WiDir: A Wireless-Enabled Directory Cache Coherence Protocol

Antonio Franques, Apostolos Kokolis, Sergi Abadal†, Vimuth Fernando, Sasa Misailovic, Josep Torrellas
University of Illinois at Urbana-Champaign †*Universitat Politècnica de Catalunya*
{franque2, kokolis2}@illinois.edu, abadal@ac.upc.edu, {wvf2, misailo, torrella}@illinois.edu

***Abstract*—As the core count in shared-memory manycores keeps increasing, it is becoming increasingly harder to design cache-coherence protocols that deliver high performance without an inordinate increase in complexity and cost. In particular, sharing patterns where a group of cores frequently reads and writes a shared variable are hard to support efficiently. Hence, programmers end up tuning their applications to avoid these patterns, hurting the programmability of shared memory.**

To address this problem, this paper uses the recently-proposed on-chip wireless network technology to augment a conventional invalidation-based directory cache coherence protocol. We call the resulting protocol *WiDir*. *WiDir* seamlessly transitions between wired and wireless coherence transactions for a given line based on the access patterns in a programmer-transparent manner. In this paper, we describe the protocol transitions in detail. Further, an evaluation using SPLASH and PARSEC applications shows that *WiDir* substantially reduces the memory stall time of applications. As a result, for 64-core runs, *WiDir* reduces the execution time of applications by an average of 22% compared to a conventional directory protocol. Moreover, *WiDir* is more scalable. These benefits are obtained with a very modest power cost.

***Index Terms*—Multicore, Wireless Network on chip, Directory cache coherence protocol**

I. Introduction

Exploiting a combination of innovative chip manufacturing techniques and reduced semiconductor feature sizes, computer manufacturers continue to increase the core counts of processor chips. With more on-chip cores and bigger caches, systems can run bigger problems with limited cost increase. For example, Ampere's Altra [1] uses 7nm technology to support up to 80 ARM cores and a 32 MB Last-Level Cache (LLC) with a coherent mesh interface on a single die. As another example, AMD's EPYC 7742 processor [2] uses a chiplet organization and 7nm technology to support up to 64 cores (2-way SMT) and a 256 MB LLC with the Infinity Fabric interconnect. Further, Intel's Xeon Platinum 9282 processor [3] uses a dual-die package and 14nm technology to host up to 56 cores (2-way SMT) and a 77 MB LLC connected with a mesh interconnect. In the near future, it is likely that on-chip core counts will continue to increase.

For these manycores, shared memory is the most popular programming and execution model. The reasons are shared-memory's ease of programming, well-developed existing algorithms, and widespread libraries such as POSIX threads and OpenMP. To support shared memory at this scale, designers build directory-based hardware cache-coherence protocols [4]. Designing such protocols is an arduous process. Importantly, as the core count goes up, it is becoming harder to engineer cache-coherence protocols that deliver high performance without an inordinate increase in complexity and cost.

As an example, consider patterns where a group of cores frequently reads and writes a shared variable. Coherence protocols rely on invalidations to keep coherence [5]–[7], and do not support these patterns efficiently. As soon as a core writes the variable, all the other sharers get invalidated, and any subsequent read by another core suffers a costly cache miss. Sending invalidations and then re-reading the data creates long-latency transactions in a large on-chip interconnect. Update-based protocols [8] are not the solution either, as they have shortcomings for many common patterns, which has prompted designers to eschew them. As a result, if programmers want to attain high performance, they have to carefully tune the sharing behavior of their applications, ensuring that patterns like the ones mentioned appear infrequently.

Wireless Network on Chip (WNoC) is a new technology that has recently seen a lot of interest [9]–[16]. In WNoCs, each core or group of cores in a manycore has a transceiver and an antenna, which it uses to communicate wirelessly with other cores. While the bandwidth of a WNoC is limited (only one or a few messages can be transmitting at a time), a WNoC supports low-latency transfers, since a message can cross a large manycore in about 5ns [17]. Further, WNoCs naturally support update multicasts and broadcasts. As a result, WNoCs have been proposed to speed-up applications on manycores. For example, WiSync [18] uses it to speed-up synchronization, while Choi *et al.* [19] use it to accelerate CNN training in CPU-GPU heterogeneous architectures, and Replica [20] uses it to speed-up the execution of communication-intensive and approximate computations.

An intriguing question is whether a conventional, wired cache coherence protocol can be augmented to use a wireless network so that patterns like the ones described above can be supported efficiently. Frequent read-write sharing by multiple cores can indeed be efficiently supported by a WNoC: writes update all sharers without complicated multi-hop coherence transactions or lengthy message routing; reads can access data locally. However, to implement a complete coherence protocol, one needs to carefully combine wired and wireless transactions in a seamless manner.

In this paper, we present such a protocol, which we call *WiDir*. *WiDir* extends an invalidation-based directory cache coherence protocol with some wireless transactions. The goal is to efficiently support frequent read-write sharing within a group of cores — which has largely eluded conventional coherence protocols. *WiDir* seamlessly transitions between

wired and wireless transactions for the same datum based on the access patterns in a programmer-transparent manner. The end result is higher performance without the need to tune applications and hurt programmability.

For the design in this paper, we start with a conventional invalidation-based MESI protocol over a wired NoC. For a given cache line, *WiDir* uses this protocol when there are few sharers. Then, when the number of sharers goes over a certain threshold, the line transitions to the Wireless (W) state, where sharing between cores uses wireless transactions. When the number of sharers falls back down to the threshold, the line goes back to using MESI over the wired NoC. The paper describes the *WiDir* protocol transitions in detail.

Our evaluation using simulations running SPLASH and PARSEC applications shows that *WiDir* substantially reduces the memory stall time of the applications. As a result, for 64-core runs, *WiDir* reduces the execution time of applications by an average of 22% compared to plain MESI. Moreover, *WiDir* is more scalable than MESI. These benefits are obtained with a very modest power cost.

Overall, the contributions of this paper are:

- The novel use of WNoCs to enhance a cache coherence protocol. This use further builds up the case for WNoCs.
- The *WiDir* directory-based hardware cache-coherence protocol that seamlessly combines wired and wireless transactions.
- An evaluation of *WiDir*.

II. Background & Motivation

A. Motivation for Wireless Network On Chip

Traditionally, a wired Network on Chip (NoC) uses a packet-switched interconnection fabric with each processor connected to a router as shown in Figure 1. Routers enqueue packets, compute routes, arbitrate, and dequeue packets in each hop towards the destination, incurring delays and energy consumption. To connect the routers, a mesh topology is typically used due to its simple layout and low link length [21]. However, the average hop count scales proportionally to $\sqrt{N}$, where N is the core count. There are other, high-radix topologies that reduce the network diameter [22], but at the expense of area and energy cost at the routers.

In contrast, WNoC architectures [9]–[16] are attractive because they can transfer a message chip-wide with a latency of only a few clock cycles, regardless of the size of the chip or the number of cores. Each core or group of cores has an antenna and a transceiver, as shown in Figure 1. Small antennas in the mmWave bands and beyond [23]–[26] can be feasibly integrated, and broadcast messages [18], [20], [27]. Figure 1 shows vertical monopole antennas based on Through-Silicon Vias (TSVs), which perforate the bulk silicon [26]. Information coming from a core is modulated by the transceiver and sent by the antenna. The resulting signals propagate inside the chip package, bouncing off the metallic heat sink until they reach the receivers. Propagation causes signals to be attenuated a few tens of dBs, mainly due to spreading loss and the relatively high transmission loss in the bulk silicon [28]–[30]. However, these losses are tolerable [17], and prevent the enclosed package from acting as a reverberation chamber.

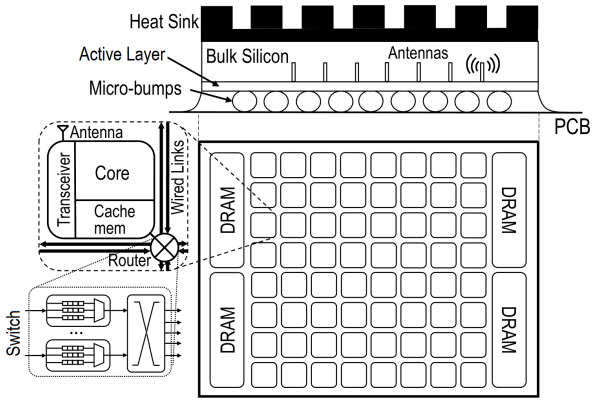


Fig. 1: Wired NoC mesh augmented with a wireless NoC within a conventional flip-chip package with one vertical monopole antenna and transceiver per core.

B. Uses of Wireless Network On Chip

There are multiple proposals to use WNoCs to enhance the architecture of manycores [18]–[20], [27]. In general, these works leverage the low latency and affordable broadcast of a WNoC to accelerate key patterns. Mondal *et al.* [27] and WiSync [18] use a WNoC to transfer all the accesses to synchronization variables, which are often traffic hot spots. In WiSync, synchronization variables are placed in a special memory called Broadcast memory. Using the WNoC only requires re-targeting the synchronization libraries or macros and, therefore, is transparent to programmers. Choi *et al.* [19] use a WNoC to transfer a fraction of the traffic generated during the training of a Convolutional Neural Network (CNN) in a CPU-GPU integrated architecture. The network design is architecture-aware and, like WiSync, transparent to the programmer. Replica [20] uses a WNoC to carry all the accesses to communication-intensive variables. The programmer has to explicitly identify these variables, which requires some effort. These variables are then placed in a special memory accessible by the WNoC. Replica also introduces approximate transformations, both in hardware and in software, which exploit the characteristics of WNoC communication. Mondal *et al.*, WiSync, and Replica partition the data structures into those that use the wired and those that use the wireless NoC, whereas Choi *et al.* do not.

In this paper, we focus on a different problem: enlisting both NoCs in supporting cache coherence protocol transactions. Both NoCs need to operate seamlessly together in a programmer-transparent manner.

C. Scalable Cache Coherence Protocols

Scalable cache coherence is attained through the use of directories [5]–[7], [31]–[33]. Directory cache coherence is widely used commercially, such as in Intel's Core i7 systems. Commercial systems use invalidation-based schemes [5]–[7]

rather than update ones [8]. This is because, in general, update protocols create more traffic and are subject to pathological cases. Examples of such cases are when data is left behind in a cache after the process migrates to another core, or when a process simply initializes data structures for several other processes, which will be running on other cores. There are proposals for hybrid invalidation-update protocols [34], [35].

In multiprocessors with large core counts, it is very costly for the directory to keep presence bits for all possible cores that could share a line at a time. As a result, designers use limited pointer schemes [6], [7], where a directory entry can only keep pointers to a handful of cores for each line. When the number of cores that want to share a line overflows the available limited pointers, a special action is taken, such as setting a broadcast bit, evicting a pointer, or re-configuring the directory entry. From then on, the directory will record sharing information in a less precise or less efficient manner.

The motivation for limited-pointer schemes is experimental data showing that an individual write often invalidates only a few caches [36]. But, as the machine's core count increases, the average write invalidates more caches. Further, if the write did not invalidate the sharers, more sharers would accumulate, and a later write with invalidation would be more expensive.

To gain insight into this issue, we modeled writes that update rather than invalidate, and measured the number of sharers that a line accumulates until the line is evicted from the LLC. For the 64-core machine and applications described in Section V, this number is on average 21 sharers. We then considered the cores that shared the line before a write, and measured what fraction of them re-read the line after the write. Such fraction is, on average, 56%. This data suggests that if, under some circumstances, we allow a write to perform a wireless update to the sharers rather than invalidating them, we may improve performance.

III. Design of WiDir

A. Main Idea

State-of-the-art directory protocols for large core counts use invalidation-based limited pointer (or coarse-vector) schemes [6], [7], [37]. As more and more cores read a shared line, the limited pointers for the line overflow, and a special action is taken in the directory entry for the line, such as setting a broadcast bit, evicting a pointer, or reconfiguring the directory entry. A subsequent write (and sometimes even a read) becomes more expensive. For example, it may trigger the broadcast of an invalidation to all the cores in the machine. As a result, existing cache coherence protocols for large core counts are unable to efficiently support groups of cores frequently reading and writing a shared line.

Wireless communication is ideally suited to this type of sharing pattern. The writer core broadcasts the update to all the current sharers in a short, fast, multicast transaction. As the sharers issue subsequent reads, they obtain the up-to-date version of the datum from their caches.

In this paper, we present a hybrid directory-based cache coherence protocol that combines a wired and a wireless component. We call the protocol *WiDir*. The directory entry for any given shared line can dynamically transition between wired and wireless coherence transactions during program execution, depending on the current access pattern.

Initially, a line uses an invalidation-based protocol operating on the wired NoC. When the number of sharers for the line reaches a certain threshold count (e.g., when all the pointers to sharers are used up), the line transitions to using wireless coherence transactions.

Later, the protocol may revert back to using wired coherence transactions for the line. This occurs when the directory realizes that the number of active sharers of the line has decreased below a certain threshold count. The directory is aware of this case because it is informed when a sharer invalidates or evicts the line from its cache — either because the sharer is not interested in the line anymore or because it needs the cache space for another line.

The *WiDir* protocol is supported by a manycore like the one shown in Figure 2. Each node in the manycore contains: a core with private caches (i.e., L1 instruction and data caches in the figure), a local slice of the shared last level cache (LLC) and its corresponding directory slice, a network interface, a local router to connect to the wired network, a transceiver, and two antennas to communicate wirelessly. To reduce costs in a large manycore, multiple neighboring nodes could share a single antenna pair.

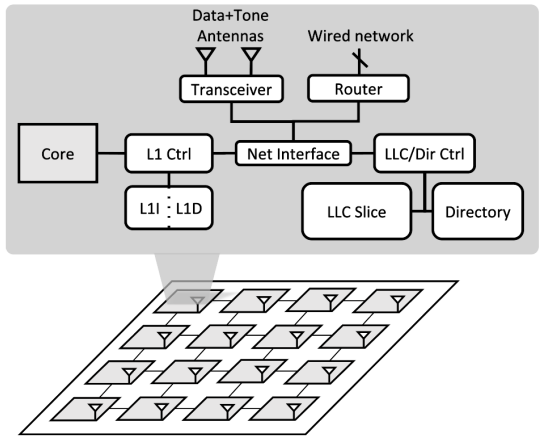


Fig. 2: Manycore that supports the *WiDir* protocol.

Following Abadal et al. [18], we use a *data* antenna and a *tone* antenna to communicate via a wireless data channel and a wireless tone channel, respectively. The data channel is centered at the 60 GHz frequency, and is used for data and most coherence messages; the tone channel is centered at 90 GHz and is used as a special-purpose *acknowledgment* channel. The data channel uses the BRS wireless protocol [38]. In this protocol, when a node has data to transmit, it first listens to the medium. When the medium is free, the node transmits a 1-cycle preamble, and leaves the second cycle empty to find out if there was a collision. If there was a collision, the node squashes the transmission and, after potentially an exponential back-off period, it restarts the transmission from the beginning; otherwise, the node completes the transmission in the next few

cycles. No further collision is possible because no other node will attempt to transmit until the current transmission uses up its allocated wireless cycles.

Cache coherence protocol messages issued by the local core or the local directory controller are passed via the network interface to the transceiver or the local wired NoC router, depending on the type of message that is being sent. For incoming messages from either of the two networks, the process is the same but in reverse.

B. Basic WiDir Operation

WiDir augments a vanilla invalidation-based directory coherence protocol with a new state, called *Wireless* or Wireless Shared (W). In this state, a write by one of the sharers sends the fine-grain update, rather than the cache line, through the wireless network, and updates the caches of all the other sharers. A read by a sharer gets the latest version of the datum from its cache. The wireless network *serializes* all the updates to any lines in W state.

In W state, the directory does not record which cores share the line, but only *how many* do. A line enters the W state from the Shared (S) state of the wired protocol, when the number of sharers goes above a *MaxWiredSharers* threshold. Cores with a line in W state are supposed to actively share the line, by regularly reading/writing the line. If a core does not do that, the hardware invalidates the local copy of the line and sends a signal to the directory, which decreases the count of wireless sharers. When the count decreases to *MaxWiredSharers*, the line transitions to the S state of the wired protocol.

The directory structure changes little from a conventional design. Without loss of generality, *WiDir* builds on top of a conventional MESI protocol, with a directory implementation that uses i shared pointers with broadcast (Dir_iB) [6], [7]. However, many other implementations, such as a Coarse Vector design (Dir_iCV_r) [7] can easily be adapted as well. The one constraint is that *MaxWiredSharers* is no higher than the number of sharer pointers supported by the directory scheme (i.e., i in Dir_iB or Dir_iCV_r).

Figure 3 shows the structure of the directory and caches in the conventional protocol augmented with *WiDir*. The changes added by *WiDir* are in bold. The changes are as follows. First, one of the states is W. Second, when a directory entry changes to W, the field of sharer pointers becomes a *count of the sharer cores* (*SharerCount*). Third, the Broadcast bit is always zero. Finally, each line in the private caches has a field called *UpdateCount*, which will be described later. Note that the *SharerCount* field needs to have as many as $\log_2 N$ bits, where *N* is the core count in the manycore. This is because a wireless line may be shared by all the cores in the manycore.

To understand how *WiDir* works, we describe the two main transactions, namely the S→W transition and the W→S transition. In the process, we will see the need for two new primitives for wireless cache-coherence protocols, namely the *Selective Data-Channel Jamming* (*Jamming*) and the *Tone-Channel Acknowledgment* (*ToneAck*). The former gives the directory the ability to reject incoming transactions for a

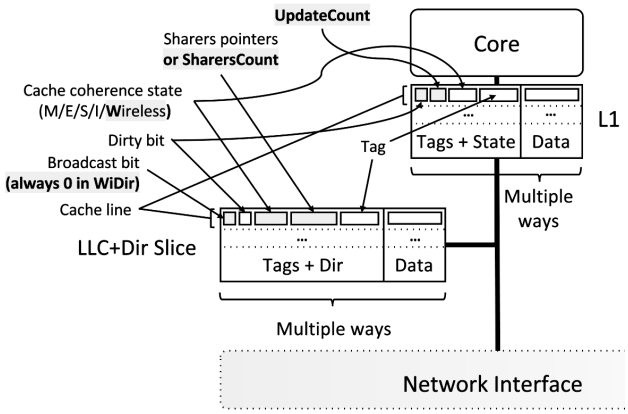


Fig. 3: Cache and directory structure of a conventional protocol augmented (in bold) for *WiDir*.

directory entry that the directory is currently operating on; the corresponding primitives in wired protocols are *bouncing* or buffering incoming transactions if the directory entry is busy. The second primitive gives the directory the ability to receive acknowledgment messages from many cores very cheaply. The primitives are discussed in Section III-C.

1) Transition from Shared to Wireless: When the directory receives a read/write request from a new core for a line that is already shared by *MaxWiredSharers* cores, the directory initiates the transition of the line to W. It does so by broadcasting a BroadcastWirelessUpgrade message (*BrWirUpgr*) for the line on the wireless data channel. It also sends a WirelessUpgrade response (*WirUpgr*) with the line to the requesting cache using the wired network.

Immediately on reception of *BrWirUpgr*, all the Tone antennas in the manycore (including the one in the node with the directory that initiated the message) start a *ToneAck* operation. For the initiating directory, this operation fully terminates only when each of the nodes in the machine completes one of the following operations: (i) the node determines that it does not contain the line in its private cache, (ii) the node finds out that it contains the line in its private cache and sets its cache state for the line to W, (iii) the node had requested the line using the wired network and has finally received either the line or a bounced response from the directory; if it has received the line, it has set its cache state for the line to W.

Once the *ToneAck* operation is complete for the initiating directory, the directory stores a count of the sharer nodes (*SharerCount*) in the sharer pointers field, and changes the state to W. From now on, all writes by the sharer processors use the wireless data network. In particular, the core that, by issuing a request triggered the transition to W, if it intended to write, it now retries the write using the wireless data network.

However, other nodes complete their *ToneAck* operation at different times before the initiating directory does. When they do, they resume execution. It is possible that one of the sharer nodes now issues a write (now using the wireless data network) before the initiating directory has fully terminated the *ToneAck* operation. The directory has to prevent this from happening because its transition is not completed. Consequently, after

starting the *ToneAck* operation, the antenna in the node with the initiating directory starts a *Jamming* operation for this line in the data channel. This prevents any core from successfully updating this line using the wireless data channel.

After the transition is fully completed, if a new core issues a read/write request to the line, the transaction will reach the directory using the wired network. In this case, the directory responds to the requester with *WirUpgr* and the line, using the wired network and, importantly, increments *SharerCount*.

2) Transition from Wireless to Shared: When cores keeping a line in W state are not interested in frequently reading/writing the line anymore, the line should return to S. To be able to do so, *WiDir* augments each line in each private cache with a short counter (e.g., 2 bits) called *UpdateCount*, which detects when the local core is not interested in the line anymore. When a cached line enters the W state, *UpdateCount* is cleared. From then on, every time that the cache receives a wireless update, *UpdateCount* is incremented; every time that the local core accesses the line, *UpdateCount* is reset. If *UpdateCount* reaches a certain threshold count, it is assumed that the local core is uninterested in the line. Hence, the hardware invalidates the line in the cache and sends a PutWireless message (*PutW*) to the directory indicating that the core is no longer a sharer. *PutW* is sent through the wired network to avoid consuming wireless bandwidth for such a non-critical message.

When the home directory receives a *PutW* for a line, it decrements the *SharerCount* for the line. If the counter reaches *MaxWiredSharers*, the line should transition to S.

To do so, the directory broadcasts a WirelessDowngrade message (*WirDwgr*) for the line on the wireless data channel. As each node receives the message, its cache controller checks if the cache indeed has the line. If it does not, no action is taken, otherwise, an acknowledgment message is sent to the directory with the sender's node ID. These messages use the wired network to save wireless bandwidth. When the directory receives all the *MaxWiredSharers* acknowledgments expected, it stores the sharer IDs in the Sharer Pointer field in the directory entry. In addition, if the LLC line is Dirty, it is written to memory. Finally, the state is set to S. The transaction is now complete and the directory accepts new requests. From now on, all communication occurs via the wired network.

When a core evicts a W line from its private cache because it needs the space for another line, the hardware also sends a *PutW* through the wired network to the directory, which will decrease *SharerCount* and may trigger a *WirDwgr*. In addition, to keep the directory up to date, a node always informs the directory when any line is evicted from its private cache. While this is not strictly needed for non-W lines to attain the functionality we desire, we do it for simplicity.

C. Primitives for Wireless Protocols

To support efficient wireless cache coherence protocols, we propose the following two primitives.

1) Selective Data-Channel Jamming: Conventional cache coherence protocols provide support for a directory to stop (i.e., buffer) or reject (i.e., bounce) new transactions directed to a directory entry that is currently busy. We propose to provide a similar primitive, called *Jamming*, for wireless directory protocols. Jamming builds on the BRS wireless protocol [38]. As indicated in Section III-A, in BRS, the second cycle of every transmission is left idle, so that the transmitting transceiver can listen if any transceiver reports (with a brief negative-Ack) a collision in the first cycle.

With this support, when a directory temporarily wants to prevent any new wireless transaction on a line, it proceeds as follows. It directs its transceiver to listen to every message initiation in the wireless data-channel network. If the first cycle of the message includes a destination address equal to the line's address (or potentially equal if all the address bits were available), the transceiver forces an interruption of the message by sending a negative-Ack, similarly as if a collision occurred. As a result, the message will be aborted. With this support, the directory prevents transactions to the line (with some false positives), while enabling transactions to other lines.

2) Tone Channel Acknowledgment: In conventional cache coherence protocols, some transactions require a directory to collect acknowledgment messages from multiple nodes. Such messages take a long time to arrive and, in addition, cause network contention. We propose to support an equivalent primitive for wireless directory protocols called *ToneAck* that allows the directory to receive acknowledgments from many processors very cheaply. In fact, since wireless messages are broadcasted, ToneAck involves an acknowledgment from all the cores.

ToneAck is triggered when a transceiver initiates a certain packet transmission in the wireless data channel — e.g., a *WirUpgr* message requested by the local directory. In ToneAck, all the transceivers except the initiating one produce a continuous tone in the Tone channel; the initiating transceiver simply monitors for the existence the tone. In parallel, each node performs a certain operation (which may be a simple check to determine that no action is needed) and, once completed, removes its tone from the Tone channel. Once the initiating transceiver notices that there is no tone in the channel, it knows that all nodes have completed their task.

Effectively, ToneAck enables a fast global acknowledgment operation. We have used ToneAck in Section III-B1 to perform a global transition from Shared to Wireless state. A similar support has been proposed by TLSync [39] and WiSync [18] for efficient core synchronization. However, this is the first time that this idea is used as part of the transactions of a cache coherence protocol.

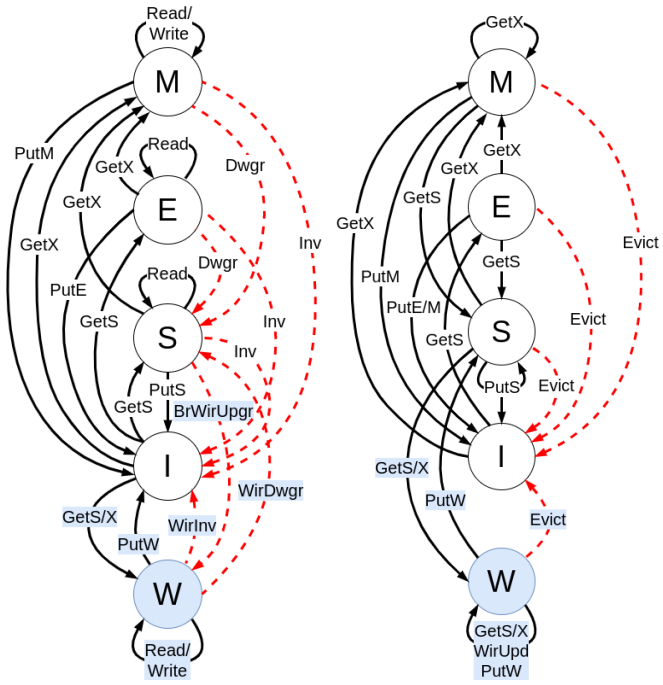
D. Summary: Why Adding Wireless Support To Coherence

Adding support for wireless communication in a large manycore provides the ability to perform several types of coherence transactions very efficiently, especially those involving multicasting. While the bandwidth of a wireless network is limited, the latency of any given transaction is very small. These properties perfectly fit the sharing pattern considered in this paper: groups of cores frequently reading and writing

a shared location. Read and write operations do not need the complicated multi-hop protocol transactions required by invalidation-based protocols, or the lengthy routing of messages required by invalidation- or update-based protocols in wired NoCs. Instead, writes transfer a fine-grained update (rather than a cache line) with about 5ns [18], and reads are local.

IV. Detailed Design

Figures 4a and 4b show the diagram of all possible transitions between stable states of *WiDir* in the controllers of the private cache and directory, respectively. The transitions are annotated with descriptive labels. As seen in the figures, we have the four MESI states plus the wireless (W) state.



(a) Private cache controller. (b) Directory controller.

Fig. 4: Transitions between stable states of *WiDir* in the controllers of the private cache (a), and directory (b). In the figures, black lines are core-initiated transactions; dashed red lines are directory-initiated transactions; and the blue text is the added wireless coherence transitions.

Rather than describing the complete protocol in detail, we focus only on the transitions that come from W or go to W. Tables I and II describe such transitions for the controllers of the private cache and directory, respectively. Each row considers one transition and describes when the transition happens and the action taken.

A. Private Cache Controller Transitions

As shown in Table I, there are four cases of I→W transitions. The first two are when the directory is in W state and the cache issues a read request (i.e., a GetS) or a write request (i.e., a GetX) (while indicating that the core is not a sharer) to the directory. In this case, the cache receives, via the wired network, a wireless upgrade message (*WirUpgr*) plus the line from the directory, and sends back a wireless upgrade acknowledgment (*WirUpgrAck*) to the directory via the wired network. The cache line transitions to W. Then, if the request was a GetX, the cache issues the update wirelessly.

The other two cases are when the cache issues a GetS or GetX (again, while indicating that the core is not a sharer) to the directory that triggers a directory transition to W. In this case, the local transceiver receives a broadcast wireless upgrade message (*BrWirUpgr*) via the wireless network, and turns on the tone channel. Then, when the cache receives, via the wired network, a *WirUpgr* plus the line from the directory, it transitions to W and tells the transceiver to turn off the tone channel. Then, if the request was a GetX, the cache issues the update wirelessly.

There are two cases of S→W transitions. The first one is when the local transceiver receives a *BrWirUpgr* message via wireless from the directory because the latter transitions to W. In this case, the transceiver turns on the tone channel, the cache transitions to W, and the transceiver turns off the tone channel.

The second case is when the cache issues a GetX to the directory (while indicating that it is already a sharer) and, by the time it gets to the directory, the latter has transitioned to W. In this case, the transceiver receives a *BrWirUpgr* via wireless, and turns on the tone channel. The cache transitions to W and the transceiver turns off the tone channel. Finally, the cache issues the update wirelessly.

There are two cases of W→W transitions. The first one is when the core reads; in this case, the hardware reads from the cache and clears the *UpdateCount* for the line. The second case is when the core writes. In this case, the transceiver broadcasts the updated word (*WirUpd*) via the wireless data network. When the transceiver indicates that the broadcast has succeeded, the local cache is also updated. The *UpdateCount* for the line in the cache is cleared.

There is one case of W→S transition. It is when the local transceiver receives a wireless downgrade message (*WirDwgr*) from the directory via wireless because *SharerCount* decreased to *MaxWiredSharers*. In this case, the cache controller sends a wireless downgrade acknowledgment message (*WirDwgrAck*) that includes the core ID to the directory via the wired network. The line state is changed to S.

There are two cases of W→I transitions. The first is when the cache evicts a line in W state. In this case, the cache controller informs the directory by sending it a *PutW* message via the wired network. The second case is when the local transceiver receives a wireless invalidate message (*WirInv*) for a line from the directory via wireless because the directory is evicting the line. The cache invalidates the line and, if the core has a pending write on the line, it squashes it and retries it.

B. Directory Controller Transitions

As shown in Table II, the transition S→W occurs when the directory receives a GetS or GetX from a non-sharer cache

TABLE I: State transitions that come from W or go to W for the controller of the private cache.

Transition	When?	Action
$I \rightarrow W$	Cache issues GetS to directory and directory is in W	Cache receives a *WirUpgr*+line via wired, sends *WirUpgrAck* back to directory via wired, and transitions to W
	Cache issues GetX to directory (while indicating that it is not a sharer) and directory is in W	Cache receives a *WirUpgr*+line via wired, sends *WirUpgrAck* back to directory via wired, and transitions to W. The cache issues the update wirelessly
	Cache issues GetS to directory triggering a directory transition to W	Transceiver receives a *BrWirUpgr* via wireless, and turns on the tone channel. When the cache receives a *WirUpgr*+line via wired, it transitions to W and notifies the transceiver to turn off the tone channel
	Cache issues GetX to directory (while indicating that it is not a sharer) triggering a directory transition to W	Transceiver receives a *BrWirUpgr* via wireless, and turns on the tone channel. When the cache receives a *WirUpgr*+line via wired, it transitions to W and notifies the transceiver to turn off the tone channel. The cache issues the update wirelessly
$S \rightarrow W$	Transceiver receives a *BrWirUpgr* from the directory via wireless because the latter transitions to W	Transceiver turns on the tone channel, the cache transitions to W, and the transceiver turns off tone channel
	Cache issues GetX to directory (while indicating that it is already a sharer) and, by the time it gets to the directory, the latter has transitioned to W	Transceiver receives a *BrWirUpgr* via wireless, and turns on the tone channel. The cache transitions to W and the transceiver turns off the tone channel. Finally, the cache issues the update wirelessly
$W \rightarrow W$	Core reads	*UpdateCount* is cleared
	Core writes	Transceiver broadcasts the updated word (*WirUpd*) via wireless. When the transceiver indicates that the broadcast succeeded, the local cache is updated and *UpdateCount* is cleared
$W \rightarrow S$	Transceiver receives a *WirDwgr* from the directory via wireless because *SharerCount* decreased to *MaxWiredSharers*	Cache sends *WirDwgrAck* (including core ID) to directory via wired, and changes the state to S
$W \rightarrow I$	Cache evicts W line	Cache notifies the directory with *PutW* via wired
	Transceiver receives a *WirInv* via wireless from the directory because the directory wants to evict the line	Cache invalidates the line and, if the core has a pending write on the line, it squashes it and retries it

TABLE II: State transitions that come from W or go to W for the directory controller.

Transition	When?	Action
$S \rightarrow W$	Directory receives a GetS or GetX from a non-sharer cache via wired and the new number of sharers for the line is now higher than *MaxWiredSharers*	Transceiver broadcasts *BrWirUpgr* via wireless and turns on jamming for the line. Directory sends *WirUpgr*+line to the requester via wired. Once the tone channel is silent, directory sets the state to W and sets *SharerCount*, and jamming is turned off
$W \rightarrow W$	Directory receives from a cache via wired a GetS or GetX (while indicating that it is not a sharer)	Transceiver turns on jamming, and directory sends *WirUpgr*+line via wired to the requester. When the directory receives its *WirUpgrAck* via wired, it increments *SharerCount* and the transceiver turns off jamming
	Directory receives a GetX from a cache via wired (while indicating that it already is sharer). The cache does not know that the directory is already in W	Directory discards message since a previously-sent *BrWirUpgr* via wireless already provided the information about the new directory state
	Transceiver receives an update (*WirUpd*) via wireless from a sharer cache	*SharerCount* is incremented
	Directory receives a *PutW* via wired	*SharerCount* is decremented and is still higher than *MaxWiredSharers*
$W \rightarrow S$	Directory receives a *PutW* via wired	*SharerCount* is decremented and now becomes *MaxWiredSharers*. Transceiver broadcasts *WirDwgr* wirelessly and directory waits for *WirDwgrAck* acknowledgments via wired. Upon receiving all *MaxWiredSharers WirDwgrAck*s, the directory records their core IDs in the sharer pointers, writes the line to memory if it is dirty in the LLC, and sets the state to S
$W \rightarrow I$	Directory evicts a line shared wirelessly	Transceiver broadcasts a *WirInv* via wireless. If the line is dirty in the LLC, the line is written to memory

via the wired network, and the new number of sharers for the line is now higher than *MaxWiredSharers*. In this case, the local transceiver broadcasts a *BrWirUpgr* via the wireless network and turns on jamming in the wireless data network for the line. The directory sends *WirUpgr* plus the line to the requester node via the wired network. Once the transceiver detects that the tone channel is silent, the directory sets the state to W and sets *SharerCount* to the count of sharers, and jamming is turned off.

The transition W→W occurs in four cases. The first one is when the directory receives from a cache via the wired network a GetS or GetX (while indicating that the cache is not a sharer). In this case, the transceiver turns on jamming and the directory sends, via the wired network, a *WirUpgr* plus the line to the requester. When the directory receives a *WirUpgrAck* via the wired network, it increments *SharerCount* and the transceiver turns off jamming.

The second case is when the directory receives a GetX from a cache via the wired network (while indicating that the cache is already a sharer). The cache does not know that the directory is already in W. In this case, the directory discards the message since a previously-sent *BrWirUpgr* via the wireless network already provided the information about the new directory state.

The third case is when the transceiver in the node of the directory receives an update (*WirUpd*) via the wireless network from a sharer cache. In this case, the directory increments

SharerCount. The fourth case is when the directory receives a *PutW* message via the wired network, as a result of a wireless sharer evicting the line from its cache. In this case, the directory decrements *SharerCount* and finds that the value is still higher than *MaxWiredSharers*.

The W→S transition occurs when the directory receives a *PutW* message via the wired network as before but, as the directory decrements *SharerCount*, it finds that the value is *MaxWiredSharers*. In this case, the transceiver broadcasts *WirDwgr* wirelessly, and the directory waits for *WirDwgrAck* acknowledgments from *MaxWiredSharers* cores via the wired network. Upon receiving them, the directory records their core IDs in the sharer pointers, writes the line back to memory if the Dirty bit in the LLC is set, and sets the state to S.

The transition W→I occurs when the directory evicts a line shared wirelessly. In this case, the transceiver broadcasts a *WirInv* message via the wireless network. If the Dirty bit of the line in the LLC is set, the line is written to memory.

C. Correct Operation of Wireless Transactions

Writes that use the wireless network, like all other writes, are kept in a core's write buffer until they complete. The process of performing such a write involves multiple steps. First, the local transceiver has to obtain access to the wireless data network. Then, the transceiver has to succeed in sending the packet without collisions. Once the packet is guaranteed to successfully transmit (i.e., when the second cycle in the transmission indicates that there is no collision), the transceiver signals the write buffer to merge the write into the local cache. Once the merging is done, the write is complete.

Before the transceiver manages to successfully send a packet through the wireless network, the transceiver may receive wireless updates to the line from remote cores. In this case, such updates are performed before the local one. It is also possible that the transceiver receives a wireless invalidation for the line (*WirInv*) because the line's entry in the directory in a remote node is being evicted. In this case, the local cache line is invalidated and the local write is retried. As the local write retries, it will issue a transaction using the wired network that will cause the directory to allocate a new entry for the line. In all of these transactions, successful transmission in the wireless network acts as a serialization point. These wireless transactions cannot change local state in caches or directories until the packet is guaranteed to successfully transmit in the wireless network.

A wireless read-modify-write (RMW) instruction is implemented in a similar way. When the RMW instruction reaches its turn to execute according to the consistency model, the hardware issues a read followed by a write to the corresponding line. The read reads the line from the L1 and marks it as non-evictable; the write does not update the line yet and, instead, reaches the local transceiver, to be sent to the wireless network. From the time the write is issued until it is guaranteed to successfully transmit in the wireless network, the hardware monitors for any incoming wireless transaction that updates or invalidates the line. If one such transaction is received, the write fails and is not sent to the wireless network. The hardware then retries the whole RMW instruction. Otherwise, as soon as the transceiver acknowledges that the write is guaranteed to successfully transmit in the wireless network, the write updates the L1 and the RMW instruction completes.

TABLE III: Architecture modeled. RT means round trip.

General Parameters	
Architecture	Manycore with 64 cores
Core	Out of order, 4-issue wide, 1GHz, x86 ISA
ROB; ld/st queue	180 entries; 64 entries
Write buffer	64 entries
L1 I+D caches	Private 64KB, WB (data), 2-way, 2-cycle RT, 64B lines
L2 cache	Shared, per-core 512KB bank (32 MB total), WB
L2 bank	8-way, 12-cycle RT (local), 64B lines
On-chip network	2D-mesh, 1 cycle/hop, 128-bit links
Off-chip memory	Connected to 4 memory controllers, 80-cycle RT
WiDir Parameters	
Cache Coherence	Enhances a Dir_3_B MESI directory protocol
Data Wireless channel	60GHz, 20Gb/s, 4 cyc. transfer + 1 cyc. collision detect.
Tone Wireless channel	90GHz, 1Gb/s, 1 cycle transfer latency
MaxWiredSharers	3 sharers/line
MAC Protocol	BRS [38] (exponential backoff)
Transceiver	At 65nm: 0.25mm^2, 30mW [17], [41], [42]
Data converter	At 65nm: 0.03mm^2, 0.72mW [43]
Serializer/Deserializer	At 65nm: 0.04mm^2, 10.8mW [44]
Data+tone antennas	At 65nm: 0.08mm^2 [45]
All RF circuit in node	At 65nm: Area: 0.4mm^2, Transmitting: 39.4mW, Receiving: 39.4mW, Idle: 26.9mW (power gating analog amplifier, with transient energy of 1.14 pJ)

V. Evaluation Methodology

We use the SST [40] simulation framework as a cycle-level, execution-driven simulator to model a server architecture with 64 cores. The architecture parameters, as well as the energy and area of the wireless components are shown in Table III. Each processor tile is composed of an out-of-order core with private L1 instruction and data caches. L2 is shared and physically distributed across the processor tiles. For the NoC, we use a 2D-mesh topology with a latency of 1 cycle per hop. We augment the simulator to model in detail the transmissions and collision handling required by the wireless network. The data channel of the wireless network has a bandwidth of 20 Gb/s. This is adequate to transmit a 64-bit word and its address in 4 cycles. Collision detection adds one additional cycle.

We use McPAT [46] and CACTI [47] to model the energy consumed by cores and memory hierarchy, as well as a calibrated DSENT [48] to model the energy of the wired links and routers. To compute the power and area consumed by the wireless hardware (transceiver, data converter, serializer, deserializer, and antennas), we use and adapt published data in 65 nm technology [17], [41]–[45], [49]. Because some of the components' data were given only for 16 Gb/s, we linearly scaled up their power and area to match our 20 Gb/s bit rate. The next step would be to scale these numbers down to the 22 nm technology used for the rest of the system. The power and area of the wireless components would be lower, as proposed by other researchers [50], [51]. However, in this paper we choose to be conservative and not scale them down. Finally, the energy parameters of the wireless components in Table III also take into consideration the fact that the transmitter's power

amplifier and the receiver's low noise amplifier can be power gated when not in use [17], [52].

Our evaluation assumes a highly reliable wireless technology. In the past, wireless on-chip links encountered sizable error rates [53]–[55]. However, improvements in recent years (e.g., [56]) have enabled wireless on-chip links with error rates like those of on-chip wires (10^{-15}), making corrupted messages extremely infrequent. Note also that the rate of these noise-related errors can be made arbitrarily low by increasing the signal strength.

To evaluate the efficacy of our design, we compare it to a *Baseline* manycore system that uses the Dir_3B MESI directory protocol without the wireless support of *WiDir*. We evaluate a wide range of multi-threaded applications from SPLASH-3 [57] and PARSEC [58]. These applications are listed and characterized by their L1 misses-per-kilo-instruction (MPKI) for *Baseline* in Table IV. They have different levels of fine-grained data sharing, as well as different access patterns.

TABLE IV: Evaluated applications characterized by L1 MPKI in *Baseline*.

SPLASH-3 [57]				**PARSEC** [58]	
Name	**MPKI**	**Name**	**MPKI**	**Name**	**MPKI**
water-spa	0.49	*fft*	5.05	*blackscholes*	0.13
water-nsq	2.86	*lu-nc*	21.52	*bodytrack*	7.51
ocean-nc	16.05	*lu-c*	1.9	*canneal*	23.21
volrend	2.44	*radix*	9.41	*dedup*	4.1
radiosity	5.28	*barnes*	9.53	*fluidanimate*	1.27
raytrace	10.05	*fmm*	1.88	*ferret*	6.34
cholesky	5.92			*freqmine*	8.84

For the SPLASH-3 applications, we use the default input sets as described in [57] and, for PARSEC, we use the standard *simsmall* [58]. The input set sizes were chosen to allow running the default region of interest of each application to completion within a realistic time frame of up to a few days of simulation.

VI. Results

In this section, we analyze the data sharing, and then evaluate performance, energy, and area. Finally, we perform a sensitivity analysis.

A. Data Sharing Analysis

Figure 5 considers only the wireless updates in *WiDir*, and shows a histogram of the number of sharers that are updated per write. We group the number of sharers in bins: up to 5, 6-10, 11-25, 26-49, and 50 or more. The figure shows that, for many applications, a wireless write updates many caches. For other applications, only a few caches are typically updated. On average, we see that updates with few sharers (up to 5) account for 36% of these writes. On the other hand, updates with many sharers (50+) account for 37% of these writes. The latter typically correspond to highly-shared variables, such as locks and barriers. This category is the one that offers the highest benefit from the wireless mode.

Note that the detection of lines with many sharers does not require any user effort. The hardware automatically identifies them and switches them to wireless mode.

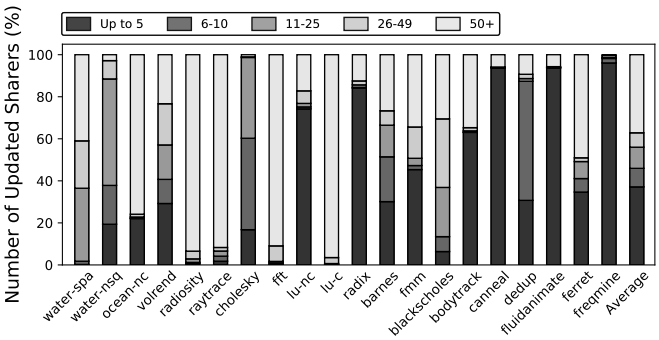


Fig. 5: Number of sharers updated upon a wireless write in *WiDir*.

B. Performance Analysis

Cache Misses. Figure 6 shows the L1 misses per kilo instruction (MPKI) in *WiDir* and *Baseline*, normalized to *Baseline*. The figure is broken down into read and write misses. We can see that, on average, the MPKI is reduced by 15%. The misses eliminated are coherence misses. *WiDir* removes them by virtue of updating the sharers of a wireless line upon a write, as opposed to invalidating them.

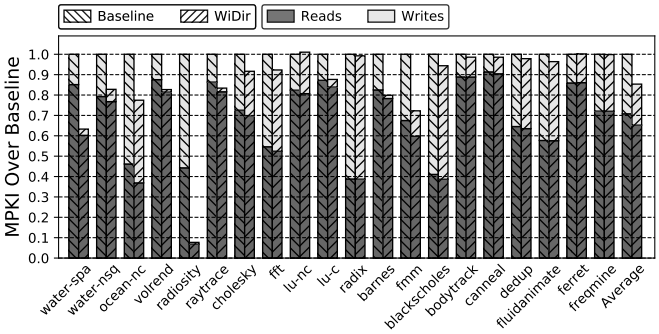


Fig. 6: Misses-per-kilo-instruction in *WiDir* and *Baseline*, normalized to *Baseline*.

The reduction in MPKI is especially large in *radiosity*. For this application, Figure 5 showed that over 90% of the wireless writes in *WiDir* update 50+ sharers. Updating so many sharers, as opposed to invalidating them, achieves a large reduction in MPKI. Other applications that see a sizable reduction in MPKI are *water-spa, ocean-nc, barnes*, and *fmm*. Most of these applications have a large average number of wireless sharers updated per write (Figure 5). However, the two figures are not perfectly correlated because there are also misses to non-wireless lines.

Memory Latency. Figure 7 shows the overall latency of memory operations in *WiDir* and *Baseline*, normalized to *Baseline*. This latency is computed by counting, for a each request, the number of cycles from the time the request enters the ROB until it retires from the ROB, and then adding over all the loads and over all the stores. The figure is broken down into loads and stores. Note that this computation does not take into account the fact that some accesses stall the ROB while others

are simply overlapped with the former. However, it gives some insight into the relative importance of memory cycles in the two architectures.

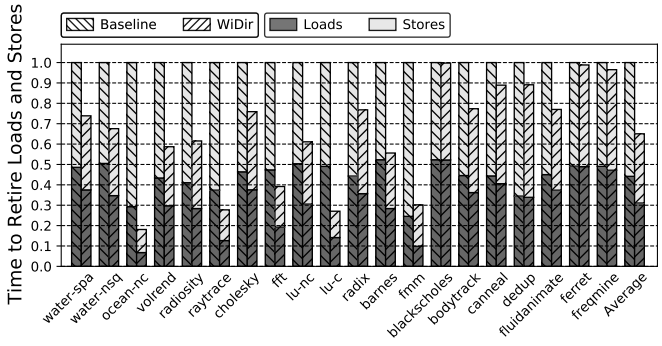


Fig. 7: Overall latency of memory operations in *WiDir* and *Baseline*, normalized to *Baseline*.

We see that *WiDir* reduces the total latency of memory access in nearly all of the applications. Moreover, the reductions are similar in both loads and stores. Some applications such as *ocean-nc, raytrace, fft, lu-c*, and *fmm* have large latency reductions. These applications are likely to be sped-up significantly by *WiDir*. Overall, on average, *WiDir* reduces the total latency of memory accesses by 35% over *Baseline*.

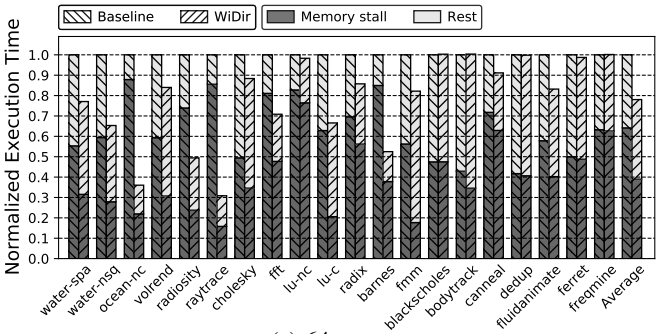
Wired Network Hops. In a conventional wired protocol, read and write miss transactions often require multiple coherence hops (or *legs*), as the directory forwards messages to other sharers. In a large chip, such as the 64-core manycore that we are considering, each of these legs of a coherence transaction ends up incurring a high cost due to the large number of network hops that each message has to go through in the wired mesh.

To assess this cost, we count, for each leg of a coherence transaction in *Baseline*, the number of network hops that the message has to go through. Table V shows the fraction of messages in our applications that need a certain number of network hops in a leg. We group the number of hops in bins of 0-2, 3-5, 6-8, 9-11, and 12-16 hops. We can see that more than half of all the messages in the baseline have to perform at least 6 network hops per leg to reach their destination. In contrast, in *WiDir*, writes to wireless lines are broadcasted to all sharers in a single hop. This capability reduces memory and communication latency.

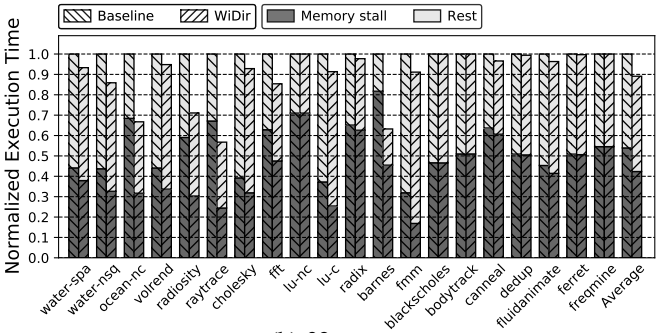
TABLE V: Distribution of number of network hops per leg for messages sent through the wired mesh, in the 64-core *Baseline* architecture.

Number of Hops per Leg	***0-2***	***3-5***	***6-8***	***9-11***	***12-16***
% of Messages	17%	22%	31%	21%	9%

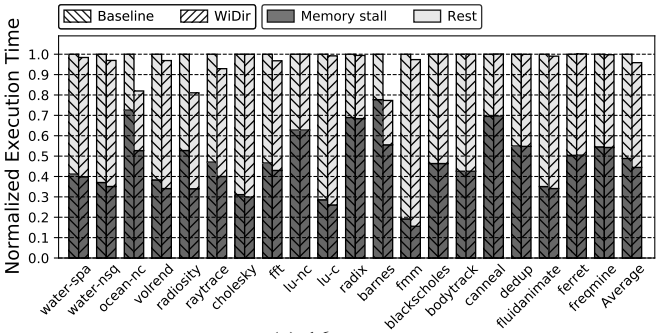
Execution Time. Figure 8 presents the execution time of the applications in *WiDir* and *Baseline*, normalized to *Baseline*. From top to bottom, the graphs show runs for 64 cores (a), 32 cores (b), and 16 cores (c). Each bar is broken down into cycles when the execution is stalled due to a pending memory access (*Memory stall*) and the rest of cycles (*Rest*).



(a) 64 cores



(b) 32 cores



(c) 16 cores

Fig. 8: Execution time in *WiDir* and *Baseline*, normalized to *Baseline* for 64 (a), 32 (b), and 16 (c) core executions. The bars are broken down into memory stall cycles and the rest of cycles.

Consider first the 64-core runs (Figure 8a). We can see that, in *Baseline*, many of the cycles are wasted to memory stall. On average, nearly 65% of the cycles fall into this category. With *WiDir*, many applications reduce their execution time, sometimes substantially. Such reductions are caused by decreases in the memory stall cycles. The *Rest* cycles change only a little, although sometimes they go up noticeably. The reason for the change is that the timing of the instructions changes between *WiDir* and *Baseline*. On average, *WiDir* manages to reduce over one third of the memory stall time. As a result, on average, the total execution time of the applications reduces by 22%.

Generally, the applications that benefit the most from *WiDir* have some of the highest percentages of memory stall cycles

in *Baseline*. They include *ocean-nc, radiosity, raytrace*, and *barnes*. Wireless transactions directly reduce the number of misses and, hence, the memory stall time. However, there are also applications where *WiDir* has practically no impact. They include *blackscholes, bodytrack, dedup, ferret*, and *freqmine*.

As we decrease the number of cores in the architecture, the speed-ups decrease. Specifically, for 32 cores (Figure 8b) and 16 cores (Figure 8c), the average execution time reduction of the applications is 11% and 4%, respectively. With fewer cores, the latency of the wired network is smaller and, in addition, variables are shared by fewer cores.

C. Energy and Area Analysis

Energy. Figure 9 shows the energy consumed by *WiDir* and *Baseline*, normalized to *Baseline*. The energy is broken down into the energy of the core, the private L1s, the shared L2 plus directory, the wired NoC, and the WNoC.

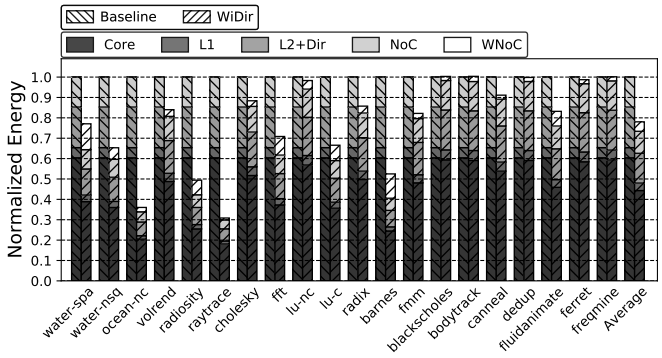


Fig. 9: Energy consumed by *WiDir* and *Baseline*, normalized to *Baseline*.

We see that *Baseline* spends on average about 60% of the energy in the core, 5% in the instruction and data L1 caches, 20% in the shared L2 and directory, and 15% in the wired NoC. In *WiDir*, we have the same categories plus the energy contribution of the WNoC.

The WNoC and coherence protocol of *WiDir* reduce the cost of polling operations and long distance communications. Therefore, they reduce the energy consumption. On average, the energy consumed by *WiDir* is 21% lower than in *Baseline*. Across applications, the energy reductions are very similar to the execution time reductions (Figure 8a). In addition, we can see that the energy contribution of the WNoC is modest: on average, it is 5.9% of the *WiDir* energy.

Finally, since the energy reduction of *WiDir* is roughly similar to the execution time reduction, we conclude that the power consumed by *Baseline* and *WiDir* are very similar.

Area. Table III showed that an estimate of the area overhead of the transceiver, data converter, serializer, deserializer, and antennas at 65nm technology is 0.4 mm^2. Several authors [50], [51] argue that the area reduces linearly with the feature size. Consequently, we expect that, for a current technology like 10 nm or lower, the area overhead of the wireless network support is modest.

D. Speedup and Sensitivity Analysis

As the number of cores in the manycore increases, so does the overhead of traditional coherence protocols, as well as the cost of traversing the wired mesh. With *WiDir*, the added wireless coherence protocol reduces the overhead of the cache lines that are highly shared and would suffer the most. The result is a better scalability of *WiDir* with the number of cores than *Baseline*.

Figure 10 shows the execution speedup of *WiDir* and *Baseline* as the number of cores increases. The speedups are computed relative to the execution time of *Baseline* with four cores. From the figure, we can see that, up to 16 cores, the difference in speedup between *WiDir* and *Baseline* is small. This is because the cost of traversing the wired network is small, and the number of cores sharing a cache line is typically modest. As a result, *WiDir* cannot provide much benefit. However, as the size of the chip increases to 32 and 64 cores, *WiDir* benefits from having more cache lines in wireless mode, while *Baseline* is harmed by the increased cost of traversing the wired network. Thus, the average speedups of *WiDir* and *Baseline* diverge. Hence, *WiDir* is more scalable.

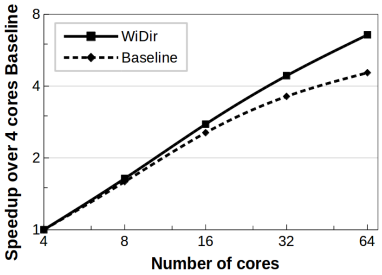


Fig. 10: Average execution time speedup of *WiDir* and *Baseline* over the 4-core *Baseline*.

Max Wired Sharers	Sp.	Coll. prob.
2	*1.22x*	*6.93%*
3	*1.43x*	*3.14%*
4	*1.38x*	*2.24%*
5	*1.31x*	*1.70%*

TABLE VI: Speedups of *WiDir* over *Baseline*, and collision probability in *WiDir*, for different values of *MaxWiredSharers*.

We now consider the effect of the threshold for the number of cores that need to be sharing a line, for the line to switch to wireless (*MaxWiredSharers*). Recall that the default value of *MaxWiredSharers* is 3. In this section, we change its value to 2, 4, and 5.

For the different values of *MaxWiredSharers*, Table VI shows two measures: (i) the average execution time speedup of *WiDir* over *Baseline* for 64-core runs (*Sp.*), and (ii) the probability of collisions of messages in the wireless network in *WiDir* (*Coll. prob.*). The values for *MaxWiredSharers* equal to 3 are 1.43x and 3.14%, respectively.

When we switch to wireless mode sooner (i.e., *MaxWiredSharers* equal to 2), more lines are in wireless mode. This increases contention and collisions in the wireless medium (from 3.14% to 6.93%), and in turn hurts speedups (from 1.43x to 1.22x) compared to the default *MaxWiredSharers* value of 3. It can be shown, however, that some applications such as *lu-nc, fmm*, and *canneal* actually attain higher speedups when switching to wireless mode sooner.

When we increase the threshold to switch to wireless later (*MaxWiredSharers* equal to 4 and 5), fewer lines are in

wireless mode. Hence, we reduce the amount of traffic in the wireless medium, which in turn decreases the probability of wireless collisions (from 3.14% to 2.24% and 1.70%). However, the result is reduced speedups (from 1.43x to 1.38x and 1.31x), due to missing opportunities to use the wireless medium. No application attains a higher speedup with these higher *MaxWiredSharers* values.

Overall, our default *MaxWiredSharers* value of 3 works best.

VII. Related Work

Wireless Architectures. We described WiSync [18], Choi *et al.* [19], Mondal *et al.* [27], and Replica [20] in Section II. Other works use a WNoC to accelerate the communication patterns of applications such as graph analytics [59], molecular dynamics simulations [60], and brain-machine interfaces [61]. These other works differ from *WiDir* in that the wireless links are used to reduce the average network latency regardless of the coherence state. Further, the network is optimized for a particular set of applications only.

There are many proposals for Medium Access Control (MAC) protocols for WNoCs [62]–[65]. *WiDir* has used BRS, but practically any other WNoC MAC protocol could be used as well.

Snooping-based Protocols. *WiDir*'s wireless network provides a totally-ordered interconnect and efficient broadcasting to satisfy misses with low latency. Snooping protocols [4], [34] also use a totally-ordered interconnect. Several works extend snooping coherence to unordered interconnects to improve its scalability [66]–[72]. The idea is to provide ordering guarantees, either at the protocol level via tokens associated to each cache line [67] or at the network level using global timestamps [66], snoop-order numbers [69], a logical embedded-ring [68], or a dedicated ordering network [70]–[72]. Depending on the protocol, the scalability is fundamentally limited by the increasing ordering buffer requirements, latency of serialization, or inefficiency of filling a mesh NoC with broadcasts.

Protocols with Emerging Interconnect Technologies. Optical/RF transmissions via shared nanophotonic waveguides [73]–[77] or transmission lines (TLs) [39], [78]–[82] can provide ordered broadcast. Some proposals use these global interconnects to implement conventional snooping coherence [73], custom limited directory protocols [75], race-free protocols via globally shared locks [74], or to speed-up synchronization [39], [83]. Compared to wireless networks, both nanophotonics and TLs are more energy efficient and provide higher bandwidth because of their guided nature. However, network design becomes more complex than wireless because a maze of physical waveguides/TLs needs be planned and laid down. Both technologies also have scalability issues in globally-shared networks. In nanophotonics, there are losses added by each and every power divider, modulator, coupler, and receiver along the path. In TLs, there is the need of a centralized arbiter for the bus, and of overcoming signal reflections with amplifying stages between segments, which are costly and complicate the design.

Scalable Directory Protocols. We outlined a variety of protocols with scalable directories [6], [7], [33] in Section II. Another way to scale the directory is by dividing the manycore into coherence domains [84], where only the nodes in the same domain are kept coherent with each other. *WiDir*, instead, not only allows the directory to scale without domain restrictions, but also boosts performance by enabling broadcast updates of highly-shared lines.

VIII. Conclusion

To handle sharing patterns where a group of cores frequently reads and writes a set of shared variables, this paper used on-chip wireless network technology to augment a conventional invalidation-based directory cache coherence protocol. The resulting protocol, called *WiDir*, seamlessly transitions between wired and wireless coherence transactions for the same line based on the program's access patterns in a programmer-transparent manner. In this paper, we described the protocol in detail. Further, an evaluation showed that *WiDir* substantially reduces the memory stall time of applications. For 64-core runs, *WiDir* reduced the execution time of applications by an average of 22% compared to MESI. Moreover, *WiDir* was shown to be more scalable than MESI. These benefits were obtained with a very modest power cost.

Acknowledgments

This work was funded in part by NSF Grant No. CCF-1629431 and by EU's Horizon 2020 Research and Innovation Programme Grant No. 863337 (WiPLASH).

References

[1] *Ampere Altra 64-Bit Multi-Core ARM Processor*. https://amperecomputing.com/altra/, accessed April 16, 2020.

[2] *AMD Epyc 7742 Processor*. https://www.amd.com/en/products/cpu/amd-epyc-7742, accessed April 16, 2020.

[3] *Intel Xeon Platinum 9282 Processor*. https://www.intel.com/content/www/us/en/products/processors/xeon/scalable/platinum-processors/platinum-9282.html, accessed April 16, 2020.

[4] M. Dubois, M. Annavaram, and P. Stenström, *Parallel Computer Organization and Design*. Cambridge University Press, 2012.

[5] D. Lenoski, J. Laudon, K. Gharachorloo, W. Weber, A. Gupta, J. Hennessy, M. Horowitz, and M. Lam, "The Stanford DASH Multiprocessor," *IEEE Computer*, 1992.

[6] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An Evaluation of Directory Schemes for Cache Coherence," in *International Symposium on Computer Architecture (ISCA)*, 1988.

[7] A. Gupta, W.-D. Weber, and T. Mowry, "Reducing Memory and Traffic Requirements for Scalable Directory-Based Cache Coherence Schemes," in *International Conference on Parallel Processing (ICPP)*, 1990.

[8] H. Grahn, P. Stenstrom, and M. Dubois, "Implementation and Evaluation of Update-Based Cache Protocols Under Relaxed Memory Consistency Models," in *Future Generation Computer Systems*, 1995.

[9] S. Abadal, J. Torrellas, E. Alarcón, and A. Cabellos-Aparicio, "OrthoNoC: A Broadcast-Oriented Dual-Plane Wireless Network-on-Chip Architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 628–641, 2017.

[10] G. Ascia, V. Catania, S. Monteleone, M. Palesi, D. Patti, J. Jose, and V. M. Salerno, "Exploiting Data Resilience in Wireless Network-on-Chip Architectures," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 16, no. 2, pp. 1–27, 2020.

[11] D. DiTomaso, A. Kodi, D. Matolak, S. Kaya, S. Laha, and W. Rayess, "A-WiNoC: Adaptive Wireless Network-on-Chip Architecture for Chip Multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3289–3302, 2015.

[12] A. Karkar, T. Mak, N. Dahir, R. Al-Dujaily, K.-F. Tong, and A. Yakovlev, "Network-on-Chip Multicast Architectures Using Hybrid Wire and Surface-Wave Interconnects," *IEEE Transactions on Emerging Topics in Computing*, vol. 6, no. 3, pp. 357–369, 2018.

[13] K. Duraisamy, Y. Xue, P. Bogdan, and P. P. Pande, "Multicast-Aware High-Performance Wireless Network-on-Chip Architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 3, pp. 1126–1139, 2017.

[14] D. W. Matolak, A. Kodi, S. Kaya, D. Ditomaso, S. Laha, and W. Rayess, "Wireless Networks-on-Chips: Architecture, Wireless Channel, and Devices," *IEEE Wireless Communications*, vol. 19, no. 5, pp. 58–65, 2012.

[15] S. H. Gade and S. Deb, "HyWin: Hybrid Wireless NoC With Sandboxed Sub-Networks for CPU/GPU Architectures," *IEEE Transactions on Computers*, vol. 66, no. 7, pp. 1145–1158, 2017.

[16] N. Mansoor, P. J. S. Iruthayaraj, and A. Ganguly, "Design Methodology for a Robust and Energy-Efficient Millimeter-Wave Wireless Network-on-Chip," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 1, no. 1, pp. 33–45, 2015.

[17] X. Yu, J. Baylon, P. Wettin, D. Heo, P. Pratim Pande, and S. Mirabbasi, "Architecture and Design of Multi-Channel Millimeter-Wave Wireless Network-on-Chip," *IEEE Design & Test*, vol. 31, no. 6, 2014.

[18] S. Abadal, A. Cabellos-Aparicio, E. Alarcón, and J. Torrellas, "WiSync: An Architecture for Fast Synchronization Through on-Chip Wireless Communication," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.

[19] W. Choi, K. Duraisamy, R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "On-Chip Communication Network for Efficient Training of Deep Convolutional Networks on Heterogeneous Manycore Systems," *IEEE Transactions on Computers*, vol. 67, no. 5, pp. 672–686, 2018.

[20] V. Fernando, A. Franques, S. Abadal, S. Misailovic, and J. Torrellas, "Replica: A Wireless Manycore for Communication-Intensive and Approximate Data," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[21] D. Sánchez, G. Michelogiannakis, and C. Kozyrakis, "An Analysis of on-Chip Interconnection Networks for Large-Scale Chip Multiprocessors," *ACM Transactions on Architecture and Code Optimization*, vol. 7, no. 1, p. Article 4, 2010.

[22] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Kilo-NoC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees," in *International Symposium on Computer Architecture (ISCA)*, 2011.

[23] H. M. Cheema and A. Shamim, "The Last Barrier: On-chip Antennas," *IEEE Microwave Magazine*, vol. 14, no. 1, pp. 79–91, 2013.

[24] Q. J. Gu, "THz Interconnect: The Last Centimeter Communication," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 206–215, 2015.

[25] J. Wu, A. Kodi, S. Kaya, A. Louri, and H. Xin, "Monopoles Loaded With 3-D Printed Dielectrics for Future Wireless Intra-Chip Communications," *IEEE Transactions on Antennas and Propagation*, vol. 65, no. 12, pp. 6838–6846, 2017.

[26] V. Pano, I. Tekin, I. Yilmaz, Y. Liu, K. R. Dandekar, and B. Taskin, "TSV Antennas for Multi-Band Wireless Communication," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 1, pp. 100–113, 2020.

[27] H. K. Mondal, R. C. Cataldo, C. A. M. Marcon, K. Martin, S. Deb, and J.-P. Diguet, "Broadcast- and Power-Aware Wireless NoC for Barrier Synchronization in Parallel Computing," in *International System-on-Chip Conference (SOCC)*, 2018.

[28] S. Abadal, C. Han, and J. M. Jornet, "Wave Propagation and Channel Modeling in Chip-Scale Wireless Communications: A Survey From Millimeter-Wave to Terahertz and Optics," *IEEE Access*, vol. 8, pp. 278–293, 2019.

[29] Y. Chen and C. Han, "Channel Modeling and Characterization for Wireless Networks-on-Chip Communications in the Millimeter Wave and Terahertz Bands," *IEEE Transactions on Molecular, Biological and Multi-Scale Communications*, vol. 5, no. 1, pp. 30–43, 2019.

[30] I. El Masri, T. Le Gouguec, P.-M. Martin, R. Allanic, and C. Quendo, "Electromagnetic Characterization of the Intrachip Propagation Channel in Ka- and V-Bands," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 9, no. 10, pp. 1931–1941, 2019.

[31] L. M. Censier and P. Feautrier, "A New Solution to Coherence Problems in Multicache Systems," *IEEE Transactions on Computers*, 1978.

[32] D. Chaiken, J. Kubiatowicz, and A. Agarwal, "LimitLESS Directories: A Scalable Cache Coherence Scheme," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1991.

[33] D. Sanchez and C. Kozyrakis, "SCD: A Scalable Coherence Directory With Flexible Sharer Set Encoding," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2012.

[34] S. J. Eggers and R. H. Katz, "Evaluating the Performance of Four Snooping Cache Coherency Protocols," in *International Symposium on Computer Architecture (ISCA)*, 1989.

[35] F. Dahlgren, "Boosting the Performance of Hybrid Snooping Cache Protocols," in *International Symposium on Computer Architecture (ISCA)*, 1995.

[36] A. Gupta and W.-D. Weber, "Cache Invalidation Patterns in Shared-Memory Multiprocessors," *IEEE Transactions on Computers*, 1992.

[37] J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," in *International Symposium on Computer Architecture (ISCA)*, 1997.

[38] A. Mestres, S. Abadal, J. Torrellas, E. Alarcón, and A. Cabellos-Aparicio, "A MAC Protocol for Reliable Broadcast Communications in Wireless Network-on-Chip," in *International Workshop on Network on Chip Architectures (NOCARC)*, 2016.

[39] J. Oh, M. Prvulovic, and A. Zajic, "TLSync: Support for Multiple Fast Barriers Using on-Chip Transmission Lines," in *International Symposium on Computer Architecture (ISCA)*, 2011.

[40] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob, "The Structural Simulation Toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, 2011.

[41] X. Yu, H. Rashtian, and S. Mirabbasi, "An 18.7-Gb/s 60-GHz OOK Demodulator in 65-nm CMOS for Wireless Network-on-Chip," *IEEE Transactions on Circuits And Systems -I: Regular Papers*, vol. 62, no. 3, 2015.

[42] X. Yu, S. P. Sah, H. Rashtian, S. Mirabbasi, P. P. Pande, and D. Heo, "A 1.2-pJ/bit 16-Gb/s 60-GHz OOK Transmitter in 65-nm CMOS for Wireless Network-on-Chip," *IEEE Transactions on Microwave Theory and Techniques*, vol. 62, no. 10, 2014.

[43] B. Xu, Y. Zhou, and Y. Chiu, "A 23-mW 24-GS/s 6-bit Voltage-Time Hybrid Time-Interleaved ADC in 28-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, 2017.

[44] S. Saxena, G. Shu, R. K. Nandwana, M. Talegaonkar, A. Elkholy, T. Anand, W. S. Choi, and P. K. Hanumolu, "A 2.8 mW/Gb/s, 14 Gb/s Serial Link Transceiver," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 5, 2017.

[45] F. Gutierrez, S. Agarwal, K. Parrish, and T. S. Rappaport, "On-chip Integrated Antenna Structures in CMOS for 60 GHz WPAN Systems," *IEEE Journal on Selected Areas in Communications*, vol. 27, no. 8, 2009.

[46] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *International Symposium on Microarchitecture (MICRO)*, 2009.

[47] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[48] C. Sun, C.-H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "DSENT - A Tool Connecting Emerging Photonics With Electronics for Opto-Electronic Networks-on-Chip Modeling," in *International Symposium on Networks-on-Chip (NOCS)*, 2012.

[49] C. W. Byeon, K. C. Eun, and C. S. Park, "A 2.65-pJ/bit 12.5-Gb/s 60-GHz OOK CMOS Transmitter and Receiver for Proximity Communications," *IEEE Transactions on Microwave Theory and Techniques*, 2020.

[50] S. Abadal, M. Iannazzo, M. Nemirovsky, A. Cabellos-Aparicio, H. Lee, and E. Alarcón, "On the Area and Energy Scalability of Wireless Network-on-Chip: A Model-Based Benchmarked Design Space Exploration," *IEEE/ACM Transactions on Networking*, vol. 23, no. 5, pp. 1501–1513, 2015.

[51] M. F. Chang, J. Cong, A. Kaplan, M. Naik, G. Reinman, E. Socher, and S.-W. Tam, "CMP Network-on-Chip Overlaid With Multi-Band RF-Interconnect," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2008.

[52] H. K. Mondal, S. Kaushik, S. H. Gade, and S. Deb, "Energy-Efficient Transceiver for Wireless NoC," in *International Conference on VLSI Design and International Conference on Embedded Systems (VLSID)*, 2017.

[53] S. Abadal, E. Alarcón, A. Cabellos-Aparicio, M. C. Lemme, and M. Nemirovsky, "Graphene-Enabled Wireless Communication for Massive Multicore Architectures," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 137–143, 2013.

[54] A. Ganguly, P. Pande, B. Belzer, and A. Nojeh, "A Unified Error Control Coding Scheme to Enhance the Reliability of a Hybrid Wireless Network-on-Chip," in *2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems*, pp. 277–285, IEEE, 2011.

[55] S. Deb, A. Ganguly, P. P. Pande, B. Belzer, and D. Heo, "Wireless NoC as Interconnection Backbone for Multicore Chips: Promises and Challenges," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 2, pp. 228–239, 2012.

[56] X. Timoneda, S. Abadal, A. Franques, D. Manessis, J. Zhou, J. Torrellas, E. Alarcón, and A. Cabellos-Aparicio, "Engineer the Channel and Adapt to it: Enabling Wireless Intra-Chip Communication," *IEEE Transactions on Communications*, vol. 68, no. 5, pp. 3247–3258, 2020.

[57] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros, "SPLASH-3: A Properly Synchronized Benchmark Suite for Contemporary Research," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016.

[58] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008.

[59] K. Duraisamy, H. Lu, P. P. Pande, and A. Kalyanaraman, "High-Performance and Energy-Efficient Network-on-Chip Architectures for Graph Analytics," *ACM Transactions on Embedded Computing Systems*, vol. 15, no. 26, pp. 1–26, 2016.

[60] X. Li, K. Duraisamy, J. Baylon, T. Majumder, G. Wei, P. Bogdan, D. Heo, and P. P. Pande, "A Reconfigurable Wireless NoC for Large Scale Microbiome Community Analysis," *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1653–1666, 2017.

[61] X. Li, K. Duraisamy, P. Bogdan, J. R. Doppa, and P. P. Pande, "Scalable Network-on-Chip Architectures for Brain–Machine Interface Applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 10, pp. 1895–1907, 2018.

[62] A. Franques, S. Abadal, H. Hassanieh, and J. Torrellas, "Fuzzy-Token: An Adaptive MAC Protocol for Wireless-Enabled Manycores," in *Design, Automation and Test in Europe Conference (DATE)*, 2021.

[63] K. Duraisamy, R. G. Kim, and P. P. Pande, "Enhancing Performance of Wireless NoCs With Distributed MAC Protocols," in *International Symposium on Quality Electronic Design (ISQED)*, 2015.

[64] N. Mansoor and A. Ganguly, "Reconfigurable Wireless Network-on-Chip With a Dynamic Medium Access Mechanism," in *International Symposium on Networks-on-Chip (NOCS)*, 2015.

[65] S. Jog, Z. Liu, A. Franques, V. Fernando, S. Abadal, J. Torrellas, and H. Hassanieh, "One Protocol to Rule Them All: Deep Reinforcement Learning Aided MAC for Wireless Network-on-Chips," in *Symposium on Networked Systems Design and Implementation (NSDI)*, 2021.

[66] M. M. Martin, D. J. Sorin, A. Ailamaki, A. R. Alameldeen, R. M. Dickson, C. J. Mauer, K. E. Moore, M. Plakal, M. D. Hill, and D. A. Wood, "Timestamp Snooping: An Approach for Extending SMPs," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[67] M. Martin, M. D. Hill, and D. A. Wood, "Token Coherence: Decoupling Performance and Correctness," in *International Symposium on Computer Architecture (ISCA)*, 2003.

[68] K. Strauss, X. Shen, and J. Torrellas, "Uncorq: Unconstrained Snoop Request Delivery in Embedded-Ring Multiprocessors," in *International Symposium on Microarchitecture (MICRO)*, 2007.

[69] N. Agarwal, L.-S. Peh, and N. K. Jha, "In-Network Snoop Ordering (INSO): Snoopy Coherence on Unordered Interconnects," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2009.

[70] B. K. Daya, C.-H. O. Chen, S. Subramanian, W.-C. Kwon, S. Park, T. Krishna, J. Holt, A. P. Chandrakasan, and L.-S. Peh, "SCORPIO: A 36-Core Research Chip Demonstrating Snoopy Coherence on a Scalable Mesh NoC With in-Network Ordering," in *International Symposium on Computer Architecture (ISCA)*, 2014.

[71] W.-C. Kwon and L.-S. Peh, "A Universal Ordered NoC Design Platform for Shared-Memory MPSoC," in *International Conference on Computer-Aided Design (ICCAD)*, 2015.

[72] B. K. Daya, L.-S. Peh, and A. P. Chandrakasan, "Low-Power on-Chip Network Providing Guaranteed Services for Snoopy Coherent and Artificial Neural Network Systems," in *Design Automation Conference (DAC)*, 2017.

[73] N. Kirman, M. Kirman, R. Dokania, J. F. Martinez, A. B. Apsel, M. A. Watkins, and D. H. Albonesi, "Leveraging Optical Technology in Future Bus-Based Chip Multiprocessors," in *International Symposium on Microarchitecture (MICRO)*, 2006.

[74] D. Vantrease, M. H. Lipasti, and N. Binkert, "Atomic Coherence: Leveraging Nanophotonics to Build Race-Free Cache Coherence Protocols," in *International Symposium on High Performance Computer Architecture (HPCA)*, 2011.

[75] 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," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2010.

[76] C. Batten, A. Joshi, V. Stojanovic, and K. Asanovic, "Designing Chip-Level Nanophotonic Interconnection Networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 2, pp. 137–153, 2012.

[77] C. Thraskias, E. Lallas, N. Neumann, L. Schares, B. Offrein, R. Henker, D. Plettemeier, F. Ellinger, J. Leuthold, and I. Tomkos, "Survey of Photonic and Plasmonic Interconnect Technologies for Intra-Datacenter and High-Performance Computing Communications," *IEEE Communications Surveys and Tutorials*, vol. 20, no. 4, pp. 2758–2783, 2018.

[78] A. Carpenter, J. Hu, J. Xu, M. Huang, and H. Wu, "A Case for Globally Shared-Medium on-Chip Interconnect," in *International Symposium on Computer Architecture (ISCA)*, 2011.

[79] A. Carpenter, J. Hu, O. Kocabas, M. Huang, and H. Wu, "Enhancing Effective Throughput for Transmission Line-Based Bus," in *International Symposium on Computer Architecture (ISCA)*, 2012.

[80] J. Oh, A. Zajic, and M. Prvulovic, "Traffic Steering Between a Low-Latency Unswitched TL Ring and a High-Throughput Switched on-Chip Interconnect," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2013.

[81] J. W. Holloway, G. C. Dogiamis, and R. Han, "Innovations in Terahertz Interconnects: High-Speed Data Transport Over Fully Electrical Terahertz Waveguide Links," *IEEE Microwave Magazine*, vol. 21, no. 1, pp. 35–50, 2020.

[82] B. M. Beckmann and D. A. Wood, "TLC: Transmission Line Caches," in *International Symposium on Microarchitecture (MICRO)*, 2003.

[83] J. L. Abellán, J. Fernández, and M. E. Acacio, "Efficient Hardware Barrier Synchronization in Many-Core CMPs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1453–1466, 2012.

[84] Y. Fu, T. M. Nguyen, and D. Wentzlaff, "Coherence Domain Restriction on Large Scale Systems," in *International Symposium on Microarchitecture (MICRO)*, 2015.