Optical Overlay NUCA: A High Speed Substrate for Shared L2 Caches

Eldhose Peter, Anuj Arora, Akriti Bagaria and Smruti R Sarangi
Computer Science and Engineering, Indian Institute of Technology Delhi, New Delhi, India
{*eldhose,mcs122812,mcs132541,srsarangi*}*@cse.iitd.ac.in*

***Abstract*—In this paper, we propose to use optical NOCs to design cache access protocols for large shared L2 caches. We observe that the problem is unique because optical networks have very low latency, and in principle all the cache banks are very close to each other. A naive approach is to broadcast a request to a set of banks that might possibly contain the copy of a block. However, this approach is wasteful in terms of energy and bandwidth. Hence, we propose a novel scheme in this paper, *TSI* , which proposes to create a set of virtual networks ($overlays$) of cache banks over a physical optical NOC. We search for a block inside each overlay using a combination of multicast, and unicast messages. We additionally create support for our overlay networks by proposing optimizations to the previously proposed R-SWMR network. We also propose a set of novel hardware structures for creating and managing overlays, and for efficiently locating blocks in the overlay. The performance of the *TSI* scheme is within 2-3% of a broadcast scheme, and it is faster than traditional static NUCA schemes by 50%. As compared to the broadcast scheme it reduces the number of accesses, and consequently the dynamic energy by 20-30%.**

***Keywords*-Nano-photonics; NUCA caches; L2 cache; LLC cache access protocol; Optical networks**

I. Introduction

On-chip optical networking is a disruptive technology that has the potential to profoundly impact the design of processors in the near future. Optical networks have already proved their superiority over traditional electrical links in local and wide area networks. Optical networks are expected to have a similar impact in on-chip networks in large many-core processors [1] of the future. Consequently, computer architects have already started building prototypes of chips with embedded optical networks, and there is a plethora of research in designing efficient optical networks [2, 3, 4]. The main issues in the design of optical networks is the reduction of static power, minimizing the number of wavelengths, and quickly setting up a light path for both unicast and multicast traffic.

The main advantages in on-chip optical NOCs are latency and bandwidth. Optical signals can travel at 7 ps/mm (as compared to 35-70 ps/mm for global electrical links). Additionally, electrical links require complicated routers that buffer packets, assign them to virtual channels, perform route computation, and arbitrate for an output port. Even with fast lookahead routers, the router delay is at least 2-3 cycles [5]. For a chip with 32 cores and 32 cache banks, and a toroidal interconnect, the diameter of the network measured in terms of the time it takes to transmit a flit is at least 32 cycles with no contention. In comparison, it is possible to transfer a 64 bit flit in less than 3 cycles using an optical network (includes the time for O/E and E/O conversion). Since, we can multiplex 32-64 wavelengths on a single optical waveguide (optical channel), optical links can provide very high bandwidth also. Lastly, a large class of optical networks such as SWMR and MWSR [6, 7] naturally support broadcast and multicast traffic. It does not take any additional effort to broadcast a flit to all the optical stations (optical nodes that can send/receive data). Researchers believe that the aforementioned advantages make a compelling case for optical networks.

Computer architects are now in the process of finding uses for optical networks and retargeting protocols to leverage optical networks. Researchers have in specific focused on two problems that are natural fits for optical networks. The first is implementing barriers [8]. Barriers require a core to broadcast a message to the rest of the cores, when it is about to release a barrier. Given the minimal latency of optical networks, barriers can be made ultra-fast. The second problem that has received a lot of attention is coherence protocols. Optical networks can be used to implement fast snoopy protocols for block location, and invalidation. Researchers have obtained appreciable speedups (20-40%) using optical networks.

In this paper, we consider a natural extension to this line of research, which is to use optical networks in the L2 cache and lower levels (if present). Optical networks for optimizing accesses to the L2 cache has not been looked at in the past (to the best of our knowledge), and thus we focus our efforts on designing methods to design large non-uniform shared L2 caches that use optical networks. We start from state of the art implementations with electrical networks, and then try to adapt the protocols for optical networks, and propose various important optimizations along the way. The most effective L2 cache access mechanisms use NUCA (non-uniform cache access) protocols. In these caches, one block is associated with a set of banks. For locating a block, we need to search all the banks that might possibly contain it. Additionally, if a block needs to be evicted from a bank, then we have the option to write it to another bank in its *bank set*.

The main insight into our design is as follows. When we are using optical networks, all the L2 cache banks are conceptually adjacent to the requester. They are all 2-4

cycles away. Hence, we can trivially broadcast a request to a block's *bank set* (*Broadcast* method). However, this method may cause a prohibitive amount of contention. Additionally, it will also increase the dynamic energy consumption of the L2 cache.

Hence, we propose a novel approach. We create a set of overlapping virtual networks ($overlays$) on the physical optical network. Each bank set is assigned an overlay. Messages in the overlay are transmitted using a phased approach, where we multicast a request to a small set of banks within a bank set. If a bank does not contain a copy of the Messages travel along the overlay according to a prespecified set of rules. Whenever, there is a hit in the overlay it sends a message to the requester. Additionally, to decrease contention it sends special control messages to cancel any outstanding messages for the block. The banks in each overlay are decided after a short profiling run, and additionally it is possible to change the structure of each overlay on the fly (not performed in our experiments). To support our overlay network, we propose optimizations to the R-SWMR optical network (originally proposed by Pan et. al. [6]), we design the hardware to form overlay networks, and then we show the design of a protocol that uses the overlay to locate blocks in a set of banks. We call this the *TSI* method. It reduces the dynamic energy of the L2 cache by 20-30%, reduces the port contention, and has a performance comparable to the *Broadcast* scheme.

Both our optical protocols are faster than optimized NUCA cache protocols such as D-NUCA and R-NUCA by 21 and 28% respectively for a suite of Splash2 [9], Parsec [10], and Parboil [11] benchmarks. The maximum speedup obtained was 178% over a baseline design.

The paper is organized as follows. We present related work in Section II, describe the optical network in Section III, explain the operation of the optical overlay network in Section IV, and show the results of architectural simulations in Section V, and finally conclude in Section VI.

II. Background and Related Work

A. Optical Communication

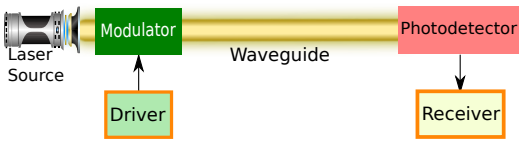

Figure 1. An optical communication system

The basic components of an optical communication system are shown in Figure 1. The light source can either be an off chip laser, or an on-chip VCSEL laser. Unless, the laser is modulated, it continuously generates light at a wavelength of 1550 nm. Subsequently, most networks use a specialized optical device to generate multiple wavelengths out of the original signal. These set of signals are sent to optical stations (nodes that can send of receive data). To send data it is necessary to encode 0s or 1s in the signal. This can be done with the help of an optical modulator (microring resonator) that can either let the optical signal pass through or absorb all of it. The signal is then sent along waveguides, which are thick 1.5 μm wires made of silicon covered with a cladding of silicon dioxide. The optical signal passes along the waveguides. It can branch out to other waveguides via optical splitters that split the signal into two parts (not necessarily equal). The receiver has a photodetector that converts the optical signal to an electrical signal.

To create an optical network, we need to arrange the optical stations, waveguides, splitters, and photo-detectors such that it is possible for all pairs of stations to communicate (non necessarily concurrently). Note that an optical station is a node that gets a part of the laser power, and has the hardware to transmit and receive requests. It is typically connected to a set of cores and cache banks using very short (< 1 cycle delay) electrical links.

There are many different kinds of optical networks with different power, throughput, and latency tradeoffs. We can broadly divide the space of optical networks into two types. The first class of optical networks allows all pairs of nodes to simultaneously communicate with each other [1, 6, 7]. The second class of optical networks allows a limited number of concurrent connections between stations [4, 12, 13, 14]. In these networks, we need to first set up a *light path* between a transmitter node and a set of receiver nodes, we need to allocate wavelengths, and then the transmitter is ready to transmit its data. These classes of networks might appear to be restrictive. With recent advances in all-optical routing [3, 15], the additional latency in these class of networks is small, and they have been adapted to support multicast traffic very well. Our scheme per se is not dependent on the topology of the optical network, and is designed to be agnostic to the network topology. However, for the purpose of evaluation, we use the widely used R-SWMR scheme proposed in [6]. In the basic SWMR (single writer, multiple reader) network, each station is connected to all the other stations. If a station transmits, then all the other stations (or alternatively the stations that need to) can receive the message.

In the R-SWMR scheme, all the receivers are turned off by default to save laser power. When we need to send some data, we turn on the receivers by an out-of-band signalling mechanism (using a dedicated reservation waveguide). Each receiver uses a ring resonator, and a Y junction. If the ring resonator is off, then a part of the optical signal reaches the receiver, otherwise the receiver does not get any part of the signal. This optimization is done to save power. The R-SWMR scheme supports both unicast and multicast

traffic. The reservation assist mechanism typically adds 1-2 cycles of overhead. We propose several optimizations to this protocol in Section III-A.

B. Cache Coherence Protocols Using Optical Networks

Cheng et. al. [16], Kirman et. al. [1], Cianchetti et. al. [17] and Kurian et. al. [18] have proposed electo-optical NOCs for implementing different flavors of snoopy based cache coherence protocols. A recent proposal, PULSE [4], implements a nanophotonic tree based network for implementing the snoopy cache coherence protocol. The scheme tries to predict the sharers of each block such that it is possible to switch off some receivers (that are not sharers). Additionally, Debaes et. al. [19] consider reconfigurable optical networks for cache coherence, and Vantrease et. al. [20] provide the details of a framework for implementing a simple cache coherence protocol that minimizes the number of transient states.

C. Non-uniform Cache Architecture

Most manycore processors have a tiled structure, where each *tile* contains a set of cores, their L1 and I caches, and a set of shared L2 cache banks (the set of L2 banks is also known as a *slice*). In optical networks each tile additionally contains an optical station, and in electrical networks it contains a router to connect to the NOC. In such architectures, the delay to access different banks varies from 2-100 cycles. The S-NUCA [] protocol associates a set of blocks with each bank. It is a very basic protocol that is aware of the fact that different banks have different delays. However, to decrease the bank access delay, it is necessary to have the notion of bank sets, and bring the data closer to the requesting core. Some of the earliest schemes such as the D-NUCA protocol [21, 22] allocate a bank set to each set of blocks. Such NUCA scheme have three aspects – (1) placement, (2) location, and (3) migration. We typically place data as close as possible to the requesting core in the bank set. For location, it is typically necessary to query all the banks in the bank set (sequentially or in parallel). NUCA schemes also migrate frequently accessed blocks towards the requesting core (to minimize delay). Subsequently, this basic protocol has been thoroughly optimized by a host of proposals [22].

Later proposals have considered the nature of data (shared or private), access frequency, and the contention in banks [23]. We compare our results with the R-NUCA [24] scheme, which maps private data very close to the requesting core, maps read-only data in a set of overlapping bank sets, and maps shared data to fixed banks using a static scheme. Some protocols additionally consider block replication [25, 26]. Since we use optical networks, we have a different set of constraints such as managing contention, and achieving a tradeoff between unicast and multicast messages. These issues are not present in electrical networks.

III. Optical Network

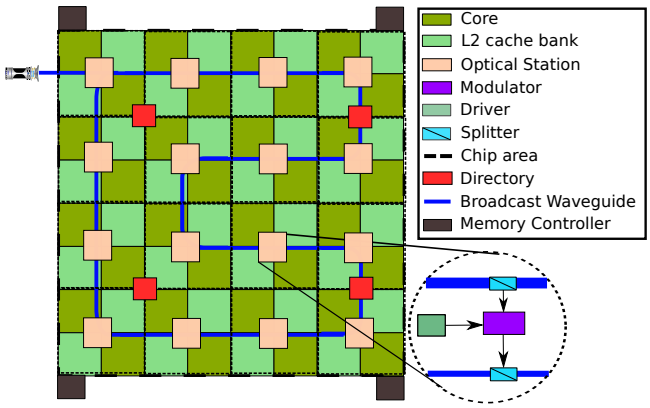

Figure 2. Overview of the system

The proposed system is shown in Figure 2. We have 32 cores, each with a private L1 cache. The L2 is a shared NUCA cache, with 32 cache banks. All these elements are arranged as a set of tiles, and the cores and cache banks are arranged as a chess board. Each tile consists of 2 cores, and 2 cache banks. We use a distributed directory-based coherence protocol, with the 4 directories placed as shown in Figure 2. We place one optical station in each tile.

A. Optical Communication Infrastructure

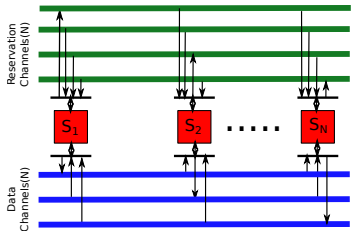

Figure 3. Reservation assisted SWMR

1) Reservation Channel: We have a total of $N = 16$ optical stations and they are connected using a serpentine structured waveguide as shown in Figure 2. Before sending a message, it is necessary to turn on the receivers of the destination stations, and also inform them about the type of the message such that they know for how long they need to keep themselves turned on. To setup the destinations, the transmitter needs to broadcast a bit vector that is N bits long to all the stations. Out of these N bits, we use 1 bit to indicate the type of the message (data or control), and use the rest of the $N - 1 = 15$ bits to indicate the set of receivers that need to be turned on. If a receiving station finds its corresponding bit to be 1, then it turns on the receiver, and also reads the type of the message. Now, we can send 64 different wavelengths using DWDM (dense wavelength division multiplexing) on the same waveguide.

Thus, in this case, 4 stations can share the same waveguide and send 16 bits each using different wavelengths to the rest of the stations. Consequently, for implementing the reservation waveguide we need a bundle of 4 waveguides. In comparison, Firefly [6] uses unicast messages in its reservation waveguides.

The additional bit, *Type of Message -TM Bit*, denotes whether the upcoming message is a control or data message. Depending on this bit, the receiver stays turned on for 3 or 7 cycles(including propagation delay) respectively. According to the authors of the Firefly [6] scheme, the area and power overhead of these reservation channels are negligible.

2) Data Channel: We use 64 wavelengths per waveguide. Thus, we can send 64 bits (1 bit per wavelength) simultaneously. Additionally, we transfer data using a double pumped clock (transfer data at both the rising and falling edge similar to Corona [7]). We can thus transmit (128 bits or 16 bytes) of data per CPU clock cycle. Note that optical channels can support signal frequencies up to 40 GHz, and thus the frequency is not the bottleneck here. We thus require 1 waveguide per station for transmitting data. Since 4 stations share a reservation waveguide, we in effect require 1.25 waveguides per station.

As discussed earlier, there can be two types of messages – *data* and *control*. The cache line width in our architecture is 64 bytes and we can thus send a cache line in 5 flits, which includes the head flit. The total bandwidth of the system is equal to $2 \times 3.4GHz \times 16 \times 64$, which is 0.87 TBPS. Control messages are single flit messages containing 86 bits, which can be sent in a single cycle. The optical parameters that we have considered are shown in Table I.

Optical Parameters	
Wavelength (λ)	$1.55\mu m$
Width of waveguide (W_g)	$3\mu m$
Slab height	$1\mu m$
Rib height	$3\mu m$
Refractive Index of $SiO_2(n_r)$	1.46
Refractive Index of Si (n_c)	3.45
Input Driver Power	76 μW
Output Driver Power	166 μW
Insertion Coupling Loss	50%
Output Coupling Loss	13%
Photodetector quantum efficiency	0.8 A/W
Photodetector minimum power	36 μW swing (4 μW(0), 40 μW(1))
Combined transmitter and receiver delay	180-270 ps
Optical propagation delay	7 ps/mm
Electrical propagation delay	35 ps/mm
Bending Loss	1 dB
Waveguide Loss	1 dB
Coupler Loss	1 dB
Photodetector	0.1 dB
Wall Plug Efficiency	30 %
Splitter Loss	0.36 dB
Ring Heating	$26\mu W/ring$
Ring Modulation	$500\mu W/ring$

Table I
OPTICAL PARAMETERS [1, 4, 27]

IV. OPTICAL OVERLAY

In implementations of NUCA caches with electrical interconnects, researchers have proposed the notion of a *bank set*, which is a set of banks that might contain a copy of the block. Different banks in the bank set have different access delays measured from the requesting node. Consequently, NUCA caches try to bring data *closer* to the core by migrating data between banks. Most papers have proposed bank sets that are a single row or column of banks. However, the case is very different in the case of optical networks. The situation is similar to wide area networks where latency is not particularly a concern because there are other factors that preponderate over the network latency.

In our optical network, the maximum latency between any pair of stations is 2 cycles; thus, all the stations can be thought of to be equidistant from each other. We can thus create any kind of an overlay network (network interposed over a network) for the bank set (see Figure 4). It need not be a row or column. It can be a tree, or any kind of complex graph even consisting of nodes in different corners of the chip. It is essentially a logical layer consisting of cache banks, where they are arbitrarily connected to each other using equi-weighted edges.

A. Details of the Overlay Network

We performed several experiments with different numbers of cache banks(4, 8 and 16) in an overlay, and also with different kinds of overlay networks (trees and rings). We found the ring overlay with 8 cache banks to be the best. In this experiment, requests proceed bidirectionally along the edges of the ring (1 hop in each direction every 2 cycles). For a representative benchmark, *Water-spatial* [9], we observed speedups of 20% and 30% in the case of an overlay with 8 nodes as compared to overlays with 16 and 4 nodes respectively. A small overlay network (4 nodes) has a higher miss rate because of the small size of the bank set. A large overlay network has higher contention, and thus has resultant slowdowns. We did not find any advantages with tree based overlays in our system. Here, all requests are sent to the root, and then they descend the edges of the tree (1 level every 2 cycles).

We also experimented with 4 statically determined bank sets of size, 8. We observed that some bank sets are highly accessed as compared to others, which remain inactive most of the time (explained later in Section V). The banks under high pressure offer poor search latency, and suffer more evictions which reduces the hit-rate.

Hence, our aim is to design a set of overlay networks that can accommodate the different levels of contention (which was measured to be non-uniform across bank sets). We proceed as follows. At the outset, we do not create an overlay for the running application. Every block in the L2 cache is uniquely mapped to a bank (referred to as its *home bank*). The mapping is decided based on its physical address.

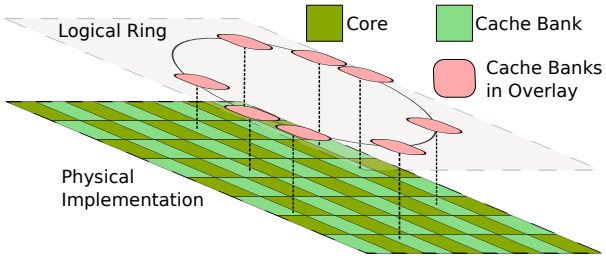

Figure 4. Logical ring based overlay

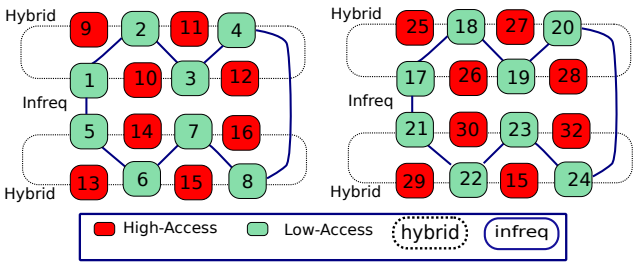

Figure 5. Structure of the $hybrid$ and $infreq$ overlays

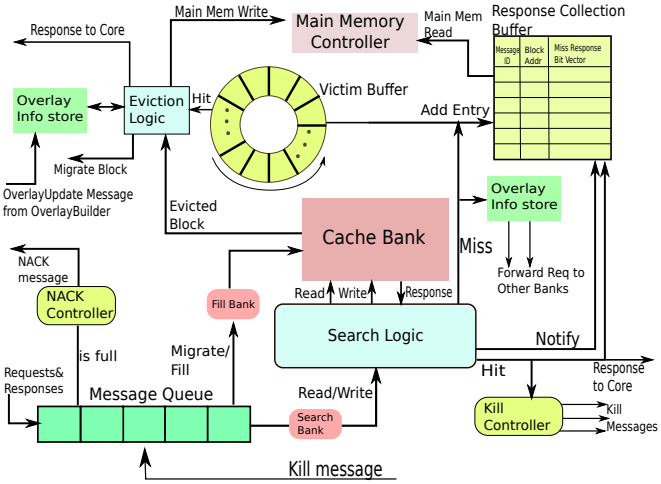

Figure 6. Home bank controller

Initially, every block can only be stored in its home bank. We run our benchmarks till they reach steady state (in the case of parallel benchmarks execute 100 million instructions in the parallel section). We subsequently, collect the access counts of each bank, and divide them into two sets namely *high-access* and *low-access*.

Now, our aim is to construct a set of overlays where data can migrate between different banks of the overlay (similar to traditional NUCA schemes). Overlays can also be overlapped with each other. Our aim is to homogenize the access counts across the blocks. We propose one such scheme here, which was empirically found to produce good results for our spectrum of benchmarks.

We create two kinds of overlays – $hybrid$ and $infreq$ (infrequent). The logical division of banks into these overlays is shown in Figure 5. A $hybrid$ overlay consists of 4 high-access and 4 low-access banks. We have 4 of such overlays. We have two more $infreq$ overlays, which consist of 8 low-access banks each. Every low-access bank is shared between a single $hybrid$ overlay and an $infreq$ overlay. This is done to bring an uniformity in the access counts for each bank.

B. Algorithm to Build the Overlay

The process of the creation of the set of overlays at the moment happens only once for an application. Instead of dedicating logic in hardware to build the overlay, we assume that the bank access counts are written to a dedicated location in memory, and the hardware invokes a custom software routine, or a firmware routine to compute the overlays as shown in Figure 5. Each bank contains a list of all the other banks in its overlay, and also a list of its neighbors. The details of the overlay are saved in the structure *Overlay Info Store* (see Section IV-C).

C. Home Bank Controller

Each bank in the overlay has a home bank controller as shown in Figure 6. Each *HBC* has its own Message Queue(*MQ*). All the requests are enqueued in it. Messages are enqueued and dequeued from this structure every 2 cycles (our pipelined cache bank accepts two new requests every 2 cycles). If there is a L1 cache miss, the request for the data block is sent to the HBC optically where it is enqueued in the MQ. If no space is left in the MQ, then a *NACK* message is sent back to the requester. The requester retries after 2 cycles, and follows an exponential backoff scheme if it receives another *NACK* message. The requests in the MQ are processed in FIFO order. A processed request is then sent to the search logic of the HBC. If there is a hit in the home bank, then a *Response* is sent to the core. Otherwise the request is forwarded to the home bank's successors in its overlay. The details of the searching mechanisms are explained in Section IV-D. The *Eviction logic* along with the *Overlay info store* store the rules for eviction, and the structure of the overlay respectively.

D. Searching Mechanisms

We implemented two types of searching mechanisms 1) Two side Incremental and 2) Broadcast.

1) Two side incremental (TSI): In this mechanism(see Figure 7), we first search in the home bank. If there is a miss, then the home bank allocates an entry in the *RCB* and sends requests to its left and right successors (two branches of the ring). The left and right successors forward the requests (in

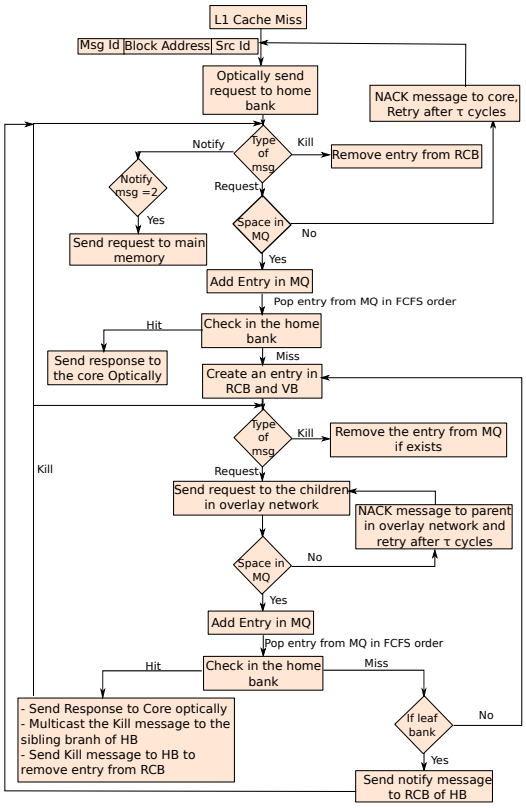

Figure 7. *TSI* Protocol

case of a miss) to their successor banks and so on. Here, we are searching simultaneously on both sides (branches) of the ring. Note that all the messages have the same ID (discussed later). If there is a hit in any of the successor banks, it stops forwarding the request, and sends a *Response* message to the requesting core and a set of *Kill* messages in the opposite branch of the ring. A *Kill* message accesses the MQ in the destination cache bank and dynamically invalidates any copies of the original message. This helps us in controlling contention in the rest of the cache bank. Once there is a hit, we ensure that we do not search any more cache banks. Additionally, a *Hit* message is sent to the *RCB* to free the entry. If there is a miss, then the request is sent to the successor bank. This process is continued till the last bank is reached. If a miss occurs at the last bank, a *Miss* message is sent to the *RCB* of the home bank. If the *RCB* of the home bank receives two *Miss* messages, it declares a L2 cache miss and requests the data block from main memory.

2) Broadcast: We have implemented a broadcast based search mechanism that searches all the cache banks if a miss occurs at the home bank. This gives us a lower access latency at the cost of higher contention at each bank.

E. Additional Structures

1) Victim Buffer (VB): Note that we can have race conditions in our search protocol.

- **False Miss** - Let us assume that a block is originally present in bank A. At the same time another bank C decides to search for the block. It first searches in bank B, and it does not find the block there. It subsequently, sends the request, req, to the successor of B, which is A. Exactly at that time, bank A decides to evict the block, and put it in bank B. After A has removed the block, it gets the request, req. It also returns a miss. req travels to the end of the ring, and concludes that the block is not present in the L2 cache, which is false. A request is sent to main memory.
- **Multiple copies problem** - Because of a false miss, a request is sent to main memory, which places the block again at the home bank although the data block was present in some other cache bank.

The main issue is that we do not have a method of querying the contents of messages that are in flight. The first option is to add a mechanism to query the contents of messages in the MQ. However, this feature was increasing the number of ports in the MQ, and increasing its complexity also. Hence, we introduce a victim buffer (VB) at the node that evicts the block. It has $M + K$ entries, where M is the size of the MQ. Unless a *NACK* message is received, the evicted block will be written to the successor bank after at the most M cycles. We arrive at M cycles by considering the fact that we dequeue two messages from the MQ every 2 cycles. K is the worst case network delay (3 cycles). We are guaranteed to find the block in either bank A or B. Every request needs to check the cache bank as well as the VB.

2) Response-Collection Buffer: The *RCB* is used in each bank to collect the *Miss* messages coming from the cache banks. When there is a miss at the home bank, it creates an entry for the requested data block in its *RCB*. In the *TSI* protocol, when a miss has been identified in one branch of the ring overlay, the last bank of that branch notifies the home bank by sending a *Miss* message. If the *RCB* receives two such messages, then it implies that there were misses in both the branches. The entry is removed if the *RCB* receives either a *Hit* message or two *Miss* messages. The size of the RCB is also determined experimentally. The maximum size it requires is 128 entries. Each entry contains the *Message ID*, block address, and the *Miss Response Bit Vector (MRBV)*. The *Message ID* is a 32 bit number, which is generated by the home bank. The 5 MSB bits indicate the bank number, and the 27 LSB bits are a per bank sequence number.

For the *broadcast* protocol, the *MRBV* contains a bit vector whose size is equal to the number of banks in the overlay. In the case of *TSI*, it contains two bits (one for each branch of the ring). A bit in this vector is set to 1 whenever a *Miss message* is received from the bank corresponding to this bit. The *RCB* is also responsible for requesting a data block from main memory. When all bits in the *MRBV* are set to 1, the *RCB* sends the request to main memory. All the

communication between banks, buffers and cores happens though the optical network.

F. Migration Policy

The placement policy in our architecture is the same as that of S-NUCA [21]. The least significant set-index bits of the address are used to statically determine the home bank for a data block. When an eviction of a data block occurs from a bank, then instead of evicting it from the L2 cache, we migrate the data to the next neighboring bank (follow an clockwise order in the ring). In our ring overlay, we created a migration/spilling mechanism in which each block when evicted from the home bank is given $n-1$ chances in subsequent evictions to remain in the L2 cache, where n is the number of banks in one ring overlay. When the block reaches the last bank, it is finally evicted from the L2 cache and written into the main memory.

The associated circuitry is contained in the *Eviction logic* block in Figure 6. The circuitry accesses the *Overlay info store* to determine whether the bank from where the block is evicted is the last bank, and also to collect the id of the successor bank. Our main aim in designing the overlays was to use the under-utilized space in low-access banks for the evicted blocks of high-access banks. Consequently, we have designed hybrid overlays to alternately contain high-access and low-access banks (arranged as a ring). If a block gets evicted from a high-access bank, then it goes to a low-access bank. No two high-access banks are adjacent to each other in the logical ring. This design helps in absorbing the additional capacity requirements of high-access banks. For the $infreq$ overlay, all banks are of the type low-access, and the blocks migrate among these low-access banks only.

G. Message Format

On a broader level we have following two kinds of messages for communication : *Data Access* messages (Request, Response) and *Control* messages *(NACK, Kill, Hit, Miss)*, as shown in Figure 8. We use six types of messages, two for data access and four control messages. A 3-bit field is alloted in each message to specify the type. The structure of the messages is shown in Figure 8. In the control message description, the core-id denotes the requesting core that requested the data block. In our system, each cache bank and core is assigned an ID. The source-id or the destination-id can be either a core-id or a bank-id.

V. EVALUATION

A. Experimental Setup

Table II shows the experimental setup of our simulations. We use the cycle accurate Tejas architecture simulator [28] for simulating the benchmarks. We use 32 OOO cores, 32 cache banks, and a Torus based electrical NOC for comparison with RNUCA and SNUCA. and a We used a suite of Parsec [10], Splash-2 [9] and Parboil [11] benchmarks

Parameter	**Value**	**Parameter**	**Value**
Cores	32	Technology	18 nm
Frequency	3.4 GHz		
Pipeline			
Retire Width	4	Integer RF (phy)	160
Issue Width	6	Float RF (phy)	160
ROB size	168	Branch Predictor	Tournament
IW size	54		(Pag-Pap)
LSQ size	64	Bmispred penalty	14 cycles
iTLB	128 entry	dTLB	128 entry
Integer ALU	4 units	Int ALU latency	1 cycle
Integer Mul	1 unit	Int Mul latency	2 cycles
Integer Div	1 unit	Int Div latency	4 cycles
Float ALU	2 units	FP ALU latency	2 cycles
Float Mul	1 unit	FP Mul latency	4 cycles
Float Div	1 unit	FP Div latency	8 cycles
L1 i-cache, d-cache			
Write-mode	Write-back	Block size	64 bytes
Associativity	4	Size	32 kB
Latency	2 cycles	MSHRs	32
Directory	fully mapped, 4-distributed, MOESI, total 4096 entries, 8-way		
Shared L2			
Write-mode	Write-back	Block size	64 bytes
Associativity	8	# banks	32
Latency (per bank)	8 cycles	Bank size	256 KB
Main Memory			
Latency	250 cycles	Mem. controllers	4
Electrical NOC			
Topology	2-D Torus	Routing Alg.	X-Y
Flit size	16 bytes	Hop-latency	1 cycle
Routing delay	2/3 cycles	# Virt. channels	4
(w/wo bypassing)		Buffers/port	8
Auxiliary structures (size in number of entries)			
RCB	128	VB	20
MQ	16		

Table II
SIMULATION PARAMETERS

Application	Input size
PARSEC (simlarge)	
Blackscholes	64KB options
Bodytrack	4 cameras, 4 frames, 4,000 particles, 5 annealing layers
Fluidanimate	300,000 particles, 5 frames
Streamcluster	16KB input points, 16KB points, 128 point dimensions
Swaptions	64 swaptions, 20,000 simulations
Splash-2	
Barnes	16KB particles
Fmm	8KB particles
Lu	512 x 512 matrix (lu contiguous)
Radiosity	batch, largeroom
Water-nsq	512 molecules (water nsquared)
Water-sp	512 molecules (water spatial)
Parboil	
Histo	256 x 8192 matrix
Stencil	512 x 512 x 64 3D matrix

Table III
CONFIGURATION OF BENCHMARKS

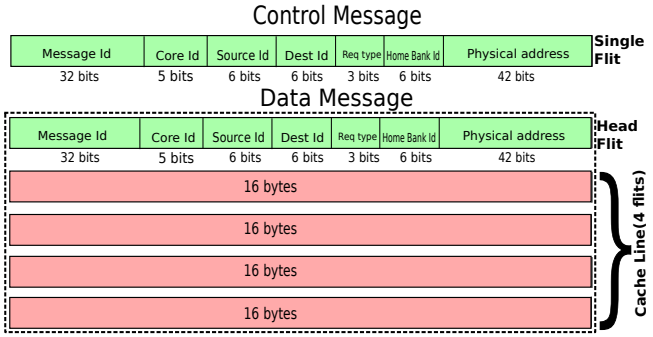

Figure 8. Message format

(details shown in Table III). We use the optical network for all the inter-tile traffic in the experiments with optical NOCs.

B. Benchmark Characterization

Before discussing the results, let us characterize the behavior of our benchmarks. We mainly focus on four parameters: L2 requests per 1000 instructions, L2 hit-rate, average number of banks searched using the *TSI* mechanism, and the number of *Kill* messages generated per L2 hit, along with percentage of *Kill* message hits in the MQ. Table IV shows the results.

The number of L2 requests per 1000 instructions helps in determining the effect of the L2 cache on the final performance of the benchmark. The last two columns in Table IV show statistics related to the *TSI* protocol. The average banks searched is the number of banks that are accessed in an overlay per L2 request. As shown, most of the values lie between 5 to 6. This means that messages on both the branches traverse a large part of the bank set. This is why we need *Kill* messages to cancel the request in the MQ, such that there is reduced contention at the ports of the cache bank. The number of *Kill* messages range from 1 to 3, and they individually have a hit rate of 20-30%. This effectively reduces contention by a third to a quarter.

C. Performance

Figure 9 shows the performance of our *Overlay NUCA* scheme with the standard electrical implementations (static NUCA, dynamic NUCA, and R-NUCA). We have shown two types of overlay implementations here - *Broadcast* and *Two Side Incremental*. Other than $Stencil$ the geometric mean speedup of the $broadcast$ protocol is $2-3\%$ better than *TSI* . However, as compared to S-NUCA, D-NUCA, and R-NUCA, *TSI* has a speedup of 50%, 18%, and 24% respectively. In *Swaptions*, and *Water-sp* the speedup of *TSI* over S-NUCA is 161% and 167% respectively. For $Water-nsq$, $Radiosity$, Lu and Fmm the speedups are low because there are very few requests that go to the L2 cache. The mean speedup of the R-NUCA scheme over S-NUCA is 20% and for the *TSI* implementation over S-NUCA is 50%.

D. L2 Hit Rate and Latency

Let us first look at the filtering effect of the home bank in Figure 12 that shows the percentage of accesses that hit in the home bank. We observe that across benchmarks the hit rate varies from 30-50% with some very high values for $Blackscholes$ and $Streamcluster$ ($> 80\%$). This explains the low speedup of optical schemes for $Blackscholes$ and $Streamcluster$. Let us now consider the behavior of the accesses that are sent to the overlay network. Figure 11 shows the normalized hit latency (normalized to S-NUCA) for the different configurations. We observe that S-NUCA and D-NUCA have similar hit latencies. The normalized hit latency of R-NUCA is between 0.4-0.8, and the latency of the optical schemes is typically between 0.2-0.55. Let us now study the hit rate in Figure 11. The hit rate of S-NUCA is definitely far inferior to that of other schemes (typically 70-97% lower). However, for rest of the four schemes, the hit rates are in the same ballpark. We can thus conclude that the performance advantage is accrued through a reduction in the hit latency.

E. Contention and Number of Accesses

Our main aim of introducing the *TSI* protocol was to reduce the dynamic energy of the L2 cache. Since the dynamic energy is proportional to the number of accesses, we focused on reducing the number of accesses in the *TSI* protocol. Figure 13 shows the number of accesses for all the configurations. We can see that the performance improvement in *Broadcast* is not more than 2-3% as compared to *TSI* , but the dynamic energy savings in *TSI* is approximately 20-30% across all the benchmarks. Both our optical schemes perform poorly as compared to the other three schemes. On an average, they have 3.0, 1.47 and 3.17 times more accesses than the S-NUCA, D-NUCA, and R-NUCA schemes respectively. Among the rest of the electrical schemes, D-NUCA is the worst. *Swaptions* is an outlier. This is because the number of hits in home banks is low and we have to search in the non-home bank caches (see Figure 12).

VI. Conclusion and Future Work

In this paper, we proposed and evaluated a set of protocols for designing a flexible and efficient access mechanism for large shared L2 caches using optical networks. When we use optical networks, the latency between banks decreases dramatically, and thus we can conceptually consider all banks to be equidistant from each other.

The crux of our idea is to create a set of overlapping virtual networks called $overlays$ on top of the physical optical network. Each overlay contains a set of banks. A

Application	% I-cache hit-rate	% D-cache hit-rate	% Directory hit-rate	L2 reqs per 1000 instructions	% L2 hit-rate with TSI	Avg. Banks searched with TSI	Kill messages per L2 Hits (% Hits) with TSI
Parsec							
Blackscholes	99.99	95.69	35.15	8.74	88.47	5.39	1.16 (21.47)
Bodytrack	99.99	99.10	45.01	2.98	77.24	5.58	2.14 (22.23)
Fluidanimate	99.99	91.20	23.48	7.35	37.92	4.91	1.20 (25.60)
Streamcluster	99.95	81.51	27.50	4.18	38.48	5.27	1.62 (23.86)
Swaptions	99.96	78.04	10.82	34.52	99.40	5.63	2.53 (19.45)
Splash-2							
Barnes	99.98	92.33	78.55	4.12	97.89	5.37	2.12 (24.05)
Fmm	99.94	93.52	56.80	2.35	85.84	5.13	2.07 (25.30)
Lu	99.99	94.02	41.80	3.25	67.52	5.77	1.45 (20.96)
Radiosity	99.99	97.58	93.80	0.31	70.82	3.84	1.11 (32.66)
Water-nsq	99.96	94.48	71.07	2.95	36.50	5.37	1.77 (23.37)
Water-sp	99.95	89.15	11.54	9.57	98.56	5.92	2.84 (20.44)
Parboil							
Histo	99.99	69.19	95.55	2.75	55.42	6.31	1.94 (16.31)
Stencil	99.99	91.35	80.25	11.99	62.25	5.21	2.14 (23.12)

Table IV
MEMORY HIERARCHY & OVERLAY STATISTICS

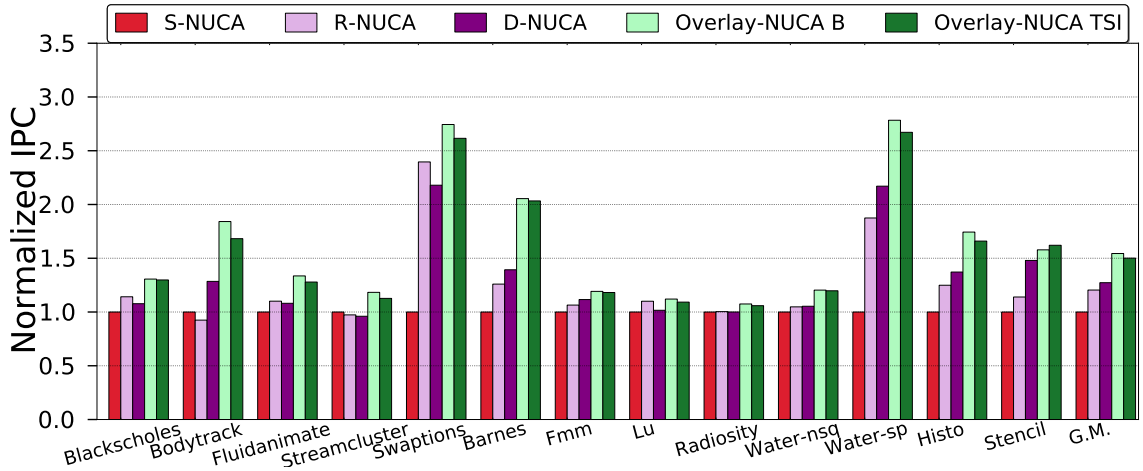

Figure 9. Normalized IPC

naive access protocol sends a request to a dedicated home bank, and if there is a miss in the home bank, then the request is broadcast to all the banks in the overlay. However, this approach is not appropriate in terms of energy, and bank contention. Hence, we propose a novel approach *TSI* , that sends a limited number of messages along the overlay in a sequential fashion. This limits the wasted work. To support our scheme, we design the hardware to create and maintain overlays dynamically, design protocols for locating blocks on an overlay, and also propose optimizations to existing optical networks for efficiently supporting our overlay networks.

We obtain a mean speedup of 50% over the static NUCA (S-NUCA) scheme using our approaches. The performance difference between the broadcast and *TSI* schemes is minimal (2-3%). However, the *TSI* reduces the number of accesses (proportional to the dynamic energy) by roughly 20-30%.

REFERENCES

[1] N. Kirman, M. Kirman, R. K. Dokania, J. F. Martinez, A. B. Apsel, M. A. Watkins, and D. H. Albonesi, "Leveraging optical technology in future bus-based chip multiprocessors," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, 2006.

[2] M. Mohamed, Z. Li, X. Chen, and A. R. Mickelson, "Hermes: A hierarchical broadcast-based silicon photonic interconnect for scalable many-core systems," *CoRR*.

[3] N. Kirman and J. F. Martínez, "A power-efficient All-optical On-chip Interconnect Using Wavelength-based Oblivious Routing," ser. ASPLOS XV, 2010.

[4] R. Morris, E. Jolley, and A. Kodi, "Extending the performance and energy-efficiency of shared memory multicores with nanophotonic technology," *Parallel and Distributed Systems, IEEE Transactions on*, 2014.

[5] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel

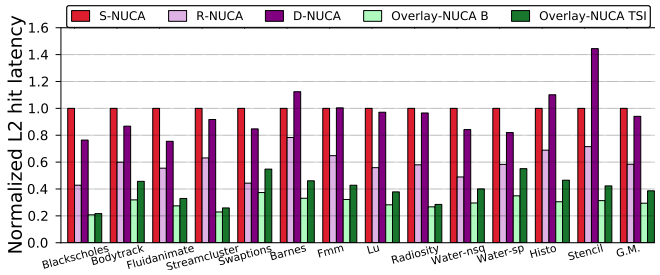

Figure 10. Normalized average hit latency

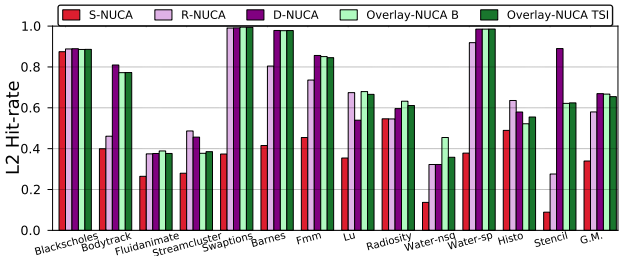

Figure 11. L2 hit rate

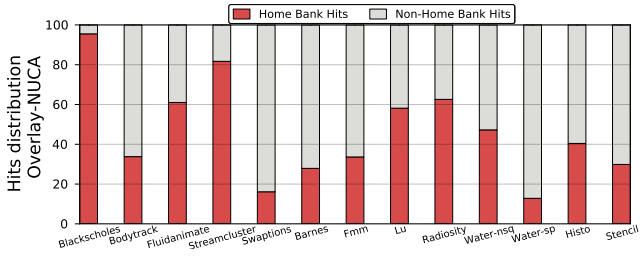

Figure 12. Hits in the home bank

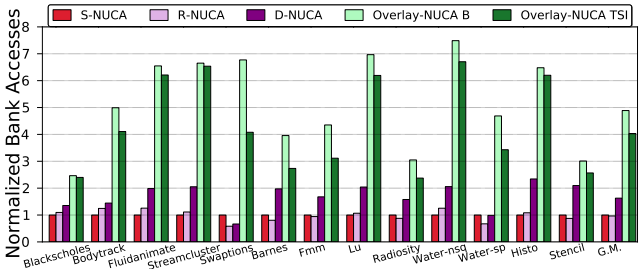

Figure 13. Normalized total bank accesses

routers for on-chip networks," *ACM SIGARCH Computer Architecture News*, 2004.

[6] Y. Pan, P. Kumar, J. Kim, G. Memik, Y. Zhang, and A. Choudhary, "Firefly: Illuminating Future Network-on-Chip with Nanophotonics," ser. ISCA, 2009.

[7] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. G. Beausoleil, and J. H. Ahn, "Corona: System Implications of Emerging Nanophotonic Technology," ser. ISCA '08, 2008.

[8] N. Binkert, A. Davis, M. Lipasti, R. Schreiber, and D. Vantrease, "Nanophotonic Barriers," in *Workshop on Photonic Interconnects & Computer Architecture (in conjunction with MICRO 41)*, 2009.

[9] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *ACM SIGARCH Computer Architecture News*, 1995.

[10] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: characterization and architectural implications," in *PACT*, 2008.

[11] J. A. Stratton, C. Rodrigues, I. J. Sung, N. Obeid, L. W. Chang, N. Anssari, G. D. Liu, and W. W. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," *Center for Reliable and High-Performance Computing*, 2012.

[12] Y. Pan, J. Kim, and G. Memik, "Flexishare: Channel sharing for an energy-efficient nanophotonic crossbar," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE, 2010, pp. 1–12.

[13] S. Le Beux, J. Trajkovic, I. O'Connor, G. Nicolescu, G. Bois, and P. Paulin, "Optical ring network-on-chip (ornoc): Architecture and design methodology," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, 2011.

[14] A. Zulfiqar, P. Koka, H. Schwetman, M. Lipasti, X. Zheng, and A. Krishnamoorthy, "Wavelength stealing: an opportunistic approach to channel sharing in multi-chip photonic interconnects," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, 2013.

[15] S. Koohi and S. Hessabi, "All-optical wavelength-routed architecture for a power-efficient network on chip," 2012.

[16] L. Cheng, N. Muralimanohar, K. Ramani, R. Balasubramonian, and J. B. Carter, "Interconnect-aware coherence protocols for chip multiprocessors," *ACM SIGARCH Computer Architecture News*, 2006.

[17] M. J. Cianchetti, J. C. Kerekes, and D. H. Albonesi, "Phastlane: a rapid transit optical routing network," in *ACM SIGARCH Computer Architecture News*, 2009.

[18] G. Kurian, J. E. Miller, J. Psota, J. Eastep, J. Liu, J. Michel, L. C. Kimerling, and A. Agarwal, "ATAC: A 1000-Core Cache-Coherent Processor with On-chip Optical Network," ser. PACT '10, 2010.

[19] C. Debaes, I. Artundo, W. Heirman, J. Dambre, K. B. Viet, and H. Thienpont, "Architectural study of the opportunities for reconfigurable optical interconnects in distributed shared memory systems," in *Proceedings Symposium*. Citeseer, 2004.

[20] D. Vantrease, M. Lipasti, and N. Binkert, "Atomic coherence: Leveraging nanophotonics to build race-free cache coherence protocols," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*.

[21] K. Changkyu, D. Burger, and S. Keckler, "Nonuniform cache architectures for wire-delay dominated on-chip caches," *Micro, IEEE*, 2003.

[22] C. Kim, D. Burger, and S. W. J. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," in *ASPLOS*, 2002.

[23] A. Huang, J. Gao, W. Guo, W. Shi, M. Zhang, and J. Jiang, "PSA-NUCA: A pressure self-adapting dynamic non-uniform cache architecture," in *NAS*, 2012.

[24] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive nuca: near-optimal block placement and replication in distributed caches," in *ISCA*, 2009.

[25] J. Merino, V. Puente, and J. A. Gregorio, "Esp-nuca: A low-cost adaptive non-uniform cache architecture," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE, 2010.

[26] B. M. Beckmann, M. R. Marty, and D. A. Wood, "Asr: Adaptive selective replication for cmp caches," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*. IEEE, 2006, pp. 443–454.

[27] G. T. Reed, *Silicon Photonics: The State of the Art*. John Wiley & Sons, 2008.

[28] G. Malhotra, P. Aggarwal, A. Sagar, and S. R. Sarangi, "ParTejas: A parallel simulator for multicore processors," in *ISPASS*, 2014.