sized 16-way baseline cache.

Between zero and full accelerator activity we observe how critical is the activity period T: a caching window of 10ms (1% acc. activity) yields performance and energy efficiency within—respectively—10% and 20% of that without accelerator activity. The higher energy overhead is due to the flushing of blocks upon accelerator reclamation.

A shorter caching window (T=2.5ms) results on average in less than half the zero-activity gains in performance and negligible gains in energy efficiency over the baseline. Moreover, it leads to the highest peak number of flushes (330) upon accelerator reclamation observed over all simulations. An even shorter caching window (T=1ms) is hardly an improvement over the full-activity scenario: performance is similar to that of the baseline, yet energy efficiency is 2% below.

5.5 Summary of Results

Our results show that accelerators that are not used frequently (i.e. with idle windows of 10ms or longer) are prime candidates for ROCA. The average performance and energy efficiency improvements obtained from them for non-accelerated workloads are, respectively, 70% and 68% of those from regular cache banks of the same area, while providing orders-of-magnitude improvements for workloads suitable for acceleration. Furthermore, our results show the importance of allocating a certain portion of the ROCA LLC to host banks (e.g. 2MB out of 6MB in our study) in order to limit performance and efficiency degradation for non-accelerated workloads when most accelerators are active.

Not all memory-rich accelerators are suitable for ROCA, however. An exception to this recommendation are accelerators with rigorous real-time constraints, with deadlines in the order of microseconds; the delay to flush dirty blocks to DRAM when the accelerator is reclaimed from ROCA could compromise the meeting of such deadlines. For example, flushing 330 64-byte blocks, assuming a sufficiently buffered DRAM controller and 128b NoC flit length, would take around 10560 cycles, i.e. 10.5us at a 1GHz clock rate.

6. ADDITIONAL RELATED WORK

The need for greater heterogeneity as a response to the end of Dennard scaling [12] was made evident by Venkatesh et al. [37], who proposed the combination of traditional processor cores and energy-efficient specialized cores in the same die. Heterogeneity, by integrating high-performance accelerators on-chip, has been growing in interest, with work coming from academia [5, 31, 38] and industry [19, 20]. In this context, the importance of PLMs is confirmed by the amount of area dedicated to accelerators' memory [24]; this key role of PLMs has sparked recent efforts to automate and optimize the design of heavily banked memory subsystems for accelerators [30].

Decoupling the utility of specialized hardware from particular workloads has motivated proposals, such as Smart Memories [25], CHARM [7] and LSSD [29], that attempt to partially match the performance and energy efficiency gains of accelerators without giving up programmability. Our approach differs from theirs in that we relinquish programmability not to compromise on performance or efficiency. With Roca, accelerator utility is however decoupled from the workload by augmenting the LLC with the abundant accelerator memories when accelerators would otherwise be inactive.

An interesting approach that shares Roca's objective of reducing the opportunity cost of accelerator integration is Stash [18], whose goal is to minimize on-chip copies of data by implementing a hybrid storage element for accelerators, thereby combining the benefits of both caches and software-managed scratchpads. Stash differs from Roca in that it requires changes to the cache coherence protocol, and similarly to the proposals discussed in Section 2 (i.e. [8, 13, 24]), it is not a good fit as a substitute for high-bandwidth PLMs, since hiding the additional latency of accessing these memories would significantly complicate accelerator designs.

Recent NUCA research has put emphasis on trading cache capacity to reduce hit latency via controlled block placement and replication, e.g. ASR by Beckmann et al. [3], R-NUCA by Hardavellas et al. [15], and Locality-Aware Data Replication by Kurian et al. [21]. These techniques were designed for always-available, equally-sized banks; extending them to support the requirements of banks such as the ones coming from Roca accelerators, which are of diverse sizes and intermittently available, would be valuable future work.

7. CONCLUSION

We have presented Roca, a technique to exploit the abundant private local memories in accelerators to mitigate the opportunity cost of their integration. Roca enables accelerators to provide *utility* even when they cannot directly speed up a workload, by exposing their private local memories to the cache substrate. Our implementation of Roca is practical, requiring minimal modifications to both accelerators and the cache substrate, and incurring a modest area overhead that is almost entirely due to additional tag storage.

Our results show that, relative to a 2MB S-NUCA LLC, a 6MB Roca LLC built upon typical accelerators (i.e. whose area is 66% memory) can, on average, realize 70% of the performance and 68% of the energy efficiency benefits of a same-area 8MB S-NUCA configuration. Further, our results suggest that accelerators with windows of inactivity of 10ms or longer are prime candidates for Roca.

8. ACKNOWLEDGMENTS

This work is supported in part by DARPA PERFECT (C#: R0011-13-C-0003), the NSF (A#: 1219001), and C-FAR (C#: 2013-MA-2384), an SRC STARnet center.

9. REFERENCES

[1] D. H. Albonesi. Selective cache ways: On-demand cache resource allocation. In *Proc. of the Intl. Symp. on Microarchitecture (MICRO)*, pages 248–259, 1999.

[2] K. Barker, T. Benson, D. Campbell, D. Ediger, R. Gioiosa, A. Hoisie, D. Kerbyson, J. Manzano, A. Marquez, L. Song, N. Tallent, and A. Tumeo. *PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual*. Pacific Northwest National Laboratory and Georgia Tech Research Institute, December 2013.

[3] B. M. Beckmann, M. R. Marty, and D. A. Wood. ASR: Adaptive selective replication for CMP caches. In *Proc. of the Intl. Symp. on Microarchitecture (MICRO)*, pages 443–454, 2006.

[4] S. Borkar and A. A. Chien. The future of microprocessors. *Communications of the ACM*, 54(5):67–77, May 2011.

[5] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proc. of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 269–284, 2014.

[6] Z. Chishti, M. D. Powell, and T. N. Vijaykumar. Distance associativity for high-performance energy-efficient non-uniform cache. In *Proc. of the Intl. Symp. on Microarchitecture (MICRO)*, pages 55–66, 2003.

[7] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman. Charm: a composable heterogeneous accelerator-rich microprocessor. In *Proc. of the Intl. Symp. on Low Power Electronics and Design (ISLPED)*, pages 379–384, 2012.

[8] J. Cong, M. A. Ghodrat, M. Gill, C. Liu, and G. Reinman. BiN: a buffer-in-NUCA scheme for accelerator-rich CMPs. In *Proc. of the Intl. Symp. on Low Power Electronics and Design (ISLPED)*, pages 225–230, 2012.

[9] E. G. Cota, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. An analysis of accelerator coupling in heterogeneous architectures. In *Proc. of the Design Automation Conference (DAC)*, pages 202:1–6, 2015.

[10] E. G. Cota, P. Mantovani, M. Petracca, M. R. Casu, and L. P. Carloni. Accelerator memory reuse in the dark silicon era. *IEEE Comput. Archit. Lett.*, 13(1):9–12, Jan. 2014.

[11] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. MemScale: Active low-power modes for main memory. In *Proc. of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 225–238, 2011.

[12] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proc. of the Intl. Symp. on Computer Architecture (ISCA)*, pages 365–376, 2011.

[13] C. F. Fajardo, Z. Fang, R. Iyer, G. F. Garcia, S. E. Lee, and L. Zhao. Buffer-integrated-cache: a cost-effective sram architecture for handheld and embedded platforms. In *Proc. of the Design Automation Conference (DAC)*, pages 966–971, 2011.

[14] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proc. of the Intl. Symp. on Computer Architecture (ISCA)*, pages 37–47, June 2010.

[15] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Reactive NUCA: near-optimal block placement and replication in distributed caches. In *Proc. of the Intl. Symp. on Computer Architecture (ISCA)*, pages 184–195, June 2009.

[16] C. D. Kersey, A. Rodrigues, and S. Yalamanchili. A universal parallel front-end for execution driven microarchitecture simulation. In *Proc. of the Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools (RAPIDO)*, pages 25–32, 2012.

[17] C. Kim, D. Burger, and S. W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *Proc. of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 211–222, 2002.

[18] R. Komuravelli, M. D. Sinclair, J. Alsop, M. Huzaifa, M. Kotsifakou, P. Srivastava, S. V. Adve, and V. S. Adve. Stash: Have your scratchpad and cache it too. In *Proc. of the Intl. Symp. on Computer Architecture (ISCA)*, pages 707–719, 2015.

[19] A. Krishna, T. Heil, N. Lindberg, F. Toussi, and S. VanderWiel. Hardware acceleration in the IBM PowerEN processor: Architecture and performance. In *Proc. of the Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, pages 389–400, 2012.

[20] N. Kurd, M. Chowdhury, E. Burton, T. P. Thomas, C. Mozak, B. Boswell, M. Lal, A. Deval, J. Douglas, M. Elassal, A. Nalamalpu, T. M. Wilson, M. Merten, S. Chennupaty, W. Gomes, and R. Kumar. Haswell: A family of IA 22nm processors. In *ISSCC Digest of Technical Papers*, pages 112–113, 2014.

[21] G. Kurian, S. Devadas, and O. Khan. Locality-aware data replication in the last-level cache. In *Proc. of the Intl. Symp. on High-Performance Computer Architecture (HPCA)*, pages 1–12, 2014.

[22] O. Lempel. 2nd generation intel core processor family: Intel core i7, i5 and i3. In *Hot Chips*, 2011.

[23] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proc. of the Intl. Symp. on Microarchitecture (MICRO)*, pages 469–480, 2009.

[24] M. J. Lyons, M. Hempstead, G.-Y. Wei, and D. Brooks. The accelerator store: A shared memory framework for accelerator-based systems. *ACM Trans. Archit. Code Optim.*, 8(4):48:1–48:22, Jan. 2012.

[25] K. Mai, T. Paaske, N. Jayasena, R. Ho, W. J. Dally, and M. Horowitz. Smart memories: A modular reconfigurable architecture. In *Proc. of the Intl. Symp. on Computer Architecture (ISCA)*, pages 161–171, 2000.

[26] M. M. Martin, M. D. Hill, and D. J. Sorin. Why on-chip cache coherence is here to stay. *Communications of the ACM*, 55(7):78–89, 2012.

[27] P. Michaud. Demystifying multicore throughput metrics. *Computer Architecture Letters*, 12(2):63–66, July 2013.

[28] N. Muralimanohar, R. Balasubramonian, and N. Jouppi. Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In *Proc. of the Intl. Symp. on Microarchitecture (MICRO)*, pages 3–14, 2007.

[29] T. Nowatzki, V. Gangadhar, K. Sankaralingam, and G. Wright. Pushing the limits of accelerator efficiency while retaining programmability. In *Proc. of the Intl. Symp. on High-Performance Computer Architecture (HPCA)*, Mar. 2016.

[30] C. Pilato, P. Mantovani, G. Di Guglielmo, and L. P. Carloni. System-level memory optimization for high-level synthesis of component-based SoCs. In *Proc. of the Intl. Symp. on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, Oct. 2014.

[31] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz. Convolution engine: Balancing efficiency & flexibility in specialized computing. In *Proc. of the Intl. Symp. on Computer Architecture (ISCA)*, pages 24–35, 2013.

[32] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks. Machsuite: Benchmarks for accelerator design and customized architectures. In *Proc. of the IEEE Intl. Symp. on Workload Characterization (IISWC)*, pages 110–119, 2014.

[33] R. Sampson and T. F. Wenisch. Zcache skew-ered. In *Proc. of the 9th Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD)*, 2011.

[34] D. Sanchez and C. Kozyrakis. The zcache: Decoupling ways and associativity. In *Proc. of the Intl. Symp. on Microarchitecture (MICRO)*, pages 187–198, 2010.

[35] A. Seznec. Bank-interleaved cache or memory indexing does not require euclidean division. In *Proc. of the 11th Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD)*, 2015.

[36] C. Sun, C.-H. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic. DSENT - A tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proc. of the Intl. Symp. on Networks on Chip (NoCS)*, pages 201–210, 2012.

[37] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation cores: reducing the energy of mature computations. In *Proc. of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 205–218, 2010.

[38] L. Wu, A. Lottarini, T. K. Paine, M. A. Kim, and K. A. Ross. Q100: The architecture and design of a database processing unit. In *Proc. of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 255–268, 2014.