Journal of Supercomputing manuscript No.
(will be inserted by the editor)

FOS: a Low Power Cache Organization for Multicores

José Puche · Salvador Petit · Julio Sahuquillo · María Engracia Gómez

Received: date / Accepted: date

Abstract The cache hierarchy of current multicore processors typically consists of one or two levels of private caches per core and a large shared last level cache (LLC). This approach incurs area and energy wasting due to oversizing the private cache space, data replication through the inclusive cache levels, as well as the use of highly set-associative caches. In this paper, we claim that although this is the common adopted approach, it presents important design issues that can be addressed by a more energy efficient organization.

This work proposes Flat On-chip Storage (FOS), a novel cache organization that, aimed at addressing energy and area on low-power processors, resolves the mentioned issues. For this purpose, FOS combines L2 and L3 cache levels into a single one, organized as a flat space, and composed of a pool of private small cache *slices*. These slices are initially powered off to save energy and they are powered on and assigned to cores provided that the system performance is expected to improve. To provide fast and uniform access from the private L1 caches to the FOS's cache slices, multiple architectural challenges are overcome, which entails the design of a custom Optical Network-on-Chip (ONoC).

Experimental results show that FOS achieves significant energy savings on both static and dynamic energy over conventional cache organizations with the same storage capacity. FOS static energy savings are as much as 60% over an electrically-connected shared cache; these savings grow up to 75% compared to optically-connected baselines. Moreover, despite deactivating part of the cache space, FOS achieves similar performance values as those achieved by conventional approaches.

Keywords Cache Hierarchy · Multicores · Energy efficiency

1 Introduction

Multicore processors have highly evolved during the last decade. Early multicores consisted of only two cores, and progressively the core count increased with technology advances. Multicores, especially tiled multicores, are designed as a set of cores, each one including

J. Puche E-mail: jopucla@gap.upv.es
S. Petit E-mail: spetit@disca.upv.es
J. Sahuquillo E-mail: jsahuqui@disca.upv.es
M. E. Gómez E-mail: megomez@disca.upv.es

private L1 and L2 caches. Private caches are of fixed size, thus the storage capacity can be larger than that needed for some applications, which wastes energy and area, especially in second level caches. Additionally, the size could not be sufficient for other applications, which causes performance losses. This means that when running multiprogram workloads, if the working set of a given application exceeds its L2 cache capacity, its performance can be harmed even if other L2 private caches are scarcely used or even not used at all. Cooperative caching approaches [8] address this issue by enabling a core to access the L2 cache of neighboring cores.

Nevertheless, recent advances in memory technologies have allowed, mainly in high performance processors, to hide the huge memory latencies by providing large shared L3 caches, which makes cooperative caching less effective. However, a large fraction of the processor die is used to implement these L3 caches. Therefore, because of area and power requirements [41], this design choice is not considered in low-power or embedded processor designs. On the other hand, cooperative caching is not a suitable solution for embedded processors since it does not reduce neither the total cache area nor its energy consumption.

The previous rationale means that there are important design concerns dealing with performance, area, and energy consumption of caches in embedded and low-power processors. This paper proposes *Flat On-chip Storage* (FOS), a novel cache hierarchy that introduces two main cache design features to deal with these concerns: unified cache space and efficient coarse-grain energy management. Unified cache space refers to the fact that FOS replaces the second and subsequent cache levels (e.g. third level if any) of a conventional cache hierarchy with a single flat storage space that can be accessed by all the processor cores.

To ease energy management, FOS is organized as consisting of relatively small and independent *cache slices* (e.g. 64KB). Slices, initially powered off for energy, are powered on and allocated dynamically at run-time to a particular application or deallocated from it according to specific criteria. This way reduces energy management and hardware complexity, since static energy is managed at coarse granularity (i.e. cache slice) instead of deactivating small cache lines (e.g. 64B) [22, 28], whose overload area required to layout the additional wires could be prohibitive.

The proposed approach is suitable for both inclusive and exclusive caches; however, to focus the research this paper concentrates on inclusive caches, which are dominant in current processors. Exclusive caches present important shortcomings, mainly because they complicate the implementation of the coherence protocol when dealing with multicore processors.

To trade off performance and energy, FOS' design must face three main challenges: i) efficient policies should be devised to assign/deassign slices to applications, ii) strategies/mechanisms should be defined to identify the FOS slice to be accessed on an L1 cache miss, and iii) an efficient interconnection that communicates the processor cores to the FOS slices needs to be designed. This work presents the first attempt to address these challenges and makes the following main contributions:

– We propose a novel cache organization that combines L2 and L3 cache levels. This approach consists of a common pool of cache slices, which are dynamically powered on and allocated to specific cores.
– We present a low-overhead algorithm to assign (or deassign) cache slices.
– We devise an ad-hoc optical network on chip (ONoC) that provides uniform access latency to the pool of slices.

Experimental results show that FOS reduces dynamic energy consumption by up to $4\times$ and leakage power up to 60% over conventional cache organizations with the same storage

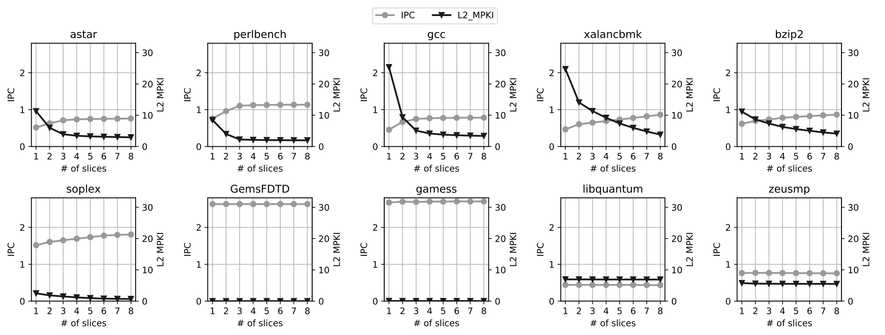

Fig. 1 IPC and L2 MPKI values varying the number of slices (1 slice is 64KB - 8 slices are 512KB).

capacity. Moreover, performance is not harmed by these energy savings, since FOS achieves similar performance values as those achieved by conventional approaches.

2 Taxonomy of Applications

This section characterizes the cache demands of a representative subset of the SPEC CPU2006 benchmarks to provide insights on the cache space required by the applications and to guide the design of FOS. In these experiments, L2 caches are assumed to be composed of slices of 64KB, and the maximum FOS cache space has been fixed to 512KB (8 slices). The experimental setup is described in Section 5. To characterize the applications, we vary the number of slices from 1 to 8, and measure the performance (i.e. IPC) and the number of misses per kilo instruction (MPKI) of the L2 cache. The goal of this study is twofold: i) to analyze the relationship between these metrics and the cache space, and ii) to evaluate the potential performance gains of increasing the cache capacity up to the entire 512KB FOS space for individual applications. Figure 1 presents the results. Applications can be categorized into three main groups according to how applications evolve when adding more cache space:

- **Minimum Slice Needs (MSN)**: Applications in this group are characterized by their performance insensitivity to the number of available slices due to two main reasons. First, some applications, like `libquantum` and `zeusmp`, present an inherent low locality so their MPKI does not change regardless of the amount of available cache slices. Second, the working set of applications like `GemsFDTD` and `gamess` fits in a single 64KB slice, thus their performance scarcely improves or even remains constant with additional slices.
- **Limited Slice Needs (LSN)**: Applications like `astar`, `gcc` or `perlbench` reach their maximum performance when they are executed with 4 or 5 slices. In this category, increasing the number of slices over the saturation point does not impact on the application performance.
- **Non-limited Slice Needs (NSN)**: Applications belonging to this group are called *cache-hungry*, since they always improve performance with higher cache capacities. For instance, both in `xalancbmk` and `bzip2` performance keeps growing with the number of slices.

This study shows that a flexible distribution of slices among the competing applications can potentially achieve important benefits. First, it can provide significant energy savings over conventional L2 caches by powering off slices that are not in use, especially for applications falling in the first two categories. Second, in multiprogram workloads, the performance of LSN and NSN applications can be enhanced since LSN applications benefit from the reduction of interferences and cache trashing while NSN ones can get a high number of slices. Additionally, applications which, due to the small size of their working set, are classified as MSN can also experience performance improvements. This occurs because FOS can prevent other cache-hungry applications from replacing the blocks exhibiting a high cache-locality of the MSN applications, which is critical for their performance. Finally, low locality MSN applications are expected to present the same performance in FOS as in any conventional cache organization.

3 Flat On-chip Storage

As mentioned above, the key idea behind the proposal is to have a common pool of cache slices that replaces all the cache levels (e.g. second and third) except the private L1 caches implemented in the core pipeline. All these slices should be properly interconnected with the cores to avoid high NoC latency deviations. A high-level block-diagram of this proposal is depicted in Figure 2.

Slices of the pool are assigned to specific cores at run-time based on the predicted application requirements, and once a slice is assigned to a core it is set to private mode for that core. Having a common pool avoids the constraint of a fixed-size private cache, which has been identified as a key concern in previous research [30], since it usually incurs area and energy wasting due to over-provisioning cache space. Previous approaches [22] handle energy savings at a very small cache line granularity (e.g. 64B), which complicates the layout of wires and increases area overhead, which makes this approach impractical from an implementation perspective. However, acting on the entire slice itself as an activation/deactivation unit (e.g. 64KB) helps the implementation of cache energy saving mechanisms.

A flexible distribution of the cache resources, in which each application is provided with the cache size that is predicted for performance, can help improve both energy savings and performance. To provide a good trade-off between energy and performance, however, three main architectural challenges must be addressed:

- When should a slice be allocated/deallocated to/from a core?
- Which slice must be accessed by a given core upon an L1 cache miss?
- Which interconnect should be used, considering that slices are not beside the core?

The design decisions made to deal with these issues are discussed below.

3.1 Slice Management Mechanism

This section presents the Slice Management Mechanism (SMM), which implements both a *Slice Allocation Algorithm* and a *Slice Deallocation Algorithm*, discussed below.

Slice Allocation Algorithm: This algorithm, summarized in Algorithm 1, assigns slices to applications based on performance and energy considerations. The algorithm is triggered at fixed-length (X execution cycles) intervals during the application execution.

During each interval, the number of cache misses in the FOS pool is measured to quantify the MPKI of the current interval ($MPKI_{n-1}$). Then, the algorithm predicts if the performance of each application would improve in case the application was provided with

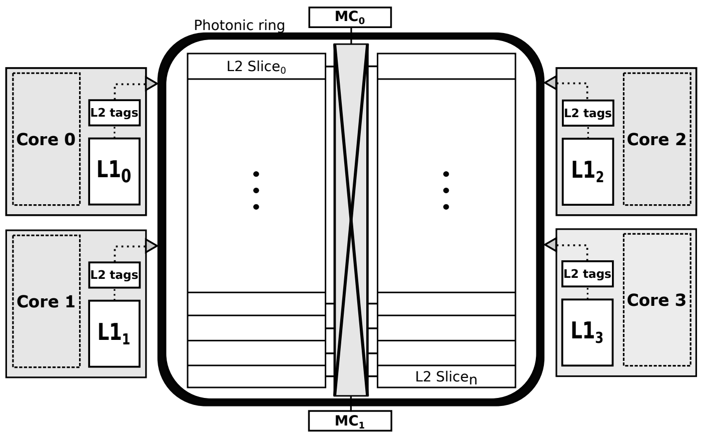

Fig. 2 Schematic of the proposed architecture.

one additional slice. The key challenge is that the algorithm needs to estimate the number of cache misses that each application would have experienced with an additional cache slice $s+1$, while running with only s slices. To deal with this issue, we have modeled extra hardware that works similar to the *Auxiliary Tag Directory* (ATD) [30]. To reduce the area of this structure, it only implements a tag array corresponding to 32 sets, which are randomly selected among all the sets in the cache.

Using the collected data, the algorithm estimates, for each interval, how much the MPKI would have improved with an additional slice. For this purpose, the expected MPKI decrease rate ($MPKI_{dec\_rate}$) of each application with $s+1$ slices is computed in step 2. This metric alone, however, only considers performance, so it is complemented with the $MPKI_{hist}$ metric, which helps improve energy efficiency by providing the average $MPKI$ during a sliding window composed of the last w intervals. This metric can be used, for instance, to check if adding an extra slice provides marginal system performance benefits, which might not justify in terms of energy consumption the activation of a new slice. Lastly, one more metric is computed to obtain the weight of the last interval MPKI in the mentioned sliding window ($MPKI_{weight}$).

Step 3 aims to check the conditions that must be fulfilled for energy efficiency. The $MPKI_{hist}$ must be greater than a minimum threshold (Thr_{window}), and the number of FOS misses ($MPKI_{n-1}$) must be greater than a threshold (Thr_{min}). If none of these conditions is satisfied Step 4 is skipped and no slice is activated.

Finally, Step 4 checks the conditions for performance provided that Step 3 conditions have been fulfilled. The $MPKI_{dec\_rate}$ of the application must be higher than the Thr_{dec} threshold, or the weight of the last interval MPKI in the sliding window ($MPKI_{weight}$) must exceed threshold Thr_{weight}. The last condition enables the proposal to react to sharp application phase changes that cannot be detected by $MPKI_{dec\_rate}$.

The proposal requires keeping track of which slices are assigned to each core. To do so, the *slice allocation logic* is accessed at the end of each interval through the devised optical network (ONoC). When a slice is requested, the allocation logic selects, powers on and assigns a slice to the requesting core by sending the corresponding acknowledgment to it. If several requests take place simultaneously, the allocation logic serves them following a

Algorithm 1 Slice Management Mechanism.

```
Algorithm Inputs: $n$: current interval number; $s$: slices currently assigned to the application; $w$: size of the history window (measured in number of intervals); $\{Thr_x\}$: thresholds.

————-Allocation Algorithm————-

1. Initialization. At the beginning of interval $n$, for each running application, gather its $MPKI_{n-1}$ and $MPKI_{n-1}(s+1)$.
2. Metric computation. For each running application, calculate:
       $MPKI_{dec\_rate} = \frac{MPKI_{n-1}(s+1)}{MPKI_{n-1}} - 1$,
       $MPKI_{hist} = \frac{\sum_{i=n-w}^{n-1} MPKI_i}{w}$, and
       $MPKI_{weight} = \frac{MPKI_{n-1}}{MPKI_{hist}}$.
3. Filtration. Skip step 4) iif:
       $MPKI_{hist} < Thr_{window}$ or $MPKI_{n-1} < Thr_{min}$.
4. Request. Request a new slice iif:
       $MPKI_{dec\_rate} > Thr_{dec}$ or $MPKI_{weight} > Thr_{weight}$.

———- Deallocation Algorithm ———-

1. Initialization. At the beginning of interval $n$, for each running application, gather its $MPKI_{n-1}$ and $MPKI_{n-1}(s-1)$.
2. Metric computation. For each running application, calculate:
       $MPKI_{inc\_rate} = 1 - \frac{MPKI_{n-1}}{MPKI_{n-1}(s-1)}$,
3. Release. Release an allocated slice iif:
       $MPKI_{inc\_rate} < Thr_{inc}$ and $IddleInts > Thr_{rel}$.
```

simple FIFO order. Note that the allocation logic is simple and only works between intervals upon requests of slices.

Slice Deallocation Algorithm: The operation of releasing a slice allows feeding the common pool of free slices and prevents an application from unnecessarily holding a high number of slices after a hungry phase. The release mechanism is summarized in the bottom of Algorithm 1. This algorithm aims to estimate whether the performance of a given application on a long-enough steady state would remain the same or similar in case a slice is deallocated from it. For these estimates, the ATD is used.

First, on each interval, the algorithm obtains the $MPKI_{n-1}(s-1)$ of the application. This value is used to estimate the number of cache misses that a given application would have experienced during the last interval with one slice less. Using the value provided by the ATD, we obtain the estimated $MPKI_{inc\_rate}$ metric, which is the ratio that the MPKI is expected to be increased in case one slice was deallocated from the cache space of each application. The $MPKI_{inc\_rate}$ is compared to the Thr_{inc} threshold and, if it is smaller enough and the application is in a steady state (i.e. it has not requested any new slice during the last Thr_{rel} intervals), one slice is released. The Least Recently Used (LRU) policy is employed to choose which slice is deallocated.

The deallocation process is carried out by draining all the blocks in *modified* state that are stored in the slice to be powered off. During the draining interval, the dirty blocks are written back to main memory over time, and the blocks not being evicted are kept accessible

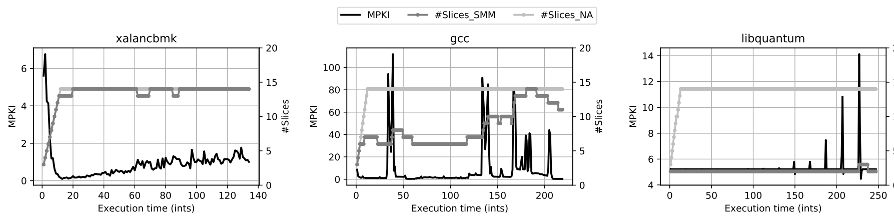

Fig. 3 MPKI and allocated slices for `xalancbmk`, `gcc` and `libquantum` along execution time.

to the application while the draining process is being carried out. When the draining phase finishes, the slice is powered off and becomes available to be assigned.

3.2 Slice Management Mechanism Evaluation

We evaluate the behavior of the allocation/deallocation algorithms of the Slice Management Mechanism (SMM) across different types of applications. To this end, Figure 3 depicts a histogram of the MPKI and the number of slices allocated to `xalancbmk`, `gcc` and `libquantum` along the execution time. The figure also shows the number of slices used by FOS under no slice allocation restrictions (NA). Under this setup, slices are assigned without restriction as soon as more cache space is required, they are never deallocated and there is no constraint regarding the number of slices that a single core can allocate.

In `xalancbmk`, an application previously identified as NSN, it can be observed that SMM allocates slices from the beginning of the execution until the application gets 14 slices, which is the maximum number of slices per core that has been set in these experiments. Notice that the deallocated slices are quickly recovered by the SMM since MPKI improvements are expected along all the execution time. This means that the devised algorithm correctly identifies the slice needs for `xalancbmk`.

In `gcc`, an LSN application, the algorithm must face different stages during the execution. During the initialization stage (intervals 0 to 40), SMM assigns up to 6-7 slices to the application. Then, a sharp increase in the MPKI is observed, thus the allocation algorithm increases the number of slices up to 8. After that, the application returns to a steady state and the deallocation algorithm turns off two slices, coming back to 6. At the end, the application enters in an irregular stage where several MPKI peaks are observed. The final stage is managed by the algorithm by reacting to these sharp increases turning on more slices, while simultaneously trying to deallocate slices. This example shows how SMM is able not only to identify slice needs in different applications but also to react to sharp increases in the MPKI.

Lastly, `libquantum` is studied as an example of an MSN application. As observed, the SMM only allocates two slices even when the MPKI is higher than that observed in `xalancbmk`. This means that the *ATD-based* algorithm correctly detects that allocating more space for this application is pointless because an MPKI reduction is not expected.

To conclude this study and provide insights on the effect of powering off slices, Figure 4 plots the average number of slices allocated depending on whether the deallocation algorithm is used or not. As can be seen in the figure, the allocate/deallocate approach reduces in some applications like `astar` and `milc` up to almost 2 slices the number of allocated slices. This directly translates (as it will be shown later in Section 6.1) to a reduction of up

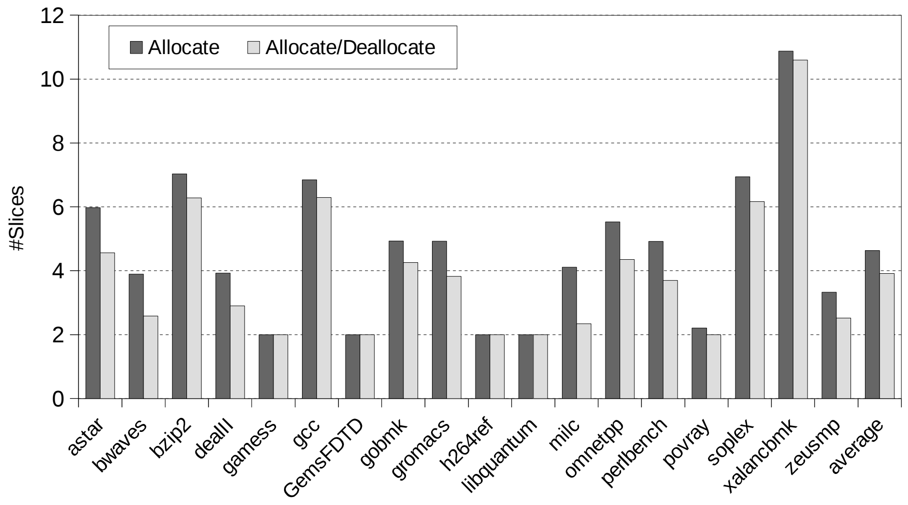

Fig. 4 Average amount of slices allocated with and without the deallocation algorithm.

to 30% in the static energy consumed due to leakage currents. Only those applications that do not allocate any slices (NSN applications) keep the same amount of slices under both approaches, which means that energy savings are achieved for any other types of applications. Therefore, activating the deallocation algorithm has a significant effect on the energy consumed by FOS. Moreover, performance is scarcely affected when the deallocation algorithm is active (less than 1% performance degradation in the worst case).

3.3 Implementation Issues and Shared Data Support

Private Tag Array: Each application will have assigned a subset of the FOS slices. Then, upon a L1 cache miss, a given core only needs to check those slices assigned to it to locate the block in the pool of slices. For this purpose, we decoupled the tag arrays of the slices from the data arrays and moved these tag arrays close to the L1 caches similarly as done in some processors (e.g. the L3 of the IBM POWER5).

The tags of the decoupled slices are managed on each core by the Private Tag Array (PTA), which corresponds with the *L2 tags* modules shown in Figure 2. The PTA block diagram is presented in Figure 5. As observed, the PTA is organized in multiple (e.g. $MaxSlices$) banks, where each bank, when active, keeps the tags associated to one of the assigned slices. Those banks that are not in use (depicted in darker color in the figure) are switched off. The operation of the PTA is described as follows:

1. The PTA is indexed with the address of the block to be accessed in the FOS pool.
2. The active banks are accessed in parallel by the logic and the results (hit or miss) are notified to the *Selection Logic*.
3. Upon a hit, the Selection Logic provides the slice, set, and way identifiers where the target block is stored.

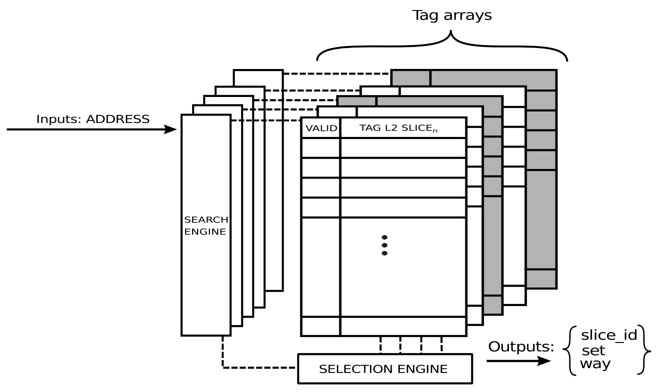

Fig. 5 High-level hardware schematic of a PTA module.

4. If the tag is not present in any bank, i.e. on a FOS pool miss, the Selection Logic replaces the LRU line in the LRU slice. The decision of retrieving the LRU slice follows a hierarchical LRU approach [2], much simpler than a strict LRU.

Hardware Overhead: The main hardware overheads introduced by FOS are the PTA and the two ATD structures required by the SMM. Below, the overhead associated to these structures is studied. First, a PTA requires as many banks as the maximum number of slices that a single core can be assigned. Theoretically, the maximum number of slices in the PTA is given by the equation $MaxSlices = NumSlices - 2 * (n - 1)$, where n is the number of cores. For scalability reasons we limit this parameter to 12 slices. This parameter should be limited in order to provide scalability to the system. However, this can not be done at the cost of performance. In this work, the effect of this parameter on performance has been experimentally explored, and results showed that setting this parameter to 12 slices allows the system to achieve the maximum performance in almost all the studied cases. Hence, this is a reasonable value that trades off performance and scalability. Then, the hardware overhead per PTA is obtained by multiplying $MaxSlices$ by: i) the number of blocks that can be stored in a slice $Slice_{size}/Block_{size}$ and ii) the number of bits in a tag entry $Valid_{bit} + Tag_{bits}$. Second, the overhead associated to an ATD that leverages *Set Sampling* is up to 32 replicated sets per core [30], which is translated to 960 bytes.

Overall, the tag area required for PTAs and ATDs in a 2-core, 16-slice CMP is 82 KB (39 KB PTA + 2KB ATD on each core), which is by 7.4% of the total area (data plus tag array) of a 1MB shared cache. Compared to a conventional cache (which occupies 4.65% of the total area), this represents a relatively small (i.e. by 2.97%) overhead. To clearly expose this overhead, Table 1 lists the different tag capacity of FOS and a conventional cache scaling with the number of cores and slices. In this table, the tag space of FOS has been computed assuming $MaxSlices = 12$ and a 25-bit tag size.

Finally, we discuss the hardware cost associated to the implementation of the Slice Management Mechanism. First, little hardware is needed to compute the four MPKI related metrics (i.e. $MPKI_{n-1}(s)$, $MPKI_{n-1}(s+1)$, $MPKI_{hist}$, $MPKI_{weight}$) used in the algorithm. This hardware compute the metrics with a small set of hardware counters that keep track of the target event. For instance, two counters are needed to gather the number of committed instructions and the amount of cache misses, which are required to obtain $MPKI_{n-1}(s)$. In addition, five registers are used to store the threshold values. As dis-

Table 1 Evolution of size devoted to tag-storing structures in FOS .

	Conventional Cache		**FOS**		
#Cores	**Data Array**	**Tag Array**	**Data Array**	**Tag Array**	**Overhead**
2	1024 KB (95.35%)	50 KB (4.65%)	$16 \times 64 = 1024$ KB (92.6%)	82 KB (7.41%)	32 KB (2.98%)
4	2048 KB (95.35%)	100 KB (4.65%)	$32 \times 64 = 2048$ KB (92.6%)	164 KB (7.41%)	64 KB (2.98%)
6	3072 KB (95.35%)	150 KB (4.65%)	$48 \times 64 = 3072$ KB (92.6%)	246 KB (7.41%)	96 KB (2.98%)
8	4096 KB (95.35%)	200 KB (4.65%)	$60 \times 64 = 4096$ KB (92.6%)	328 KB (7.41%)	128 KB (2.98%)
10	5120 KB (95.35%)	250 KB (4.65%)	$72 \times 64 = 5120$ KB (92.6%)	410 KB (7.41%)	160 KB (2.98%)
12	6144 KB (95.35%)	300 KB (4.65%)	$88 \times 64 = 6144$ KB (92.6%)	492 KB (7.41%)	192 KB (2.98%)

cussed below, there is no need to use additional logic neither to compute the metrics nor to check the thresholds, since these fast computations can be carried out with the available functional units of the core at the time the core is stalled.

Execution Time Overhead: Note the SMM does not need to act at strictly fixed-length periods of time. Therefore, to mitigate the overhead in time, the logic of the devised approach could be triggered during the core stall cycles (e.g. when the ROB is blocked due to long memory latencies). According to our experiments, the period of time when the processor is blocked, referred to as the core slack time, is on average a 24% of the total time. Therefore, taking into account the low complexity of the required computations (e.g. just tens of processor cycles every 40M cycles), there is no appreciable impact on the system performance even if the core is stopped to perform these computations.

Shared Data Support: This work focuses on multiprogram workloads composed of sequential applications. To support the execution of shared memory multithreaded applications, FOS needs to be extended to avoid multiple non-coherent copies of the same block to be present in the slice pool. To address this issue, FOS could be extended with a small number of distributed directory nodes, similarly to distributed cooperative caches schemes (e.g. [18]). With this structure, after a PTA miss, FOS would be able to notify the accessing core if the block is already present in the slice pool, so preventing the replication of shared blocks. In addition, the sharer vector and other coherence protocol information could be stored either in the distributed directories or in the slices. The design of such structures and FOS for multi-threaded applications is out of the scope of this paper.

4 Optical Network-on-Chip

FOS provides a novel and flexible management of a common pool of slices. This disrupting approach relies on a Network on-Chip (NoC) that must fulfill two main conditions: speed and low latency variability. Speed is needed since most of the accesses that miss in the L1 cache result in (by using the PTA) hits in the FOS pool, thus the network must be fast enough to avoid delaying these hits. With respect to low latency variability we mean that the access to the target slice must present similar latency regardless of the accessed slice. Distinct NoCs might be devised fulfilling these conditions, although in this work we focus on *Optical Networks-on-Chip* (ONoCs).

Current advances in silicon nanophotonics allow the integration of a complete functional optical network on a single chip [9, 27, 36]. Optical networks present several features that meet the mentioned requirements. First, fast transmissions are possible even with a reduced number of optical resources. Second, the impact of distance on performance in optical networks is much lower than in traditional electrical networks. Nevertheless, an ONoC must

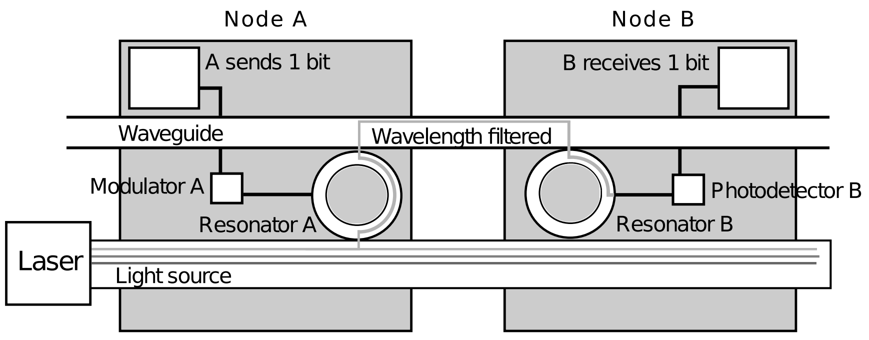

Fig. 6 End-to-end transmission using optical interconnects.

be properly designed in order to avoid a prohibitive increase in the overall network energy consumption.

4.1 Background on ONoC Components

To make this paper self-contained, next we describe the main components needed to fully integrate an ONoC in a silicon die and to achieve a point-to-point communication with optical interconnects. First, a laser source is employed to inject light into the chip. On-chip lasers are considered the most suitable solution in ONoCs in terms of energy efficiency and energy proportionality [14]. The light signal introduced by the laser is transmitted through waveguides, which are narrow slots made over one layer of the die and confined between cladding layers [37]. The signal transmitted over a single waveguide can be multiplexed in different wavelengths using the *Dense Wavelength Division Multiplexing* (DWDM) technique. DWDM allows higher bandwidth density and avoids introducing a prohibitive number of waveguides into the chip [7]. Resonators are ring-shaped components that filter a certain wavelength from the multiplexed light signal. Microring resonators can be tuned to filter different wavelengths by applying an electrical pulse to them or by rising their temperature (using a well-known process called *thermal tuning*). Finally, electro-optical and opto-electrical conversions are performed by modulators and photodetectors, respectively. Modulators are electro-optical components that convey the digital signal inside the wavelengths filtered by the resonators. Similarly, photodetectors transform the filtered optical signal in an electrical signal.

Figure 6 shows an example of an end-to-end optical transmission between two nodes A and B. In the figure, the ring resonator in node A absorbs a given λ_i wavelength and modulates a bit stream over it. The modulated optical signal is carried along the waveguide to the destination node (node B), whose resonator filters the mentioned wavelength and uses its photodetector to reconstruct the electrical bit stream, completing the transmission.

4.2 FOS ONoC

The main features of the devised network are:

– A rather uniform and low transmission latency regardless of the accessed slice.

Table 2 Optical network-on-chip parameters and latencies.

ONoC Parameters	
Frequency	10 GHz
Wavelengths per channel	32 - 128 - 8 λ
Signal propagation	11.4 ps/mm
Modulation bandwidth	10 Gbps
ONoC Latencies (ps)	
Token transmission	Varying in range [100..500]
Microring tuning delay	400
Data modulation (64b-576b)	100 - 500
Trans. latency	Varying in range [100..900]

– Reduced number of optical resources and network complexity.

Table 2 summarizes the modeled ONoC parameters and latencies. Considering the aforementioned network characteristics, a ring topology can be used since it does not involve the use of optical switches and it has an inherent low complexity. Optical rings [16] have been studied in previous research work and have been proved to work with a reduced number of optical components.

With the aim of reducing the data access time, the FOS ring is configured with three separated, spatially multiplexed channels C_0, C_1 and C_2, which interconnect the L1 and the pool of slices. Channel C_0, provided with 32 wavelengths, is used to send requests and data from L1 caches to the FOS level. Next, the C_1 channel is used to deliver the requested blocks from the FOS slices to the L1 caches. This channel is critical for performance, so in order to provide a reduced latency, it is configured with 128 wavelengths. Finally, a third channel is employed to communicate the FOS slices with the PTA structure described in section 3.3. FOS slices use this channel to notify the PTA when writing and replacement operations are finished. Since notification messages are small (4B), this channel is provided with just 8 wavelengths. Channels use different waveguides, so transmissions can take place in all of them simultaneously.

Communication on each channel follows a custom *Multiple Writer Multiple Reader* (MWMR) [7] approach. MWMR requires several steps before a transmission is performed. First, a sender node must get access to the channel, which is granted by an arbitration logic. Token-based arbitration is widely used in optical rings since it guarantees collision-free transmissions and barely introduces extra overhead [42]. Token-based arbitration requires to acquire 1-bit token before transmitting data, and optical tokens help reduce the token acquiring latency [42]. The implementation of optical tokens only requires an additional wavelength on each channel. Token transmission latency mainly depends on the silicon lightspeed and the path length; assuming a signal propagation of light of 11.4 ps/mm [43] in silicon, a 10GHz ONoC, and a ring length of 44.8 mm, token transmission latencies range from 100 ps to 500 ps, depending on the distance.

After getting ring access, the sender must notify the receiver that a message is going to be sent using a given wavelength λ_i. This action is performed using several extra wavelengths to ask the receiver to turn on its ring resonators in order to read the received wavelength. Regarding tuning/detuning latencies, recent work [43] assumes a 400 ps tuning delay.

Once a communication path has been established between two nodes, the data transmission can be carried out. The overall transmission latency (leaving apart communication setup) comes from delays associated to electrical-to-optical conversion, data transmission

Table 3 Loss values of the photonic components.

Component	Value	Reference
Laser Efficiency	5 dB	[43]
Coupler	1 dB	[25]
Waveguide path loss	0.1 dB/mm	[43]
Waveguide bend/cross	0.005/0.5 dB	[43]
Ring drop	1 dB	[25]
By/Through ring loss	0.001/0.1 dB	[25]
Photodetector	0.1 dB	[25]
Receiver sensitivity	-25 dB	[25]

along the waveguide, and optical-to-electrical conversion. The former conversion introduces a delay by 9.5 ps/bit and the latter by 4.0 ps/bit [3]. In the studied system, request messages are 8B (64 bits) long, data messages are 72B (576 bits) long and notification messages are 4B (32 bits) long. Considering the number of wavelengths available on each channel (32, 128, 8) and 10 Gbps conversion speeds, each conversion requires 200 ps, 500 ps and 400 ps, respectively, on each channel. Analogously, the data transmission delay also depends on the length and the width (number of wavelengths) of the optical path. For a 44.8 mm ring and depending on the number of wavelengths, the maximum transmission latencies for 64-, 576- and 32-bit messages are 600, 900 and 900 ps, respectively. These latencies can be lower for path lengths shorter than 44.8 mm.

Overall, for a 2 GHz core clock frequency and assuming no contention, the whole transmission latency (i.e. considering the latencies of tuning resonators, both conversions, and the message size) of any message varies, depending on the distance, between 2 and 4 clock cycles. In short, from an experimental perspective, the NoC latency variability does not exceed two clock cycles regardless of the location of the end-to-end points, hence becoming rather uniform in the studied system. Notice that other alternative networks, like an electrical mesh or a crossbar, might be much slower and present a higher latency variability.

4.3 Energy Consumption in the FOS ONoC

To satisfy the latency requirements, in addition to include a relatively high number of wavelengths on the C_1 channel, FOS ONoC replicates some optical resources to allow simultaneous transmissions on channels C_0 and C_2. This section analyzes the energy consumption of these components.

The components that determine the power consumption in an ONoC are mainly the laser and the microring resonators. First, regarding the microrings power consumption, according to previous research [13] we consider 5μW/ring for the thermal tuning of resonators. Although a huge number of microrings performing filtering actions might increase the power loss and the crosstalk noise power, the FOS ONoC does not present this technological constraint because all its rings never work simultaneously. The worst case scenario for FOS occurs when all the three channels are communicating simultaneously and, in this case, only requires a reduced number of tuned microrings.

Second, laser power in ONoCs depends on the total losses that optical devices introduce along the communication path. To estimate the minimum laser power needed to reach all the components in the FOS ONoC we use the power model provided by Morris et al. in [25], given by the equation $P_{laser} = P_{rx} + C_{loss} + M_s$, where P_{laser} is the laser power, P_{rx} is the receiver sensitivity, C_{loss} is the channel loss, and M_s is the system margin. The model

Table 4 Baseline system parameters.

	Core
Number of cores	2 - 4, OoO, 4 issue/commit width
Frequency	2 GHz
ROB size	128 entries
	Cache Hierarchy
L1 Inst-Data cache	Private, 32KB, 8-way, 64Bytes, 2 cc
L2	Private, 512KB, 16-way, 64Bytes, 8 cc
	Interconnect L1-L2
Frequency	2 GHz
Bandwidth	64 Bytes/cycle
	Main Memory & Memory Controller
DRAM bus freq.	1066MHz
DRAM device	DDR3 (2133 Mtransfers/cycle) 8 banks
Latency	t_{RP}, t_{RCD}, t_{CL} 13.09ns each

is fed with the loss values listed in Table 3, and the outcome provides the minimum laser power to carry a signal strong enough to be possibly received by the photodetectors on every node.

The energy per bit for the FOS ONoC has been computed using the average number of transferred bits and execution time across the executed workloads. According to the model, the energy per bit consumed by the ONoC is up to 1.5 pJ/bit. In contrast, the energy dissipation value expected in electrical links is 0.25 pJ/bit (estimated with ORION 2.0 [21]). As expected, the ONoC presents higher energy consumption than conventional electrical links, but it should be noticed that electrical private links only implement point-to-point communications, while the FOS network interconnects every L1 cache to every FOS slice. Nevertheless, the power savings reached by FOS mainly come from slice deactivation as it is discussed in Section 6.

5 Experimental Framework and Studied Approaches

5.1 Simulation Setup

We have widely extended the code of the Multi2Sim simulation framework [40] to model and evaluate our approach. To improve the accuracy of the DRAM memory subsystem, Multi2Sim has been linked to the DRAMSim2 framework [32], which is a hardware-validated DRAM simulator. Also, the CACTI v6.5 [26] tool has been used to estimate the energy consumption and access times of the studied cache structures for a 32nm technology node. Experiments have been carried out using the SPEC CPU2006 benchmark suite [38]. Applications have been executed until they commit at least 500M instructions after fast-forwarding the initial 300M instructions. When evaluating mixes composed of multiple applications, they are kept running until the slowest one commits the target number of instructions, but statistics for each individual application are gathered at the time it commits the targeted 500M instructions. This way prevents some applications to run in isolation during the last part of the execution of the mix, which would result in better IPC values.

5.2 Studied Approaches

The FOS implementation mainly modifies the cache hierarchy and the NoC. In order to study where the achieved benefits come from, the proposal has been compared with four different systems modifying these components; that is, different L2 cache organizations and L1-L2 interconnection networks have been considered. All the four combinations that have been considered will be referred to as a tuple X-Y, where X indicates the L2 cache organization (e.g. shared or private) and Y the underlying NoC technology (e.g. electrical or optical). For instance, *Shared-OPT* refers to a system with an L2 shared cache with optical NoC. Below, the four baseline schemes are discussed.

- **Private-ELC:** This scheme presents 512KB L2 private caches, connected to the corresponding cores through electrical links. This configuration is used to study the performance constraints associated to fixed-size private caches. Table 4 summarizes the main parameters of this baseline system.
- **Shared-ELC:** Unlike the previous one, this scheme presents a unified shared L2 cache connected to the L1 cache level with electrical links. This scheme is aimed at comparing FOS against a common shared space, which exhibits inter-application interference.
- **Shared-OPT:** This configuration replicates the same cache hierarchy as the *Shared-ELC* approach, but electrical links are replaced by an optical ring. The parameters and latency values used in this ring match those of FOS 's ONoC (see Section 4.2). This scheme, together with *Shared-ELC*, contribute to discern whether the performance enhancements come from reducing the network latency or from improved performance of the shared organization.
- **NUCA-OPT:** A NUCA-based approach is introduced to replicate the FOS system without the SMM algorithm. For this purpose, this scheme is configured as a pool of n cache slices of k size, where n and k match the values selected for the FOS setup. This scheme presents the same optical network as FOS. The motivation behind this scheme is twofold: i) exposing clearly the energy efficiency benefits in terms of cache management brought by the SMM algorithm, and ii) comparing FOS against a scheme with faster cache modules and equal network latencies.

Finally, **FOS** has been configured in the same way as *NUCA-OPT*, that is, as a pool of n slices of k size. Experimental results consider k equal to 64KB and n equal to 8 times the number of cores. In this way, the compared schemes have the same storage capacity, i.e. 1024KB and 2048KB for 2- and 4-core systems respectively.

The reason why we select 64KB sized slices for FOS is twofold. First, reducing the slice size below 64KB increases the demands of networks resources, hence increasing the energy consumption and the hardware complexity. Second, the larger the granularity, the lower the number of slices (i.e. the pool size) that SMM has available to distribute, hence the lower the flexibility of the allocation and the deallocation algorithms which can also affect the performance. Taking these considerations into account, we experimentally concluded that 64KB is the most suitable slice size for evaluation purposes.

Regarding the SMM algorithm setup, a wide set of experiments has been performed to tune the SMM parameters. These parameters could be further refined with more experiments, however, this refinement is out of the scope of this work. To illustrate the potential of FOS, this work presents the results with the parameter values that showed the best energy efficiency on average through most of the experiments; that is, Thr_{min}=0.2, Thr_{window}=0.8, Thr_{dec}=0.25, Thr_{weight}=1.5, Thr_{rel}=25. Finally, w and the interval length have been set to 10 intervals and 40k cycles, respectively.

Table 5 Composition of the evaluated 2- and 4-benchmark mixes. Legend: M: Minimum Slice Needs, L: Limited Slice Needs, N: Non-limited Slice Needs.

	4-benchmark mixes	**2-benchmark mixes**
Mix0	astar (L), dealII (L), milc (M), soplex (L)	astar (L), bzip (N)
Mix1	GemsFDTD (M), soplex (L), xalancbmk (N), h264ref (M)	astar (L), gcc (L)
Mix2	xalancbmk (N), gromacs (L), omnetpp (L), zeusmp (M)	astar (L), xalancbmk (N)
Mix3	libquantum (M), omnetpp (L), milc (M), xalancbmk (M)	gobmk (L), gromacs (L)
Mix4	perlbench (L), povray (M), gamess (M), gromacs (L)	gobmk (L), h264ref (M)
Mix5	omnetpp (L), astar (L), libquantum (M), milc (M)	milc (M), h264ref (M)
Mix6	gamess (M), perlbench (L), omnetpp (L), xalancbmk (N)	omnetpp (L), perlbench (L)
Mix7	gcc (L), libquantum (M), milc (M), zeusmp (M)	povray (M), libquantum (M)
Mix8	astar (L), gcc (L), dealII (L), zeusmp (M)	soplex (L), zeusmp (M)

5.3 Mix Design

To evaluate our approach, experiments have been launched for applications running alone and for multiprogram workloads in conventional 2-core and 4-core multicores. Experiments with more cores are not required since, as explained in Section 6.3, our approach assigns the same cache space to each application regardless of the number of co-runners. The multiprogram mixes were randomly generated varying the ratio of applications belonging to each one of the three categories identified in Section 2. The list of studied multiprogram workloads for 2-core and 4-core experiments is presented in Table 5.

6 Experimental Results

This section analyzes the impact of the devised FOS's cache management and slice distribution on energy and performance of multiprogram workloads. By design, FOS assigns slices exclusively as private to cores, which means that, provided that there is enough cache space, FOS will assign the same cache space to each application as it assigns in individual execution. In other words, FOS behaves similar for each application in multiprogram workloads as it behaves in individual execution in terms of performance and energy. This behavior is a key issue of FOS's design, which is aimed at scaling with the number of cores or concurrently running applications.

6.1 Energy Consumption of Multiprogram Workloads

This section evaluates the energy consumption of the 2 and 4-benchmark multiprogram workloads across all the studied approaches. Regarding 2-benchmark mixes, Figure 7 plots the consumed energy normalized to the *Shared-ELC* approach and broken down into: i) static energy consumption due to cache leakage, ii) dynamic energy consumption due to cache lookups, and iii) network energy consumption. We show the latter energy component in order to highlight the energy consumption differences between both electrical and optical approaches, since in this work FOS is implemented with a custom optical ring. Notice that energy savings directly coming from the proposal, however, are found in the cache's (static and dynamic) energy values, and these savings are independent of the underlying network supporting FOS. In other words, in case that FOS was implemented over a high-performance electrical network, the energy savings achieved by the cache management would remain.

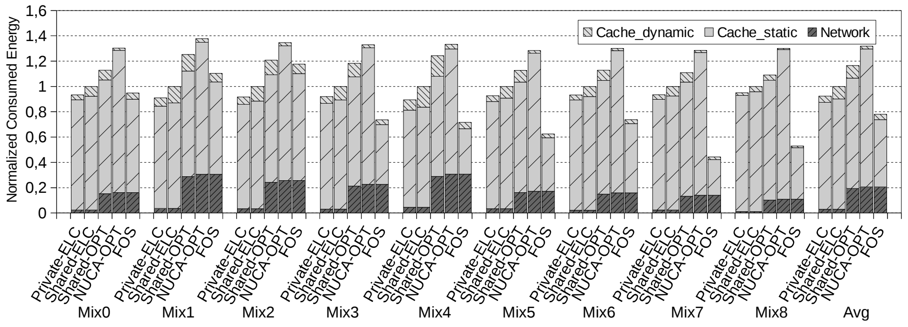

Fig. 7 Normalized energy consumption of 2-benchmark mixes.

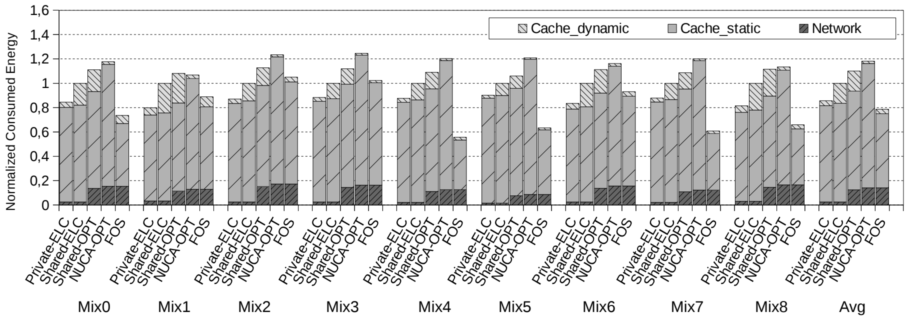

Fig. 8 Normalized energy consumption of 4-benchmark mixes.

As observed, the static energy (i.e. leakage currents) consumption of caches is the component that clearly dominates the overall energy consumption across all the conventional cache organizations. Notice that this energy component is largely reduced by FOS thanks to the number of slices that are turned off dynamically at runtime. These savings are especially significant in `Mix4` or `Mix7`, where FOS improves leakage currents by 60% to 70% over conventional organizations. Nevertheless, when executing cache-hungry mixes like `Mix2` the whole pool of slices needs to be activated for performance, which means that FOS is correctly working. Of course, performance improvement is achieved at the cost of energy, which in this case matches the one consumed by *Shared-ELC* and *Shared-OPT* approaches.

Dynamic energy consumption (shown in the upper component of each bar) represents the smallest energy component. *NUCA-OPT* presents the best results, thanks to the reduced complexity of their cache modules. FOS implements cache modules with the same size, although its dynamic energy consumption is slightly higher than in *NUCA-OPT* because of the PTA overhead. Since tag lookups in the PTA structure are performed only for the active slices, however, the corresponding overhead reduces the dynamic energy consumption on average by a $4\times$ factor over *Shared-ELC* and *Shared-OPT* approaches. Moreover, when executing mixes with a low number of active slices like `Mix4` and `Mix7`, FOS achieves a dynamic energy consumption similar to that of *NUCA-OPT*.

Regarding the network energy, it can be observed that, on average, the energy consumption of optical interconnects is by $10\times$ higher than that of electrical links. This overhead translates to an overall energy consumption increase by 20% from *Shared-ELC* to *Shared-*

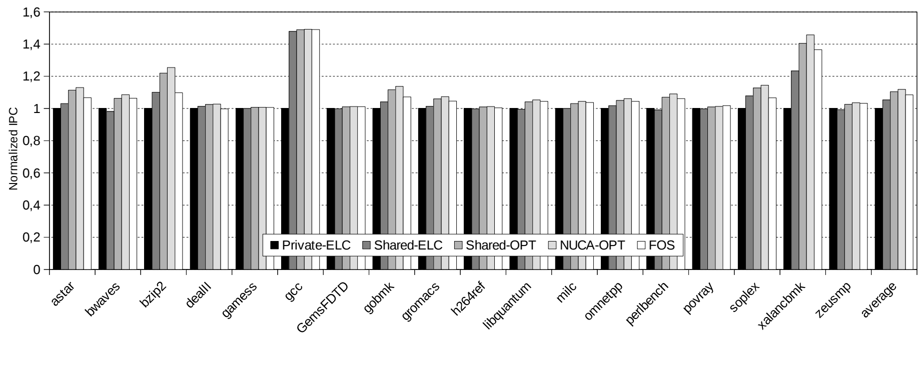

Fig. 9 Normalized performance of the individual applications.

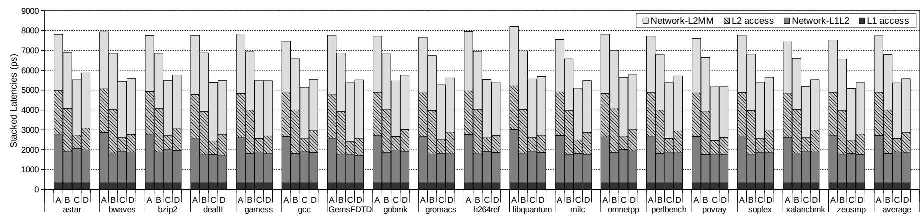

Fig. 10 Memory subsystem latencies broken down in four main categories. Legend: A: Shared-ELC, B: Shared-OPT, C: NUCA-OPT, D: FOS.

OPT. In the *NUCA-OPT*, the optical network overhead, together with the higher leakage values of 64KB modules, leads to an energy consumption by 35% higher than the *Shared-ELC* baseline. On the contrary, and despite experiencing also this network overhead, FOS presents an energy consumption by 23% on average lower than *Shared-ELC*, improving energy consumption over the electrically-connected approaches in seven out of nine mixes.

In summary, it can be observed that the energy savings provided by FOS coming from cache management nicely compensate the energy expenses coming from the optical network. These results show the high potential of the proposal, which is one of the main aims of this paper, and let us to conclude that FOS would provide higher energy savings while sustaining the performance if it was implemented with a low power NoC able to fulfill the latency requirements.

Lastly, Figure 8 plots the energy results for the 4-benchmark multiprogram workloads. As observed, FOS provides cache energy savings across most of the mixes. These savings come from both dynamic and static energy consumptions. Dynamic energy is highly improved, in percentage, compared to *Shared-ELC* and *Shared-OPT* approaches. Cache static energy is also significantly reduced in a wide set of mixes, being in some cases less than half the energy consumed by conventional approaches. Again, in cache intensive workloads like `Mix1`, FOS keeps an energy consumption similar as that exhibited by the *Shared-ELC* system, since as mentioned above, FOS activates slices for performance.

6.2 Performance Evaluation of Individual Applications

This section evaluates the FOS's performance considering each benchmark running alone in one of the cores of a 2-core system, while the other core is left empty. First, we analyze and compare the normalized performance (in terms of IPC) of FOS and the four studied schemes: *Private-ELC*, *Shared-ELC*, *Shared-OPT* and *NUCA-OPT*. Figure 9 shows the results normalized with respect to *Private-ELC*. As observed, in spite of the fact that the main goal of FOS is not related to achieve better performance, our approach improves performance, on average, over *Private-ELC* by 8% (i.e. by 3% higher than that of *Shared-ELC*). The schemes with optical network (i.e. *Shared-OPT* and *NUCA-OPT*) achieve the highest speedups, improving performance by 10% and 12% over the *Private-ELC* scheme.

To provide further insights on where performance improvements come from, we analyzed the memory subsystem latency, breaking this latency down into five main components: i) L1 cache access time, ii) L1 to L2 cache NoC latency, iii) L2 cache access time, iv) L2 cache to memory controller NoC latency, and v) memory controller latency to handle the main memory access. Figure 10 plots the results for the first four components, since the main memory subsystem is the same across all the studied approaches. Results are shown for the *Shared-ELC*, *Shared-OPT*, *NUCA-OPT* and *FOS* systems, which are referred to as *A*, *B*, *C* and *D*, respectively, in the figure. As observed, optical interconnects provide a $1000ps$ faster L1-L2 network latency, on average, than the electrical links employed by the *Shared-ELC* scheme. This latency reduction explains, at a first glance, why all the optical approaches outperform both *Private-ELC* and *Shared-ELC* systems in individual execution.

Latency differences across the studied systems can also rise, however, because of the different access times of the L2 cache modules. *NUCA-OPT* system divides its cache space in 64KB modules, while *Shared-ELC* and *Shared-OPT* implement a single 1024KB module, whose access time is $3\times$ higher[1]. On the other hand, the access time presented by FOS depends on the number of slices actually being allocated. For instance, in `xalancbmk`, a cache intensive application, the FOS memory subsystem latency is up to $1100ps$; however, in `h264ref`, a compute intensive application, this time drops down to $700ps$. Notice that the L2 latency of FOS is slightly higher than that of the *NUCA-OPT* system. This small increase is due to the sequential access to the tag and data arrays that FOS performs.

In summary, optical NoC and small L2 cache slices help improve performance for individual execution. In this regard, both the optical NUCA and FOS present similar performance.

6.3 Performance and Cache Space Management Evaluation for Multiprogram Workloads

In addition to the performance in isolated execution, this section presents results for 2- and 4-program mixes. Figure 11 shows the system performance (i.e. IPC harmonic mean) across the 2-program mixes. As expected, the *NUCA-OPT* approach is on average slightly the best performing approach, closely followed by *Shared-OPT* and FOS. Results, however, are not homogeneous across all the studied workloads since FOS has to face different scenarios regarding the cache behavior of each mix.

To provide deeper insights and identify these behaviors, we measured the MPKI of FOS on the allocated cache slices, and the MPKI of the L2 cache on the remaining schemes. Figure 12 presents the results for *Private*, *Shared*, *NUCA* and *FOS* systems, which are referred

[1] According to CACTI 6.5 cache simulator with a $32nm$ technology node.

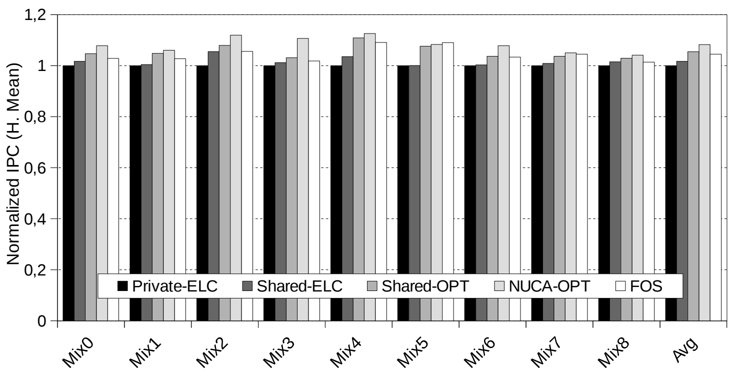

Fig. 11 Normalized performance of 2-benchmark mixes.

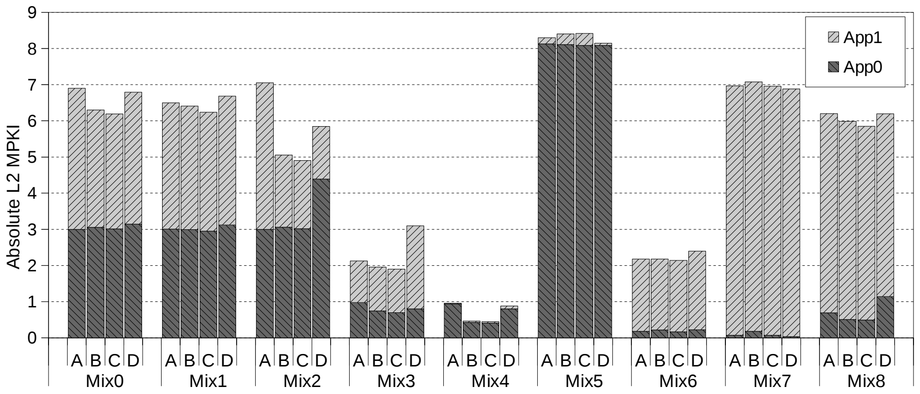

Fig. 12 MPKIs of 2-benchmark mixes for: A: Private, B: Shared, C: NUCA, D: FOS.

to as *A*, *B*, *C* and *D* respectively. The legends App 0 and App 1 refer to the order of the application in the mix as stated in Table 5. Since FOS trades off performance against cache space, we should consider MPKI jointly with the number of allocated slices in the analysis. Figure 13 shows the average number of slices that the SMM algorithm allocates to each application.

First, we analyze the cache behavior of each mix according to the categories previously identified in Section 2. Applications with *Minimum Slice Needs* (MSN) like `milc` or `libquantum` (i.e. App0 in `Mix5` and App1 in `Mix7`, see Table 5) do not present major differences in their MPKIs among the studied approaches due to their inherent low locality. The SMM algorithm realizes this fact and only allocates 3 and 2 slices to them on each mix. Secondly, applications with *Limited Slice Needs* (LSN), like `astar` in mixes 0 and 1 (App0) or `gcc` in `Mix1` (App1), are provided with 5 slices each and present MPKI values similar as the ones shown by conventional configurations. These values, however, are slightly higher than those with shared L2 caches since FOS has to progressively activate slices as they are predicted to be needed. Finally, applications included in the *Non-limited Slice Needs* (NSN), like `xalancbmk` in `Mix2` (App1) or `bzip2` in `Mix0` (App1), are provided with 10 and 7 slices on average and clearly present better MPKI values than the *Private* cache configuration.

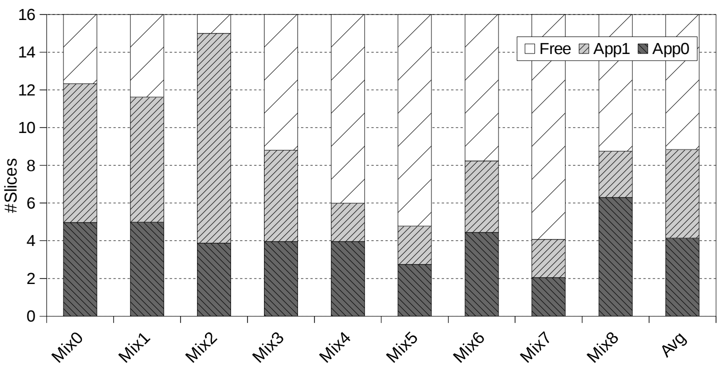

Fig. 13 Average number of slices assigned to each application in FOS.

These MPKI results match the slice sensitivity study carried out in Section 2. When executing multiprogram workloads, however, MPKI can increase due to the inter-application interference. Nevertheless, since FOS assigns slices as private to cores, the inter-application interference at the shared space is avoided. For instance, in `Mix5`, `h264ref` (App1) reduces its MPKI compared to both *Shared* and *NUCA* systems even though in this mix there are only 4.5 active slices on average during execution. In short, reducing the inter-application interference in the shared space allows FOS to improve performance over the *NUCA* approach in some mixes like `Mix5`. Notice that MSN applications with small working sets (like `h264ref`) are very sensitive to block replacements, hence preventing other applications from accessing their cache slices translates to relevant performance gains. This is the reason why `h264ref` does not present significant performance improvements in individual execution, but it does in `Mix5`.

The SMM algorithm covers a wide set of cache demanding scenarios, however, there are two *corner cases* where the SMM algorithm does not achieve the best behavior. Next, we discuss these cases and how they could be addressed. These cases appear due to two different situations: i) slices are assigned to applications in FIFO order, which means that an application with high cache needs may obtain a big amount of slices in the earlier stages of its execution while reducing the available space for the co-running application, and ii) irregular and fast-changing cache demands are hard to be identified with the experimental SMM thresholds. An example of the first case can be seen in `Mix2`, where `xalancbmk` (App0) slightly reduces the available space for `astar` from 5 slices to 4 and, due to this reason, the MPKI of the latter is higher in this mix than `Mix0` and `Mix1`. This behavior, however, can be prevented in FOS by adjusting the corresponding threshold to limit the maximum number of slices that an application can be assigned. An example of the second case is `gromacs` (App 1 in `Mix3`). It can be seen that, when executed in the FOS system, this application almost doubles the MPKI of the *Private* system, in spite of having by 45% of the cache space still available. A more aggressive setup of the SMM algorithm would limit this behavior but, in this work, only the setup that provides the highest energy efficiency is shown (see Section 6.1).

To conclude the analysis of 2-benchmark mixes, results have shown that, on average, FOS assigns only by 55% of the total cache space that the other evaluated approaches, with none or negligible performance losses. This cache space is dynamically distributed

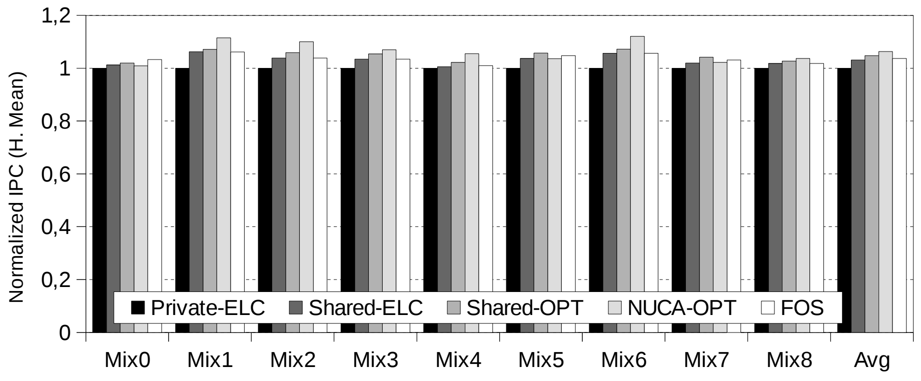

Fig. 14 Normalized Performance of 4-bench mixes.

according to application needs, which allows some applications to work with a smaller cache space.

Figure 14 presents the performance results for the 4-benchmark mixes. These mixes have been evaluated with 2MB cache storage capacity for all the evaluated cache approaches. Results show that, on average, FOS improves performance by 4% over the *Private-ELC* system, while *Shared-ELC*, *Shared-OPT* and *NUCA-OPT* improve by 4%, 5% and 6%, respectively. Performance differences are lower than in 2-benchmark mixes between electrical and optical approaches, mainly because optical rings present slightly higher contention. Performance, however, could be enhanced with an improved optical NoC, but this analysis is out of the scope of this paper. The performance improvements achieved by FOS are especially significant in mixes 0, 5 and 7 because in these mixes FOS prevents `milc` from interfering the other applications in the shared space. Moreover, despite turning off part of the cache space and making an efficient use of the cache space, FOS just presents by 2% performance degradation with respect to *NUCA-OPT*.

6.4 Putting it All Together: Energy Efficiency

To summarize the previously exposed performance and energy results and conclude the experimental evaluation, in this section it is discussed the overall energy efficiency of FOS with respect to conventional approaches. To this end, Figures 15 and 16 show the energy delay squared product (ED2P) for 2- and 4-benchmark mixes, respectively. It can be seen that, on average, FOS is the approach presenting the best results, reducing the ED2P in seven out of nine mixes both in 2- and 4-benchmark multiprogram workloads. Moreover, in some cases this reduction is over 60%. An interesting observation is that, according to the performance results presented below, the best performing approach (i.e. *NUCA-OPT*) shows the worst ED2P values in both figures, while the worst performing approach (i.e. *Private-ELC*) presents the second lowest ED2P values. This means that in the former case, *NUCA-OPT* increases performance at the expense of energy, mainly due to the optical NoC; while in the latter it occurs the opposite, that is, energy savings are achieved at the expense of performance.

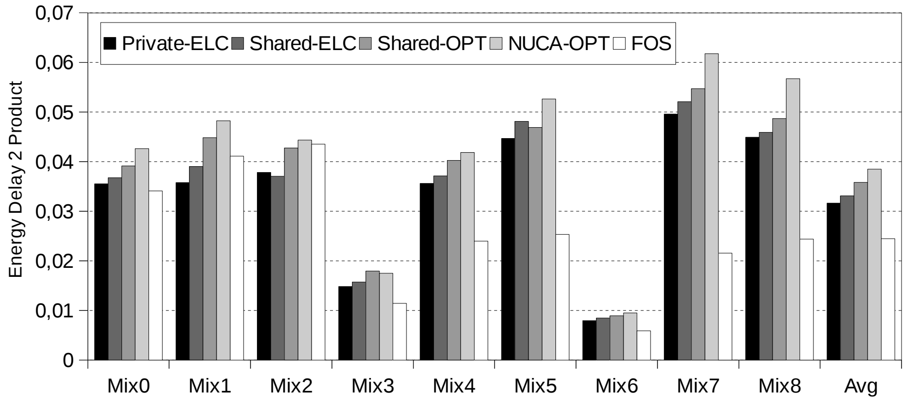

Fig. 15 Energy Delay Squared Product of 2-benchmark mixes.

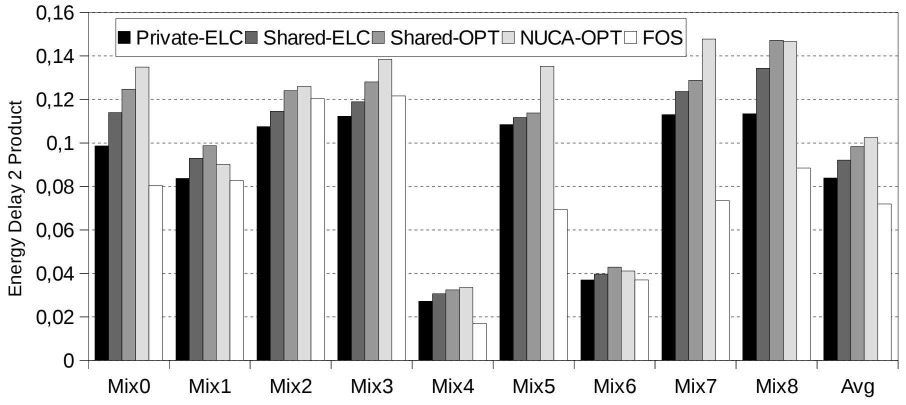

Fig. 16 Energy Delay Squared Product of 4-benchmark mixes.

FOS addresses these trade-offs showing that, in spite of presenting an energy consumption even lower than that of *Private-ELC* in most mixes, it achieves similar performance to *NUCA-OPT*, which translates to better ED2P results.

Finally, Table 6 summarizes a quantitative comparison of the energy, performance and energy efficiency values corresponding to all the studied approaches. Results show that FOS is the most energy-efficient approach, improving by 16.55% the energy efficiency with respect to *Private-ELC*. *NUCA-OPT* is shown as the less efficient scheme, since in spite of being the best performing approach, it prohibitively increases energy consumption by 27.61% over *Private-ELC*. Similarly, *Shared-ELC* and *Shared-OPT* configurations achieve marginal performance gains but increase the energy consumed, which also makes *Private-ELC* a more efficient organization than these approaches. This means that, among all the studied approaches, FOS is the only one that improves performance over the private baseline while also reducing the energy consumed.

Table 6 Energy efficiency, performance and energy savings achieved by each architecture normalized to the Private-ELC approach.

Architecture	Energy efficiency	Performance	Energy savings
Shared-ELC	−8.91%	3.33%	−14.42%
Shared-OPT	−14.8%	4.87%	−22.29%
NUCA-OPT	−18.24%	7.03%	−27.61%
FOS	16.55%	3.79%	8.45%

7 Related Work

Some seminal work [31, 33, 34] showed in the past that splitting the cache organization in small and independent cache structures does not only allows energy consumption to be reduced but also, when properly managed, to enhance the system performance. In this context, research approaches aimed at improving the utilization and efficiency of the cache hierarchy can be classified in two main categories depending on whether they apply to shared caches [23, 30] or to private caches [8, 18, 19].

Approaches belonging to the former category pursue to improve the system performance by properly assigning specific cache ways to the different applications running on the processor cores An interesting piece of research that falls in the first category is the work [30] by Qureshi and Patt, which partitions the cache ways of the LLC among the co-running applications depending on the reduction in cache misses that each application is likely to obtain for a given amount of cache resources. Other works also distributing cache ways of the LLC are [23, 29, 35], some of them focusing on improving performance while others focusing on improving system fairness [23, 35]. Regarding the second category, some approaches [8, 18] allow applications with high cache demands to borrow part of the private cache from the neighbor cores. Unlike our approach, both strategies highly restrict the maximum cache space that is available for a given core.

With the aim of reducing the latency and energy consumption of large caches, other approaches focus on Non Uniform Cache Access (NUCA) architectures [1, 5, 10, 11, 20]. In these approaches, the cache is organized as a set of interconnected banks. Conventional NUCA present interleaved access which suffer typical shortcomings associated to rigid cache hierarchies like multiple lookups and excessive data movement and thrashing. To deal with these issues some approaches [15, 17, 24] have concentrated on placement/migration approaches in such organization. Recent works like Jigsaw and Jenga [6, 39] propose an evolution of this organization by allocating specific banks to each core according to the cache needs of the running applications, hence the number of banks accessed by each core is not necessarily a power of two. The latter works are the most closely related to our proposal; however, these approaches present important differences: i) both Jigsaw and Jenga are hybrid software-hardware implemented; ii) allocated banks need to be located near the core which presents important constraints to the assignment algorithm; and iii) switching off strategies are not devised.

In summary, the aforementioned works focus on improving performance by carrying out a more efficient management of the available cache resources, but they do not switch off the *unneeded cache space* and therefore they do not leverage of the corresponding energy savings. Moreover, the majority of these approaches are limited by the rigid location of the cache banks, presenting a lack of flexibility.

Finally, similar to our approach, other works [4,12,43] make use of optical interconnects to enhance specific aspects of the CMP architecture that affect the cache access.

8 Conclusions

In this work, FOS has been presented as a novel cache organization and management approach to trade off performance for energy consumption. The FOS architecture replaces conventional low level (e.g. L2 and L3) caches with a single level consisting of a pool of cache slices, and introduces a novel management approach especially suited for low power processors. Slices are much larger (i.e. by a thousand times larger) than individual cache lines, which eases the implementation of practical energy-aware approaches working at this granularity. For the sake of exploring the impact of FOS on performance, it has been implemented and evaluated with an underlying Optical Network on Chip, since it provides a rather fast and uniform access latency to the slices. Other networks (not necessarily optical) might be used, whenever latency requirements are satisfied.

Experimental results have shown that FOS reduces dynamic energy consumption on average by a factor of up to $4\times$ over a shared cache organization with the same storage capacity. Energy savings on static energy consumption are also achieved, reaching a reduction by up to 60% of the total static energy over conventional approaches. As a result, FOS is the least energy consuming cache approach among all the studied cache organizations. Overall energy results also include the overhead introduced by the devised ONoC, which means that energy benefits can be even higher.

Moreover, FOS energy savings do not come at the expense of performance, since moderate performance improvements come from the network and the cache side. That allows FOS to achieve a similar performance to an optically connected NUCA cache, in spite of using, on average, by 50% of the cache space. In summary, we have shown that FOS is the most energy efficient approach, since it does not only achieves the highest energy savings but also sustains performance.

References

1. Awasthi, M., Sudan, K., Balasubramonian, R., Carter, J.: Dynamic hardware-assisted software-controlled page placement to manage capacity allocation and sharing within large caches. In: 2009 IEEE 15th International Symposium on High Performance Computer Architecture, pp. 250–261 (2009). DOI 10.1109/HPCA.2009.4798260
2. Baer, J., Low, D., Crowley, P., Sidhwaney, N.: Memory hierarchy design for a multiprocessor look-up engine. In: 12th International Conference on Parallel Architectures and Compilation Techniques (PACT 2003) (2003)
3. Bahirat, S., Pasricha, S.: Meteor: Hybrid photonic ring-mesh network-on-chip for multicore architectures. ACM Trans. Embed. Comput. Syst. **13**(3s), 116:1–116:33 (2014). DOI 10.1145/2567940
4. Bartolini, S., Grani, P.: A simple on-chip optical interconnection for improving performance of coherency traffic in cmps. In: 15th Euromicro Conference on Digital System Design, pp. 312–318 (2012). DOI 10.1109/DSD.2012.13
5. Beckmann, B.M., Marty, M.R., Wood, D.A.: Asr: Adaptive selective replication for cmp caches. In: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 39, pp. 443–454. IEEE Computer Society, Washington, DC, USA (2006). DOI 10.1109/MICRO.2006.10. URL `https://doi.org/10.1109/MICRO.2006.10`
6. Beckmann, N., Sanchez, D.: Jigsaw: Scalable software-defined caches. In: Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques, PACT '13, pp. 213–224. IEEE Press, Piscataway, NJ, USA (2013). URL `http://dl.acm.org/citation.cfm?id=2523721.2523752`
7. Bergman K. Carloni L. P. Bibermani, A.C., G., H.: Photonic Network-on-Chip Design, vol. 68 (2014)
8. Chang, J., Sohi, G.S.: Cooperative caching for chip multiprocessors. In: Procs. 33rd Annual Int. Symp. on Computer Arch., pp. 264–276 (2006). DOI 10.1109/ISCA.2006.17
9. Chen, G., Chen, H., Haurylau, M., Nelson, N., Fauchet, P.M., Friedman, E.G., Albonesi, D.: Predictions of cmos compatible on-chip optical interconnect. In: Procs. of Int. Workshop on System Level Interconnect Prediction, SLIP '05, pp. 13–20 (2005)

10. Chishti, Z., Powell, M.D., Vijaykumar, T.N.: Optimizing replication, communication, and capacity allocation in cmps. SIGARCH Comput. Archit. News **33**(2), 357–368 (2005). DOI 10.1145/1080695.1070001. URL `http://doi.acm.org/10.1145/1080695.1070001`
11. Cho, S., Jin, L.: Managing distributed, shared l2 caches through os-level page allocation. In: 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), pp. 455–468 (2006). DOI 10.1109/MICRO.2006.31
12. Cianchetti, M.J., Kerekes, J.C., Albonesi, D.H.: Phastlane: A rapid transit optical routing network. In: Procs. of the 36th Annual International Symposium on Computer Architecture, ISCA'09, pp. 441–450 (2009). DOI 10.1145/1555754.1555809
13. Demir, Y., Hardavellas, N.: Parka: Thermally insulated nanophotonic interconnects. NOCS '15, pp. 1:1–1:8 (2015). DOI 10.1145/2786572.2786597
14. Duan, G.H., Fedeli, J.M., Keyvaninia, S., Thomson, D.: 10 gb/s integrated tunable hybrid iii-v/si laser and silicon mach-zehnder modulator. In: European Conference and Exhibition on Optical Communication (2012). DOI 10.1364/ECEOC.2012.Tu.4.E.2
15. Dybdahl, H., Stenstrom, P.: An adaptive shared/private nuca cache partitioning scheme for chip multiprocessors. In: 2007 IEEE 13th International Symposium on High Performance Computer Architecture, pp. 2–12 (2007). DOI 10.1109/HPCA.2007.346180
16. García, A., Fernández, R., Garca, J.M., Bartolini, S.: Managing resources dynamically in hybrid photonic-electronic networks-on-chip. Concurrency and Computation: Practice and Experience **26**(15), 2530–2550 (2014). DOI 10.1002/cpe.3332
17. Hardavellas, N., Ferdman, M., Falsafi, B., Ailamaki, A.: Reactive nuca: Near-optimal block placement and replication in distributed caches. SIGARCH Comput. Archit. News **37**(3), 184–195 (2009). DOI 10.1145/1555815.1555779. URL `http://doi.acm.org/10.1145/1555815.1555779`
18. Herrero, E., González, J., Canal, R.: Distributed cooperative caching. In: Procs. of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT '08, pp. 134–143 (2008). DOI 10.1145/1454115.1454136
19. Herrero, E., González, J., Canal, R.: Elastic cooperative caching: An autonomous dynamically adaptive memory hierarchy for chip multiprocessors. In: Procs. of the 37th Annual International Symposium on Computer Architecture, ISCA '10, pp. 419–428 (2010). DOI 10.1145/1815961.1816018
20. Huh, J., Kim, C., Shafi, H., Zhang, L., Burger, D., Keckler, S.W.: A nuca substrate for flexible cmp cache sharing. In: Procs. of the 19th Annual International Conference on Supercomputing, ICS '05, pp. 31–40. ACM (2005). DOI 10.1145/1088149.1088154
21. Kahng, A.B., Li, B., Peh, L.S., Samadi, K.: Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In: DATE, pp. 423–428. European Design and Automation Association (2009)
22. Kaxiras, S., Hu, Z., Martonosi, M.: Cache decay: Exploiting generational behavior to reduce cache leakage power. In: Procs. of the 28th Annual International Symposium on Computer Architecture, ISCA'01, pp. 240–251 (2001)
23. Kim, S., Chandra, D., Solihin, D.: Fair cache sharing and partitioning in a chip multiprocessor architecture. In: PACT, pp. 111–122 (2004)
24. Merino, J., Puente, V., Gregorio, J.A.: Esp-nuca: A low-cost adaptive non-uniform cache architecture. In: HPCA - 16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture, pp. 1–10 (2010). DOI 10.1109/HPCA.2010.5416641
25. Morris, R., Kodi, A.K., Louri, A.: Dynamic reconfiguration of 3d photonic networks-on-chip for maximizing performance and improving fault tolerance. In: 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 282–293. DOI 10.1109/MICRO.2012.34
26. Muralimanohar, N., Balasubramonian, R., Jouppi, N.P.: Cacti 6.0: A tool to model large caches. In: HP Laboratories (2009)
27. Pang, J., Dwyer, C., Lebeck, A.R.: Exploiting emerging technologies for nanoscale photonic networks-on-chip. In: Procs. of 6th Int. Workshop on NoC Architectures, NoCArc '13, pp. 53–58
28. Petit, S., Sahuquillo, J., Such, J.M., Kaeli, D.R.: Exploiting temporal locality in drowsy cache policies. In: Proceedings of the Second Conference on Computing Frontiers, 2005, Ischia, Italy, May 4-6, 2005, pp. 371–377
29. Pons, L., Selfa, V., Sahuquillo, J., Petit, S., Pons, J.: Improving system turnaround time with intel CAT by identifying LLC critical applications. In: Euro-Par 2018: Parallel Processing - 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27-31, 2018, Proceedings, pp. 603–615 (2018). DOI 10.1007/978-3-319-96983-1_43. URL `https://doi.org/10.1007/978-3-319-96983-1\_43`
30. Qureshi, M., Patt, Y.: Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In: MICRO, pp. 423–432 (2006)

31. Rivers, J.A., Tam, E.S., Tyson, G.S., Davidson, E.S., Farrens, M.K.: Utilizing reuse information in data cache management. In: Proceedings of the 12th international conference on Supercomputing, ICS 1998, Melbourne, Australia, July 13-17, 1998, pp. 449–456 (1998). DOI 10.1145/277830.277941. URL `http://doi.acm.org/10.1145/277830.277941`
32. Rosenfeld, P., Cooper-Balis, E., Jacob, B.: Dramsim2: A cycle accurate memory system simulator. IEEE Comput. Archit. Lett. pp. 16–19. DOI 10.1109/L-CA.2011.4
33. Sahuquillo, J., Pont, A.: The filter cache: A run-time cache management approach1. In: 25th EUROMICRO '99 Conference, Informatics: Theory and Practice for the New Millenium, 8-10 September 1999, Milan, Italy, pp. 1424–1431 (1999). DOI 10.1109/EURMIC.1999.794504. URL `https://doi.org/10.1109/EURMIC.1999.794504`
34. Sahuquillo, J., Pont, A.: Splitting the data cache: a survey. IEEE Concurrency **8**(3), 30–35 (2000). DOI 10.1109/4434.865890. URL `https://doi.org/10.1109/4434.865890`
35. Selfa, V., Sahuquillo, J., Eeckhout, L., Petit, S., Gómez, M.E.: Application clustering policies to address system fairness with intel's cache allocation technology. In: 26th International Conference on Parallel Architectures and Compilation Techniques, PACT 2017, Portland, OR, USA, September 9-13, 2017, pp. 194–205 (2017). DOI 10.1109/PACT.2017.19. URL `https://doi.org/10.1109/PACT.2017.19`
36. Shacham, A., Bergman, K., Carloni, L.: On the design of a photonic network-on-chip. In: Networks-on-Chip, NOCS 2007, pp. 53–64
37. Soref, R., Bennett, B.: Electrooptical effects in silicon. IEEE Journal of Quantum Electronics **23**(1), 123–129 (1987). DOI 10.1109/JQE.1987.1073206
38. SPEC website. URL `http://www.spec.org/`
39. Tsai, P.A., Beckmann, N., Sanchez, D.: Jenga: Software-defined cache hierarchies. SIGARCH Comput. Archit. News **45**(2), 652–665 (2017). DOI 10.1145/3140659.3080214. URL `http://doi.acm.org/10.1145/3140659.3080214`
40. Ubal, R., Sahuquillo, J., Petit, S., Lopez, P.: Multi2sim: A simulation framework to evaluate multicore-multithreaded processors. In: Int. Symp. on Computer Architecture and High Performance Computing., pp. 62–68. DOI 10.1109/SBAC-PAD.2007.17
41. Valero, A., Sahuquillo, J., Petit, S., López, P., Duato, J.: Combining recency of information with selective random and a victim cache in last-level caches. ACM Trans. Archit. Code Optim. **9**(3), 16:1–16:20 (2012). DOI 10.1145/2355585.2355589. URL `http://doi.acm.org/10.1145/2355585.2355589`
42. Vantrease, D., Binkert, N., Schreiber, R., Lipasti, M.: Light speed arbitration and flow control for nanophotonic interconnects. In: Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium, pp. 304–315
43. Werner, S., Navaridas, J., Lujan, M.: Designing low-power, low-latency networks-on-chip by optimally combining electrical and optical links. 2017 IEEE Int. Symp. of High Performance Computer Architecture