DarkCache: Energy-performance Optimization of Tiled Multi-cores by Adaptively Power Gating LLC Banks

DAVIDE ZONI, Politecnico di Milano
LUCA COLOMBO, Politecnico di Milano
WILLIAM FORNACIARI, Politecnico di Milano

The Last Level Cache (LLC) is a key element to improve application performance in multi-cores. To handle the worst case, the main design trend employs tiled architectures with a large LLC organized in banks, which goes underutilized in several realistic scenarios. Our proposal, named *DarkCache*, aims at properly powering off such unused banks to optimize the Energy-Delay Product (EDP) through an adaptive cache reconfiguration, thus aggressively reducing the leakage energy. The implemented solution is general and it can recognize and skip the activation of the *DarkCache* policy for the few strong memory intensive applications that actually require the use of the entire LLC. The validation has been carried out on 16- and 64-core architectures also accounting for two state-of-the-art methodologies. Compared to the baseline solution, *DarkCache* exhibits a performance overhead within 2% and an average EDP improvement of 32.58% and 36.41% considering 16 and 64 cores, respectively. Moreover, *DarkCache* shows an average EDP gain between 16.15% (16 cores) and 21.05% (64 cores) compared to the best state-of-the-art we evaluated, and it confirms a good scalability since the gain improves with the size of the architecture.

CCS Concepts: • **Computer systems organization** → **Multicore architectures**; • **Networks** → *Network on chip*;

Additional Key Words and Phrases: Last Level Cachey, Power Gating, Energy-Performance

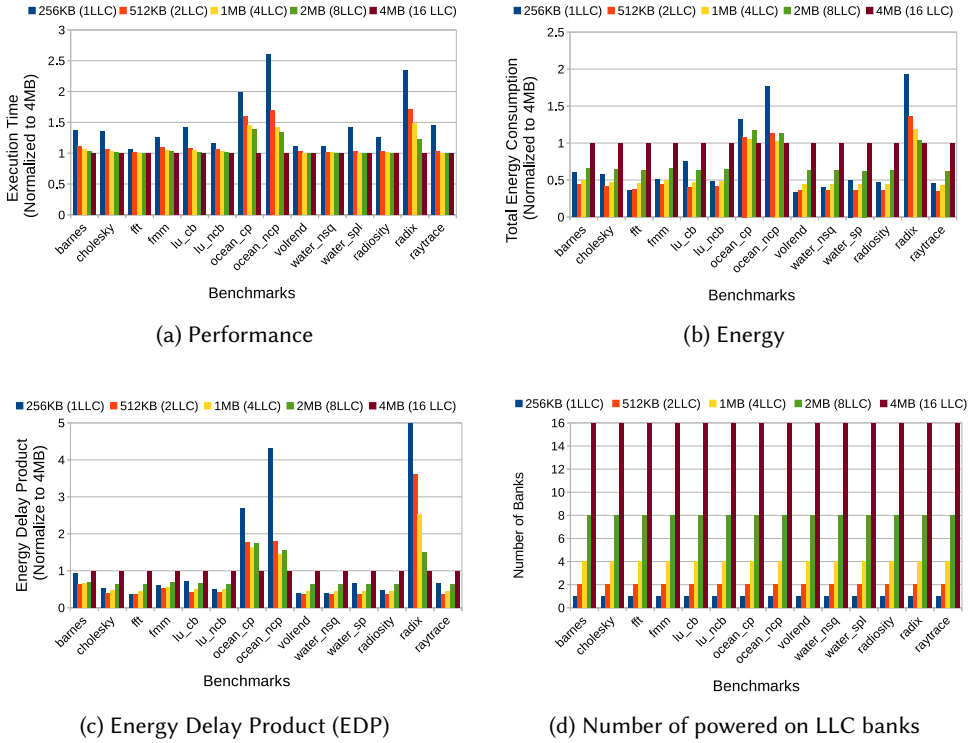
ACM Reference format:
Davide Zoni, Luca Colombo, and William Fornaciari. 2017. DarkCache: Energy-performance Optimization of Tiled Multi-cores by Adaptively Power Gating LLC Banks. *ACM Transactions on Architecture and Code Optimization* 9, 4, Article 39 (March 2017), 25 pages.
https://doi.org/0000001.0000001

1 INTRODUCTION

The steady increase in the number of CPUs in multi-cores poses a severe pressure on the memory hierarchy which is in charge of serving all the requests coming from the concurrently executed applications [5]. In this scenario, the Last Level Cache (LLC) can improve the application performance by reducing the number of time-consuming accesses to the main memory. However, the independent data access streams of the multiple applications severely affect the data locality and, consequently, the effectiveness of the LLC. This fact motivates a constant increase of the LLC size

This work was partially supported by two grants within the EU H2020 Research and Innovation Programme: "MANGO" Grant agreement no. 671668 and "M^{2}DC" Grant agreement no. 688201. Author's addresses: D. Zoni, L. Colombo and W. Fornaciari, DEIB, Politecnico di Milano, Via Ponzio 34/5, 20133 Milano, Italy.

XXXX-XXXX/2017/3-ART39 $15.00
https://doi.org/0000001.0000001



(a) Performance

(b) Energy

(c) Energy Delay Product (EDP)

(d) Number of powered on LLC banks

Fig. 1. Performance (a), Energy (b) and EDP (c) as a function of the number of the powered on LLC banks that are shown in (d). Each benchmark is simulated with a different LLC size that is kept constant for the entire simulation. Results are normalized to the 4MB (16 LLC) architecture and are collected using the *gem5* cycle accurate simulator, considering a tiled 16-core architecture implementing a four by four 2D-mesh on-chip network and the inclusive MESI cache coherence protocol.

to accommodate more data, unfortunately with two main drawbacks: *i)* the techniques to efficiently use the LLC space become increasingly complex and *ii)* the energy consumption, dominated by the leakage, constantly increases [4, 19]. The *dynamic cache reconfiguration* architectures emerged as a viable solution to adaptively power off the underutilized portion of the LLC, thus minimizing the leakage energy. Such architectures take steps from the cache partitioning schemes that are used to virtually partition the physical LLC space between multiple competing applications to avoid mutual interferences. Several solutions targeting both multi-cores [25] and CPU-GPU systems [26] were proposed in place of the Least Recently Used (LRU) policy that cannot prevent a memory-bound application from trashing the data of another less demanding tasks [2]. The typical *cache reconfiguration* schemes that have been proposed for performance optimization lead to a frequent LLC underutilization [13]. This is due to the combination of an always increasing LLC size and to the fact that several realistic scenarios exhibit either a number of running applications that is lower than the number of CPUs, or a set of applications with limited memory requirements. The combination of LLC under-utilization and low energy requirements motivates the use of power gating actuators in different *cache partitioning* proposals to power off the unused portion of the cache [9, 13, 18, 24, 30]. However, the available literature on dynamic cache reconfiguration is

limited to monolithic LLC architectures where at most a portion of the single LLC bank is turned off, thus making them not readily applied in tiled-based multi-cores.

The tiled multi-core represents a de-facto solution to ensure scalability and to cope with the time to market, which imposes to simplify the chip manufacturing stage. A tile is optimized, both from the architectural and the geometry viewpoints, before being replicated to create the final multi-core. Each tile contains one or more computing elements, the logic to interconnect the tile to the rest of the system and a slice of the LLC. Such slice is physically split in banks (one per tile) while implementing a unique and shared address space for all the running applications. In tiled multi-cores, the single LLC bank is smaller than the monolithic LLC designs, thus calling for novel dynamic cache reconfiguration strategies capable of adaptively powering off and on at bank granularity to ensure valuable energy savings. Tiled multi-cores enforce a Non Uniform Cache Access (NUCA) time, i.e., the time to access a cache line depends on the physical location of the accessed cache bank. The NUCA architecture is classified either as static (SNUCA) or dynamic (DNUCA) according to the mapping policy between the address of the cache line and the cache bank. SNUCA maps each address to a specific cache bank using a deterministic policy, while the mapping between an address and the cache bank can change over time in DNUCA [1]. Figure 1 depicts performance (Figure 1a), energy (Figure 1b) and Energy-Delay-Product (EDP) (Figure 1c) for the complete set of *Splash2x* benchmarks (part of the Parsec3 suite [32]), in a 16-core processor with a four by four 2D-mesh on-chip network and an LLC made of 16 banks featuring 256KB of cache each. Each benchmark is simulated several times with a different number of powered on LLC banks that remains constant for the entire simulation (see Figure 1d) to mimic a power gating capable architecture. The results show the positive impact on the energy consumption due to the reduction of the LLC size. Conversely, the performance strongly depends on the considered application, thus imposing the design of a dynamic power gating-capable cache reconfiguration scheme to effectively manage the energy-performance trade-off. Indeed, the cache underutilization is observed for the majority of the considered applications (11 out of 14), for which no significant performance loss is reported while the LLC is bigger or equal to 1MB. Only few applications, namely *ocean-cp*, *ocean-ncp* and *radix*, show a significant performance loss due to the LLC size reduction. Moreover, the highest energy consumption is associated with the smallest LLC size due to the exponential increase in the execution time. The results reveal a strong memory-bound behavior, which, in turn, demands a bigger LLC. These applications cannot benefit from the use of a dynamic LLC resize scheme that would severely affect both the performance and the energy consumption.

This paper presents *DarkCache*, a dynamic cache reconfiguration approach encompassing tiled and non-tiled multi-cores. *DarkCache* optimizes the system-wide Energy Delay Product (EDP) by dynamically optimizing the static energy of the LLC, also considering the application performance. *DarkCache* splits the LLC in two partitions: *i)* the powered on LLC banks and *ii)* the powered off LLC banks. By exploiting a power gating policy, *DarkCache* dynamically reshapes the partitions at the LLC bank granularity to minimize the LLC leakage energy. Moreover, *DarkCache* can dynamically switch between the two implemented operating modes: *normal* and *energy saving* modes. The *normal* mode disables all the optimizations within *DarkCache*, leaving the partition of the powered off LLC banks empty. It also exploits a virtualized Static Non Uniform Cache Access (SNUCA) architecture to map the data to the LLC banks. This mode is employed to optimize the EDP for the applications that fully use the available LLC space (close to 20% of the applications in the analyzed benchmark suite). Conversely, the *energy saving* mode allows to reshape the LLC size by dynamically monitoring the actual memory requirements. In particular, *DarkCache* improves the state-of-the-art by introducing a dynamic LLC reconfiguration scheme that works at bank granularity, and that is able to optimize the system-wide EDP also enforcing three properties:

- **Wide Applicability** - *DarkCache* proposes a system-oriented cache reshaping scheme that always identifies two partitions regardless of the number of the running applications or threads. The main advantages are: *i)* the management of the partitions is greatly simplified and *ii)* the architecture scales up regardless of the number of running applications. *DarkCache* makes no assumptions on the running applications. Since the partitions are reshaped depending only on the actual LLC pressure, *DarkCache* neither requires any a-priori off-line profiling nor application-depended parameters to be used at run-time.
- **Comprehensive solution for tiled and monolithic LLC architectures** - *DarkCache* is neither limited to a specific coherence protocol nor to a specific on-chip interconnect and suits both monolithic and tiled LLC architectures. The optimization of the EDP metric accounts for the LLC, the main memory, the on-chip interconnect and the CPU. As a representative use case, *DarkCache* is implemented in the inclusive MESI-based cache coherence protocol within a tiled multi-core architecture, considering both 16 and 64 cores.
- **Scalable and partially non-blocking infrastructure** - *DarkCache* implements a *partially* non-blocking reconfiguration scheme that allows to dynamically resize the LLC without freezing the running applications, thus minimizing the performance penalties. During the reconfiguration, each CPU continues to execute the application and it stalls only if an L1 miss occurs. Moreover, *DarkCache* shows an EDP improvement over the state-of-the-art which increases with the number of cores.

The rest of this paper is organized in five parts. Section 2 discusses the state-of-the-art on the cache partitioning and reconfiguration schemes. Section 3 overviews *DarkCache* while the in depth presentation of the architecture and the policy are confined in Section 4 and Section 5. The results are detailed in in Section 6 and, finally, Section 7 draws the main conclusions.

2 RELATED WORKS

Cache partitioning and reconfiguration have been extensively studied for multi-cores to efficiently use the LLC resources that are shared between multiple competing applications. To provide the big picture of the shift towards dynamic cache reconfiguration for tiled multi-cores, this section summarizes the cache partitioning schemes by clustering them in two groups: partitioning for performance and partitioning for energy efficiency.

Performance-aware Cache Partitioning Schemes - The performance-aware cache partitioning schemes represent a key research branch focused on the optimal allocation of the LLC resources. In particular, it optimizes the system-wide performance by minimizing the cross interferences between the running applications. In contrast to *DarkCache* which optimizes the Energy Delay Product (EDP), in these solutions the information of the LLC underutilization is not directly accounted, thus no static energy saving mechanisms are usually considered. [17] proposed a utility-based cache partitioning scheme to improve the performance of monolithic LLC architectures, i.e., a single LLC bank is shared among all the cores. Each running application enforces the creation of a new LLC partition and each partition in the system is dynamically reshaped according to the utility curves, one per application, that are continuously updated using the monitored architectural statistics. Different cache replacement policies have been proposed to minimize the cache contention between concurrently executed threads [7, 28]. Conversely, [29] identifies the running threads that induce data trash in the LLC and isolates them in a dedicated cache partition. The use of a per thread priority level has been presented in different schemes to offer a quality of service layer [12, 20]. [20] leverages highly associative caches and creates an additional unmanaged partitions side by side with all the other per executing thread partitions. A victim cache line is moved to the unmanaged partition before being actually flushed from the cache. The unmanaged partition acts as a victim

cache and its cache lines are monitored to decide if the corresponding thread partition has to be reshaped. *PriSM* [12] proposes a cache partitioning scheme that partitions the entire cache at a fine granularity; a critical advantage offered by *PriSM* is the possibility, for each cache set, to independently control the cache occupancy of each competing thread.

Energy-aware Cache Partitioning and Reconfiguration Schemes - [13] leverages the LLC underutilization by proposing two step methodology for the energy-aware LLC reconfiguration: *i)* the LLC partitions are evaluated and enforced and *ii)* the leakage mechanism turns off the unused portions of the cache. Different cache partitioning techniques address the energy saving in the LLC by turning off the unused cache space at the granularity of cache colors [14, 31], ways [9, 24] or sets [18, 30]. Several proposals in the literature address both dynamic [23, 24] and static [13, 23, 24] energy saving. [24] enforces a single owner for each cache way to reduce the dynamic energy consumption, since at each access the requester looks up in the owned cache lines only. A migration mechanism is used to guarantee the single ownership of the cache lines while an additional per core hardware component tracks the owned cache lines. A similar cache partitioning scheme is proposed in [23]. *RECAP* [23] partitions the cache lines between the private and the shared ones without enforcing a migration mechanism to ensure a single owner per cache line. The mechanism reduces the dynamic energy due to the migration mechanism compared to [24]. Conversely, the unused cache lines are turned off as in [24]. [13] proposes an energy-aware cache partitioning scheme that specifically addresses the leakage energy in the LLC by powering off the unused cache portions. *DarkCache* also leverages the LLC underutilization with two critical differences with respect to [13]. First, *DarkCache* addresses the EDP metric thus reducing the LLC energy and ensuring minimal performance impact. Second, *DarkCache* works at LLC bank granularity since it is suitable for tiled LLC architectures while [13] fits monolithic LLCs. Despite the variety of proposed solutions, *DarkCache* addresses the EDP optimization for tiled LLC architectures, i.e., multiple physical LLC banks form the shared address space. In contrast to *DarkCache* which enables a system-wide LLC optimization even for tiled multi-cores, current state-of-the-art techniques require to be applied to each LLC bank in isolation. This produces two main design drawbacks. First, the LLC reconfiguration can be sub-optimal; second, each LLC requires dedicated statistics to compute its own optimization, thus severely affecting the scalability of the design. A detailed evaluation that compares these methodologies [17, 24] with *DarkCache* is provided in Section 6.

3 DARK CACHE AT A GLANCE

DarkCache is a dynamic cache reconfiguration scheme used to optimize the system-wide Energy Delay Product (EDP) in tiled multi-cores. It dynamically powers off the unused LLC banks also satisfying two requirements: *i)* monitoring of the LLC demand to timely satisfy the application needs, and *ii)* balancing of the benefit offered by a dynamic LLC resizing scheme against the performance and energy overheads due to the required management infrastructure.

Reference multi-core architecture - Without lack of generality, the rest of the manuscript introduces *DarkCache* considering a tiled multi-core that implements an inclusive, two-level cache hierarchy and an on-chip network. The on-chip network permits out of order message delivery, thus imposing the most challenging scenario for the design of the coherence protocol [6]. The cache hierarchy features a private L1 cache and a Last Level Cache (LLC) that is shared between the L1s but physically split in multiple banks. The LLC controllers ensure the coherence between the L1 cache and the LLC banks while the directory implemented at the memory controller level ensures coherence between the memory controller and the LLC banks [21]. *DarkCache* leverages the presence bit information within the memory controller's directory to implement its architecture. Depending on the actually implemented coherence protocol, the information corresponding to

the presence bit for a cache line can be spread across several cache coherence states and can be complemented by other information, e.g., the dirtiness and the ownership bits. However, such baseline information is vital for any coherence protocol since it tells if a copy of the corresponding cache line is in the LLC. Considering the two-level, inclusive, MESI implementation within the *gem5* cycle accurate simulator [3], the directory stores the coherence information, including the presence bit, when a copy of the cache line is in the LLC. The reference multi-core leverages the information of the presence bit to avoid data races. For example, the presence bit prevents uncoherent behavior when the LLC bank dumps a dirty cache line and, right after, it fetches the line again. Due to the out of order delivery property of the on-chip interconnect, the memory controller can observe the two coherence messages, i.e. *put* and $fetch$, in any time order. We note that the fetch request can be immediately processed if the presence bit of the corresponding cache line is not set. Otherwise, the fetch request is stalled, since another incoming message is expected to provide the most up-to-date copy of the data, e.g., a *put* message in the above example. From the energy and performance viewpoints, we assume an on-chip memory controller for which the actual memory access is only executed after the directory look up, i.e., only in the case of an actual data fetch or writeback.

***DarkCache* philosophy -** *DarkCache* elaborates on the system-wide architectural statistics to constantly monitor the memory requirements of the applications, and to reshape the LLC size or toggling between two operating modes: *normal* and *energy saving*. In the *normal* mode, *DarkCache* operates as the reference multi-core: *i)* the entire LLC is powered on and *ii)* all the mechanisms to reshape the LLC are not in use to minimize the performance overhead. Conversely, *DarkCache* enters into the *energy saving* mode when the system-wide statistics highlight a certain level of LLC underutilization. In this operating mode, to minimize the EDP, *DarkCache* exploits the implemented power gating mechanism to dynamically switch on and off the LLC banks.

Within *DarkCache*, the LLC is organized in two power partitions (*on* and *off*) regardless of the number of executing applications. Each LLC bank falls in one of the two partitions and it is dynamically toggled from one to the other when *DarkCache* operates in the *energy saving* mode, while all the LLC banks stay in the *on* partition when *DarkCache* operates in *normal* mode. It is worth noticing that *DarkCache* operates on the LLC memory which is a critical source of leakage; thus the cache controllers are supposed to be always active.

DarkCache is made of two parts: *i)* the mechanisms (see Section 4), namely the *multicast* and the *handshake*, that extend the baseline coherence protocol, and *ii)* the *policy* (see Section 5) which defines the logic to reshape the LLC size or to switch between the two operating mode. The *multicast mechanism* operates between an L1 and the powered on LLC banks. It is triggered by an L1 miss to retrieve the requested data in the LLC, if available, before fetching it from memory. The *handshake mechanism* (see Section 4.2) contains the coherence extensions to support the system reconfiguration in terms of switching between the two operating modes and the resizing of the LLC when operating in the *energy saving* mode. The *policy* periodically monitors some selected architectural statistics (see Section 5) to generate a power command that triggers a change of either the *LLC State* or the operating mode. The *pwrOff* and *pwrOn* commands are used to power off and on the LLCs when *DarkCache* operates in *energy saving* mode, while the *pwrSav* and *pwrNrm* power commands are used to change to *energy saving* and *normal* modes, respectively.

4 DARK CACHE ARCHITECTURE

The coherence protocol has been extended to support both the dynamic LLC reshape and the switch between the two operating modes. The *DarkCache architecture* is designed to deliver an architectural reconfiguration layer that is transparent to the applications. When *DarkCache* operates in *energy saving* mode, the *multicast mechanism* (see Section 4.1) allows to retrieve data from the

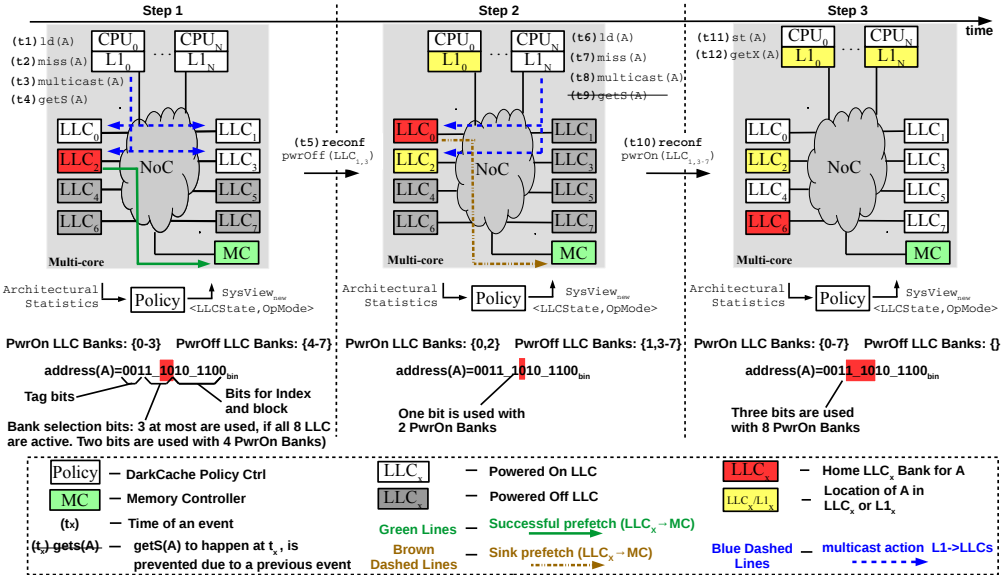


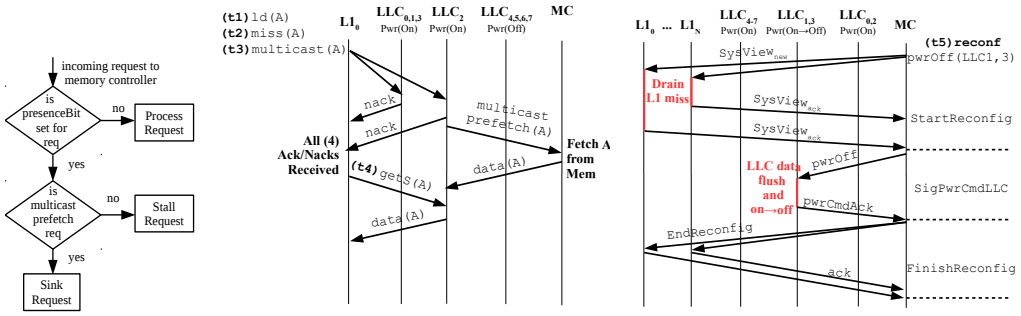
Fig. 2. Timing evolution of *DarkCache* on a multi-core made of 8 LLC banks and N CPUs with private L1 cache. The policy imposes two *LLC State* reconfigurations at time ($t5$) and ($t10$), thus enforcing both a DNUCA architecture. Three relevant steps are highlighted: *Step 1*, *Step 2* and *Step 3*. Each step shows the sequence of coherent requests due to a load or store instruction for A, i.e., $ld(A)$ or $st(A)$, respectively. In *Step 1* A is not present in both *LLCs* and $L1_0$. In *Step 2* data in not present in $L1_N$ but it is available in LLC_2. In *Step 3*, A is available in both $L1_0$ and LLC_2 where `st(A)` imposes an A permission upgrade in $L1_0$.

LLC preventing uncoherent behaviors. The *handshake mechanism* (see Section 4.2) defines the coherence extensions to both reconfigure the LLC and to switch between the operating modes.

4.1 Multicast Mechanism

The *energy saving* mode selectively powers off and on the LLC banks to minimize the EDP, thus enforcing a Dynamic Non Uniform Cache Access (DNUCA) mapping of the addresses onto the active LLC banks. Considering a DNUCA architecture, the same address can be mapped onto different LLC banks at different points in time. Traditionally, DNUCA architectures implement a broadcast mechanism to retrieve a cache line after a cache miss in the upper cache level [11]. Differently, *DarkCache* proposes a multicast mechanism at the L1 level to reduce both the performance penalty and the additional on-chip traffic. It is worth noticing that we define the term broadcast as the action of sending a message to all the implemented LLC banks. In contrast, the term multicast is defined as the action of sending a message to all the active, i.e., powered on, LLC banks that, in general, are a subset of the implemented ones.

In case of an L1 miss, *DarkCache* triggers a multicast action to look up the powered on LLCs for the requested data. In case the requested data is not present in the powered on LLCs, a unicast coherence request is generated according to the coherence protocol of the reference architecture. To further reduce the negative impact of the multicast mechanism, each cache line in the L1 is augmented with the *location bits*, i.e., $log_2(\#LLC)$ bits where $\#LLC$ identifies the number of implemented LLC banks. The *location bits* are kept updated until the cache line is valid in the LLC. They are also used to prevent a multicast action when the cache line is present in the L1 but an access to the LLC bank is required, e.g., data dump or permission upgrade. We note that each L1



(a) MC algorithm (b) Multicast: Step 1, *(t3) multicast* (c) Handshake: *Step* 1 → 2, *(t5) reconf*

Fig. 3. *DarkCache* details: (a) the memory controller algorithm to manage multicast prefetch requests, (b) multicast sequence of messages related to time events (t1-t4) in Figure 2, and (c) handshake sequence of messages corresponding to time event (t5) in Figure 2.

stores the actual operating mode and the current *LLC State* to correctly generate the multicast requests.

Figure 2 shows a three-step evolution of a *DarkCache* system while serving few coherence requests for data A and two LLC reconfigurations. The platform implements a memory controller and eight LLC banks, which means that at most three bits of the requested address are used to select the LLC home bank. Without lack of generality, the addresses are made of 12 bits, i.e., 0 to 11, and bits from 6 to 8 are used to identify the LLC home bank for each address. The destinations of the multicast actions are limited to the powered on LLC banks to minimize the additional on-chip traffic (see the blue dashed lines in Figure 2). At `Step 1` two bits of $address(A)$ are used to set the home bank for A, i.e., LLC_2, among the 4 powered on LLCs. The *LLC State* reconfiguration at time $(t5)$ powers off all the LLCs but LLC_0 and LLC_2. A single bit of $address(A)$ is enough to assign the new home bank for A, i.e., LLC_0, in *Step 2*. The multicast action at time $(t8)$ is restricted to the two powered on LLCs. This is essential to preserve the coherent state of the system, since a copy of A is already present in LLC_2 while a direct coherent request, e.g. $getS(A)$ at time $(t9)$, would address LLC_0, i.e., the home bank for A in the current *LLC State*. A second *LLC State* reconfiguration at time $(t10)$ powers on all the LLC banks (see *Step 3* in Figure 2). LLC_6 becomes the home bank for A since three bits of $address(A)$ are used to possibly addressing the 8 powered on LLC banks. However, both $L1_0$ and $L1_N$ have a copy of A, so that any access to the LLC from the L1 involving A will leverage the already set location bits and prevent an expensive multicast action, e.g., a unicast $getX(A)$ to LLC_2 at time $(t12)$ to acquire write permission on A.

The LLC prefetch scheme is part of the multicast mechanism and speculatively fetches data from the memory to the LLC to increase the overall system performance. Considering the current *LLC State*, any multicast action sets a prefetch bit in the message destined to the LLC home bank for the specific request. The LCC home bank receives the multicast request with the prefetch bit set and speculatively fetches from memory, i.e., it sends a *multicast prefetch* request to the memory controller, if the requested data is not present. Note that the LLC home bank sends the *multicast prefetch* request even if another LLC bank has a copy of the requested data. In this scenario, the memory controller can observe both *multicast prefetch* and standard coherence requests that are eventually out of order delivered. *DarkCache* enhances the memory controller with the algorithm depicted in Figure 3a to prevent a non coherent behavior in case of a *multicast prefetch* request

for which the requested data is already present in the LLC. In particular, the memory controller processes the incoming request if the corresponding presence bit in the directory is not set. This means that the cache line corresponding to the address in the incoming request is not present in any LLC bank. Conversely, the memory controller sinks a *multicast prefetch* request for which the directory controller reports a valid entry. A copy of the requested data is in fact already in the LLC and the request is due to a speculative *multicast prefetch*. Finally, the memory controller stalls the processing of the incoming request if the presence bit is set and if it is not a *multicast prefetch* request. This latter scenario is originated by the out of order deliver property of the on-chip interconnect and it is possible even in the reference multi-core architecture and at least another coherence message will sooner or later reach the memory controller for the same address to unblock the stalled coherence request. Figure 2 shows two *multicast prefetch* requests, i.e., during *Step 1* and during *Step 2*. The one in *Step 1* successfully prefetches A to LLC_2, since A is present in the LLC. Conversely, the memory controller correctly sinks the *multicast prefetch* request from LLC_0 during *Step 2*, since a copy of the data is already present in LLC_2.

Figure 3b reports the detailed sequence of coherence messages due to the multicast action at time ($t3$) in Figure 2. $L1_0$ sends a multicast message only to $LLC_{0,1,2,3}$, since $LLC_{4,5,6,7}$ are powered off. A is not present in any LLC bank, thus, once all the *nack* messages are collected, $L1_0$ sends a unicast $getS(A)$ request to LLC_2, that is the LLC home bank for A in the current *LLC State* (see (t4) $getS(A)$ in both Figure 3b and Figure 2). We note that LLC_2, as its incoming multicast request has the home bank bit set, starts a *multicast prefetch* request immediately after sending the *nack* to $L1_0$. The memory controller (MC) satisfies the multicast prefetch request, since by executing the algorithm in Figure 3a, and discovers that no copy of A is available in the LLC. To this extent, LLC_2 can timely answer to the incoming $getS(A)$ request (see ($t4$) $getS(A)$ in Figure 3b).

4.2 Handshake Architecture

The *Handshake Mechanism* (HM) manages the LLC reshape and the change of the operating mode according to the *policy* decisions. The *HM* reconfiguration protocol is organized in three separate stages: *StartReconfig*, *SigPwrCmdLLC* and *FinishReconfig*. We note that the reconfiguration protocol is designed to minimize the performance and energy penalties by implementing a protocol level deadlock-free scheme that avoids freezing the execution of the applications during the reconfiguration until the latter experiences an L1 miss. Without lack of generality, *DarkCache* implements the *HM* controller in the memory controller, even though it can be implemented in any part of the processor that provides access to the on-chip interconnect to send and receive information.

***StartReconfig* stage -** The *HM* informs all the L1 controllers of the upcoming reconfiguration by sending a $SysView_{new}$ token defined as:

$$SysView_{new} :=< LLCState_{new}, OperatingMode_{new} >$$

where $LLCState_{new}$ and $OperatingMode_{new}$ describe the final LLC configuration and operating mode. The $SysView_{new}$ represents the decision from the policy (see the output from the *Policy* block in Figure 2). The L1 controller stores the $SysView_{new}$ token and takes two separate actions: *i)* wait for the completion of all the cache coherence transactions started because of an L1 miss before acknowledging the *HM* controller, and *ii)* block any subsequent L1 miss until the end of the reconfiguration. It is worth noticing that all the other types of coherent requests are allowed during the reconfiguration. For example, any CPU request that resolves in the L1 with a hit, allows the CPU to continue the execution. To this extent, the *HM* protocol is defined as partially non-blocking, since it permits the execution of the application of the CPU as long as no L1 misses appear.

A global synchronization point is enforced in the *handshake* controller when the last acknowledged message from the L1 controllers is received, since no more transactions started because of an L1 miss, i.e., multicast transactions, are active in the system. This is an essential condition to avoid protocol level deadlock scenarios during the LLC reconfiguration. For an inclusive cache coherence architecture, in fact, the L1 miss can trigger a memory fetch at LLC level for which, during the reconfiguration, the mapping in the LLC bank can be unstable or undefined.

***SigPwrCmdLLC* stage -** The *HW* sends a *PwrCmd* token ($\in \{pwrOn, pwrOff, modeNrm, modeSav\}$) to those LLC banks that have to acknowledge it. In particular, the *PwrCmd* containing the new operating mode is sent to all the LLC controllers if *DarkCache* is changing the operating mode, otherwise it is sent only to the target LLC banks that have to change their power state. The *modeNrm* and *modeSav* token are used to signal a transition to the *normal* or to the *energy saving* mode, respectively. When *DarkCache* is operating in the *energy saving* mode, *pwrOn* and *pwrOff* commands are used to power on or off specific LLC banks.

The LLC that receives a *pwrOn* command triggers the power gating network for which the wake up latency of the associated cache memory is a technology parameter (see Section 6 for further details). Conversely, a *pwrOff* command forces the receiving LLC bank to coherently flush all the cache lines before turning off the cache memory of the bank. The *modeSav* signals the *energy saving* as the new operating mode to be used. All the LLC controllers receiving the *modeSav* command immediately acknowledge it. The reason for this is that they are already powered on and the current SNUCA mapping, i.e., the address to LLC banks mapping is the same of the baseline architecture, and it can be seen as a particular instance of a DNUCA. Conversely, the *modeNrm* signals the *normal* mode as the new operating mode to be used. Indeed, since the current mapping (DNUCA) can differ from the one used in the new operating mode (SNUCA), all the LLC controllers are forced to flush all the cache.

The *Handshake Mechanism* implements a partially non-blocking reconfiguration protocol that makes possible for an LLC to receive unicast L1 requests even when the LLC bank is under reconfiguration and when it is switching to the power off state. In this case, the L1 request arrives to the LLC that is flushing all the cache lines and an ad-hoc coherence protocol extension allows to correctly manage the L1 request without inducing either a protocol level deadlocks or starvation.

The coherence protocol extension is only required for handling the case of a powering off LLC bank that receives an L1 request. In particular even if such LLC bank has the required data, these data are going to be flushed and can change their LLC home bank before the end of the reconfiguration.

The LLC coherence protocol has been extended to cope with such a possibility by grouping in three classes all the possible combinations of *L1 request* and *state of powering off LLC*:

- **Present and stable -** L1 requests an address for which the corresponding data are stored in the LLC with a stable non-busy coherent state, which means that the LLC controller normally processes the incoming request. It is worth noticing that the LLC is powering off, so sooner or later after the invalidation of the copy in the requesting L1, the requested cache line will be evicted from the LLC.
- **Not present -** the address required by L1 corresponds to a data for which a copy is not present in the LLC because either the data has been already flushed, or it has been requested for the first time. Regardless of the actual scenario that induces an LLC miss, the powering off LLC cannot fetch any data from memory. This constraint is enforced to keep the coherence protocol simple and to ensure a finite time for the LLC flushing. The powering off LLC controller signals the requesting L1 a *retryFreeze* replay to prevent the L1 from sending any further request for the line, before the end of the reconfiguration. Indeed, the cache

line is not in the LLC anymore but a memory fetch to retrieve it can only be processed at the end of the reconfiguration, i.e., when the new home bank of the address will be stable.

- **Present but busy** L1 requests an address for which the corresponding data is present in the powering off LLC but has a busy coherent state. A busy state is a transient state from the coherence protocol viewpoint and it forces to wait until the cache line moves back to a stable one before taking any action. Thus, the LLC sends a *retry* message to the requesting L1, that then issues the request again to the same LLC. In particular, the busy state of the requested cache line in the LLC can be the result of two scenarios: *i)* the LLC is dumping the cache line, thus the next stable state will be `Not Present` and so the control falls back to an already discussed scenario, i.e., second item above, or *ii)* a previous L1 request requires some process on the target cache line for which the possible next stable state can be either `Not Present` or `Present and Stable`. In both cases the control falls back in a previously discussed class.

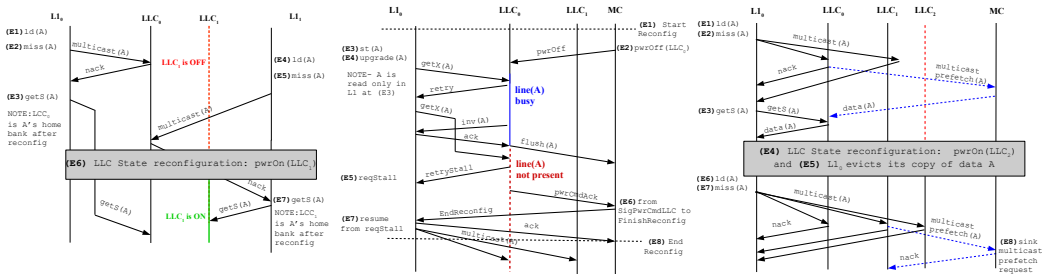
***FinishReconfig* stage -** The LLC acknowledges the *HM* controller once the flush of the cache lines is over and the cache memory is powered off or immediately if it is powering on. The *HM* enters the *FinishReconfig* once the last acknowledge from the reconfiguring LLCs is received and then it sends the *EndReconfig* message to all the L1. The L1 controllers will then start using either the new LLC state or the new operating mode and acknowledge the *HM* that, as a consequence, closes the reconfiguration process. The *FinishReconfig* stage is used to avoid the overlap between different reconfigurations.

Figure 3c reports the detailed sequence of coherence messages due to the *LLC State* reconfiguration at time ($t5$) in Figure 2. After having drained all the transactions started with a multicast action, the *MC* sends a $SysView_{new}$ message to all the L1 controllers that have to acknowledge it. The *HM* moves from *StartReconfig* to *SigPwrCmdLLC*, once the last $SysView_{ack}$ is received. A $pwrOff$ signal is sent to both LLC_1 and LLC_3 that has to power off. Note that all the other LLCs are not targeted by any power command. Both LLC_1 and LLC_3 acknowledge the *HM* after flushing their cache memory (see *pwrCmdAck* in Figure 3c). *HM* signals the end of the reconfiguration to all the L1s once all the expected *pwrCmdAck* messages are collected. Note that all the L1s have to acknowledge the $EndReconfig$ message to correctly close the reconfiguration protocol, thus preventing the time overlap of different reconfigurations.

4.3 Protocol Level Deadlock Analysis

All the cache coherence protocol level extensions and *DarkCache* itself have been designed and analyzed to tackle possible deadlock scenarios. To stress the coherent architecture, *DarkCache* has been tested against a large variety of synthetic traffic benchmarks coupled with random power on and power off policies, topology sizes between 2 and 16 LLC banks and LLC bank sizes of 128KB, 256KB and 1MB. A scoreboard coupled with a watchdog counter, whose limit is set to 1 million cycles, is implemented on top of the traffic generator to detect the possible deadlock occurrences; each test is run for 500 millions transactions.

During the reconfiguration of the *LLC State* and unless an L1 miss occurs, the CPUs continue the execution of the applications. Such relaxed constraints highlight three possible deadlock situations. It is worth noticing that *DarkCache* enforces a single active reconfiguration at once by means of the *FinishReconfig* stage (see Figure 3c). This prevents inconsistent *LLC States* or operating modes and it has been experimentally observed not to limit the system-wide performance. Indeed, at the beginning of the reconfiguration, the L1 controllers are informed of the final LLC State and operating mode. Moreover, as the final stage of the reconfiguration protocol, the *handshake* controller collects all the acknowledge messages from the L1s and LLCs. A reconfiguration that starts within another



(a) Deadlock handshake (b) Deadlock data dump (c) Deadlock multicast prefetch

Fig. 4. Handled deadlock scenarios within *DarkCache*. For each deadlock scenario only the relevant events, i.e., (E_i) where $i \in (1..8)$ are reported.

reconfiguration overwrites the $SysView_{new}$ token configured by the already active reconfiguration process. The reconfiguration actions can terminate out of order, thus delivering an inconsistent system state.

Global synchronization to drain all multicast transactions - At the beginning of the reconfiguration, each L1 is required to complete any already active transaction started because of an L1 miss. No more L1 misses are processed until the end of the reconfiguration protocol. *DarkCache* statically maps addresses to the LLCs depending on the actual *LLC State*, thus the same address can be mapped onto different LLC banks based on the *LLC State* before and after a reconfiguration. In this scenario, an L1 miss is the only event that can potentially trigger a memory fetch if the required data is not present in the LLC. However, during the reconfiguration, the mapping between the address and the LLC bank can possibly change. This means that two requests from two different L1s to the same address can be mapped to different LLC banks thus breaking the coherence. Figure 4a highlights a possible deadlock scenario due to the concurrent execution of a multicast, after an L1 miss request and the *LLC State* reconfiguration. The $L1_0$ sends a $getS(A)$ unicast request to LLC_0 once the multicast transaction returns all *nacks*. The $getS(A)$ is delayed due to the on-chip interconnect and, before reaching the LLC_0, the reconfiguration mechanism completes the change of the *LLC State* that powers ON LLC_1, and that turns to be the home LLC bank for A. However, the on-the-flight $getS(A)$ from $L1_0$ will force a fetch from LLC_0, thus leading to LLC data duplication in the case of a second $getS(A)$ to the correct LLC_1, e.g. the $getS(A)$ from $L1_1$ to LLC_1 in Figure 4a.

Retry Request on LLC dump - The LLC bank enqueues any incoming request that refers to a busy cache line to be later processed once the cache line state changes. During the reconfiguration, a cache line can be busy due to *i)* an in-progress request process from another cache controller or *ii)* because it is the victim of a flush action if the LLC bank is powering off. The cache line is guaranteed to exit the busy state after a finite amount of time, while its final state depends on the request that made it busy. In particular, a busy cache line due to a flush action on the LLC exits the busy state in the not-present coherent state. Moreover, *DarkCache* ensures a finite reconfiguration time by imposing that a powering off LLC bank that started the LLC flush cannot fetch any cache line until the end of the reconfiguration. Thus, the enqueued incoming request that is waiting for the requested cache line to exit the busy state, can incur in a deadlock. Since no complex mechanism is implemented in the LLC bank to check if the cache line will exit the busy state in the not-present state, *DarkCache* implements a retry mechanism to prevent such deadlock scenarios. The implemented retry mechanism returns a retry response to the sender of a request to

a powering off LLC bank for which the requested cache line is busy. The sender continues to retry the request until the cache line exits the busy state, thus leading to two different scenarios. First, the cache line exits the busy state in a state different from the not-present one, thus a subsequent retry request from the sender can be satisfied. Second, the cache line is flushed and its final LLC state is not-present, thus a *retryFreeze* response is sent back to the sender that will wait the end of the reconfiguration before issuing the new request. It is worth noticing that after a *retryFreeze* message, the requested data is not anymore present in the LLC, thus a memory fetch is required but it is impossible to be triggered during the reconfiguration (see Figure 4b).

Allow to Sink LLC Prefetch Requests - The *Multicast Mechanism* can *prefetch* data from memory to minimize the performance penalty of the L1 misses. However, the dynamic reconfiguration of the *LLC State* changes the LLC home bank for a specific cache line overtime. Moreover, the LLC home bank can prefetch a data without knowing if the data is already present in the LLC. *DarkCache* enhances the memory controller with the algorithm depicted in Figure 3a that leverages the presence bit in the directory to correctly manage, i.e., to process or to sink, the *multicast prefetch* requests (see Section 4.1 for details). Both the directory and the presence bit information are already available in the reference multi-core (see Section 3), thus no additional storage is required.

Figure 4c shows two prefetch requests for the same data, namely A, while a reconfiguration happens in between. We note that $L1_0$ evicts its copy of A at time ($E5$). Once the first prefetch request succeeded, the memory controller returns the required data. At this point, the reconfiguration changes the LLC home bank for address A from LLC_0 to LLC_1. Thus the *multicast(A)* action after the *miss(A)* event at time ($E7$), forces LLC_1 to issue a *multicast prefetch* request without knowing that A is already stored in LLC_0. However, the memory controller sinks the prefetch request after checking its directory state to avoid the possible deadlock due to data duplication.

5 DARK CACHE POLICY

DarkCache policy aims at minimizing the Energy Delay Product (EDP) by leveraging the power gating mechanism at LLC bank granularity. Figure 5 depicts the control loop where the *DarkCache policy* is inserted. The *Energy and Performance Indexes* (EPIs) are computed from the monitored architectural statistics and quantify the energy and performance impact of different LLC States compared to the current LLC configuration. In particular, the *Energy and Performance Indexes* are employed in the *LLC State Update Algorithm* to take the next best *LLC State*, that is then actuated through the power gating mechanism (see *Power Gating Actuation* in Figure 5).

The rest of this section overviews the key concepts of the *policy* while the *Energy and Performance Indexes* and the *Decision Algorithm* are presented in Section 5.1 and Section Section 5.2.

One-Dimension Optimization - The design of the LLC energy-performance policy is challenging due to conflicting requirements for energy and performance metrics. The *EPIs* have been carefully designed to allow an energy-aware optimization for which the performance is considered in terms of the additional energy spent due to the longer application execution.

Two Operating Modes - The *policy* operates in the *energy saving* mode to dynamically reshape the LLC size. Conversely, the *normal* mode is tailored for those scenarios where the running applications fully leverage the LLC, or if a small reduction in the LLC size severely increases the performance penalty. In these scenarios, all the additional logic to manage the run-time LLC resize is disabled to minimize the energy and performance overheads. The policy dynamically switches between the two operating modes without blocking the execution of the applications, that is with minimal impact on the system performance.

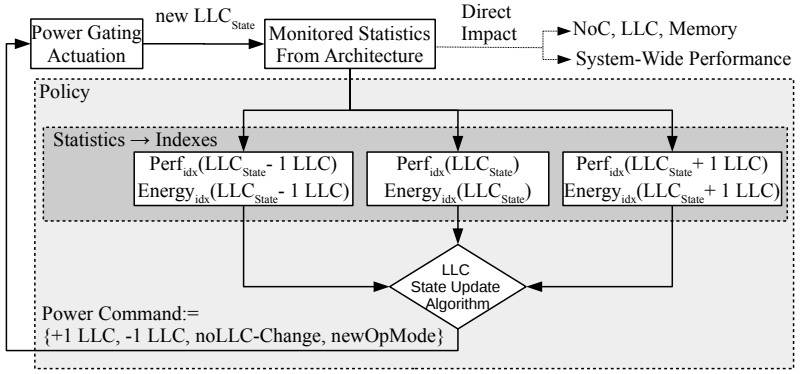


Fig. 5. The monitored architectural statistics are leveraged to quantify the energy and performance indexes that are later used by the *LLC State* decision algorithm of the policy (the speculative indexes are not shown). The policy imposes a new *LLC State* by powering on or off one LLC bank at most. The change in the *LLC State* directly affects the NoC memory and LLC energy profile as well as the system-wide performance. Moreover, the monitored architectural statistics are also expected to change according to the new *LLC State*.

Policy Search Space - *DarkCache* ensures the fast computation of the new *LLC State* and a simple implementation of the policy due to: *i)* the reduced search space for the new *LLC State*, and *ii)* the small number of monitored architectural statistics. A change in the *LLC State* is constrained to modify the power state of a single LLC bank at most, thus reducing the *LLC State* search space to three options, in addition to the possible change in the *Operating Mode* (see bottom of Figure 5). The *Energy and Performance Indexes* require to monitor only four architectural statistics, thus imposing a negligible traffic overhead on the interconnect (see Section 5.1). Moreover, *DarkCache* updates the *LLC State* with a period of half millisecond, a value that our experiments confirmed to be at least two orders of magnitude smaller than the changes in the application requirements. However, *DarkCache* ensures a fast response to the changes in the LLC requirements as discussed in Section 6.

5.1 Energy and Performance Indexes

The *Energy and Performance Indexes* (EPIs) capture the energy and the performance variations between different *LLC States* or operating modes to drive the selection of the next *LLC State*. In particular, the *EPIs* focus on the components in the multi-core that are directly affected by a change in the *LLC State*. Despite the simple structure of the EDIs does not account for higher order energy and performance effects caused by a dynamic LLC reshape, they accurately track the energy and performance variations to correctly drive the policy.

Assumptions - The EPIs are evaluated every half of a millisecond and we assume that is the shortest possible distance in time (time epoch) between two LLC reconfigurations. All the L1s and LLCs forward the collected architectural statistics to the policy module at the end of each time epoch. The experimentally estimated traffic overhead on the on-chip interconnect within 0.0024%. In particular, the policy exploits four architectural statistics: *i)* the LLC replacements, *ii)* the LLC accesses, *iii)* the L1 accesses and *iv)* the average number of sharers for the evicted LLC addresses. It is worth noticing that the LLC accesses are equal to the L1 misses and that the number of LLC replacements are supposed to be equal to the LLC misses after the cache warmup period.

The structure of the EDIs assumes a uniform distribution of the LLC accesses for which a change in the *LLC State* imposes a redistribution of the accesses between the powered on LLC banks, while their number (accesses) is supposed not to change because of the reconfiguration.

Indeed, our assumption does not affect the performance nor the accuracy of *DarkCache* because the new *LLC State* is decided considering the global pressure on the LLC banks, namely number of accesses and misses, for which the detailed statistics for each bank are almost of *no use*. In contrast, the state-of-the-art methodologies working on a physically split LLC architecture do reshape the available space for each LLC bank in isolation using a set or way granularity approach. In this latter scenario the actual distribution of the accesses and misses on each LLC bank is required.

DarkCache also assumes that the change in the LLC State does not affect the absolute number of LLC accesses and misses in the near future. Indeed, the reconfiguration can change at most the power state of a single LLC bank, with a limited impact on the monitored statistics. Moreover, the EDIs directly model the effect of a change in the *LLC State* on three subsystems of the multi-core, i.e., on-chip interconnect, main memory and LLC, without directly accounting for the L1 and the CPU (see the discussion on both Energy Performance Index in the rest of this section).

Starting from the assumption that the change of the power state of a single LLC bank cannot modify the total number of the LLC cache accesses, the LLC energy index is defined in Equation 1, for each time epoch t. In particular, the *LLC Energy Index* ($E_{LLC,t}$) only accounts for the change in the static energy of the LLCs after a reconfiguration.

$$E_{LLC,t} = (\#LLCs_{On,t} * E_{bankOn}) + (\#LLCs_{Off,t} * E_{bankOff}) \tag{1}$$

The *On-chip Interconnect Energy Index* $Comm_{EnIdx,t}$ for each time epoch t is defined as:

$$E_{NoC,t} = E_{hop,pktSize} \times \overline{HopCount} \times (1 + \hat{\overline{S_t}}) \times \sum_{i \in LLC} R_{LLC,i,t} \tag{2}$$

where $E_{hop,pktSize}$ is the energy to traverse one hop of the interconnect for a packet, that is $pktSize$-fold bigger than the channel width. The $\overline{HopCount}$ defines the number of hops a packet has to traverse from source to destination, on average, $\overline{S_t}$ defines the number of sharers to a replaced cache line in the LLC and $R_{LLC,i,t}$ represents the number of replacements in the i-th LLC bank. Equation 2 leverages data packets only, since they are directly traced by means of the two monitored architectural statistics, i.e., the number of accesses and misses to the LLC banks. Indeed the on-chip network traffic is also made by short control packets, while an LLC reconfiguration that increases (decreases) the LLC size is expected to eventually decrease (increase) the number of the data packets traversing the on-chip interconnect due to the increase (decrease) of the LLC misses.

The *Memory Energy Index* considers the variations in the number of reads and writes due to a change in the *LLC State*. For each time epoch t the index is defined as:

$$E_{Mem,t} = \sum_{i \in LLC} E_{memWrite} \times R_{LLC_{dirty},i,t} + E_{memRead} \times R_{LLC,i,t} \tag{3}$$

where $E_{memWrite}$ and $E_{memRead}$ capture the write and read energy spent for a single operation, respectively, and $R_{LLC_dirty,i,t}$ defines the fraction of dirty replacements in the i-th LLC bank during the time epoch t. The energy model assumes that a replacement always enforces a read, i.e., the missed data, and an additional write if the victim LLC cache line is dirty. Note that the dynamic energy in the memory is the only quantity influenced by the *LLC State*.

DarkCache does not define the *CPU Energy Index*, since a change in the *LLC State* is not easily and directly observable on the CPU without monitoring additional architectural statistics. In particular, all the other energy indexes indirectly account for the CPU contribution, given that an increase in the number of LLC misses affects the LLC, the on-chip interconnect and the memory. Conversely, the effect of an *LLC State* change on the CPU is better explained by means of the *Performance*

Index focusing on the performance loss in terms of the additional time to complete the application execution due to the reduction of the LLC size.

It is important to underline that the actual energy consumption of the architecture is evaluated in the experimental section by means of the widely accepted power models and tools, i.e., *McPat*, *dsent* and *Cacti*, that consider the complete multi-core made of CPU, cache hierarchy, memory and on-chip interconnect. The role of the *EDIs* is to estimate the first order variations in terms of energy and performance between different *LLC States* to drive the *LLC State* selection and they are not intended to provide an accurate estimate of the energy consumption.

Energy Index - The *Energy Index* estimates the energy benefit achieved by switching from the current *LLC State* to a new one that only differs by the power state of a single LLC bank. Equation 4 defines the *Energy Index* for the current *LLC State* resulting from the contributions of the LLC, the memory and the on-chip interconnect. Equation 6 and Equations 5 quantify the *Energy Index* for a configuration that uses an LLC bank more and less, respectively.

$$E_{cur,t} = E_{mem,t} + E_{noc,t} + E_{LLC,t} \tag{4}$$

$$E_{-1,t} = \frac{\#LLC_{tot}}{\#LLC_{On,t}} \times (E_{mem,t} + E_{noc,t}) + (E_{LLC,t} - E_{bankOn} + E_{bankOff}) \tag{5}$$

$$E_{+1,t} = \frac{\#LLC_{On,t}}{\#LLC_{tot}} \times (E_{mem,t} + E_{noc,t}) + (E_{LLC,t} + E_{bankOn} - E_{bankOff}) \tag{6}$$

The $\frac{\#LLC_{tot}}{\#LLC_{On,t}}$ coefficient grows with the inverse of the number of switched on banks, it is harder to switch off an additional LLC bank when the number of the active LLC banks is lower than the total implemented banks, i.e. $\#LLC_{tot}$. Moreover, Equation 5 also accounts for the turned off LLC bank, i.e., $(E_{LLC,t} - E_{bankOn} + E_{bankOff})$. The $E_{+1,t}$ estimate is computed in a similar fashion but the slope of the coefficient is reverted and the energy due to the LLC banks is updated considering an additional one powered on.

The energy overhead due to the *multicast mechanism* is defined by Equation 7 and it is used to support the policy in the selection of the best operating mode. $\#L1_{miss,t} \times \#LLC_{ON,t}$ accounts for the additional injected flits due to the multicast. Moreover, $\#numHops_{avg} \times E_{nocHop,t}$ defines the energy spent to traverse an NoC hop, i.e., link and router, times the average number of hops from source to destination evaluated from the topology at hand.

$$E_{multicast,t} = \#L1_{miss,t} \times \#LLC_{ON,t} \times \#numHops_{avg} \times E_{nocHop,t} \tag{7}$$

Last, the saved energy due to the switched off LLC banks in the time epoch t is defined by Equation 8.

$$E_{LLC_{saved},t} = (E_{bankOn} - E_{bankOff}) \times \#LLC_{Off,t} \tag{8}$$

Performance Index - *DarkCache* actuates on the LLC size to optimize the Energy Delay Product (EDP). Equation 9 defines the *Performance Degradation Index* PDI for each time epoch t as the performance degradation of the actual *LLC State* with respect to the performance achieved by the baseline architecture where the *LLC State* reconfiguration mechanism is not active. It is worth noticing that the *Performance Degradation Index* is accounted in terms of energy to allow an easier, single-objective optimization policy.

The *Performance Degradation Index* (PDI) is made of three terms and it leverages on the intuition for which the performance degradation is better observable as the cost of a miss in the L1 due to its

Algorithm 1 LLC State Update

```
1: if R_{LLC,t} > #Lines_{LLCbank} AND E_{cur,t} > E_{+1,t} then
2:     b := selectTargetBank(LLCs_current_config)
3:     sendSwitchOn(b)
4: else if R_{LLC,t} < #Lines_{LLCbank} AND E_{cur,t} > E_{-1,t} then
5:     b := selectTargetBank(LLCs_current_config)
6:     sendSwitchOff(b)
7: end if
```

capability of actually stalling the CPU.

$$E_{PerfDegr,t} = \frac{\#L1_{miss,t}}{\#L1_{access,t}} \times \left(\frac{Time_{lastAck} - Time_{avgAck}}{Time_{avgAck}} - 1\right) \times E_{cur,t} \tag{9}$$

The $\frac{\#L1_{miss,t}}{\#L1_{access,t}}$ quantity, i.e., the L1 miss rate, represents the fraction of the requests that trigger the multicast when the system is operating in the *energy saving* mode. Note that no performance degradation is expected if the system is operating in *normal* mode, since this is the baseline architecture. Moreover, *DarkCache* aims at optimizing the EDP by minimizing the energy consumption without affecting the overall system performance.

The performance degradation due to the use of a multicast request in place of the corresponding unicast one is defined by $\frac{Time_{lastAck} - Time_{avgAck}}{Time_{avgAck}} - 1$, i.e., by the difference of time between the last received ack/nack response and the average time to receive an ack/nack of a multicast request. In particular, the latter is used in place of the time taken to solve the L1 miss using a unicast request. Thus, $\frac{\#L1_{miss,t}}{\#L1_{access,t}} \times \left(\frac{Time_{lastAck} - Time_{avgAck}}{Time_{avgAck}} - 1\right)$ actually defines the performance overhead due to the multicast transactions within epoch t. The energy overhead due to the performance degradation, i.e., $E_{PerfDegr,t}$, is computed by multiplying the percentage overhead and the current energy consumption estimates, namely $E_{cur,t}$.

5.2 Policy Algorithm

The *DarkCache* policy is made of two parts: *i) Algorithm 1 updates the* LLC State *when* DarkCache *operates in the* energy saving *mode and* ii) Algorithm 2 drives the operating mode selection.

The update of the *LLC State* leverages the observation for which, if the number of replacements in all the active LLCs ($R_{LLC,t}$) is bigger than the LLC bank size in terms of number of lines ($\#Lines_{LLCbank}$), new LLC bank is powered on if and only if $E_{cur,t} > E_{+1,t}$. Conversely, an LLC bank is powered off if $E_{cur,t}$ is less than $E_{-1,t}$ subject to $R_{LLC,t}$ less than $\#Lines_{LLCbank}$. To this extent, the *LLC State* update depends on both the consumed energy and the actual number of LLC replacements compared to the actual LLC bank capacity. It is important to notice that the *Performance Degradation Index* is not directly accounted in Algorithm 1 for two reasons. First, the PDI compares *DarkCache* and baseline in terms of performance, thus pointing out when the additional overhead due to the additional mechanism of *DarkCache* is not providing a real EDP benefit. Second, the performance degradation between different *LLC States* are already indirectly accounted in terms of the number of LLC accesses and misses. In particular, an increase in the LLC misses signals a possible performance degradation due to an higher stall probability for the CPUs that are waiting for data. In contrast, Algorithm 2 employs the *PDI* to drive the operating mode selection. *DarkCache* switches to the *normal* mode when the sum of the energy spent due to the multicast and the performance degradation overcomes the saving due to the powered off LLC banks. This condition should persist throughout the whole *THRESHOLD* period. Conversely,

Algorithm 2 Operating Mode Selection

1: **if** $(E_{multicast,t} + E_{PerfDegr,t}) > E_{LLC_{saved},t}$ AND $cnt == 10$ **then**
2: SwitchTo(normal mode); cnt=0;
3: **else**
4: cnt++;
5: **end if**
6: **if** $(E^{spec}_{multicast,t} + E^{spec}_{PerfDegr,t}) < E^{spec}_{LLC_{saved},t}$ AND $cnt == 10$ **then**
7: SwitchTo(energy saving mode); cnt=0;
8: **else**
9: cnt++;
10: **end if**

the *Speculative Energy and Performance Indexes*, $E_{multicast,t}$, $E_{PerfDegr,t}$ and $E_{LLC_{saved},t}$, are used when *DarkCache* is operating in *normal* operating mode to foreseen the benefit of a switch to the *energy saving* operating mode.

Equation 10 defines the number of speculative L1 misses that would be observed if the system were to work in the *energy saving* mode starting from the actual L1 accesses and misses in the considered time epoch t. Equation 11 defines the performance overhead due to the multicast in terms of the hop count to traverse the on-chip interconnect in place of the actual ack/nack response time that is not available when the system is operating in *normal* mode (see $(\frac{Time_{lastAck} - Time_{avgAck}}{Time_{avgAck}} - 1)$ in Equation 9). To avoid spurious reconfigurations , the constant values in Equation 10 and Equation 13 have been experimentally evaluated to force the switch from the *normal* to the *energy saving* mode when the memory requirements allows to roughly powering off one third of the LLC banks.

$$\#L1^{spec}_{miss,t} = \frac{\#L1_{access,t}}{\#LLC_{tot}} \times (\frac{1}{3} \times \#LLC_{tot}) + \#L1_{miss,t} \tag{10}$$

$$TimeDegr^{spec}t = \frac{\#numHops_{max} - \#numHops_{avg}}{\#numHops_{avg} - 1} \tag{11}$$

Starting from Equation 10 and Equation 11, the speculative quantities corresponding to $E_{LLC_{saved},t}$, $E_{multicast,t}$ and $E_{PerfDegr,t}$) are defined in Equation 13, Equation 14 and Equation 12, respectively. These quantities are then used in Algorithm 2 (see lines $6-10$).

$$E^{spec}_{LLC_{saved},t} = (E_{bankOn} - E_{bankOff}) \times \frac{2}{3} \times \#LLC_{tot} \tag{12}$$

$$E^{spec}_{multicast,t} = \#L1^{spec}_{miss,t} \times (\frac{2}{3} \times \#LLC_{tot}) \times \#numHops_{avg} \times E_{nocHop,t} \tag{13}$$

$$E^{spec}_{PerfDegr,t} = \frac{\#L1_{miss,t}}{\#L1_{access,t}} \times TimeDegr^{spec}t \times (E_{cur,t} - E^{spec}_{LLC_saved,t}) \tag{14}$$

6 EVALUATION

This section is organized in two parts: *i)* the energy, performance and EDP evaluation (Section 6.1) of *DarkCache* and its comparison with two state-of-the-art methodologies, and *ii)* a detailed analysis of the reconfiguration's timing and frequency for both *DarkCache* and the state-of-the-art (Section 6.2). As baseline, we employ the inclusive, two-level MESI cache coherence protocol enclosed in the *gem5*

Table 1. Architecture and technology parameters for the simulated 4x4 and 8x8 architectures.

Trace generator	Sigil2
Processor core	Synchrotrace: ALU: 1 cycle, FPU: 4 cycles
L1 cache	32+32kB 8-way set assoc. split I/D, 2 cycles latency
LLC cache	256KB per bank, 8-way associative
Coherence Prot.	MESI / DarkCache , 3 virtual networks
Router	1GHz, 2-stage wormhole, 4 VCs per VNET 128bit link width, 4 flit/VC
Topology	2D-mesh, 4x4 / 8x8 NoC
Main Memory	Size: 8GB, 4 MC at the topology corners Latency: 160 cycles
Technology	45nm at 1.1V
Power Network	Power-on time: 20 cycles, Power-off time: 1 cycle
LLC Leakage (Bank)	Power-on-leakage: 110uJ , Power-off-leakage:10nJ

Table 2. Evaluated solutions to dynamically reshape the LLC size.

Name	Details
Baseline	Baseline SNUCA; all results are normalize to this architecture.
DarkCacheSeq DarkCacheCirc	*DarkCache* using the energy saving operating mode only and the circular (Circ) or sequential (Seq) bank selection policy
DarkCacheOpt	*DarkCache* optimized; two operating modes and sequential bank selection policy
UtilityBased	The power gating actuates at cache line granularity and each LLC bank optimizes its associativity in isolation by powering off the unused lines.
UtilityBasedOpt	Improves *UtilityBased* by minimizing the spurious reconfigurations.

cycle accurate simulator [3]. We then implemented *DarkCache* and the considered state-of-the-art methodologies on top of it. The obtained results are compared against the *UtilityBased* [17] and the *UtilityBasedOpt* [24] state-of-the-art design methodologies.

Simulated Architectures and Benchmarks - *DarkCache* is validated against 16- and 64-core, 2D-mesh, tiled architectures whose detailed parameters are reported in Table 1. The two architectures share the same tile configuration that is made of an L1 cache that remains private to the CPU, an LLC cache slice that is shared between all the CPUs and a router to connect the tile to the rest of the multi-core. For each application in the *Splash2x* benchmark (part of the *Parsec 3.0* suite), an execution trace is extracted spawning 16 and 64 parallel threads. Each trace is fed into the *synchrotrace* trace replayer [16] within *gem5* to simulate the 16- and the 64-core architectures.

State of the Art Methodologies - We employ two state-of-the-art methodologies, i.e., *UtilityBased* [17] and *UtilityBasedOpt* [24], to compare the benefit of *DarkCache* in terms of performance, energy and EDP. Both [17] and [24] are presented as cache partitioning schemes for monolithic LLC architectures. In both cases the LLC is dynamically partitioned between the competing applications at cache line granularity and the unused LLC portion is powered off. At each reconfiguration, *UtilityBased* [17] computes the optimal number of ways to be assigned to each running application. This is the number that maximises the *cache utility*, i.e., the reduction in the miss count that occurs if an additional cache way is assigned to the application. The *UtilityMonitor* hardware module, which is part of the methodology, predicts the miss rate after considering different associativity levels. *UtilityBasedOpt* [24] takes steps from [17] to introduce a threshold mechanism that prevents cache associativity's unintended reconfigurations due to spurious changes in the application cache access pattern, that negatively impact the energy without adding any EDP improvement.

The two implemented state-of-the-art methodologies optimize each LLC bank in isolation working at cache line granularity. The wake-up energy and time are scaled according to the weight of a single cache line with respect to the whole LLC bank. According to *cacti* [15], the maximum leakage that can be saved by switching off all the cache lines but one per each set is 79% out of the leakage of an entire LLC bank due to the impossibility of switching off completely an LLC bank (sense amplifiers, one way per set and the decode logic) considering a utility based methodology. We impose two clock cycles to wakeup a portion of the cache, and ten for the entire LLC bank.

Evaluated Metrics - The proposed methodology is validated considering performance, energy and the Energy Delay Product (EDP) metrics. The *performance* metric of a benchmark running on a specific architecture is defined as the simulated time the benchmark takes to complete the execution on the architecture. The real energy consumption of the entire system is computed at

each time epoch. We employ McPAT [10] to compute the CPU energy accounting for the number of different types of instructions that are committed during the epoch. The *dsent* power model [22] is employed to extract the NoC energy consumption as the sum of the power of each router and link over the time epoch. The *DRAMPower* energy model [8] is used for the memory controllers and the cache energy is computed with *cacti* [15]. The power network timing and energy overheads are taken from [27], while the *EDP* for architecture a and benchmark b is defined as:

$$EDP(a,b) = EPI(a,b) \times CPI(a,b) \tag{15}$$

where the $EPI(a,b)$ is the average energy per instruction dissipated by the system in the architecture a during the execution of application b and $CPI(a,b)$ is the average number of clock cycles required to execute an instruction of the benchmark b while it is running on architecture a.

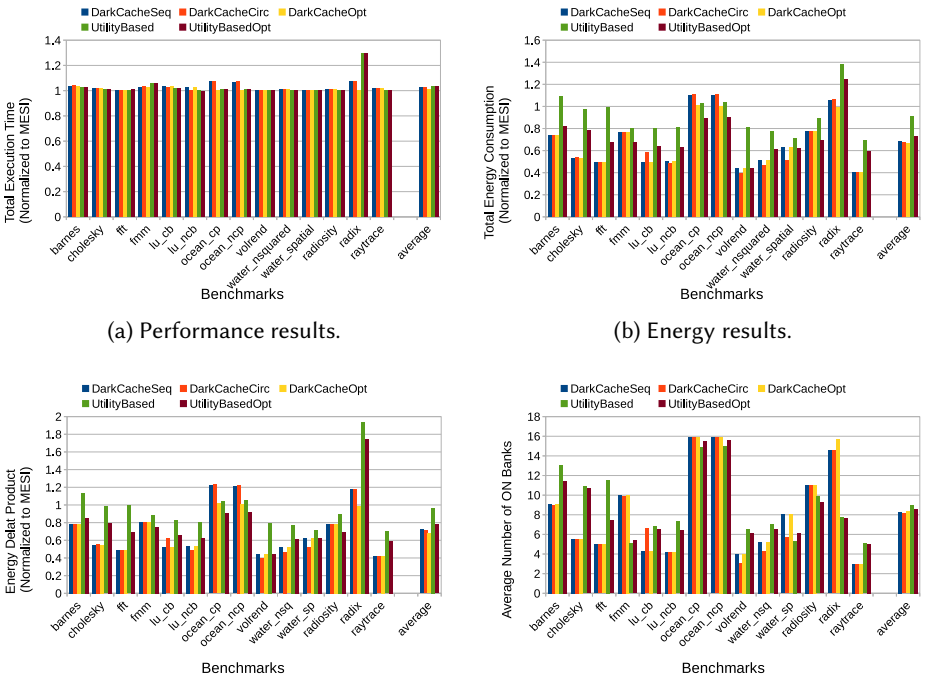
Operating Modes and Target LLC Bank Selection Policies - We report results for both *DarkCache* and *DarkCacheOpt* where the former only implements the *energy saving* operating mode and the latter implements both the *energy saving* and the *normal* modes. The comparison highlights the benefit introduced by means of the *normal* mode to manage the applications that fully exploit the available LLC for which any multicast request becomes a full broadcast to the entire LLC with a severe overhead on EDP, energy and performance.

Note that *DarkCache* is not constrained to any bank selection policy to select the target LLC banks during the reconfiguration. Such policies are strongly topology dependent and it is interesting to analyze their impact on the final system performance when coupled with the proposed *DarkCache* solution. Two LLC bank selection policies are taken from the state of the art and implemented, i.e., *DarkCacheCirc* and *DarkCacheSeq*. *DarkCacheCirc* implements the *circular policy* that powers off/on the LLC banks by following a concentric scheme where the LLC banks at the center of the topology are the last to be powered off and the first to be powered on. *DarkCacheSeq* exploits a static scheme that, from the topology viewpoint, powers off and on the LLC banks line by line in the 2D-mesh. The obtained results demonstrate a negligible impact of the bank selection policy on the considered metrics, i.e., energy performance and EDP (see Figure 6 and Figure 7). To this extent, we only report results for *DarkCacheOpt* which implements the two operating modes and employs the sequential bank selection policy, since, even for *DarkCacheOpt*, a negligible impact has been observed when the *circular bank selection policy* is used in place of the *sequential* one.

6.1 Performance, Power and EDP

The *DarkCache* assessment considers both 4 by 4 and 8 by 8 multi-cores. The collected results are discussed in terms of energy, performance and EDP against both *UtilityBased* and *UtilityBasedOpt* solutions. All the results are normalized to the baseline MESI cache coherence protocol that also represents the protocol in use when *DarkCache* operates in *normal* mode.

Figure 6 reports the results for each benchmark considering a 4 by 4 2D-mesh with both *DarkCache* and *UtilityBased* implementations. All the results are normalized to the baseline MESI (not shown), while Figure 6d reports the average number of powered on LLC banks for each simulated benchmark. In particular, the *UtilityBased* and *UtilityBasedOpt* implementations report the estimated number of powered on LLC banks obtained by aggregating the average number of powered on cache lines in the LLCe, divided by the number of lines in an LLC bank. Results in Figure 6 show an average energy saving of 31.95%, 32.74% and 33.78% considering *DarkCacheSeq*, *DarkCacheCirc* and *DarkCacheOpt*, respectively. In particular, the majority of the analyzed benchmarks underutilize the LLC, thus even with few powered on LLC banks the average performance degradation is within 1.3%. It is worth noticing that both *DarkCacheCirc* and *DarkCacheSeq* report similar results for all the considered metrics, i.e., performance energy and EDP, thus highlighting the limited impact of the bank selection



(a) Performance results.

(b) Energy results.

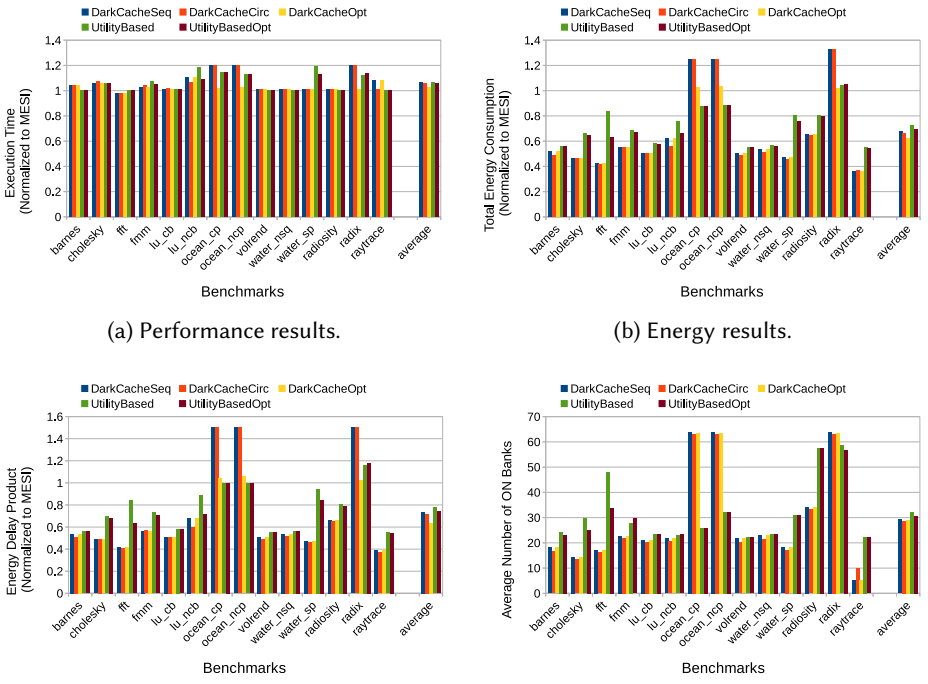
(c) Energy Delay Product (EDP) Results.

(d) Average number of powered on LLC banks.

Fig. 6. Performance (a), Energy (b) and EDP (c) results normalized to the baseline MESI architecture considering a 4 by 4 2D-mesh topology and the complete Splash2x benchmark suite. The average number of powered-ON LLC banks is shown in (d).

policy on both performance and energy saving. Few benchmarks fully employ the available LLC, thus limiting the energy saving opportunities of *DarkCache*. In particular, the average LLC bank utilization for *ocean_cp*, *ocean_ncp* is close to 15 banks while *radix* is close to 16. Considering *DarkCacheSeq*, *DarkCacheCirc*, these benchmarks have an energy and performance overheads up to 7% and 8% with an EDP degradation up to 18.1%. This is mainly due to the impossibility of disabling the multicast support that dramatically increases the on-chip network traffic with a net increase of *i)* the on-chip energy, *ii) network latency*, and *iii)* pressure onto the LLC controllers. Conversely, *DarkCacheOpt* correctly manages these benchmarks for which the operating mode is switched from *energy saving* to *normal*. Indeed, *DarkCacheOpt* limits the performance and energy overheads for the three benchmarks up to 0.02% and 1.0%, respectively, with an average EDP overhead of 1.2%.

Both *UtilityBased* and *UtilityBasedOpt* solutions allow to fast reconfigure the LLC associativity. Since each address cannot change the *LLC Home Bank*, a simpler decentralized reconfiguration mechanism is in fact capable of avoiding the complex *Handshake* protocol. However, the unconstrained number of reconfigurations severely affects the performance of the *UtilityBased* solution that behaves worse than *UtilityBasedOpt*. In contrast, *UtilityBasedOpt* limits the number of spurious reconfigurations, thus limiting both the unrequired LLC power on and off patterns and the dump of cache lines back to the main memory. This way, the net energy saving increases without affecting the performance. Despite *UtilityBasedOpt* obtains an average EDP reduction within 23.96%, *DarkCacheOpt* outperforms it by 16.15%, since the former suffers of three main



(a) Performance results.

(b) Energy results.

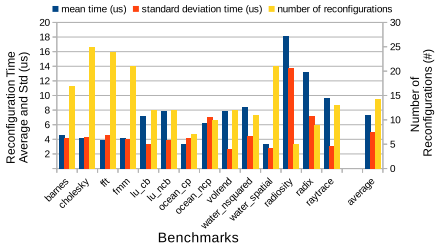
(c) Energy Delay Product (EDP) Results.

(d) Average number of powered on LLC banks.

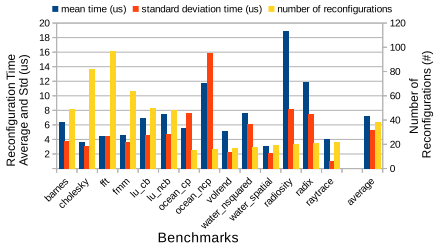
Fig. 7. Performance (a), Energy (b) and EDP (c) results normalized to the baseline MESI architecture considering a 8 by 8 2D-mesh topology and the complete Splash2x benchmark suite. The average number of powered-ON LLC banks is shown in (d).

drawbacks. First, *UtilityBasedOpt* operates a local optimization for which different LLC banks can optimize their associativity towards opposite directions, thus limiting the theoretically achievable EDP reduction. Second, the performance of the applications can be severely affected by a reduction in the cache associativity. For example *radix* exhibits a performance overhead of 29,7% and 29,5% using *UtilityBased* and *UtilityBasedOpt*, respectively. In particular, the utility based methodologies greatly reduce the average number of switched on banks, i.e., less than 8, to save energy, while the application performance is severely influenced. In contrast *DarkCacheOpt* executes *radix* in *normal* mode for most of the time, with a negligible performance overhead that, from the EDP perspective, justifies the used LLC banks. Third, *UtilityBasedOpt* cannot completely LLC power off, since at least one way per set and the decode logic have to be powered on in each LLC bank, thus limiting the energy saving opportunities.

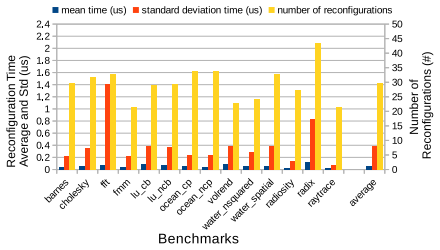
Figure 7 reports the results of *DarkCache* and *UtilityBased* considering an 8 by 8 2D-mesh architecture. The bigger topology magnifies the observations drawn for the 16-core architecture. Again, the majority of the benchmarks underutilize the LLC and the increased size of the LLC leads to better optimization opportunities for both *DarkCache* and *UtilityBased*. Indeed, there are always the same three benchmarks discussed for the 4 by 4 simulations for which an LLC size reduction severely affects the performance. Moreover, the bigger topology highlights the performance and energy overheads due to the multicast mechanism on a 64 core architecture. At the end of 21 days of simulations, the performance, energy and EDP results for those three benchmarks are limited to



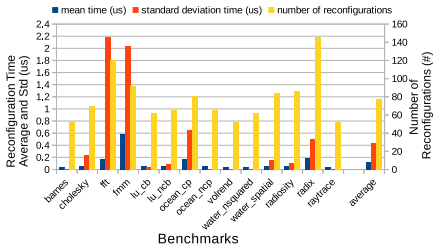
(a) DarkCacheOpt 4x4.



(b) DarkCacheOpt 8x8.



(c) UtilityOpt 4x4.



(d) UtilityOpt 8x8.

Fig. 8. Per benchmark number of reconfigurations as well as the average and standard deviation time taken by a reconfiguration considering both a 4x4 (a) and an 8x8 (b) 2D-mesh topology.

20%, 25% and 50% for both *DarkCacheSeq*, *DarkCacheCirc*. Thus, the reported numbers represent an optimistic, even if already too bad, estimate of the real overheads. The collected results are reported to demonstrate the superior flexibility of *DarkCacheOpt* compared to *DarkCacheSeq* and *DarkCacheCirc*. In particular, *DarkCacheOpt* can switch between two operating modes, which means that the benchmarks mentioned above are executed in the *normal* mode for the majority of the time. To this extent, performance, energy and EDP overheads are limited to 1.8%, 3.1% and 5.5%. It is worth analyzing the origins of the overheads reported by *DarkCacheOpt* for the *ocean_cp*, *ocean_ncp* and *radix* benchmarks are due to a pair of different aspects. First, the single operating mode switch from *energy saving* to *normal* mode requires a complete flush of the LLC banks. Second, *DarkCache* starts the execution in *energy saving* mode and requires a long enough observation window before eventually switching the operating mode. The bigger topology increases the optimization gap between *UtilityBasedOpt* and *DarkCacheOpt* where the latter overcomes the former by 21.05%. The results highlight the better scalability of *DarkCacheOpt* that delivers a system-wide *LLC State* optimization, while *UtilityBasedOpt* operates a local optimization for each LLC bank.

6.2 Detailed Analysis

This section discusses the details of *DarkCache* and *UtilityBased* in terms of the frequency and time of the reconfigurations. We consider *DarkCacheOpt* and *UtilityBasedOpt* since they are the implementations showing the best results for the proposed solution and for the considered state-of-the-art. The final goal of this discussion is to highlight the relationship between the performance and the reconfiguration dynamics of the proposed methodology compared with the state-of-the-art solution. Figure 8 reports the results for *DarkCacheOpt* (Figure 8a and Figure 8b) and *UtilityBasedOpt* (Figure 8c and Figure 8d) considering both 4 by 4 and 8 by 8 topologies. Each figure reports the

results in terms of the average and the standard deviation of the reconfiguration time, as well as the number of reconfigurations for each benchmark. Since the reconfiguration is local to each bank, *UtilityBasedOpt* has a reconfiguration time in the order of dozens of nanoseconds regardless of the size of the topology. The standard deviation is also independent from the size of the topology showing an average value close to 1.2 microseconds due to the highly variable time taken to dump a cache line when the LLC associativity is reduced. *DarkCacheOpt* has an average reconfiguration time in the order of few microseconds (dozens of microseconds for the standard deviation), thus two orders of magnitude higher than that of the *UtilityBaseOpt* solution. This is due to the more complex handshake mechanism that coherently coordinates the reconfiguration of the LLC as a single subsystem in the multi-core. However, the *DarkCacheOpt* reconfiguration protocol does not block the computation but only stalls the L1 misses (see Section 4.2). In contrast, all the accesses to the L1 resulting in a hit allow the requesting CPU to continue the execution. The results in Section 6.1 highlight the limited performance overhead imposed by the *DarkCache* reconfiguration protocol that is, in average, within 1.3% (16-core) and 2.8% (64-core).

Last, *UtilityBasedOpt* highlights a number of reconfigurations that is between 60% to 100% higher than *DarkCacheOpt*. As already discussed in Section 6.1, the utility based solutions do not explicitly consider any performance metric and the associativity level is always optimized to maximize the ratio between the reduction in the number of misses and the increase in the associativity. However, the scheme cannot easily handle the spurious reconfigurations, since the reconfiguration process does not account for the system-wide state of the memory requirements.

7 CONCLUSIONS

This work presented *DarkCache*, a dynamic cache reconfiguration scheme for tiled and non-tiled multi-cores, which optimizes the Energy Delay Product (EDP) by minimizing the static energy of the LLC also accounting for the system-wide performance. In particular, *DarkCache* leverages the LLC underutilization to power off the unused LLC banks. Moreover, for those benchmarks that fully use the available LLC, *DarkCache* implements a second operating mode, that turns off all the additional mechanisms required to correctly manage the dynamic LLC reshape. The LLC reshape is also partially non-blocking for the running applications to minimize the performance overheads.

DarkCache has been compared against two state-of-the-art methodologies and the results are reported for both 16- and 64-core architectures. *DarkCache* achieves an average EDP gain of 32.58% and 36.41% against the baseline architecture with 16- and 64-cores respectively and an average performance penalty always within 2%. Moreover, an EDP gain of 16.15% (16 cores) and 21.05% (64 cores) is achieved against the best state-of-the-art methodology. This demonstrates the good scalability properties of the proposed solution since the gains increases with the core counts.

REFERENCES

[1] Sandro Bartolini, Pierfrancesco Foglia, and Cosimo Antonio Prete. 2018. Exploring the relationship between architectures and management policies in the design of NUCA-based chip multicore systems. *Future Generation Computer Systems* 78 (2018), 481 – 501.

[2] N. Beckmann and D. Sanchez. 2016. Modeling cache performance beyond LRU. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 225–236.

[3] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 Simulator. *SIGARCH Comput. Archit. News* 39, 2 (Aug. 2011), 1–7.

[4] S. Borkar. 1999. Design challenges of technology scaling. *IEEE Micro* 19, 4 (Jul 1999), 23–29.

[5] Shekhar Borkar and Andrew A. Chien. 2011. The Future of Microprocessors. *Commun. ACM* 54, 5 (May 2011), 67–77.

[6] Andreas Hansson, Kees Goossens, and Andrei Radulescu. 2007. Avoiding Message-Dependent Deadlock in Network-Based Systems on Chip. *VLSI Design* (2007).

[7] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely, Jr., and Joel Emer. 2010. High Performance Cache Replacement Using Re-reference Interval Prediction (RRIP). In *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA '10)*. ACM, New York, NY, USA, 60–71.
[8] Chandrasekar Karthik, Weis Christian, Li Yonghui, Akesson Benny, Wehn Norbert, and Gossens. 2012 Kees. 2012. *DRAMPower: Open-source DRAM power & energy estimation tool.* Technical Report. http://www.drampower.info
[9] Isao Kotera, Kenta Abe, Ryusuke Egawa, Hiroyuki Takizawa, and Hiroaki Kobayashi. 2011. Transactions on High-performance Embedded Architectures and Compilers III. Springer-Verlag, Berlin, Heidelberg, Chapter Power-aware Dynamic Cache Partitioning for CMPs, 135–153.
[10] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *IEEE/ACM MICRO'09*. 469–480.
[11] Mario Lodde and JosÃĺ Flich. 2014. Runtime home mapping for effective memory resource usage. *Microprocessors and Microsystems* 38, 4 (2014), 276 – 291.
[12] R. Manikantan, K. Rajan, and R. Govindarajan. 2012. Probabilistic Shared Cache Management (PriSM). In *2012 39th Annual International Symposium on Computer Architecture (ISCA)*. 428–439.
[13] S. Mittal, Y. Cao, and Z. Zhang. 2014. MASTER: A Multicore Cache Energy-Saving Technique Using Dynamic Cache Reconfiguration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22, 8 (Aug 2014), 1653–1665.
[14] S. Mittal, Z. Zhang, and Y. Cao. 2013. CASHIER: A Cache Energy Saving Technique for QoS Systems. In *2013 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems*. 43–48.
[15] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. 2008. Architecting Efficient Interconnects for Large Caches with CACTI 6.0. *IEEE Micro* 28, 1 (Jan 2008), 69–79.
[16] S. Nilakantan, K. Sangaiah, A. More, G. Salvadory, B. Taskin, and M. Hempstead. 2015. Synchrotrace: synchronization-aware architecture-agnostic traces for light-weight multicore simulation. In *(ISPASS'15)*. 278–287.
[17] Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches. In *MICRO 39*. IEEE Computer Society, Washington, DC, USA, 423–432.
[18] P. Ranganathan, S. Adve, and N. P. Jouppi. 2000. Reconfigurable caches and their application to media processing. In *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*. 214–224.
[19] Samuel Rodriguez and Bruce Jacob. 2006. Energy/Power Breakdown of Pipelined Nanometer Caches (90Nm/65Nm/45Nm/32Nm). In *ISLPED '06*. ACM, New York, NY, USA, 25–30.
[20] D. Sanchez and C. Kozyrakis. 2012. Scalable and Efficient Fine-Grained Cache Partitioning with Vantage. *IEEE Micro* 32, 3 (May 2012), 26–37.
[21] Daniel J. Sorin, Mark D. Hill, and David A. Wood. 2011. *A Primer on Memory Consistency and Cache Coherence* (1st ed.). Morgan & Claypool Publishers.
[22] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. 2012. DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling. In *NOCS '12*. IEEE Computer Society, Washington, DC, USA, 201–210.
[23] K. T. Sundararajan, T. M. Jones, and N. P. Topham. 2013. RECAP: Region-Aware Cache Partitioning. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*. 294–301.
[24] K. T. Sundararajan, V. Porpodas, T. M. Jones, N. P. Topham, and B. Franke. 2012. Cooperative partitioning: Energy-efficient cache partitioning for high-performance CMPs. In *IEEE HPCA'12*. 1–12.
[25] P. H. Wang, C. H. Li, and C. L. Yang. 2016. Latency sensitivity-based cache partitioning for heterogeneous multi-core architecture. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6.
[26] X. Wang and W. Zhang. 2016. Cache locking vs. partitioning for real-time computing on integrated CPU-GPU processors. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. 1–8.
[27] Y. Wang, S. Roy, and N. Ranganathan. 2012. Run-time power-gating in caches of GPUs for leakage energy savings. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*. 300–303.
[28] Yuejian Xie and Gabriel H. Loh. 2009. PIPP: Promotion/Insertion Pseudo-partitioning of Multi-core Shared Caches *(ISCA '09)*. ACM, New York, NY, USA, 174–183.
[29] Yuejian Xie and Gabriel H. Loh. 2010. Scalable Shared-cache Management by Containing Thrashing Workloads. In *HiPEAC'10*. Springer-Verlag, Berlin, Heidelberg, 262–276.
[30] S. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar. 2001. An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches. In *HPCA'01*. 147–157.
[31] Y. Ye, R. West, Z. Cheng, and Y. Li. 2014. COLORIS: A dynamic cache partitioning system using page coloring. In *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*. 381–392.
[32] Xusheng Zhan, Yungang Bao, Christian Bienia, and Kai Li. 2017. PARSEC3.0: A Multicore Benchmark Suite with Network Stacks and SPLASH-2X. *SIGARCH Comput. Archit. News* 44, 5 (Feb. 2017), 1–16.

Received Aug 2017; Revised Dec 2017; Accepted Feb 2018