ON-CHIP WIRELESS MANYCORE ARCHITECTURES

BY

ANTONIO MARIA FRANQUES GARCIA

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Doctoral Committee:

Professor Josep Torrellas, Chair
Professor Sasa Misailovic
Professor Haitham Al-Hassanieh
Professor David Padua
Professor Tushar Krishna, Georgia Institute of Technology

ABSTRACT

Recent computer architecture trends herald the arrival of massive multiprocessors with more than a hundred processor cores within a single package. In this setting, on-chip communication becomes increasingly important, as parallel programs increase the amount of data sharing and signaling between cores. Unfortunately, traditional on-chip networks have been proven to not scale well in terms of latency or energy consumption, slowing down the computation, and jeopardizing the scalability of hundred-core processors.

On-chip wireless networks are a novel interconnect paradigm that holds considerable promise for overcoming the communication challenges left unmet by wired networks-on-chip, and for enabling such massive multicore chips.

In this thesis, we propose several applications of on-chip wireless technology for manycore architectures, namely: *Replica*, a manycore that uses wireless communication for synchronization and communication-intensive data; *WiDir*, which uses on-chip wireless technology to augment a conventional invalidation-based directory cache coherence protocol; and *WiPackage*, a chiplet-based architecture that uses a wireless network to attain scalable communication.

Besides the aforementioned applications of on-chip wireless technology, in this thesis we also address some of the challenges that come with the technology. One of the main challenges of on-chip wireless technology is the design of methods that provide fast and efficient access to the wireless channel while adapting to the constant traffic changes within and across applications. Hence, we propose *Fuzzy-Token*, a simple medium access control protocol that leverages the unique properties of the on-chip scenario to deliver efficient and low-latency access irrespective of the application characteristics.

To my mom.

ACKNOWLEDGMENTS

I sincerely believe that there is a determining factor in the success or failure of the challenges in which we embark throughout our lives: the fact that someone believes in you. That is why this section is dedicated to all those people who have supported and with great patience taught me throughout the life project I undertook when I considered starting this career.

To my advisor, Professor Josep Torrellas, for his tireless passion, patience, and guidance. For giving me the opportunity to be part of his incredible team, and for always welcoming me with the door open.

To my close collaborator and even closer friend, Sergi, for making this project possible, for his intense involvement in all my work, and for his encouragement whenever things did not go as planned.

To my undergraduate advisors, Professors Alicia Cordero and Juan Ramon Torregrosa, for sparking the interest of research in me, for giving me the skills and experience that allowed me to later enter the Ph.D. program in Illinois, for their support, and for their trust in their recommendation letters.

To my uncle, for guiding me when I needed it most, for his constant dedication and interest in my personal career. For teaching me the values that honor a person, and for awakening my interest in the knowledge of everything that surrounds us.

To my wife, the rest of my family, and my friends, for giving me their love and support, for filling my life with good times and memories; for making me happy.

TABLE OF CONTENTS

CHAPTER 1: THESIS CONTRIBUTION AND OVERVIEW

Semiconductor devices have shrunk relentlessly during the last decade, leading to stunning levels of chip integration. As a result, recent years have witnessed the emergence of computing architectures that integrate up to a thousand processor cores and memory on a single die. This includes heterogeneous systems with integrated GPUs [1], neural network accelerators [2], server processors [3], and general-purpose manycore CPUs [4]. Processor cores communicate among them and with memory to share data and synchronize execution. The importance of communication grows with the number of cores, to the point of having become the key performance bottleneck in manycore processors [5, 6].

Modern processors rely on Networks-on-Chip (NoCs) to serve inter-core traffic [7, 8, 9, 10, 11]. NoCs are essentially packet-switched networks of on-chip routers and links, typically arranged in a 2D grid as shown in Figure 1.1. Routers enqueue packets, compute routes, arbitrate, and dequeue packets in each hop towards the destination [5], incurring delays and energy consumption. To connect the routers, the mesh topology is the *de facto* standard due to its simple layout and low link length [5]. While this approach is more scalable and efficient than buses, the average hop count scales proportionally to $\sqrt{N}$, where N is core count, and therefore the latency and energy of NoCs start to become a problem at hundred cores [5, 6]. This is especially so for broadcasts and for messages that need to traverse a high number of routers before arriving to the intended destination [12]. The increasing latency raises the likelihood of stalling remote processor cores as they wait for results or to synchronize, which throttles execution. Slowdowns of $2-3\times$ have been estimated in processors with several tens of cores [12, 13] and even worse slacks are expected at higher core counts. There are other, high-radix topologies that reduce the network diameter [14], but at the cost of a non-trivial chip area and energy cost at the routers.

To guarantee the scalability of processor architectures to thousand cores and beyond, emerging interconnect technologies such as 3D [15, 16, 17], nanophotonics [18, 19, 20], or wireless [21, 22, 23] have been proposed, each with its pros and cons [24]. Wireless technology has shown promise due to its inherent broadcast nature and very low latency for chip-wide transmissions, better than in conventional NoCs by an order of magnitude [25]. Moreover, the lack of additional wiring between cores provides a network-level flexibility unattainable with other technologies [26].

Wireless Network-on-Chip (WNoC) is enabled by recent advances in CMOS RF technology, allowing the on-chip integration of millimeter-wave and sub-terahertz antennas and transceivers [22, 27, 28, 29, 30, 31, 32, 33]. WNoC architectures [34, 35, 36, 37, 38, 39, 40, 41] 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 is integrated with an

antenna and a transceiver, as shown in Figure 1.1. Small antennas in the mmWave bands and beyond [30, 42, 43, 44] can be feasibly integrated, and use a broadcast-based protocol [45, 46, 47]. Figure 1.1 shows vertical monopole antennas based on Through-Silicon Vias (TSVs), which perforate the bulk silicon [44]. 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 dielectric loss at the bulk silicon [48, 49, 50]. However, these losses are tolerable [31], and prevent the enclosed package to act as a reverberation chamber.

More importantly, complete multiprocessor architectures based on the WNoC paradigm have demonstrated execution speedups of up to 10× (1.89× in average) and reduced the multiprocessor energy consumption by an average of 38% [26, 51]. This has been achieved by considering bandwidths of a few tens of GHz and low-order modulations achievable with current technology.

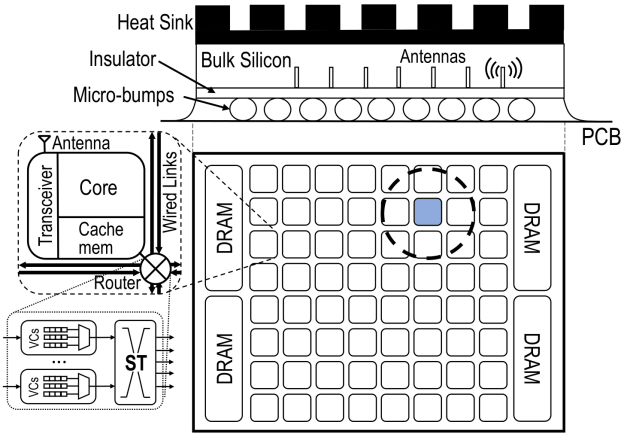


Figure 1.1: Wired NoC mesh augmented with a wireless NoC, within a conventional flip-chip package with one vertical monopole antenna and transceiver per core.

1.1 OVERVIEW

This thesis presents the application of wireless on-chip communication in manycore architectures. These architectures take advantage of the natural broadcast capabilities of wireless communication in different ways. The first work, *Replica*, proposes a manycore architecture that uses wireless communication for synchronization and communication-intensive ordinary data. To deliver high performance, *Replica* supports an adaptive wireless protocol and selective message dropping. The adaptive wireless protocol introduced and used in *Replica* is then extended and studied in much more detail in the second work, *Fuzzy-Token*. Besides describing the protocol in detail, this second work focuses on motivating the need for such an adaptive protocol in WNoC scenarios. It

also explains the different parameters available for tuning, and finally shows an analysis of different values and combinations of values for the parameters previously described, for different types of synthetic traffic and real application traces. In order to improve the programmability of the design proposed in the first work, the third work uses wireless communication to augment a conventional directory-based invalidation cache coherence protocol. The resulting protocol, called *WiDir*, seamlessly transitions between wired and wireless coherence transactions for the same data based on the access patterns, in a programmer-transparent manner. Lastly, the fourth contribution focuses in the novel chiplet paradigm, and its scalability concerns beyond a handful of chiplets. In this fourth contribution, we present *WiPackage*. *WiPackage* is comprised of a set of chiplets, each operating as a separate shared-memory domain with its own separate wired on-chip network. *WiPackage* provides a message-passing wireless-augmented interconnection network, aimed at sending data across chiplets.

1.2 SUMMARY OF CONTRIBUTIONS

The design of a wireless manycore inherently involves trade offs between several elements such as programmability, performance, and area and power overhead. We present below a brief summary of the challenges that each of our works faced, and the novel techniques we proposed to overcome them.

1.2.1 *Replica*: A Wireless Manycore for Communication-Intensive and Approximate Data

Data access patterns that involve fine-grained sharing, multicasts, or reductions have proved to be hard to scale in shared-memory platforms. Recently, wireless on-chip communication has been proposed as a solution to this problem, but a previous architecture has used it only to speed-up synchronization [45]. An intriguing question is whether wireless communication can be widely effective for ordinary shared data.

This work presents *Replica*, a manycore that uses wireless communication for communication-intensive ordinary data. To deliver high performance, *Replica* supports an adaptive wireless protocol and selective message dropping. On the software side, the published paper that came out of this work [46] describes the computational patterns that leverage wireless communication, programming techniques to restructure applications, and tools that help with automation. However, my contributions in this project focused instead on the hardware aspect of the work, such as the design and implementation of the adaptive wireless protocol, the hardware support for the selective packet dropping, and the area and energy analysis of the proposed architecture. As such, in my thesis I only describe the aforementioned hardware aspects of this work.

Our results show that wireless communication is effective for ordinary data. For 64 cores, *Replica* obtains a mean speed-up of 1.76x over a conventional machine. The mean speed-up reaches 1.89x if approximate-computing transformations are enabled. The average energy consumption is substantially reduced by 34% (or 38% with approximate transformations), and the area increases only modestly.

1.2.2 *Fuzzy-Token*: An Adaptive MAC Protocol for Wireless-Enabled Many-Core CMPs

One of the main challenges of the Wireless Network-on-Chip paradigm resides in the design of methods that provide fast and efficient access to the wireless channel while adapting to the constant traffic changes within and across applications. Existing approaches are either cumbersome or do not provide the required adaptivity. In this work we propose *Fuzzy-Token*, a simple protocol that leverages the unique properties of the on-chip scenario to deliver efficient and low-latency access irrespective of the application characteristics. We substantiate our claim via simulations with a synthetic traffic suite and real application traces. *Fuzzy-Token* consistently provides a latency among the lowest of the evaluated protocols. On average, the latency provided by *Fuzzy-Token* is 28.5× lower than previously proposed WNoC MAC protocols, such as BRS (4.4× in geometric mean), and 4.1× lower than Token (2.6× in geometric mean), as evaluated and discussed in depth in this work.

1.2.3 *WiDir*: A Wireless-Enabled Directory Cache Coherence Protocol

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 work 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 work, 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.

1.2.4 *WiPackage*: Wireless-Enabled Collective Communication in a Many-Chiplet Package

As the semiconductor industry moves to smaller process nodes, the cost of producing large dies continues to increase. In response, industry has started shifting towards disintegrated designs in which a single chip is broken down into multiple smaller *chiplets*. However, a main challenge still resides in the design of interconnection networks that provide fast and efficient chiplet-to-chiplet communication. While current low chiplet counts have allowed manufacturers to opt for simple fully-connected and adapted star topologies, more scalable solutions are required for the long term.

In this work, we propose and evaluate a hybrid wired-wireless hierarchical memory and network chiplet-based architecture, called *WiPackage*. *WiPackage* is comprised of a set of chiplets, each operating as a separate shared-memory domain with its own separate wired on-chip network. *WiPackage* provides a message-passing wireless-augmented interconnection network, aimed at sending data across chiplets. When compared to a purely wired-based chiplet architecture, we show that *WiPackage* is able to speed up common message-passing collective primitives by up to 9x, and common message-passing benchmarks by 2.05x, while adding a modest area and power cost.

In the following sections, we present each of the aforementioned works in detail along with their evaluation results.

CHAPTER 2: *REPLICA*: A WIRELESS MANYCORE FOR COMMUNICATION-INTENSIVE AND APPROXIMATE DATA

2.1 INTRODUCTION

Data access patterns where multiple threads interleave reads and writes to the same set of variables in a fine-grained manner and without much per-thread locality do not scale well in shared-memory multiprocessors. They create many network messages, inducing communication bottlenecks. To alleviate this problem, commercial vendors (e.g., [52, 53, 54, 55]) and researchers (e.g., [18, 56, 57, 58, 59, 60, 61, 62]) have proposed various hardware techniques. They include new synchronization and cache coherence protocol improvements, special networks, and new communication technologies such as optics and transmission lines.

Recently, on-chip wireless communication has emerged as a promising alternative that supports fine-grained data sharing with low-latency, and is broadcast-friendly [21, 51, 63, 64]. In this environment, broadcasting a short message of 80 bits takes about 4 ns, which is about two orders of magnitude lower than in conventional on-chip networks. For example, the recent WiSync manycore [51] augments each core with a small antenna and a transceiver. It supports low-latency implementations of synchronization primitives, such as locks and barriers. WiSync stores the state of synchronization variables in a small, per-core Broadcast Memory (BMem) that has identical contents in all of the cores. Writes to the BMem are broadcasted, updating all the BMems at the same time, while reads are satisfied from the local BMem.

While WiSync shows the attractiveness of on-chip wireless communication, it is only tailored to speed-up synchronization operations. An intriguing question is whether the wireless communication and BMem support can be used to speed-up transfers of ordinary data.

Using wireless communication for ordinary data faces two fundamental challenges: the bounded size of BMem and the limited bandwidth of the wireless communication channel. WiSync does not completely experience these challenges, as the synchronization variables typically fit in the 16KB BMem and do not consume much of the wireless channel bandwidth. In contrast, ordinary data does not fit in BMem, and its frequent updates may cause contention in the wireless channel. It is therefore necessary to judiciously select the subset of the data that will benefit the most from the wireless communication, and place it in BMem.

In this work, we present *Replica*, a manycore architecture and software interface that enables efficient use of wireless communication for ordinary data. We tailor *Replica* to speed-up *communication-intensive shared data* – whose accesses typically induce substantial overheads in standard cache hierarchies. Our analysis presents several common communication-intensive patterns. They include broadcasts, regular many-to-many interactions, irregular many-to-many interactions, and reduc-

tions. To handle these patterns, the main author of [46] presented: (i) a software API that exposes BMem to the software developer, and (ii) transformations and tools for selecting communication-intensive data and restructuring applications for improved BMem and wireless channel use. A software developer can use two operations, *approximate locks* and *approximate stores*, to optimize applications that can tolerate noise. Further, she can combine these operations with existing approximation techniques that trade accuracy for reduced communication and/or data size.

However, as mentioned in Section 1.2.2, in my thesis I do not claim the aforementioned proposed software techniques. Instead, I claim, describe, and focus on the hardware aspects of the *Replica* architecture. As such, in this work we propose two hardware-based techniques to reduce contention and latency in the wireless channel. First, we introduce an adaptive wireless protocol. The protocol dynamically identifies whether the data transmissions in the execution are sparse or bursty, and applies a random-access or a token-passing protocol, respectively.

Second, *Replica* provides hardware support for selectively dropping packets if they carry certain types of data and if the sender encounters a certain level of channel contention. Together, these techniques have a greater impact on *Replica* than on standard architectures, due to the limited BMem size and the limited wireless channel bandwidth.

Our results show that *Replica* effectively uses wireless communication for ordinary data. We evaluated *Replica* with 10 applications from graph analytics, vision, and numerical simulation. For 64-core executions, *Replica* speeds-up the applications over a conventional machine by a geometric mean of 1.76x for exact computation and 1.89x for approximate computation. Further, *Replica* substantially reduces the average energy consumption by 34% (or 38% with approximate computation). Finally, the area increase is small, and the developer effort is modest.

2.2 BACKGROUND

Figure 2.1 shows the WiSync architecture [51]. WiSync augments every core of a manycore with a *Broadcast Memory* (BMem), a wireless transceiver, and two antennas (of which we will only consider one). The transceiver has two main modules, namely the physical layer (PHY) and the Medium Access Control (MAC). The PHY module serializes and modulates the data to transmit, detects collisions, and demodulates and deserializes data at reception. The MAC module manages the access to the channel by scheduling transmissions and handling collisions [65].

The BMem is a direct-mapped memory of a size similar to an L1 cache. The BMems of all the cores contain the exact same variables that are kept coherent through wireless updates. A core accesses its BMem with plain loads and stores. Based on the physical address of the location accessed, a load or store request is sent either to the L1-L2 hierarchy or to the BMem.

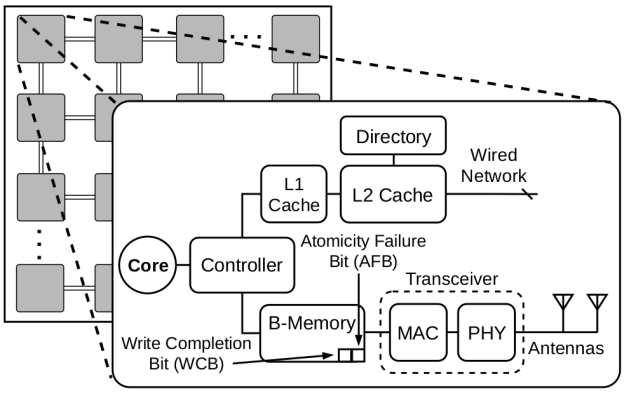


Figure 2.1: WiSync manycore.

When a core writes to a BMem location, it generates a message to be broadcasted through the wireless network. All BMems (including the local one) are updated simultaneously. This design ensures a total order of writes to BMems across all cores. It also ensures that, at all times, all cores have the same values in their BMems. Loads that access the BMem simply read the local copy of the data.

WiSync uses 5 cycles to transmit a 77-bit packet, which corresponds to a 64-bit write. In the second cycle, the transceiver listens if there was a collision with another packet in the first cycle. If there was no collision, in the next three cycles it sends the rest of the packet with guaranteed no collision. Otherwise, the transfer is aborted, and the senders will retry sending their packets after a randomized, exponentially-increasing number of cycles. This carrier-sensing protocol with exponential backoff adapted to the on-chip scenario is called Broadcast Reliability Sensing (BRS) [66].

WiSync supports read-modify-write instructions that form the basis of synchronization primitives. These instructions leverage a special hardware bit called the Atomicity Failure Bit (AFB). The AFB is set in hardware if, in between the read and write of the local core in a read-modify-write operation to a BMem address, an external core succeeds in performing a write to the same location. In this case, the write of the local core does not occur, and the atomic operation fails.

2.3 *REPLICA* OVERVIEW

Replica extends the WiSync architecture in several ways, including the ability to store ordinary (i.e., non-synchronization) *and* synchronization data in the BMem. In this section, we outline the key features of *Replica*.

Broadcast Memory. *Replica* provides an API to allocate data in BMem (*which I do not claim in my thesis*). To store an array `a` in BMem, the developer only needs to change the allocation site to

```
float* a = wireless_malloc(n*sizeof(float));
```

All accesses to the array elements are automatically directed to the BMem, and writes use the wireless channel. Programs do not require any additional developer or compiler interventions, as the BMem is memory mapped. A call to `wireless_free` deallocates the memory.

Since the amount of communication-intensive data may exceed the size of the BMem, it is essential to restructure communication-intensive data structures to fit as much as possible in BMem. We present transformations that allow the flexible storage of a fraction of communication-intensive data in BMem. Our approach rests on two observations: (i) in many applications, the size of communication-intensive data increases at a much slower rate than the full input data size, and (ii) since BMem is memory-mapped, we can transform the data structure layout with little performance penalty.

Adaptive Wireless Protocol. In *Replica*, the wireless network utilization varies across applications and even within an application. For applications with sparse transmissions, the carrier-sensing protocol from WiSync is sufficient. However, applications with high or bursty load perform better with a token-passing protocol, in which only the node that owns the token can transmit. *Replica*'s MAC module supports both protocols, and switches between the two to adapt to the characteristics of the application. This process is automatic and does not require input from the developer.

Approximate Broadcast Memory. To further reduce the wireless channel contention, *Replica* uses a section of BMem for approximate data. In this section, the messages for data updates and locking operations may occasionally be dropped, if the latency to perform the access exceeds a certain threshold. Approximate data is allocated as

```
float* a = approx_wireless_malloc(n*sizeof(float));
```

and is stored in a specially-designated section of the BMem. Approximate BMem supports two operations that selectively drop packets (*which I do not claim in my thesis*):

- **Approximate Store:** It assigns a value `val` to a variable `var` if the write succeeds within a specified latency threshold. Approximate stores can be *unchecked* or *checked*. In the former, if the message is dropped, the computation silently continues without informing the software. In the latter, software can use the call `approx_stac(var, val)` (for store approximate checked) to find out if the write succeeded. Unchecked stores use the same opcode as standard stores. Checked stores use a different opcode.
- **Approximate Lock:** `approx_lock(m)` attempts to obtain the lock `m` within a specified latency threshold. If it succeeds, it returns a success code. If it does not succeed, either

because it spins for too long on an already taken lock, or because it takes too long to obtain the wireless network to send the lock acquire update, it returns a failure code. In this case, the software skips the critical section and the unlock operation.

Tool Support. *Replica* includes a tool infrastructure (*which I do not claim in my thesis*) – a profiler, compiler passes for transformations, and an autotuner – to help the developer restructure the application. Our experience shows that these tools can automate many tasks and enable seamless adaptation of program code to versions of *Replica* with different hardware characteristics.

2.4 *REPLICA* ARCHITECTURE

This section describes the two main architectural features of *Replica* that improve over WiSync: the adaptive wireless protocol and the approximate BMem.

2.4.1 Adaptive Wireless Protocol

In WiSync, the wireless network is utilized relatively little, and its traffic patterns are simple. Therefore, a wireless MAC protocol like BRS is appropriate. In BRS, when a packet collides, the sender does not try to resend it at the next available opportunity. Instead, it waits for a backoff period before retrying. Specifically, it considers a period of $2^c - 1$ cycles (where c is the number of collisions that the packet has suffered so far), picks a random number within that period, and waits for that number of cycles. The result is a backoff that increases exponentially.

In *Replica*, the use of the wireless network is more complex, and its utilization patterns vary across applications and within an application. Hence, while *Replica* retains the BRS protocol for applications or sections of applications with sparse transmissions, it also supports a *Token Ring* protocol in applications with frequent or bursty transmissions [67].

In the Token Ring protocol, there is a logical token that is owned by different nodes at different times. At any time, only the node that owns the token can transmit. At the end of a packet transmission, or if the owner node remains silent for one cycle, the token is passed to the next node following a logical ring.

Replica introduces an adaptive wireless protocol that intelligently switches between the two protocols, adapting to the characteristics of the execution. Specifically, one of the nodes (which we call the *master node*) has a hardware mechanism in its transceiver that monitors the use patterns of the wireless channel and chooses the protocol to use. The mechanism's hardware consists of two counters and simple logic to perform a division and a comparison:

- When running in BRS mode, the counters are called *Coll* and *NoColl*. Every time any core sends a packet, the mechanism checks for a collision. If there is no collision, it increments *NoColl*; otherwise, it increments *Coll*.

- When running in Token Ring mode, the counters are called *Idle* and *Busy*. If the mechanism observes an idle cycle, a core missed its opportunity to transmit, and the *Idle* counter is incremented. Otherwise, a packet is transmitted, and the *Busy* counter is incremented.

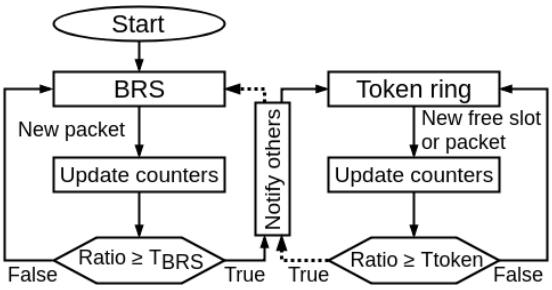


Figure 2.2: Adaptive wireless protocol.

Figure 2.2 presents the operation of the mechanism. The mechanism starts in BRS mode. Following an update to either counter, the hardware calculates the ratio *Coll*/*NoColl*. If this ratio is equal or higher than the T_{BRS} threshold, the transceiver in the master node clears the counters and informs all the nodes to switch to the Token Ring mode.

From then on, the nodes use the Token Ring protocol, and the mechanism in the master node's transceiver computes the ratio *Idle*/*Busy*. When the ratio is equal or higher than the T_{token} threshold, it means that only a few cores have data to transmit and are unnecessarily waiting to get the token. In this case, the transceiver clears its counters and notifies all the nodes to switch to the BRS mode.

2.4.2 Approximate Broadcast Memory

The programmer can tag a certain range of BMem addresses as containing approximable variables. For variables allocated in this range, *Replica* can drop wireless packets through approximate stores and locks. This support reduces the pressure when the wireless channel is highly contended.

Approximate Stores. Stores to approximable variables use the wireless channel like any other BMem variable, but the stores can be dropped if the contention for the wireless channel is high. In this case, neither the remote BMems nor the local BMem are updated.

The MAC module keeps track of the waiting delay of droppable write packets. If the waiting delay exceeds threshold T_{drop}, the packet is dropped and the store is canceled. This automatically

and dynamically reduces the contention and power in communication-intensive periods. This way, programmers can perform certain approximations only if it is strictly necessary, to avoid saturating the network.

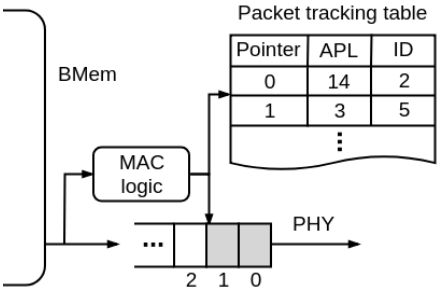


Figure 2.3: Packet tracking.

Figure 2.3 presents the hardware support. It consists of the Packet Tracking Table (PTT) in the MAC module. When a packet is sent to the BMem, it is deposited in a buffer in the transceiver. At this point, if the packet is for a droppable store, it allocates an entry in the PTT. Each PTT entry has three fields. The first one is a pointer to the position of the packet in the buffer. The second one contains the total time that the packet is expected to wait before being sent. We call it the *Accumulated Packet Latency* (APL) for the packet. When the APL for a packet reaches T_{drop}, the packet is dropped. The third field is only used by *checked* approximate stores, and contains the ID of the register that will receive the transmission outcome.

The algorithm to set the APL is protocol dependent. When running in BRS mode, the APL for a packet is set as follows. When the packet arrives at the queue, it sets its APL to the minimum number of cycles that it will have to wait to be sent. This time is equal to the number of cycles remaining in the backoff period of the first packet in the queue, plus the number of packets in the queue times the duration of a packet send (i.e., 5 cycles including the cycle listening for collisions). Note that this computation includes both droppable and non-droppable packets. Further, when a packet is sent and collides with another packet, the hardware calculates the backoff time, and this backoff time is added to the APL of all the droppable packets in the queue (including the packet that collided). Finally, when a packet is dropped, the hardware computes the number of waiting cycles to remove. This number is 5 cycles plus the remaining cycles in the backoff period (if this was the first packet in the queue). This number is subtracted from the APL of all the droppable packets that are queued after the dropped one.

When running in Token Ring mode, the APL for a packet is set as follows. When the packet arrives at the queue, it sets its APL to the number of cycles that it will have to wait assuming an idle wireless channel. This number includes: (i) for the first packet in the queue, the number of cycles remaining until it can start the sending plus the cycles to send the packet, plus (ii) for each

of the other packets in the queue, the number of nodes in the machine plus 3 (since a packet takes 4 cycles to be sent now). Further, every time that another node transmits, the hardware adds 3 cycles to the APL of all the entries in the table. Finally, when a packet is dropped, the number of waiting cycles to remove from all the subsequent droppable packets is: (i) if the dropped packet was the first one in the queue, the number of cycles remaining until it could start the sending plus the cycles to send the packet, or (ii) if the packet was not the first one in the queue, the number of nodes in the machine plus 3.

When a packet is dropped, it is removed from the buffer. Further, its PTT entry is removed and, for checked approximate stores, the register indicated in the PTT entry is set. When a droppable packet is sent successfully, the same actions occur, except that the register indicated in the PTT entry is cleared.

Checked Approximate Stores. In unchecked approximate stores, an update may be silently dropped. In the checked version, the programmer obtains the outcome of the store. The programmer uses a special *store approximate* instruction (*which I do not claim in my thesis*), `sta R1, R2, var_addr`, which takes three arguments: a register `R1` containing the value to store, a register `R2` that indicates if the store was either successfully committed (R2=0) or dropped (R2=1), and the address `var_addr` to receive the data. R2 is the register recorded in the corresponding entry of the PTT. We expose the API call `approx_stac(var, val)`, which is implemented as `sta R1, R2, var_addr; ret R2`. It takes the variable to update (`var`) and the update (`val`), and returns whether the write succeeded (zero) or not (one).

Approximate Locks. To support approximate locks, *Replica* introduces a new hardware bit in the BMem controller called the *Write Drop Bit* (WDB). The hardware sets the WDB bit when a write packet belonging to a read-modify-write instruction is dropped. The WDB bit remains set until the software reads it, at which point the bit automatically clears. In this way, the programmer is aware of whether an approximate lock has been dropped. Since there is only one read-modify-write instruction executing at a time per core, a single WDB bit is enough.

Figure 2.4 shows the lock acquire routine for an approximate lock (*which I do not claim in my thesis*). The code tries to acquire lock `m` in address `m_addr` using an exchange instruction. An approximate lock fails and returns -1 when either (i) the software has unsuccessfully tried to acquire the lock for more than MAX attempts, or (ii) the write packet of the latest read-modify-write instruction that attempted to acquire the lock is queued for T_{drop} cycles. Condition (i) is implemented in software, using variable `Num_spins` (Line 6). When condition (ii) occurs, the write packet is dropped, the WDB bit gets set, and the exchange instruction terminates without performing the write. In this case, the exchange register `R` may have read the new value, but the

```
1       Num_spins = 0
2       R = TAKEN
3      Try: exchange R, m_addr
4       if (WDB) return -1 //packet dropped
5       if (R == TAKEN){   //failure
6         if(Num_spins++ > MAX)
7           return -1  //too many failures
8         jmp Try
9       }
10      else {     //R is UNTAKEN
11        if (AFB){    //atomicity failure
12           //no write occurred
13           R = TAKEN
14           jmp Try
15        }
16        else return 0  //success
17      }
```

Figure 2.4: Approximate lock acquire.

write to `m_address` has not occurred. Hence, the first action that the software takes after the exchange is to check WDB (Line 4). Irrespective of the current value of `R`, if WDB is set, the function returns -1.

The software then checks `R`. If `R`'s value is still TAKEN and the number of tries is no higher than MAX, the software retries the exchange (Line 8). If `R`'s value is UNTAKEN, we still need to do a final check. WiSync [51] requires the software to check the Atomicity Failure Bit (AFB) (Line 11). If the AFB is set, it means that another node updated `m_address` between the local read and the local write, and that the local write failed. In this case, the software resets `R` to TAKEN and retries the exchange (Line 14).

When the software finds out that an approximate lock operation has failed, it skips the critical section and the subsequent unlock operation. The author of *Replica*'s proposed software techniques described the software transformation in the paper [46].

We have also designed a similar algorithm for compare and swap (CAS) synchronization.

2.4.3 Other Features

Two additional *Replica* enhancements over WiSync's wireless hardware are related to the capacity of BMem. First, since the BMem is bigger in *Replica* than in WiSync, it needs more address bits and is slower. Second, since different applications need different BMem sizes, *Replica* organizes the BMem in chunks. The chunks that are not allocated by the application are power-gated to save energy.

Table 2.1: Summary of the applications.

Name	Description	Input
Water [68]	Simulation of water molecules (nsquared)	1000 molecules for 10 steps
BFS [69]	Breadth-first search	p2p-gnutella31 (from [70])
SSSP [69]	Single source shortest path	p2p-gnutella31 (from [70])
Pagerank [69]	Compute pagerank for nodes in a graph	p2p-gnutella31 (from [70])
CC [69]	Compute connected components of a graph	p2p-gnutella31 (from [70])
Bodytrack [71]	Track a body pose through images	4 frames, 1000 models
Streamcluster [71]	Cluster streams of points	4096 pts, 20 centers
Volrend [68]	Render a 3D object	head
Community [69]	Compute modularity of a graph	p2p-gnutella31 (from [70])
Canneal [71]	Find optimal routing for gates on a chip	10000 elements

2.5 METHODOLOGY

To evaluate *Replica*, we perform cycle-level architectural simulations using Multi2sim [72]. We run a variety of applications from SPLASH-2 [68], PARSEC [71], and the CRONO [69] graph suite.

2.5.1 Applications

Table 2.1 lists the 10 applications, what they do, and the inputs we use in the evaluation.

Data Sharing Patterns. The benchmark applications have different data-sharing patterns. Water has broadcast communication. The graph applications (BFS, Pagerank, SSSP, CC, and Community) have irregular, mostly many-to-many communication. Volrend mainly contains communication between neighbors, but also has broadcast communication. Canneal has an irregular communication pattern, due to locks. Bodytrack and Streamcluster have one-to-many communications and reductions.

Inputs and Metrics. For the SPLASH-2 and PARSEC applications (except Streamcluster), we use the same input sets as WiSync. For the graph applications, we use input sets from SNAP [70]. The input set sizes were chosen to allow detailed simulation runs that ranged between 4 and 48 hours per run.

Other Programs. We also analyzed other applications from the SPLASH-2 and PARSEC suites. As noted in previous characterizations [73], most of the remaining programs are data-parallel (e.g., blackscholes and swaptions) or implement regular algorithms with limited sharing, typically among neighbors (e.g., fluidanimate and raytrace). Since we do not expect *Replica* to improve performance for such computational patterns, we do not evaluate such applications.

2.5.2 Architecture Configurations

We analyze three configurations of *Replica*:

- Wireless-Locks (**WL**): it allocates only synchronization variables in BMem. It extends WiSync with the adaptive wireless protocol, and a BMem size that holds all the synchronization variables: 23KB in Water, 39KB in Canneal, and less than 1KB in the rest of the applications.
- Wireless-Optimized (**WO**): it extends WL by allocating some ordinary data in the BMem and applying the Data Splitting and Lock Coarsening transformations proposed by the main author of *Replica* in [46]. These transformations preserve the program semantics.
- Wireless-Approximate (**WA**): it extends WO by applying the approximation and non-approximation transformations, and checked and unchecked approximate stores proposed by the main author of *Replica* in [46].

We compare these configurations to a conventional architecture without BMem or wireless network in three configurations: Baseline (**B**) runs the original application, Optimized (**O**) augments B with the transformations in WO, and Approximate (**A**) augments O with the transformations in WA except those that need hardware support (e.g., approximate stores).

2.5.3 Energy Models

We model the energy consumed by the cores and the memory hierarchy with McPAT [74] and CACTI [75], and the energy of the wired links and routers with DSENT [76]. For the wireless hardware, we compute the power and area consumed per core using data in the literature for 65nm. Specifically, for the transceiver, we use a micrograph and data from [31, 77, 78] to estimate an area of 0.25mm^2 (including passives) and a power of 30mW. For the data converter, based on [79], we estimate an area of 0.03mm^2 and a power of 0.72mW. For the serializer and deserializer, data from [80] indicates an area of 0.04mm^2 and a power of 10.8mW. Finally, for the antenna, [81] shows a simple dipole that consumes an area of 0.04mm^2. We double the area to 0.08mm^2 to make it more realistic. Therefore, the overall RF circuit with passives consumes 0.4mm^2 and 41.5mW.

However, following [31, 82], we can power gate the transmitter's power amplifier and the receiver's low noise amplifier when not in use. This saves 10mW for the transmitter and 10mW of the receiver. The remaining components (serializers, data converters, oscillators, mixers, and detectors) are always on. Moreover, since our data comes from 16Gb/s systems and we use a system with 20Gb/s, we need to scale up the power consumption linearly. This gives us the following total

Table 2.2: Architecture modeled. RT means round trip.

General Parameters	
Architecture	Manycore with 32–64 cores at 22nm technology
Core	Out of order, 2-issue wide, 1GHz, x86 ISA
ROB; ld/st queue	64 entries; 20 entries
L1 I+D caches	Private 32KB WB, 2-way, 2-cycle RT, 64B lines
L2 cache	Shared with per-core 512KB WB banks
L2 bank	8-way, 6-cycle RT (local), 64B lines
Cache coherence	MOESI directory based
On-chip network	2D-mesh, 4 (default), 2 or 1 cycles/hop, 128-bit links
Off-chip memory	Connected to 4 mem controllers, 110-cycle RT
Replica Parameters	
Per-core BMem	Up to 512KB, in 32KB chunks 6-cycle RT, 64-bit wide line
Wireless channel	20Gb/s; 1 cycle for collision detection
MAC Protocols	BRS (exponential backoff), token passing in ring
MAC Thresholds	$T_{BRS} = 0.4$, $T_{token} = 15$
T_{drop}	40–2500 cycles
Transceiv+Anten	Area: 0.4mm^2; TX/RX/idle: 39.4/39.4/26.9mW
Power gating	Analog amplif. (transient: 1.14 pJ), unused BMem

values for the per-core RF circuitry at 65nm: 39.4mW when the transmitter is idle, 39.4mW when the receiver is idle, 26.9 mW when both are idle, and a total area of 0.4mm^2.

The next step would be to scale these numbers to the 22nm technology assumed for the core. Several authors [83, 84] argue that the area reduces linearly with the feature size. However, we conservatively use no scaling, and keep the area at 0.4mm^2 and the power at 39.4mW with gating. Note that these numbers are much higher than those used in WiSync, which are 0.14mm^2 for the area and 18mW for the power. Finally, using the amplifier consumption of [31], and the power-gating overheads of [82], we estimate a transient energy of 1.14pJ.

2.5.4 Simulator Implementation

We use cycle-level execution-driven simulations using the Multi2sim [72] simulator. We model a manycore with 32–64 cores at 22nm technology running at 1GHz. Table 2.2 shows the parameters of the architecture. Each tile has a 2-issue out-of-order core, 32KB of private L1 instruction and data caches, and a 512KB bank of shared L2. The NoC is a 2D mesh. The per-core BMem is as large as an L2 bank, but we power-gate unused 32KB chunks as directed by the application. We will present the used fraction of BMem in the next section. The wireless network has a data rate of 20 Gb/s, enough to transmit a BMem line and its address (about 80 bits) in 4 cycles (plus one cycle for collision detection). We do not consider missing packets due to noise, since the error rate is below 10^{-16}. We augment Multi2sim with an on-chip wireless network that accurately models transmissions, collision handling, transceiver power-gating, and packet dropping.

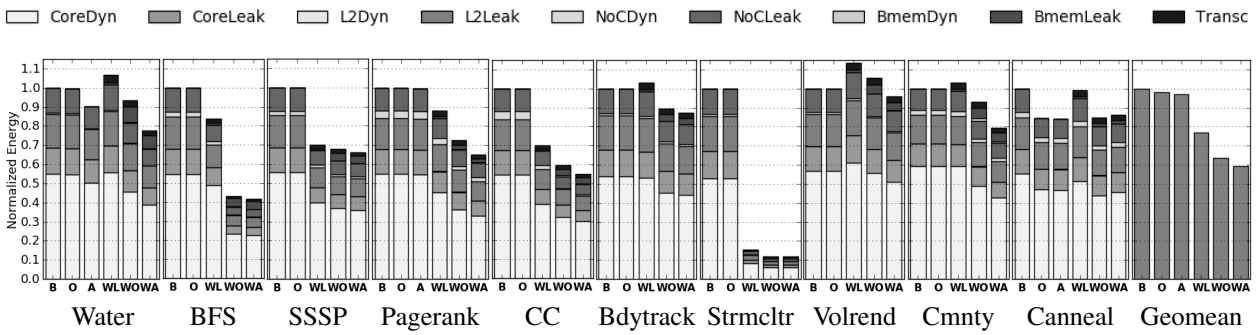


Figure 2.5: Energy consumed by the different configurations relative to Baseline (B) for 64 cores.

2.6 EVALUATION

As mentioned in Section 1.2.2, in my thesis I only claim the hardware aspects of *Replica*, therefore although the evaluation in the paper [46] also examines the performance of *Replica*, the impact of approximations on accuracy, the relationship between tuning parameters and the accuracy, and how applications are adapted for *Replica*, in this thesis I only show the effect of the adaptive wireless protocol (Section 2.6.1), and the energy consumption and area of *Replica* (Section 2.6.2).

2.6.1 Adaptive Wireless Protocol

In our experiments, we run the adaptive mechanism of Figure 2.2 for an initial section of the execution of the application, until the hardware can identify which of the two protocols is best for the application. Specifically, the execution of an application is logically divided into intervals of 10,000 cycles. At the beginning of each interval, the two counters discussed in Section 2.4.1 start at zero. Then, as execution proceeds, they get updated. At the end of the interval, based on their relative values and the values of the T_{BRS} and T_{token} thresholds (Table 2.2), the hardware decides what protocol to try in the next interval, and clears the counters. The process then starts again. After about 350 intervals on average, the hardware makes the decision to stick with the protocol that has been chosen in most of the intervals so far.

Of our applications running under WO, seven end-up sticking with the token ring protocol (BFS, SSSP, Pagerank, CC, Streamcluster, Volrend, and Community), and three with the BRS protocol (Water, Bodytrack, and Canneal). In most applications, the percentage of intervals when the dominant protocol was chosen is greater than 75%. The percentages are lower in BFS (60%), Pagerank (51%), and Volrend (51%).

To assess the performance impact of our adaptive wireless protocol, we measure how the execution time of WO would change if either all applications were using BRS or all were using token ring. Specifically, if the applications that prefer token used BRS, their individual execution times

would be higher by 28.1% (Community), by 27.1% (CC), by 21.7% (Pagerank), and by less than 2.2% (BFS, SSSP, Streamcluster, and Volrend). Conversely, if the applications that prefer BRS used token, their individual execution times would be higher by 8.6% (Bodytrack) and by less than 2.0% (Water and Canneal). With these results, we can conclude that, if all the applications used BRS, the average execution time under WO would increase by 8.4%; if all the applications used token ring, the average execution time would increase by 1.2%.

2.6.2 Analysis of Energy and Area

Energy Impact. Figure 2.5 shows the energy consumed by the different configurations relative to B for 64 cores. The energy is broken down into dynamic (*Dyn*) and leakage (*Leak*) energy for the core (including L1), L2, NoC, and BMem. We also show the contribution of the transceiver.

The results show that about half of the energy is dynamic energy in the core, and the rest is leakage energy in all the components. Across the bars, we see that the energy savings of each configuration over the Baseline (B) are broadly proportional to the performance improvements of the configuration over B. The wireless configurations reduce the cost of the polling operations and long distance communications. Therefore, they reduce the energy consumption. This effect is especially visible in Streamcluster.

The energy cost of the wireless communication itself can be estimated by adding the contributions of the BMem and the transceiver. We see that such contribution is modest. On average, it is 8.9% of the energy in WO and 8.6% of the energy in WA. The results for 32 cores show a similar behavior and are omitted to save space.

Summary of energy savings. Overall, the average energy savings of the exact version of Replica (WO) over the optimized baseline (O) are 34%. The average energy savings of the approximate Replica (WA) over the approximate baseline (A) are 38%.

Area Impact. Based on the numbers from Section 2.5.3 and Table 2.2, our tools estimate that the area overhead of supporting wireless communication is around 15.5% of the Replica architecture. This includes the contribution of the BMems (11.52%) and of the transceivers and antennas (3.97%). Note that these results are conservative, as we do not scale the area of the RF components from 65nm to 22nm (Section 2.5.3).

2.7 RELATED WORK

Wireless Architectures. We described WiSync [51] in Section 2.2. Duraisamy et al. [63] accelerate graph analytics using an NoC augmented with wireless links to better support irregular

communication patterns. In their case, the application is oblivious of the underlying architecture, and the routing mechanism of each node decides whether to use the wireless links or the regular wire lines, based on the destination address. Their work is also different from ours in that the wireless links are only used to unicast packets between distant cores, irrespective of their criticality, and just as a way to shorten the propagation time of the packets through the network. Later, Duraisamy et al. [64] propose to accelerate graph analytics by bypassing certain updates. Their approximation is exclusively software-based and reduces both the computation and the volume of data lookups, specific to a particular community detection graph algorithm. In contrast, Replica presents hardware-supported, general approximate store and approximate lock mechanisms, which we applied across multiple application domains.

Nanophotonics and Transmission Lines. Transmission of optical signals through nanophotonic waveguides [18, 59, 60, 85, 86] and transmission of radiofrequency signals through transmission lines (TLs) [87, 88, 89, 90, 91, 92] can provide broadcast. Compared to wireless networks, both nanophotonics and TLs are more energy efficient and provide higher bandwidth, because energy is guided rather than radiated. However, network design using either nanophotonics or TLs becomes more complex and less scalable than wireless. It is more complex because it requires a physical infrastructure that interconnects the nodes. Nanophotonics are less scalable due to laser power needs. Light is modulated by the transmitter and then guided to all the receivers. Each receiver extracts a fraction of the light, causing losses, and requiring high laser power for large destinations sets. TLs are less scalable due to: (1) the need to overcome signal reflections with amplifying stages between segments, which are costly and complicate the design, (2) the requirement of a centralized arbiter for the bus, (3) the fact that the analog logic in TLs cannot handle broadcast operations well, especially if the fan-out is large.

Scratchpads. While both BMem and scratchpads [93] have a finite size, BMems are automatically coherent. They do not rely on the compiler to keep them coherent, which is a major reason for the difficulty of using scratchpads. In Replica, the programmer and/or compiler just allocates the data in BMem and Replica transparently handles coherence in hardware.

Lossy NoCs. Prior work has proposed to apply lossy compression techniques to messages before sending them to the network [94]. The approximation occurs in the (wired) network interface, but could be potentially applied to wireless too. Although bufferless networks [6, 95] drop or deflect packets to undesired paths when there is contention at the switches, they are not approximate, since delivery is ensured through retransmissions.

Approximate Parallelization. Relaxed synchronization optimizations intentionally give up some synchronization for faster execution (e.g., [96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107]). The previous works mainly show the potential of many computations to successfully continue execution with relaxed synchronization and random errors on commodity hardware. Our work presents an approximate BMem architectural abstraction that is specialized for packet dropping. We show the efficiency of our hardware and software co-design and develop a toolchain to automate program adaptation.

2.8 CONCLUSION

This work presented *Replica*, a manycore that uses wireless communication for communication-intensive ordinary data. Replica supports two hardware mechanisms to reduce contention and latency in the wireless channel: an adaptive wireless protocol and the ability to selectively drop wireless packets if the sender encounters a certain level of contention.

Our results showed that *Replica* effectively uses wireless communication for ordinary data. For 64-core executions, *Replica* sped-up applications over a conventional machine by a geometric mean of 1.76x for exact computation and 1.89x for approximate computation (as shown in the *Replica* paper [46]). Further, *Replica* substantially reduced the average energy consumption by 34% (or 38% with approximate computation). Finally, the area increase is small, and the developer effort modest.

CHAPTER 3: *FUZZY-TOKEN*: AN ADAPTIVE MAC PROTOCOL FOR WIRELESS-ENABLED MANYCORE CMPS

3.1 INTRODUCTION

In recent years, processor manufacturers have steadily increased the number of cores integrated on a processor chip. Currently, Ampere's Altra processor [108] integrates up to 80 ARM cores, AMD's EPYC 7742 processor [109] supports up to 64 cores, and Intel's Xeon Platinum 9282 processor [110] hosts up to 56 cores. It is conceivable that the core count will continue to increase past one hundred.

To serve inter-core traffic, processor chips use Networks-on-Chip (NoCs) [7, 8]. Typically, manycore NoCs are packet-switched networks composed of routers and links arranged in a 2D grid. While this design is more scalable and efficient than buses, the latency and energy consumption of NoCs start to become a problem at these core counts [5, 6]. In particular, messages that need to be broadcasted or traverse a high number of routers before arriving to the intended destination are problematic [12]. The high latency may stall cores as they wait for a response or to synchronize, which throttles execution. Slowdowns of 2–3$\times$ have been estimated in processors with several tens of cores [12, 13] and worse effects are expected at higher core counts.

To improve the scalability of NoCs to high core counts, emerging interconnect technologies such as 3D [17], nanophotonics [20], and wireless [31, 111] have been proposed. Among them, wireless technology has shown promise due to its inherent broadcast nature and very low latency for chip-wide transmissions, which is lower than that of conventional NoCs by an order of magnitude [111]. Moreover, no wiring infrastructure is needed, which provides a flexibility that is unattainable with other technologies [26].

The wireless approach is enabled by recent advances in CMOS RF technology, which have allowed the integration of millimeter-wave antennas and transceivers on chip [30, 31]. Based on these, multiple Wireless Network-on-Chip (WNoC) designs have been proposed. Using simulations, these designs are shown to achieve substantial network-level improvements with respect to conventional NoCs [111, 112, 113], and help manycores attain substantial execution speedups and reduced energy consumption [26, 51]. These estimations have assumed wireless bandwidths of a few tens of GHz and low-order modulations, which are both feasible with current technology.

To fully realize the potential of WNoCs, however, it is necessary to build Medium Access Control (MAC) mechanisms that are able to cope with the heterogeneity and performance requirements of the multiprocessor scenario [114]. MAC protocols need to adapt to wide and fast changes in the on-chip traffic, while incurring little to no delay in the transmission. Unfortunately, existing efforts [26, 51, 112, 115, 116] are unable to capture such fast variations, resulting in execution time

slowdowns and energy waste.

In this work, we present *Fuzzy-Token*, a new MAC protocol capable of dynamically adapting to the traffic demands of the application, minimizing transmission latency and increasing network throughput. The *Fuzzy-Token* algorithm combines the strengths of token passing and contention-based MAC protocols [117] and, with a few simple rules, adapts to different types of workloads. The algorithm is evaluated using both a synthetic traffic model [118], and a set of real application traces. Our simulation-based evaluation shows that *Fuzzy-Token* consistently provides one of the lowest packet latencies of the evaluated WNoC MAC protocols. On average, the packet latency in *Fuzzy-Token* is 4.4× and 2.6× lower than in a state-of-the art contention-based WNoC MAC protocol and in a token-passing protocol, respectively.

3.2 BACKGROUND

Traditionally, wired NoC architectures use a packet-switched network with each processor connected to a router as shown in Fig. 1.1. Each router has several pipeline stages for enqueuing, computing the route, arbitration, and dequeuing packets [5]. Data packets move from source to destination via multiple hops, with each hop incurring delays and energy consumption. A mesh is a typical topology for manycores due to its simple layout and low inter-router link length [5]. However, the average hop count scales with $\sqrt{N}$, where N is the number of cores. Several works [12, 13, 111] have shown that execution suffers significant slowdowns as a result of high communication latency. In response to this, high-radix topologies have been proposed that reduce the network diameter, but at the cost of non-trivial router chip area and energy consumption.

Wireless technology can reduce the latency of a chip-wide communication to a few clock cycles regardless of the size of the chip or the number of cores. To that end, one antenna and transceiver are co-integrated in each core or cluster of cores, as shown in Fig. 1.1. Information coming from the cores is modulated and the resulting signals propagated through the chip package, bouncing off the metallic heat sink and reaching 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 [119]. These losses, on the other hand, prevent the enclosing package from acting as a reverberation chamber.

Antennas are either variants of planar integrated dipoles [30] or vertical monopoles implemented with vias that are drilled through the silicon die [119]. In both cases, the operating frequency is chosen within the mm-wave and sub-THz spectrum to minimize antenna size and avoid undesired near-field coupling. The link budget is generally determined by targeting a Bit Error Rate (BER) similar to that of on-chip wires, this is, below 10^{-12}. For the typical channel attenuation values seen

in the on-chip environment, it has been shown that simple modulations such as On-Off Keying (OOK) are able to minimize power consumption (toward 1 pJ/bit/cm) and chip area (toward 0.1 mm^2 per transceiver) while maintaining relatively high speeds (10+ Gb/s) [31]. This is because these modulations avoid power-hungry components such as Phase-Locked Loops (PLLs).

From the MAC design perspective, WNoCs are a very demanding scenario. Chip traffic is highly heterogeneous, meaning that the injection rate, burstiness (temporal injection distribution) and hotspotness (spatial injection distribution) of the network vary across and within applications [118]. These characteristics are generally detrimental to performance [114]. On the other hand, the WNoC scenario has some good traits. For instance, the number of nodes is fixed and known beforehand, and due to the enclosed nature of the on-chip scenario, all nodes are within the same transmission range. Therefore, the hidden/exposed terminal effects can be avoided and collisions can be detected [51].

3.3 THE FUZZY TOKEN PROTOCOL

Fig. 3.1 illustrates the basic operation of *Fuzzy-Token*. It has two operation modes: *focused* token mode and *fuzzy* token mode. During a focused period, only the token holder can transmit. If the token holder transmits, it is guaranteed to suffer no collision. If the token holder does not transmit, then the protocol switches to fuzzy mode. During a fuzzy period, the token holder can not transmit, but the nodes within what we call the *Fuzzy Area* can transmit. If only one of these nodes attempts to transmit, it succeeds. If more than one of these nodes attempts to transmit, a collision is detected using the NACK mechanism from [51]. In this case, the protocol switches to focused mode. By switching between the two modes, the protocol aims to take advantage of the capabilities of a fair and collision-free token passing protocol (*focused* mode), which works well for high, bursty, and all-core (i.e., uniformly distributed in space) traffic; and of a contention-based protocol (*fuzzy* mode) that performs better for low and few-core (i.e., hotspot) traffic.

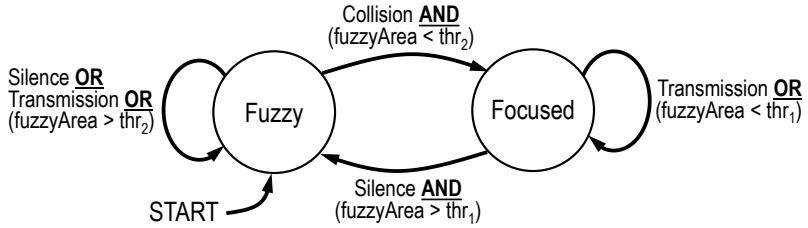


Figure 3.1: Basic state diagram of *Fuzzy-Token*.

The amount of contention during a fuzzy period is controlled with the fuzzy area. The fuzzy area includes the nodes around the token holder that may be able to transmit during the period.

The idea is to increase the fuzzy area when the load is low to give rapid access to the few nodes that want to transmit, and decrease it otherwise to minimize collisions. To this end, we always increase the fuzzy area when a silence is detected and decrease it when there is a collision. Note that a switch in mode involves a change in the size of the fuzzy area.

Fig. 3.2 (left side) shows the three areas of *Fuzzy-Token* operation, as a function of the fuzzy area size and the load. When the fuzzy area size is between two given thresholds thr_1 and thr_2, *Fuzzy-Token* follows the state diagram of Fig. 3.1 (ignore the conditions in parenthesis). We call this area *Normal* in the figure. However, in extreme cases, it is advisable to remain in one of the modes. Specifically, when, after many collisions, the fuzzy area is below thr_1, the network benefits from remaining in the focused mode. On the contrary, when, after many silences, the fuzzy area is over thr_2, the load is low enough for the network to remain in the fuzzy mode.

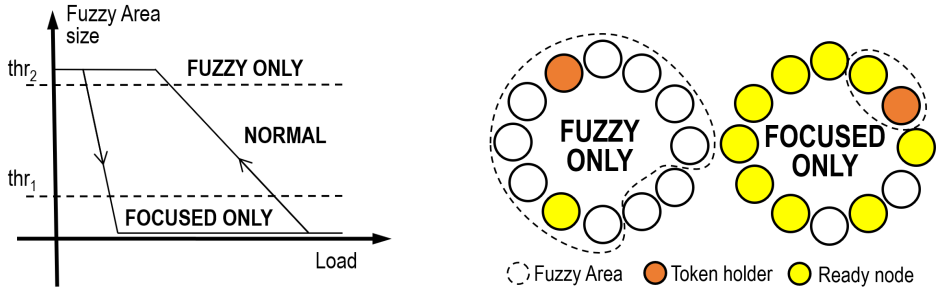


Figure 3.2: Transitions (left) and extreme cases (right) of *Fuzzy-Token*.

Fuzzy-Token ensures fairness by circulating the token around the virtual ring. In more detail, the token is passed implicitly at every event (silence, collision, or successful transmission) regardless of the protocol mode. This is important because multithreaded applications generally run as slow as the slowest of the threads. Thus, latency tails generated by unfair access will significantly slow down computation, even if they are infrequent. It is worth noting that, thanks to the unique characteristics of the on-chip scenario, all nodes have a consistent view of all events. This allows *Fuzzy-Token* to pass the token implicitly and to update the fuzzy area position and size without explicit messages or centralized control.

3.3.1 Algorithm

The algorithm is divided into steps. In each step, all the nodes in the chip are in the same mode (focused or fuzzy), and one node is the token holder. A step lasts for the duration of an event (silence, collision, or successful transmission). On termination of the step, the next node in the virtual ring becomes the token holder, the fuzzy area changes in position and maybe in size, and the mode may change.

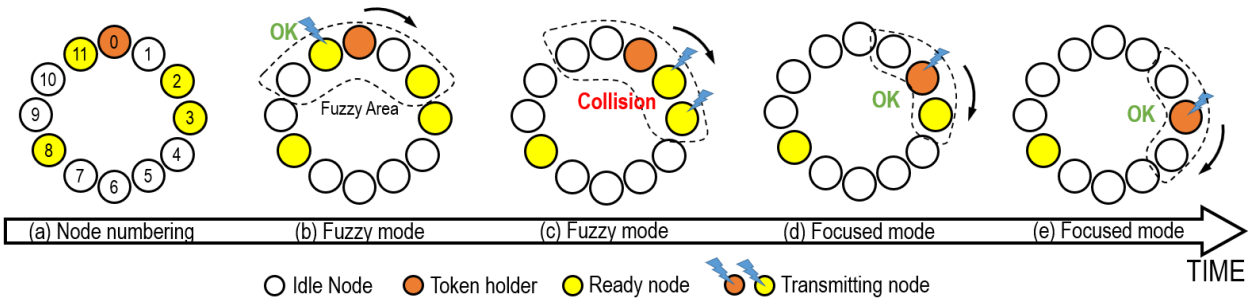


Figure 3.3: *Fuzzy-Token* protocol example. It assumes the protocol starts in a fuzzy mode step and its initial fuzzy area size is 5.

The algorithm is executed in each of the nodes of the chip. First, the node checks whether the mode is fuzzy or focused. To this end, there is a bit called *periodMode* that tells the mode of the step. Based on the mode, the operation of the node is as follows:

Fuzzy mode. In fuzzy mode, if the node is in the fuzzy area and has a ready packet, it considers transmitting it with a certain probability. The fuzzy area is encoded with set bits in a vector with as many entries as nodes in the chip. The vector also contains the transmission probability p_i for each node i. If the node decides to transmit, a variant of the BRS protocol [51] is used: the node first sends the preamble and then senses the channel. If the token holder detects a collision of preambles from multiple nodes, it sends a NACK. When a sender senses the NACK, it cancels the transmission; otherwise, it proceeds with the rest of the transfer. By using this mechanism, *Fuzzy-Token* reduces the time and energy penalty of collisions, as it avoids the unnecessary transmission of the whole message. The protocol differs slightly from that of [51]: here, it is the token holder the one that sends the NACK, saving energy — in BRS, all idle nodes detect the collision and send NACKs concurrently. For this to work, however, the token holder cannot try to send a packet.

After the packet is sent (C cycles), or the collision is detected (2 cycles), or no packet transmission is even attempted (1 cycle), the algorithm starts a new step. Since the packet size and wireless bitrate are known and static, all nodes are synchronized and proceed in lockstep.

Focused mode. The focused token mode is simpler, as it is based on a conventional token protocol. The token holder is stored in the variable *tokenID*, which is updated at the end of each step. In this mode, the node checks whether it is the token holder and whether it has a packet ready to transmit. If both conditions are true, the node transmits the packet; otherwise, it remains idle. After the C cycles of a successful transmission or 1 cycle of silence, a new step starts.

Housekeeping. At the end of each step, the variables *tokenID* and *fuzzy area* are updated, and *periodMode* may be updated. The *tokenID* is incremented by one modulo the number of nodes in the ring. The fuzzy area is rotated one position as well. In addition, the fuzzy area size is increased if a silence occurred and decreased if a collision occurred. If a successful transmission occurred,

we choose not to change the fuzzy area size, as it may be close to the optimal value. Finally, the *periodMode* bit may be updated according to the state machine rules shown in Fig. 3.1.

3.3.2 Design Decisions

Fuzzy-Token involves several design decisions, and includes a set of parameters that determine how aggressive the protocol is — e.g., how eagerly it changes the fuzzy area size. In this section, we discuss these design decisions. Default values are given in Sec. 3.4.

Probability of transmission. In fuzzy mode, the probability of transmitting may be a function of the fuzzy area size and the distance to the token holder.

Fuzzy area size update. *Fuzzy-Token* uses an additive-increase-multiplicative-decrease (AIMD) approach to update the fuzzy area size. This approach increases the fuzzy area size slowly when the load decreases and reduces it abruptly when collisions are detected. This is because collisions are more harmful than lost opportunities to transmit. Such AIMD approach has been shown to lead to high performance and stability in common protocols. In our scheme, we update the fuzzy area immediately after silence or collision events occur. This is in contrast to protocols that modify their behavior after collecting metrics during an observation epoch (see Sec. 3.6).

Activation thresholds. In the extremes, *Fuzzy-Token* becomes purely driven by the token passing protocol (focused mode) or the contention-based protocol (fuzzy mode). Fig. 3.2 (right side), shows an example of the two extremes. The figure assumes that thr_1 is 3 nodes and thr_2 is all the nodes in the chip minus 3. In this case, the chart where the fuzzy area includes all the nodes minus 2 is operating in fuzzy mode only. The chart where the fuzzy area includes only two nodes is operating in focused mode only. By setting the thr_1 and thr_2 thresholds, we affect how often the protocol stays in either extreme mode.

Token passing order. A statically-ordered virtual ring is just one of the possible ways of ordering the passing of the token. In any ordering, two hotspot nodes placed close together may lead to multiple collisions and multiple silences. To alleviate this, the token passing order can be changed at runtime, possibly in a pseudo-random way and at every collision. To ensure that all nodes agree on the same order without having to exchange messages, a synchronized random number generator seed can be assumed.

3.3.3 Walkthrough Example

Fig. 3.3 shows an example of *Fuzzy-Token*'s operation. Fig. 3.3(a) shows the node numbering, that Node 0 is the token holder, and that Nodes 2, 3, 8 and 11 want to transmit. We assume an initial fuzzy area of 5 and that the protocol is in a fuzzy mode step. Fig. 3.3(b) shows the operation

in the step. Nodes 3 and 8 are outside the fuzzy area, whereas Nodes 2 and 11 are in and could contend for the channel. Assume that only Node 11 proceeds to transmit. Because the transmission is successful, the next step will start with the same fuzzy area size and, following Fig. 3.1, in fuzzy mode. Fig. 3.3(c) shows the next step, where the token is owned by Node 1 and the fuzzy area has rotated. Both Nodes 2 and 3 are within the fuzzy area. Assume that both proceed to transmit and therefore collide. As a result, the fuzzy area size will decrease in the next step. We use an AIMD approach that sets $FA_{new} = \lceil FA_{old}/2 \rceil$, where FA is the fuzzy area size. Assume that the new fuzzy area size is less than thr_2. As a result, the next step will start with a fuzzy area size of 3 and, following Fig. 3.1, in focused mode. Fig. 3.3(d) shows the operation in the step. The token holder (Node 2) transmits successfully. As a result, both the fuzzy area size and the focused mode remain unchanged. Fig. 3.3(e) shows the operation of the next step, where Node 3 is the token holder. In this step, Node 3 transmits successfully.

3.4 SIMULATION ENVIRONMENT

We compare through simulations the average packet latency and throughput of three different protocols: BRS [51], token passing, and *Fuzzy-Token*. For a fair comparison, we optimize the token passing protocol with the same assumption as in *Fuzzy-Token*, namely, that all nodes have a consistent view of the wireless channel. Thus, the passing of the token is done implicitly, with zero delay.

Evaluations are carried out with the cycle-level Multi2sim [72] architecture simulator. We augmented Multi2sim with wireless on-chip communication modules that model collisions and MAC protocols. Multi2sim admits synthetic traffic and multithreaded applications. The architecture parameters are summarized in Table 3.1. A wireless transfer can be performed in four clock cycles: one for the preamble and three for the payload. BRS [51] and the fuzzy mode of *Fuzzy-Token* use one extra cycle to detect a potential NACK.

3.4.1 Traffic Patterns

Synthetic Traffic Model

Each of the cores acts as a generator that injects packets into the network. Typically, NoCs are evaluated with synthetic traffic models that have, as the main parameter, the injection rate λ in packets/cycle. Common models assume a Poisson process with the same average injection rate for all cores. However, in parallel applications, packet injections show a clear self-similarity caused

Table 3.1: Simulated architecture.

Architecture	
Processor chip	64 cores, x86 ISA, 1 GHz, 22-nm tech
L1 I+D	private, 32-KB, 2-way, 64B lines, 2 cycles
L2	shared, 512-KB/core, 8-way, 6 cycles
Coherence	MOESI directory + Replica [26]
Off-chip memory	4 controllers, 100 cycles delay
NoC	2D Mesh, 2 cycles/hop, 128-bit links
WNoC Parameters	
Network	64 nodes, 80-bit packets (preamble: 20 bits)
MAC protocols	BRS [51], Token, *Fuzzy-Token*
PHY layer	OOK, 20 Gb/s, power: 39 mW (TX), 39 mW (RX) [26]

by the data dependencies between threads of the application. Moreover, common memory patterns such as producer-consumer lead to some cores transmitting more often than others. Our traffic model takes these aspects into account.

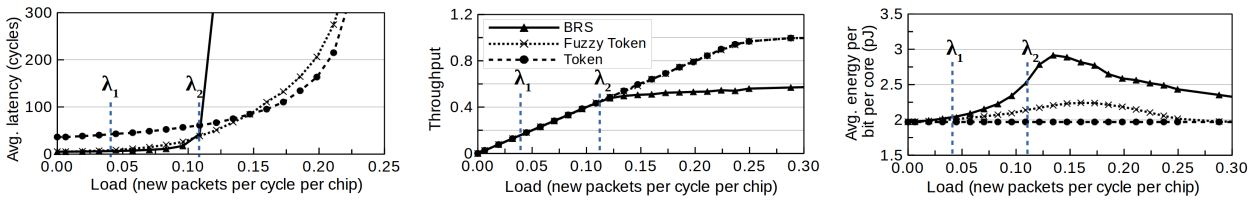
To account for the effect of self-similarity, we model a heavy-tailed distribution of traffic via a Pareto distribution [120]. The value of the Hurst exponent is $H \in [0.5, 1)$, which leads to increasing degrees of self-similarity when approaching 1 [118]. Moreover, to model an uneven injection of traffic across nodes, we make use of the hotspotness parameter σ proposed in [118]. There, it was demonstrated that the spatial injection distribution can be approximated as a Gaussian process where the value of σ represents the standard deviation. Low values of σ represent higher concentrations of traffic around a few selected nodes.

Real Applications

We model a wireless architecture in Multi2sim. This allows us to obtain the latency statistics of the three MAC protocols when executing multithreaded applications. Table 3.1 shows the details of the manycore architecture modeled. We model Replica [26] due to (1) its large speedups with respect conventional architectures, and (2) the relatively high load that it puts on the wireless network. The NoC and memory parameters are in line with the state of the art in manycore processor design. We run a selection of shared-memory multithreaded applications from the Splash2, Parsec, and Crono benchmark suites, which are suited to the characteristics of WNoCs.

3.4.2 Performance Metrics

The MAC protocols are evaluated on the basis of packet latency and throughput. Latency is defined as the time between the launching of a packet into the WNoC and its correct reception at all the intended destinations, measured in clock cycles. Throughput is measured in transmitted bits



(a) Latency (b) Throughput (c) Energy

Figure 3.4: Performance and energy comparison for different MAC protocols over increasing load.

per clock cycle. Another important metric is the energy consumed per transmitted bit, which may increase due to collisons. We calculate the energy per transmitted bit E_{bit} as

$$E_{bit} = E_{OK}\left(1 + \frac{L_{pre}}{L_{tx}}N_{re}\right), \quad E_{OK} = E_{TX} + (N-1)E_{RX}, \tag{3.1}$$

where L_{pre} and L_{tx} are the length of the preamble and of the complete packet, respectively, and N_{re} is the average number of re-transmissions per successfully transmitted packet. E_{OK} is the energy per bit of a non-colliding packet. It can be found as a function of the energy per bit of the transmitting and receiving part of the wireless transceiver (E_{TX} and E_{RX}, respectively), and the number of cores N. In turn, if P_{TX} and P_{RX} are the power consumed by the wireless transmitter and receiver, respectively, and R is the transmission speed, we compute $E_{TX} = P_{TX}/R$ and $E_{RX} = P_{RX}/R$.

3.5 EVALUATION

3.5.1 Evaluation with Synthetic Traffic

We start by comparing the protocols with synthetic traffic. By default, arrivals are distributed Poisson and the injection process is equidistributed across all cores. The default *Fuzzy-Token* configuration is: equal $1/k$ probability of transmission for all the k nodes inside the fuzzy area, increment the fuzzy area by 1 in each silence, decrement the fuzzy area to half (with ceiling) in each collision, $\text{thr}_1 = 0.1 \times N$, $\text{thr}_2 = 0.9 \times N$, and static token ring. In all cases, we repeat the simulations 10 times and obtain the geometric mean of all runs. Although we performed a sensitivity analysis for many different combinations of these parameters, including a Gaussian probability of transmission function, a wide range of fuzzy area increase/decrease factors, several pairs of thresholds, and a pseudo-random ring ordering, the choices shown in this section were found to be optimal. Due to space constraints, we do not show the sensitivity analyses.

Fig. 3.4 shows the packet latency (a), packet throughput (b), and energy per bit and core (c)

for the different MAC protocols as the load increases. Fig. 3.4(a) shows that *Fuzzy-Token* is able to deliver the low latency of BRS at low loads and almost match the latency of Token at high loads. At intermediate loads, *Fuzzy-Token* can outperform both BRS and Token thanks to its intelligent management of contention. In Fig. 3.4(b), we see that *Fuzzy-Token* achieves the same throughput as Token, leaving BRS behind. In fact, at very high loads, *Fuzzy-Token* ends up converging to Token by design. We finally note that, as Fig. 3.4(c) illustrates, *Fuzzy-Token* achieves high performance with only a moderate energy overhead over Token (less than 12%). The overhead is mainly due to collisions at intermediate loads.

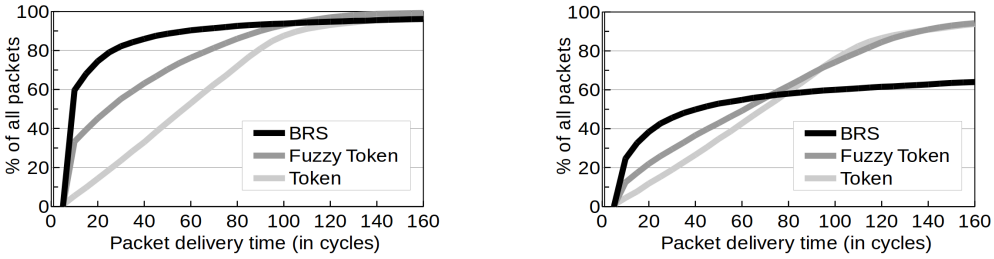
Energy-wise, it is also worth noting that the energy consumption of Token is not affected by the load, since the token is passed implicitly, and thus does not consume energy. This, however, comes at the cost of high latency at low loads. In contrast, the energy of BRS increases with the load because of increasing collisions, but then decreases. This effect is due to the finite population of the chip scenario: at very high loads, the backoff reaches high values and reduces the probability of collisions at the expense of unacceptable latency. *Fuzzy-Token* attains the low-load latency of BRS while avoiding its high energy consumption at higher loads.

Fig. 3.5 shows the latency distribution of the three protocols for two different loads: a moderate one (λ_1=0.045 packets per cycle per chip) and an intermediate one (λ_2=0.110 packets per cycle per chip). These loads are also shown in Fig. 3.4.

At λ_1=0.045 (Fig. 3.5(a)), most transmissions take less than 30 cycles with BRS, less than 60 with *Fuzzy-Token*, and less than 90 with Token. However, due to collisions, 1.29% of the packets with BRS take more than 500 cycles, with a worst-case of ~3400 cycles. On the other hand, *Fuzzy-Token* has a worst-case of ~330 cycles, which is the best among the three protocols.

At λ_1=0.110 (Fig. 3.5(b)), BRS still delivers many packets within the first 60 cycles. However, due to the high number of collisions, 28.9% of the packets take more than 500 cycles to be delivered, with a worst-case of ~110,000 cycles. On the other hand, *Fuzzy-Token* also delivers many packets within the first 60 cycles, while providing a worst-case latency of ~390 cycles (again the best among the three protocols). Overall, these plots show the difference between BRS and *Fuzzy-Token*. At moderate loads, BRS does well but, at intermediate loads, BRS generates a long tail. In contrast, *Fuzzy-Token* approaches the best of BRS and Token at all loads.

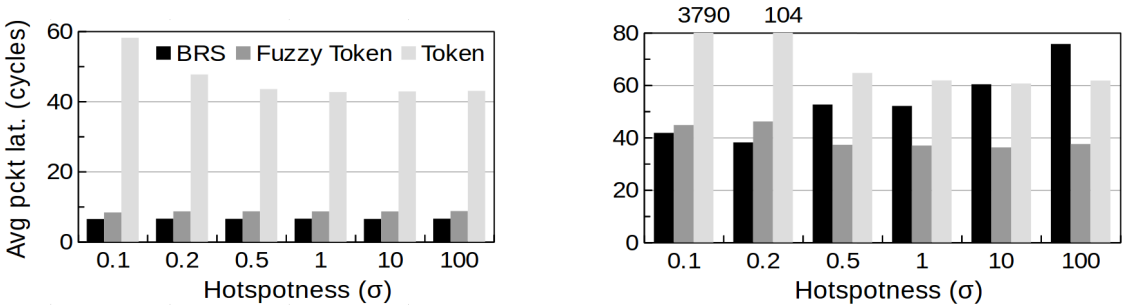
Hotspot Traffic. In hotspot workloads, a few processors inject most of the traffic. In this situation, it has been shown that contention-based protocols such as BRS outperform more rigid collision-free alternatives such as Token [115]. To confirm the hypothesis that *Fuzzy-Token* approaches the best of the two types of protocols, we increase the spatial concentration of the traffic via the σ parameter mentioned in Section 3.4.1 (i.e., low σ means more hotspot traffic). The inter-arrival time is kept exponential.



(a) Moderate load, $\lambda_1 = 0.045$. Tail: 1.29% in BRS, 0% in others.

(b) Intermediate load, $\lambda_2 = 0.110$. Tail: 28.9% in BRS, 0% in others.

Figure 3.5: Latency cumulative distribution function for the protocols at different loads. Tail is defined as delivery time over 500 cycles.



(a) Moderate load, $\lambda_1 = 0.045$

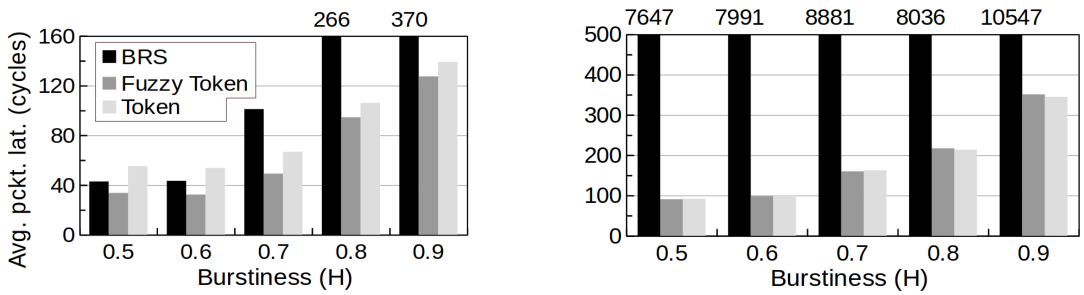
(b) Intermediate load, $\lambda_2 = 0.110$

Figure 3.6: Latency for hotspot traffic with different σ values. Lower σ means that fewer nodes inject most of the traffic.

Figs. 3.6(a) and 3.6(b) show the latency for hotspot traffic with different σ values, at $\lambda_1 = 0.045$ and $\lambda_2 = 0.110$ packets per cycle per chip, respectively. With $\lambda_1 = 0.045$, the load is moderate and contention is low. We show that *Fuzzy-Token* is just a couple of cycles slower than BRS, which maintains a very low latency regardless of the value of σ. This is because *Fuzzy-Token* has a large fuzzy area and, therefore, nodes can transmit as soon as they generate the packets, irrespective of their location. Token has a high latency, which worsens for low σ because the few transmitting nodes have to wait for their turn to issue a packet.

With $\lambda_1 = 0.110$ (Fig. 3.6(b)), contention starts to become important. In this case, BRS still benefits from a low number of contenders at small σ, as traffic becomes more concentrated. Token performs poorly in such a situation because many clock cycles are wasted passing the token between a few hotspot nodes. *Fuzzy-Token* is capable of maintaining a low latency across all situations, outperforming the two other protocols in nearly all cases. Similar tendencies are observed for loads beyond λ_2, but are not shown for brevity.

Bursty Traffic. We repeat the same set of experiments now changing the temporal distribution of traffic, assuming $\sigma = 100$. Burstiness is modeled via the Hurst exponent H mentioned in Sec.



(a) Moderate load, $\lambda_1 = 0.045$

(b) Intermediate load, $\lambda_2 = 0.110$

Figure 3.7: Latency for different H values. High H means longer bursts.

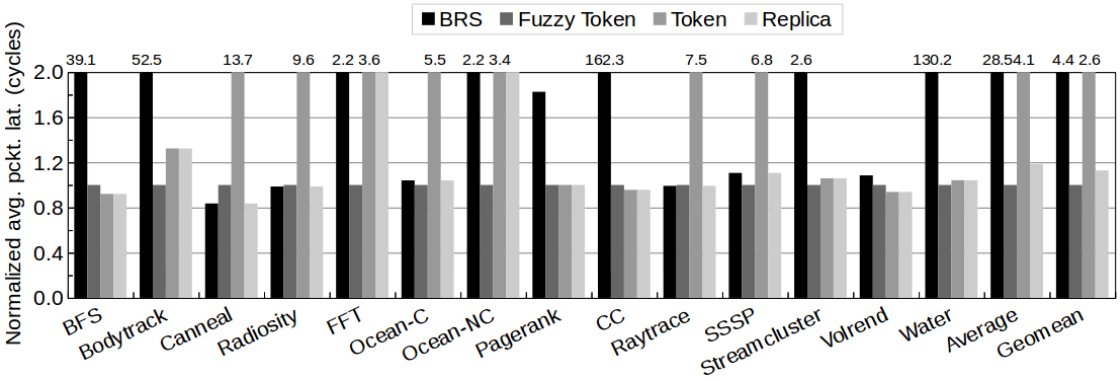


Figure 3.8: Average packet latency normalized to *Fuzzy-Token*'s latency for real applications.

3.4.1, with higher H values leading to longer bursts and longer intervals between bursts.

Figs. 3.7(a) and 3.7(b) show the packet latency for different H values, at $\lambda_1 = 0.045$ and $\lambda_2 = 0.110$ packets per cycle per chip, respectively. We observe that burstiness is detrimental for most mechanisms, especially for contention-based protocols like BRS. This is already patent at low loads: bursty injections create collisions. The latency of Token also increases, but its collision-free nature absorbs the bursts better. *Fuzzy-Token* is capable of achieving the best performance at all burstiness levels. This is because the first collisions occurring at the burst onset make *Fuzzy-Token* to become pure token passing. As the burst is being served, the fuzzy area grows gradually and allows the last nodes of the burst to access the channel earlier. Fig. 3.7(b) serves to confirm that increasing the load leads to early saturation, especially for BRS. *Fuzzy-Token* avoids contention and converges to Token to better absorb the intense bursty traffic.

3.5.2 Evaluation with Real Applications

Fig. 3.8 shows the average packet latency for different MAC protocols normalized to that of *Fuzzy-Token* in real applications. This figure showcases the strengths of *Fuzzy-Token*, which con-

sistently provides a latency that is among the lowest of all evaluated protocols. On average, *Fuzzy-Token*'s latency is 4.4× lower than BRS', 2.6× lower than Token's, and 1.13× lower than Replica.

Applications such as *canneal* and *radiosity* have a relatively low load and, therefore, Token performs poorly. Other applications such as *water* and *CC* are communication-intensive and inherently bursty. As a result, BRS suffers an average latency that is two orders of magnitude higher than that of *Fuzzy-Token*. In cases such as *bodytrack* and *ocean-nc*, *Fuzzy-Token* outperforms BRS, Token, and Replica, since traffic alternates between different types of patterns where *Fuzzy-Token* is consistently better.

3.6 RELATED WORK

Multiplexing. First MAC protocols proposed for the WNoC paradigm used time, frequency, or code orthogonal channels [36, 112]. This approach, however, does not scale well beyond a few cores due to the hardware overhead. Spatial multiplexing has also been proposed in [113], but the use of directional antennas eliminates the broadcast advantage of WNoC, and its applicability is unclear in packaged environments with lots of multipath.

Token passing. Different variants of token passing have been examined as alternatives to channelization [111, 116]. These solutions work well for distributed, high loads, but not for heterogeneous or hotspot traffic. Also, the protocol is not adaptive and does not scale. Mansoor *et al.* attempt to minimize these issues by means of a predictive scheme that estimates the optimal token occupancy of each node [116]. The work of [121] proposes token-like distributed arbitration protocol with single-bit concurrent probing. However, the probing introduces unfeasible bit-level synchronization requirements. *Fuzzy-Token* does not need such complex hardware while still leveraging the advantages of token passing at high loads. On top of that, *Fuzzy-Token* is flexible and scalable.

Random access protocols. Contention-based protocols have been explored for WNoC [115, 122] due to their low latency at low loads. BRS-MAC [122] minimizes latency also under moderate loads by using preamble transmission, collision detection, or collision notification via negative ACKs. At high loads, however, the protocol saturates early due to the many collisions. *Fuzzy-Token* borrows concepts from BRS-MAC [122], but builds a hybrid protocol that reacts before collisions become too penalizing.

Hybrid approaches. In [26, 115], token/contention hybrid protocols are proposed that attempt to leverage the benefits of both approaches. In [123], utilization metrics are gathered and used to

switch between token or random access modes, whereas [26] decides which protocol to use based on the load observed during the first thousands execution cycles of an application. This can have an negative impact in bursty and fast-changing traffic. Instead of working as two discrete protocols connected by a controller, *Fuzzy-Token* represents a continuous solution that naturally and rapidly adapts to the load without having to collect any utilization statistics. As shown in Sec. 3.5, this provides finer-grained control and better performance at for all kinds of workloads.

Hybrids in other scenarios. Traditionally, research on LAN networks has also tried to combine fixed and random access. In [124], the number of nodes is divided into fixed-size groups connected via a virtualtoken-passing ring. Groups contend for the channel with CSMA/CD, but the token-holding group has priority. Although this is similar to *Fuzzy-Token*, the fixed size of the groups and the requirement for ACKs after every transmission discourages its use on multiprocessors. Another hybrid protocol is given in [125], wherein the token holder has priority after the channel changes from busy to idle in any contention period. In the rest of cases, however, all nodes can contend for the channel, which leads to early saturation. We addressed this problem in *Fuzzy-Token* by the use of dynamic contention windows (i.e. the fuzzy area). Ehpremides *et al.* [126] propose a probabilistic TDMA scheme where nodes transmit to a preferred slot with a given probability a or elsewhere within the frame otherwise. The idea of the two extremes is laid out, but the decision on the parameter a is not discussed. *Fuzzy-Token*, instead, naturally and gradually adapts to the load.

3.7 CONCLUSIONS

This work has presented *Fuzzy-Token*, a MAC protocol uniquely tailored to the particularities of the WNoC scenario. We have shown that with a set of simple rules, *Fuzzy-Token* can achieve the low latency of contention-based protocols at low loads and the high throughput of fair collision-free protocols such as token passing. We have evaluated our protocol in a variety of synthetic traffic patterns, as well as with real application traces, demonstrating a several-fold speedup with respect to other state-of-the-art protocols. Our contribution, together with other advances in the field of on-chip networking, pave the way for scalable and efficient manycore processors for general-purpose computing, machine learning acceleration, and data center as well as high-end server processors.

CHAPTER 4: *WIDIR*: A WIRELESS-ENABLED DIRECTORY CACHE COHERENCE PROTOCOL

4.1 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 [108] 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 [109] 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 [110] 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 [127]. 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 [128, 129, 130], 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 [131] 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 [34, 35, 36, 37, 38, 39, 40, 41]. 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 [31]. Further, WNoCs naturally support update multicasts and broadcasts. As a result, WNoCs have been proposed to speed-up applications on manycores. For example, WiSync [45] uses it to speed-up synchronization, while Choi *et al.* [132] use it to accelerate CNN training in CPU-GPU heterogeneous architectures, and Replica [46] 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 work, 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 work, 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 work 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 work 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*.

4.2 BACKGROUND & MOTIVATION

4.2.1 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 1.1. 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 [5]. 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 [14], but at the expense of area and energy cost at the routers.

In contrast, WNoC architectures [34, 35, 36, 37, 38, 39, 40, 41] 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 1.1. Small antennas in the mmWave bands and beyond [30, 42, 43, 44] can be feasibly integrated, and broadcast messages [45, 46, 47]. Figure 1.1 shows vertical monopole antennas based on Through-Silicon Vias (TSVs), which perforate the bulk silicon [44]. 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 [48, 49, 50]. However, these losses are tolerable [31], and prevent the enclosed package from acting as a reverberation chamber.

4.2.2 Uses of Wireless Network On Chip

There are multiple proposals to use WNoCs to enhance the architecture of manycores [45, 46, 47, 132]. In general, these works leverage the low latency and affordable broadcast of a WNoC to accelerate key patterns. Mondal *et al.* [47] and WiSync [45] 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.* [132] 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 [46] 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 work, 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.

4.2.3 Scalable Cache Coherence Protocols

Scalable cache coherence is attained through the use of directories [128, 129, 130, 133, 134, 135]. Directory cache coherence is widely used commercially, such as in Intel's Core i7 systems. Commercial systems use invalidation-based schemes [128, 129, 130] rather than update ones [131]. 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 [136, 137].

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 [129, 130], 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 [138]. 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 4.5, 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.

4.3 DESIGN OF *WIDIR*

4.3.1 Main Idea

State-of-the-art directory protocols for large core counts use invalidation-based limited pointer (or coarse-vector) schemes [129, 130, 139]. 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 work, 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 4.1. 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.

Following Abadal et al. [45], 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

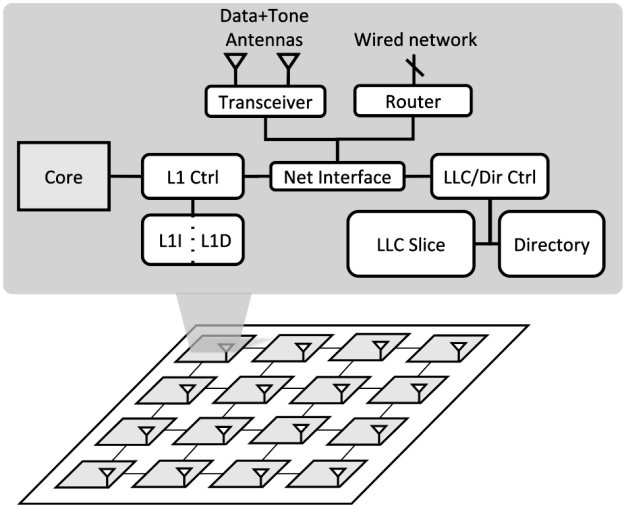


Figure 4.1: Manycore that supports the *WiDir* protocol.

at 90 GHz and is used as a special-purpose *acknowledgment* channel. The data channel uses the BRS wireless protocol [140]. 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.

4.3.2 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) [129, 130]. However, many other implementations, such as a Coarse Vector design (Dir_iCV_r) [130] 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 4.2 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.

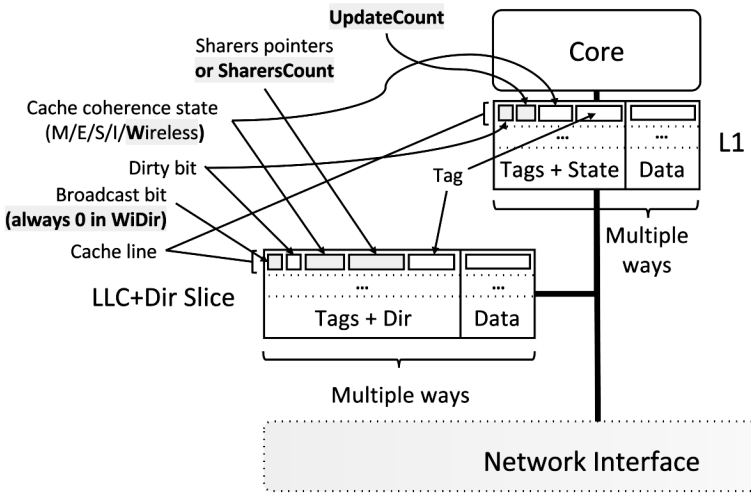


Figure 4.2: Cache and directory structure of a conventional protocol augmented (in bold) for *WiDir*.

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 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 4.3.3.

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*.

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.

4.3.3 Primitives for Wireless Protocols

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

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 [140]. As indicated in Section 4.3.1, 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.

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 4.3.2 to perform a global transition from Shared to Wireless state. A similar support has been proposed by TLSync [141] and WiSync [45] 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.

4.3.4 Summary: Why Adding Wireless Support To Coherence

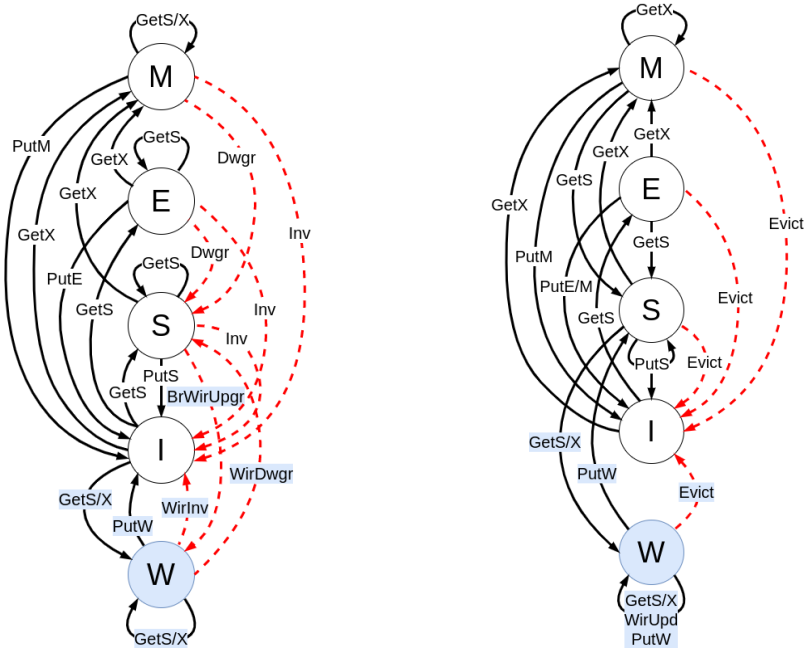
Adding support for wireless communication in a large manycore provides the ability to perform several types of coherence transactions very efficiently, especially those involving multicasting. While the bandwidth of a wireless network is limited, the latency of any given transaction is very small. These properties perfectly fit the sharing pattern considered in this work: 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 [45], and reads are local.

4.4 DETAILED DESIGN

4.4.1 Protocol Description

Figures 4.3a and 4.3b show the diagram of all possible transitions between stable states of *WiDir* in the controller of the private caches, 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) L1 cache controller.

(b) Directory controller.

Figure 4.3: Transitions between stable states of *WiDir* in the controllers of the private caches (a), and directories (b). In the figures, black lines are core-initiated transactions; red dashed 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 4.1 and 4.2 describe such transitions for the controllers of the private caches and directories, respectively. Each row considers one transition and describes when the transition happens and the action taken.

Table 4.1: State transitions that come from W or go to W for the controller of the private cache.

Trans.	When?	Action
$I \rightarrow W$	Core 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
	Core issues GetX to directory (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
	Core issues GetS to directory triggering a directory transition to W	Transceiver receives a *BrWirUpgr* via wireless, and sets the tone channel on. When the cache receives a *WirUpgr*+line via wired, it transitions to W and notifies the transceiver to turn off the tone channel
	Core issues GetX to directory (indicating that it is not a sharer) triggering a directory transition to W	Transceiver receives a *BrWirUpgr* via wireless, and sets the tone channel on. 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
	Core issues GetX to directory (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	Clear *UpdateCount*
	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
$W \rightarrow S$	Cache 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
$W \rightarrow I$	Local cache evicts W line	Cache notifies the directory with *PutW* via wired
	Cache 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

Private Cache Controller (Table 4.1). There are four cases of I→W transitions. The first two are when the directory is in W and the core issues a GetS or GetX (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

Table 4.2: State transitions that come from W or go to W for the directory controller.

Trans.	When?	Action
$S \rightarrow W$	Directory receives a GetS or GetX from a non-sharer cache via wired, and new number of sharers is higher than *MaxWiredSharers*	Directory broadcasts *BrWirUpgr* via wireless, turns on jamming on the line, and sends *WirUpgr*+line to the requester via wired. Once the tone channel is silent, directory sets the Wireless bit, sets *SharerCount*, sets state to W, and turns off jamming
$W \rightarrow W$	Directory receives from a cache via wired a GetS or GetX (indicating that is not a sharer)	Directory turns on jamming, and sends *WirUpgr*+line via wired to the requester. Upon receiving its *WirUpgrAck* via wired, increase *SharerCount* and turns off jamming
	Directory receives a GetX from a cache via wired (indicating that it already is sharer) not knowing the directory is in W	Directory discards message since a previously sent *BrWirUpgr* via wireless already resolved the conflict.
$W \rightarrow S$	Directory receives a *PutW* via wired. The *SharerCount* is decreased and now becomes *MaxWiredSharers*	Directory broadcasts *WirDwgr* wirelessly and waits for *WirDwgrAck* acknowledgments via wired. Upon receiving all *MaxWiredSharers WirDwgrAck*s, the directory records their core IDs in the sharer pointers, resets the Wireless bit, 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	Directory broadcasts a *WirInv* via wireless. If the line is dirty in the LLC, the line is written to memory

to W. If the request was a GetX, the cache now issues the update wirelessly.

The other two cases are when the core issues a GetS or GetX (again, 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. 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 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 core issues a GetX to the directory (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 W→S transition. It is when the cache 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 local 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 cache receives a wireless invalidate message (*WirInv*) for 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.

Directory Controller (Table 4.2). The transition S→W occurs when the directory receives a GetS or GetX from a non-sharer node via the wired network, and the new number of sharers for the line is now higher than MaxWiredSharers. In this case, the directory broadcasts a *BrWirUpgr* via the wireless network, turns on jamming in the wireless data network for the line and, via the wired network, sends *WirUpgr* plus the line to the requester node. Once the directory detects that the tone channel is silent, it sets the Wireless bit, sets *SharerCount* to the count of sharers, sets the state to W, and turns off jamming.

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

The second case is when the directory receives a GetX from a cache via the wired network indicating that the node is already a sharer, but not knowing that the directory is already in W. In this case, the directory discards the message since a previously-sent *BrWirUpgr* via wireless already resolved the conflict.

The W→S transition occurs when the directory receives a *PutW* message via the wired network, as a result of a wireless sharer evicting the line from its cache. After the directory decrements *SharerCount*, the latter becomes *MaxWiredSharers*. In this case, the directory broadcasts *WirDwgr* wirelessly, and waits for *WirDwgrAck* acknowledgments from *MaxWiredSharers* cores via the

wired network. Upon receiving them, the directory records their core IDs in the sharer pointers, resets the Wireless bit, 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 directory broadcasts a *WirInv* message via the wireless network. If the Dirty bit in the LLC is set, the line is written to memory.

4.4.2 Correct Operation of the Wireless Update 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. The first one is for the transceiver to obtain access to the wireless data network. The transceiver then has to succeed in sending the message without collisions. Once the message is fully transmitted (i.e., 5 cycles from the initiation), the transceiver signals the private cache controller to merge the write into the local cache. Once the merging is done, the write is complete.

Before the transceiver gains access to the wireless network, the transceiver may receive 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 an invalidation for the line (*WirInv*) because the line's entry in the directory was evicted. In this case, the local cache line is invalidated and the local write is squashed and retried. As the local write retries, it will trigger the directory to allocate a new entry for the line.

The fact that the update has been successfully transmitted via the wireless data network does not imply that the update has already updated the home directory or the remote caches. This is because, in each destination node, the update has to be routed to the controllers for the private cache and (in the home node) to the directory/LLC controller (Figure 4.1). Such routing takes some delay. For the coherence protocol to be correct, we need to ensure that the update is not overtaken by requests that are issued later. To ensure that this does not happen, *WiDir* has a single queue for the private cache controller, and a single queue for the directory/LLC controller. Further, messages are processed in strict FIFO order. Specifically, messages to the private cache controller from the core, transceiver, router, and directory/LLC in Figure 4.1 all go to a single FIFO queue. The same applies to the directory/LLC controller. Hence, as an incoming update is received by the transceiver, it gets ordered in the queue where it goes to.

Table 4.3: 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 [140] (exponential backoff)
Transceiver	At 65nm: 0.25mm^2, 30mW [31, 77, 78]
Data converter	At 65nm: 0.03mm^2, 0.72mW [79]
Serializer/Deserializer	At 65nm: 0.04mm^2, 10.8mW [80]
Data+tone antennas	At 65nm: 0.08mm^2 [81]
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)

4.5 EVALUATION METHODOLOGY

We use the SST [142] 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 4.3. 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 [74] and CACTI [75] to model the energy consumed by cores and memory hierarchy, as well as a calibrated DSENT [76] 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 [31, 77, 78, 79, 80, 81, 143]. 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 [83, 84].

However, in this work we choose to be conservative and not scale them down. Finally, the energy parameters of the wireless components in Table 4.3 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 [31, 82].

Our evaluation assumes a highly reliable wireless technology. In the past, wireless on-chip links encountered sizable error rates [21, 144, 145]. However, improvements in recent years (e.g., [119]) 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 [146] and PARSEC [71]. These applications are listed and characterized by their L1 misses-per-kilo-instruction (MPKI) for *Baseline* in Table 4.4. They have different levels of fine-grained data sharing, as well as different access patterns.

Table 4.4: Evaluated applications characterized by L1 MPKI in *Baseline*.

SPLASH-3 [146]				**PARSEC** [71]	
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 [146] and, for PARSEC, we use the standard *simsmall* [71]. 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.

4.6 RESULTS

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

4.6.1 Data Sharing Analysis

Figure 4.4 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.

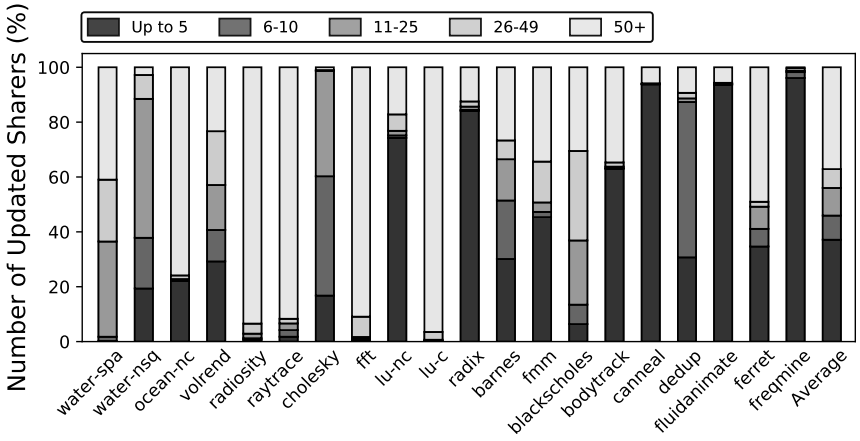


Figure 4.4: Number of sharers updated upon a wireless write in *WiDir*.

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.

4.6.2 Performance Analysis

Cache Misses. Figure 4.5 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.

The reduction in MPKI is especially large in *radiosity*. For this application, Figure 4.4 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 4.4). However, the two figures are not perfectly correlated because there are also misses to non-wireless lines.

Memory Latency. Figure 4.6 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

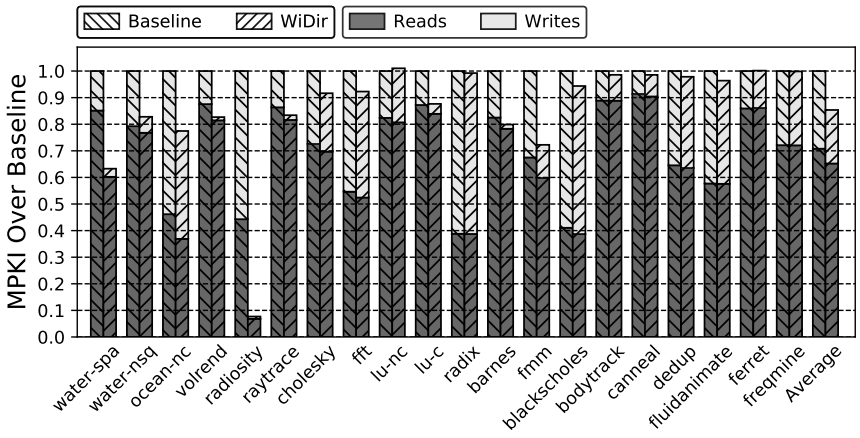


Figure 4.5: Misses-per-kilo-instruction in *WiDir* and *Baseline*, normalized to *Baseline*.

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.

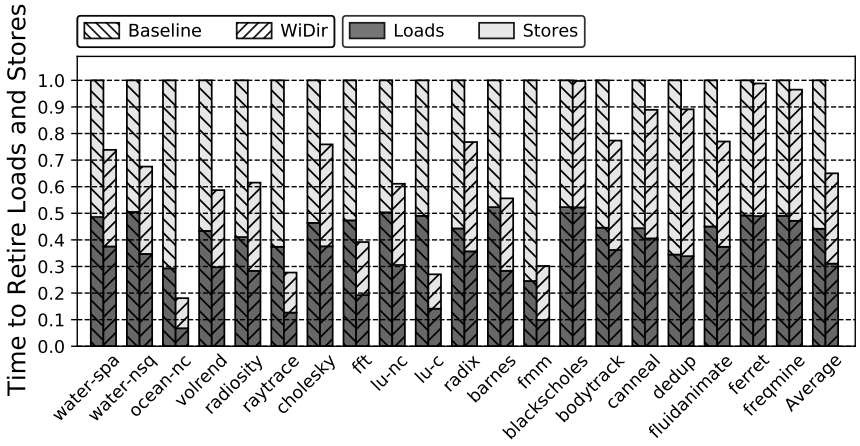


Figure 4.6: Overall latency of memory operations in *WiDir* and *Baseline*, normalized to *Baseline*.

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 4.5 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 4.5: Distribution of number of network hops per leg for messages sent through the wired mesh, in the 64-core *Baseline* architecture.

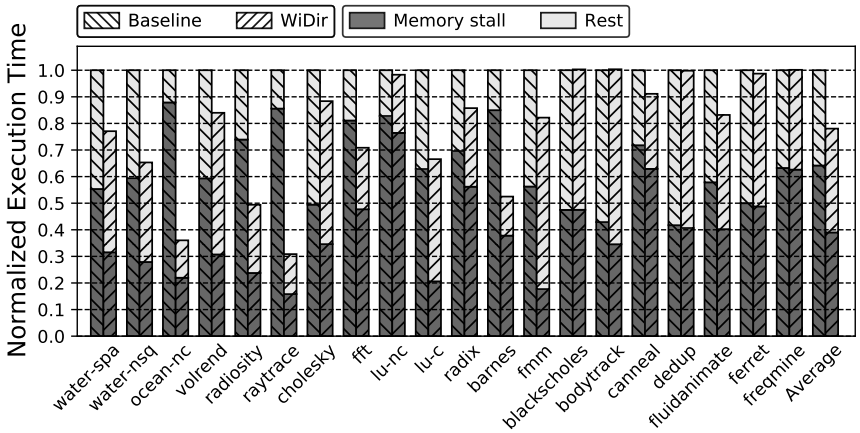
Number of Hops per Leg	***0-2***	***3-5***	***6-8***	***9-11***	***12-16***
% of Messages	17%	22%	31%	21%	9%

Execution Time. Figure 4.7 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*).

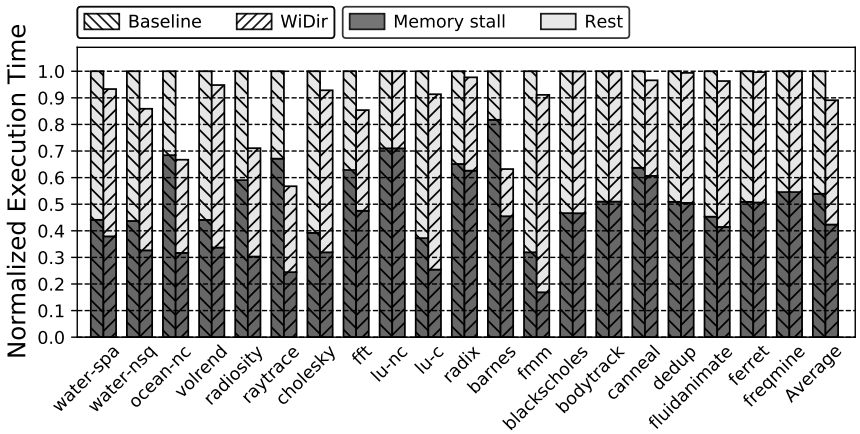
Consider first the 64-core runs (Figure 4.7a). We can see that, in *Baseline*, many of the cycles are wasted to memory stall. On average, nearly 65% of the cycles fall into this category. With *WiDir*, many applications reduce their execution time, sometimes substantially. Such reductions are caused 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*.

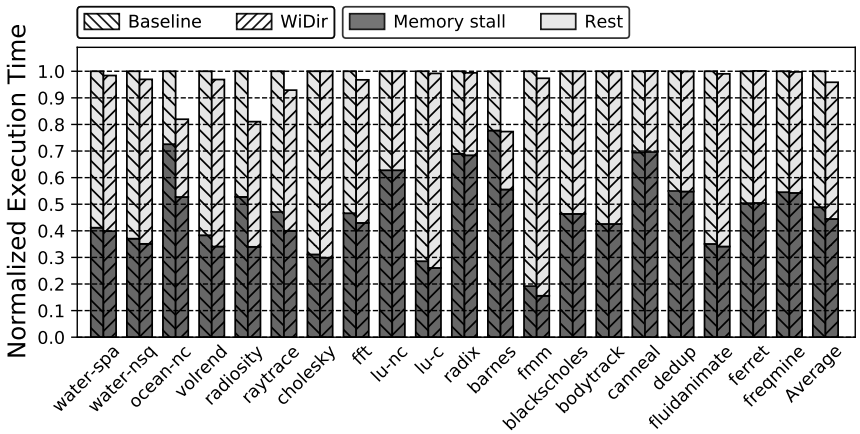
As we decrease the number of cores in the architecture, the speed-ups decrease. Specifically, for 32 cores (Figure 4.7b) and 16 cores (Figure 4.7c), the average execution time reduction of the applications is 11% and 4%, respectively. With fewer cores, the latency of the wired network is smaller and, in addition, variables are shared by fewer cores.



(a) 64 cores



(b) 32 cores



(c) 16 cores

Figure 4.7: 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.

4.6.3 Energy and Area Analysis

Energy. Figure 4.8 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.

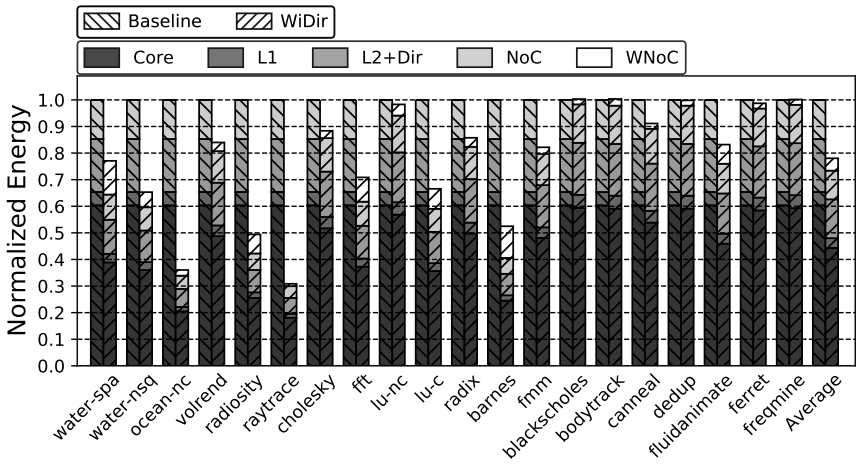


Figure 4.8: Energy consumed by *WiDir* and *Baseline*, normalized to *Baseline*.

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 4.7a). 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 4.3 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 [83, 84] 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.

4.6.4 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 coher-

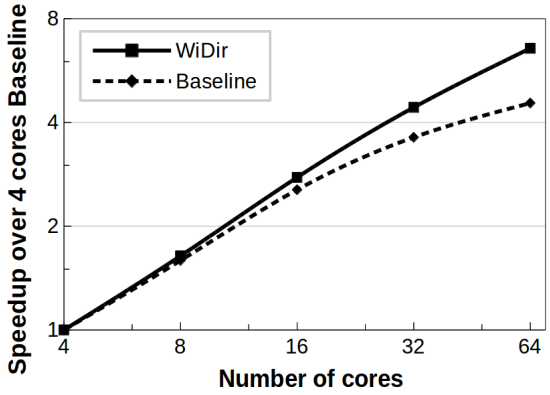


Figure 4.9: Average execution time speedup of *WiDir* and *Baseline* over the 4-core *Baseline*.

ence 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 4.9 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.

Table 4.6: Speedups of *WiDir* over *Baseline*, and collision probability in *WiDir*, for different values of *MaxWiredSharers*.

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%*

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 4.6 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.

4.7 RELATED WORK

Wireless Architectures. We described WiSync [45], Choi *et al.* [132], Mondal *et al.* [47], and Replica [46] in Section 4.2. Other works use a WNoC to accelerate the communication patterns of applications such as graph analytics [63], molecular dynamics simulations [147], and brain-machine interfaces [148]. 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 [115, 121, 149, 150]. *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 [127, 136] also use a totally-ordered interconnect. Several works extend snooping coherence to unordered interconnects to improve its scalability [151, 152, 153, 154, 155, 156, 157]. The idea is to provide ordering guarantees, either at the protocol level via tokens associated to each cache line [152] or at the network level using global timestamps [151], snoop-order numbers [154], a logical embedded-ring [153], or a dedicated ordering network [155, 156, 157]. 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 [18, 59, 158, 159, 160] or transmission lines (TLs) [141, 161, 162, 163, 164, 165]

can provide ordered broadcast. Some proposals use these global interconnects to implement conventional snooping coherence [59], custom limited directory protocols [159], race-free protocols via globally shared locks [158], or to speed-up synchronization [141, 166]. 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 [129, 130, 135] in Section 4.2. Another way to scale the directory is by dividing the manycore into coherence domains [167], 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.

4.8 CONCLUSION

To handle sharing patterns where a group of cores frequently reads and writes a set of shared variables, this work 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 work, 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.

CHAPTER 5: *WIPACKAGE*: WIRELESS-ENABLED COLLECTIVE COMMUNICATION IN A MANY-CHIPLET PACKAGE

5.1 INTRODUCTION

Over the last few years, we have observed the slowdown in the rate of transition from one technology node to the next, due to fundamental physical limitations. As a consequence of that, the costs of manufacturing new silicon wafers are sky-rocketing. Therefore, modern high-performance computing (HPC) processors partition large multi-core designs into smaller *chiplets* (Figure 5.1) that deliver better yield and lower the production cost [168, 169]. Individual chiplets can be tested separately and then reassembled using any of the different packaging technologies available, such as multi-chip modules (MCMs) [169, 170], 2.5D integration over a silicon interposer [168, 171, 172, 173], or silicon bridges [174, 175]. For example, AMD's EPYC 7742 processor [109] uses an 8-chiplet MCM organization and 7 nm technology to support up to 64 cores interconnected through its Infinity Architecture technology [176]. As another example, NVIDIA's Simba prototype [177] consists of a 36-chiplet MCM system for deep-learning inference, fabricated in 16 nm FinFET process technology, and connected through a 2D mesh network, totaling 576 processing elements (PEs). Further, Intel has already made public a detailed description of its upcoming Ponte Vecchio Xe-HPC GPGPU for supercomputers [178], which will use a variety of process technologies centered around 7 nm, and a disaggregated modular architecture consisting of 47 tiles, joined together by its proprietary Embedded Multi-Die Interconnect Bridge (EMIB) technology [175]. In the near future, it is likely that disintegration and chiplet counts will continue to increase.

For these manycores, shared memory is currently the most popular programming and execution model, due to its ease of programming, well-developed existing algorithms, and widespread libraries such as POSIX threads and OpenMP. To support shared memory at this scale, designers use directory-based cache-coherent systems [127]. However, as the number of cores per package increases, engineering massive scale directory-based cache-coherent systems that deliver high performance without an inordinate increase in complexity and cost will present challenging; the main reasons being the poor scaling of the directory storage, and most importantly the communication latency. While there have been many attempts to design scalable directories for manycores [128, 129, 130, 133, 134, 135, 179], these have all been devised for monolithic architectures, and rely on the basic premise that all cores and directory slices are on the same chip, and that network-on-chip (NoC) latencies between cores and directory slices are relatively low. This basic premise no longer holds in the chiplet scenario, since manycores are fragmented into different dies, and chiplet-to-chiplet latencies are twice as large as on-chip [177].

Therefore, while the chiplet paradigm comes with great advantages in yield and heterogeneity, its large chiplet-to-chiplet latencies leads us to rethink the way manycore chips will be programmed in the near future, and how to address the problem of the coherence wall by introducing ideas from supercomputing into the chiplet-based manycore scenario.

On-chip wireless communication is a new technology that has recently sparked a lot of interest [34, 35, 36, 37, 38, 39, 40, 41, 46, 179]. While most of these designs have focused on monolithic architectures, researchers are already extending this technology to the novel chiplet-based systems, leading to the concept of Wireless Networks-on-Package (WNoPs) [180]. In WNoPs, each chiplet in a manycore package has a transceiver and an antenna, which it uses to communicate wirelessly with other chiplets. Thanks to completely integrated on-chip antennas [30] and transceivers [181, 182], short-range wireless communication up to 100 Gb/s with small resource overheads has recently become viable. Because a wireless message can cross a large package in around 5 ns [181], a WNoP enables low-latency transfers, even though its bandwidth is limited (just one or a few messages can be transmitted at a time). WNoPs also allow multicasts and broadcasts by nature. As a result, WNoPs have been proposed as a way to speed up parallel applications. WiSync [45] uses it to speed up synchronization, Choi *et al* [132] uses it to speed up CNN training in CPU-GPU heterogeneous architectures, WIENNA [180] uses it to connect an array of DNN accelerator chiplets to the global buffer chiplet, and WiDir[179] uses it to augment a conventional invalidation-based directory cache coherence protocol.

An intriguing question is whether a chiplet-based architecture can be augmented to use a wireless network so that common supercomputer parallel programming models can be supported efficiently. Parallel programming models on supercomputers include: (1) pure message passing, which consists of one MPI process running on each core; (2) hybrid shared memory and message passing, which can be achieved with the standard MPI for inter-node communication, and with either OpenMP or the novel MPI-3.0 for shared memory intra-node communication; and (3) OpenMP only, which is effectively a distributed virtual shared memory.

Several message-passing collective primitives, such as MPI's *Barrier, Bcast, Scatter, Gather, Allgather, Reduce, and Allreduce*, can be efficiently supported by a WNoP: node to node communication (or, in our case, chiplet-to-chiplet) can take place wirelessly without the high latency associated with multi-hop chiplet transactions. To implement these MPI primitives thoroughly, however, one must carefully integrate wired and wireless operations in a seamless manner.

In this work we present the *WiPackage* architecture. *WiPackage* augments each chiplet with a multi-channel wireless transceiver, and extends an MPI implementation with some wireless transactions. This enables the use of wireless channels to send and receive data across chiplets, bypassing the high-latency multi-hop transactions of the wired network-on-package. We refer to the concept of combining message passing with shared memory in a single package, where each

chiplet is effectively a node, as a *supercomputer on a package*.

Contributions. The contributions of this work include: (1) the *WiPackage* architecture; (2) a dynamic usage of multiple wireless channels; (3) an adaptive wireless ring scheme for *n-to-1* communication; (4) a chiplet-based implementation and analysis of common message-passing collective primitives for *WiPackage*; (5) an evaluation of the primitives for *WiPackage*, and their impact in the overall performance of the studied applications.

5.2 BACKGROUND & MOTIVATION

5.2.1 Chiplet-Based Systems

When compared to a 250 mm^2 die on a 45 nm process, the 16 nm process approximately doubles the cost per yielded mm, and the 7 nm process nearly quadruples it [169]. The cost of moving to 5 nm and even 3 nm nodes is projected to rise even farther. Fabricating huge monolithic dies is becoming more and more expensive. Chiplet-based architectures, in which a single chip is divided into numerous smaller and interconnected chiplets (Figure 5.1), is one option for easing the manufacturing costs of large semiconductors.

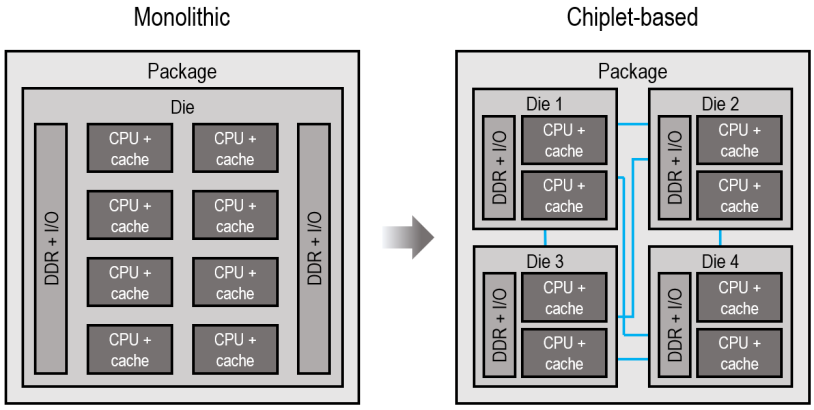
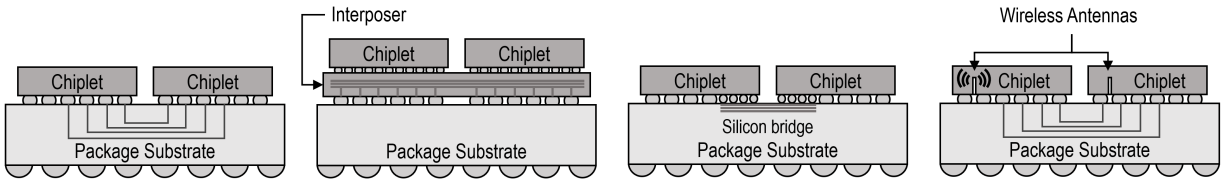


Figure 5.1: An example of a chiplet-based disintegrated design.

5.2.2 Chiplet Interconnect Technologies

Chiplet-based architectures require a fast and efficient chiplet-to-chiplet interconnect to enable data sharing and synchronization across the system. As we describe next and show in Figure 5.2, several *wired* alternatives enable this approach.

The traditional approach to interconnecting multiple chips within the same package is the Multi-Chip Module (MCM, Figure 5.2a) [109, 170, 176, 184]. MCMs rely on the integration and inter-



(a) Multi-Chip Package (b) Silicon Interposer (c) Silicon Bridge (d) Wireless-Augmented Multi-Chip Package

Figure 5.2: Chiplet-to-chiplet interconnection technologies, according to Intel [183] (a, b, c) and Guirado *et al.* [180] (d).

connection of chiplets directly on top of organic package substrate. This option offers large room for accommodating chiplets, e.g. beyond 70×70 mm^2 [170] yet at a relatively coarse I/O bump pitch over 100 µm [184]. To make up for the large pin sizes, ground-referenced signaling (GRS) serial links at multiple tens of Gb/s are implemented instead of more traditional parallel links. This, however, increases the hop latency up to a few tens of nanoseconds [177] and discourages the use of long links across the package. The scalability of the solution is thus limited by latency problems.

An alternative technology is the silicon interposer, which is effectively a *large chip upon which other smaller dies can be stacked* as shown in Figure 5.2b [168]. This allows to interconnect chiplets at a much greater density (i.e. around an order of magnitude greater) than in the classical MCM [168, 185, 186], which enables the implementation of low-latency parallel links. However, this comes at the expense of a high manufacturing cost. To be affordable, the interposer size needs to be generally limited far below the recent 1700-mm^2 demonstration by TSMC [187]. Moreover, the connectivity is still limited by the amount of pins, which discourages the implementation of high-radix topologies. Instead, the pins are used to increase the bandwidth of parallel links in a mesh topology. Therefore, cost and latency are limiting the scalability of this approach.

A third alternative recently promoted by Intel is the Embedded Multi-die Interconnect Bridge (EMIB, Figure 5.2c) [175, 178, 188]. The solution consists in integrating very small silicon dies or *bridges* within the package substrate, to which chiplets can be connected at a fine granularity. These bridges are strategically located at the edge of chiplets, allowing two adjacent chiplets to be interconnected with high bandwidth and low latency. As a result, EMIB offers the speed of an interposer without its size constraints. However, the connectivity is clearly limited to neighboring chiplets: in systems with large chiplet counts, certain data communication patterns will require many hops to complete.

In all these cases where only contiguous chiplets are interconnected for various reasons, the high latency associated with inter-chiplet communication renders multi-hop coherence transactions across chiplets extremely costly, decreasing performance and jeopardizing the scaling of the

system.

In this context, wireless technology represents an opportunity to greatly alleviate the issues of existing chiplet interconnects. As shown in Figure 5.2d, communication happens through electromagnetic waves are radiated by antennas integrated within the chiplets. The waves propagate within the package at the speed of light. That leads to system-wide low latency (comparable to that of a single MCM hop) and inherently broadcast communication without any pin or cost-related size constraints [44, 180, 189]. However, since the medium is shared, wireless technology can only offer a moderate aggregate bandwidth of a few hundred Gb/s. Hence, this technology is expected to complement a wired interconnect alternative.

5.2.3 Shared Memory and Cache Coherence in Manycores

Scalable cache coherence is attained through the use of directories [128, 129, 130, 133, 134, 135]. Directory cache coherence is widely used commercially, such as in Intel's Core i7 systems. Proposed directory-based coherence schemes make different tradeoffs in meeting their cost requirements (*i.e.*, storage overhead, latency, and energy). Traditional schemes such as full-bit vector (or full-map) directories maintain sharer information accurately. However, full-map directories are unsuitable for manycore designs, since directory overhead scales linearly with the number of processors. For example, in a 64-core system with a 64-byte cache line, the directory overhead is 12.5%, while in a 1024-core system, the overhead is 200%. To overcome the area inefficiency of full-map directories, other techniques represent sharer sets inexactly. Examples of these techniques include coarse-bit vector [190], limited pointer [129, 130], sparse [190], and tag-less [191] directories. However, while these schemes substantially reduce the directory overhead when compared to full-map directories, their inability to track sharers precisely introduces additional traffic in the form of spurious invalidations, which in turn degrades performance significantly. Most importantly, all these works have a common flaw that challenges their scalability when applied to new chiplet-based manufacturing trends: they were designed for monolithic architectures, and rely on uniform and low on-chip communication latencies.

However, current chiplet-based designs consist of low-radix and high-diameter topologies with high average hop counts and tens of nanoseconds of latency between chiplets [169, 177]. Therefore, even with a low numbers of chiplets, these designs exhibit significant variances in access latency to different addresses [169]. Recent solutions seek to make memory accesses in chiplet-based systems more uniform by implementing a central I/O die through which all memory traffic is routed [169], however these are only temporary solutions, as the switching fabric on the I/O die becomes then the bottleneck for performance and power.

Due to these latency overheads, inter-chiplet communication in general, but cache coherence

traffic in particular (which commonly involves multi-hop coherence transactions), can now significantly reduce system performance and therefore limit scaling. Unfortunately, there has yet to be discovered a cache coherence scheme that can efficiently scale to a large number of chiplets.

5.2.4 Message Passing and Hybrid Programming

The Message Passing Interface (MPI) continues to be the preferred programming model for high performance computing systems [192]. In the message passing model, the programmer explicitly orchestrates the data communication in a application using a set of communication primitives.

On large HPC systems, hybrid application programs using MPI and OpenMP have become commonplace. In these setting MPI defines parallelism between processes (with separate address spaces), and Pthreads or OpenMP provides shared memory parallelism. This mix of programming models is primarily motivated by two factors: (1) memory footprint reduction in both the application and the MPI library (e.g. communication buffers); and (2) improved performance, particularly at high core counts where the scalability of pure MPI applications starts to saturate. We believe this model is well suited for the chiplet scenario. Therefore, in the same way that nowadays node-to-node communication happens through message passing and off-chip interconnection networks such as InfiniBand, in the chiplet scenario we would have chiplet-to-chiplet communication through the on-package network, and integrated within a single package.

5.2.5 Wireless Network-on-Package

Short-range wireless communication up to 100 Gb/s with affordable resource overheads has recently become possible thanks to fully integrated on-chip antennas [30] and transceivers [181, 182]. Wireless Network-on-Package (WNoP) is one of the uses for this technology. In an WNoP, processor or memory chiplets are equipped with antennas and transceivers for communicating with other chiplets, with the system package serving as the propagation medium. This in-package channel is static and controlled, allowing for optimization. Timoneda *et al.* demonstrated that system-wide attenuation below 30 dB is possible [119]. These numbers match Yu's *et al.* 65 nm CMOS 30/60/90-GHz transceiver specifications: 48 Gb/s of aggregate bandwidth, 1.95 pJ/bit at 25 mm distance with error rates under 10^{-12}, and 0.8 mm^2 of total area [181]. WNoP provides scalable broadcast capabilities and low latency across the system by removing the requirement for wires between transceivers. It is scalable since additional chiplets simply need a transceiver to participate in the communication, and not extra off-die wiring. Because there are no I/O pin limitations the bandwidth is high, and the latency is low because transfers avoid intermediate hops. WNoP also

enables dynamic topology changes using reconfigurable media access control or network protocols [193], since receivers can determine whether or not to handle incoming transfers at run-time.

5.3 OVERVIEW OF *WIPACKAGE*

In this work, we envision a hierarchical memory and network design called *WiPackage*. On the one hand, *WiPackage* is comprised of a set of chiplets, each operating as a separate shared-memory domain, with its own separate wired on-chip network. On the other hand, *WiPackage* provides a message-passing wireless-augmented interconnection network, aimed at sending data across chiplets. The advantages of *WiPackage*'s hierarchical design are manifold:

1. The low core count of each chiplet ensures fast intra-chip wired communication (4 and 6 hops at most in an 8 and 16 core-per-chiplet scenario, respectively), and low contention to its local directory.

2. The wireless links used to communicate across chiplets can be dynamically rearranged, as opposed to the unscalable state-of-the-art designs that use static wired paths between each pair of chips.

3. The inter-chiplet wireless-based communication is latency-independent from the distance between the two chiplets, given that signals travel at the speed of light. This is not true for the currently implemented wired paths.

4. Package-wide broadcast traffic, as generated by MPI primitives like `MPI_Barrier` or `MPI_Bcast`, is inherently free, due to the natural broadcast capabilities of wireless. Alternatively, wired paths require sending as many individual packets as receivers are in the communicator pool.

5. The irregular communication patterns of the multi-threaded shared memory portions of certain applications are best suited for a small-world wired mesh NoC, as we have in each chiplet, since that allows the use of a single clock domain and single shared memory address space, and enhances the performance of tightly collaborative threads.

6. The regular communication patterns of the message-passing portions of certain applications open the possibility for wireless channel parallelism, so that different chiplets can communicate with each other concurrently, and speed up the data transfer.

5.4 *WIPACKAGE* ARCHITECTURE

Figure 5.3 illustrates the *WiPackage* architecture. In essence, *WiPackage* consists of an array of chiplets surrounded by High Bandwidth Memory (HBM) modules, acting as node memory. At the

same time, each chiplet contains an array of cores (with their corresponding private L1 caches), a shared last-level cache (LLC) with its corresponding directory, a local memory that we use as a mailbox for incoming and outgoing inter-chiplet messages, and a DRAM controller. Additionally, each chiplet contains a network-on-package router and wireless transceiver, used for chiplet-to-chiplet communication.

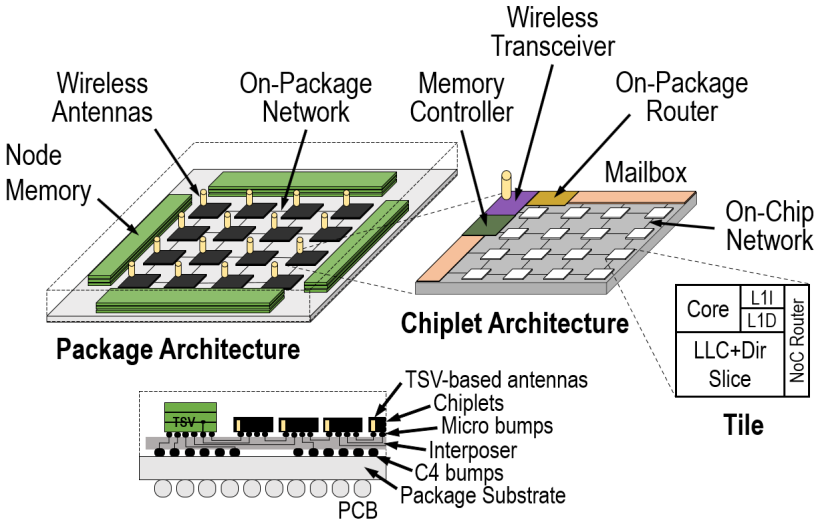


Figure 5.3: Overview of the *WiPackage* architecture.

WiPackage implements a two-level hierarchy. On the one hand, the array of chiplets and the HBM-based node memory follow a 2.5D integration scheme, interconnected through a hybrid wired/wireless network-on-package. Each chiplet has access to an independent section of the node memory. On the other hand, each chiplet operates as a separate shared-memory domain, with its own wired on-chip network. All cores within the same chiplet can access the same section of node memory. Data exchanges between chiplets consist of explicit message passing transactions, and are sent through the wired/wireless network-on-package.

Like in MemPod's design [194], or in next-gen AMD Zen 4 EPYC Genoa and Intel Sapphire Rapids Xeon CPUs, which are both expected to incorporate HBM Memory, in *WiPackage* we opt for HBM modules instead of off-chip memory due to its performance and efficiency.

5.4.1 Multiple Wireless Channels

Previous wireless-enabled monolithic architectures required the placement of one transceiver and antenna per core [45, 46, 179]. Because the floorplan of a monolithic chip is limited, these designs were only able to use a single wireless data channel across the entire chip, as multiple channels require more complex transceivers and the use of multiple antennas. As we move to chiplet-based architectures and we place wireless transceivers on a per-chiplet basis instead of per-core,

the area available for these transceivers increases, and we are therefore able to use more complex designs, with multiple wireless channels. Previous designs had already proved the feasibility of antennas and transceivers capable of enabling multiple on-chip wireless channels [44, 181, 195]. However, due to area constraints, architects were not able to integrate them into full packages. With the advent of the chiplet-based era, this is bound to change.

The *WiPackage* architecture integrates a 4-channel wireless transceiver per chiplet. This means that at any time, each chiplet has the ability to send and receive data across up to 4 channels simultaneously. The entity in charge of granting and revoking access to the different channels is the operating system (OS). Because the wireless channels are used for message passing communication across chiplets, and because each communication is associated with a communicator group, the OS grants and revokes channel usage on a communicator basis (and therefore the chiplets in which they are mapped). In turn, this enables simultaneous wireless transmissions on different wireless channels from different communicator groups, either from the same or different applications.

Breaking down the entire wireless bandwidth into multiple disjoint channels enables frequency-multiplexing the medium when multiple applications or communicator groups attempt to transmit at the same time, while still allowing a single communicator to use the whole spectrum when no one else needs it. Because the number of channels that each communicator group uses may vary at runtime, the implementation of the MPI algorithms described later in Section 5.5 is channel-independent, meaning that it does not rely on holding a certain amount of channels.

5.4.2 Barrier Mechanism With Tone Channel

WiPackage provides an additional wireless channel, called the Tone channel, as a scalable solution to the synchronization mechanism of barriers, and to reduce contention on the data channels. We apply TLSync [91] and WiSync's [45] ideas for shared-memory monolithic on-chip transmission lines and wireless, respectively, to our message-passing chiplet-based architecture. Chiplets transmit a tone rather than data on the Tone channel. The first master core to reach the barrier sends a certain message across any of its assigned data channels. Upon receiving this message, the transceivers of all other chiplets start transmitting a continuous tone in the Tone channel. As soon as all processes of a chiplet reach the barrier, the master core orders its local transceiver to stop sending the tone. As the tone fades away, we know all processes in all chiplets have reached the barrier.

Since at any given time there may be multiple active barriers pertaining to different applications or communicator groups, *WiPackage* supports multiple barriers through a single Tone channel by time-multiplexing it. The channel is divided in 1-ns slots and, similar to in WiSync [45], two tables

in each transceiver monitor which barriers are allocated across the system and which are active in each chiplet. To allow controlling the tone channel at 1-ns granularity, the associated transceiver needs to operate at 1 Gb/s only. This hugely simplifies the tone channel transceiver and reduces its power consumption to a few mW [45].

To address the long latency overhead associated with inter-chiplet communication, in Section 5.5.1 we used the Tone channel as part of the *WiPackage* implementation for the MPI barrier.

5.4.3 Adaptive Wireless Ring

While the traditional *1-to-n* wireless communication pattern can be trivially implemented (one node broadcasts a message, all others listen), the inverse *n-to-1* pattern requires an efficient orchestration of each transmission, in order to avoid bottlenecks in the shared medium. Previous designs have used random access techniques [140] that allow nodes to attempt transmitting at any time they sense the medium is free, and retry upon detecting a collision. While these techniques work well when contention is low, they suffer from long tail latencies when many nodes attempt to transmit at the same time. In the latter scenario, a time-division multiplexing technique such as token ring provides better guarantees, since each node is granted exclusive access to the medium once per round, in a round-robin fashion.

However, one drawback of token ring is the number of wasted slots when nodes are idle, since no one else is allowed to transmit instead. To overcome this limitation, in *WiPackage* we propose an adaptive wireless ring scheme, which allows the token to skip nodes that have already finished their transmission, so that the pending ones can finish sooner. Effectively, this acts as a resizable ring, where as time progresses, the number of nodes in the ring decreases, and therefore the number of time slots as well. Since in wireless the token is passed implicitly (all nodes can hear all broadcast transmissions, and therefore all nodes are synchronized), this technique does not require any explicit exchange of information. Also, this technique would otherwise not be possible in wired networks, since links are static and predefined at manufacturing time, and nodes cannot hear whether other nodes have used their time-slot to transmit. The feasibility of this technique only applies to wireless networks where all nodes can listen to each other.

Figure 5.4 shows a walkthrough example of the adaptive ring. Initially, in Step (a), nodes 2, 3, and 7 have already transmitted their data and are therefore out of the ring. Also, node 0 holds the token, however since it has no data ready to transmit, the slot is left unused and the token passes to node 1. Next, in Step (b), node 1 holds the token, and since it has data ready to send it transmits right away. Because nodes 2 and 3 had previously already finished, the token is now implicitly passed to node 4. In Step (c), node 4 holds the token but has no data to transmit, so the time slot is wasted and the token passes to 5. Finally, in Step (d), node 5 uses its assigned time slot to transmit

its data, and then implicitly pass the token to node 6.

We have used the adaptive ring technique in Sections 5.5.4 and 5.5.5 as part of the *WiPackage* implementation for the gather and reduce collective primitives, to collect the data from the multiple chiplets to the root.

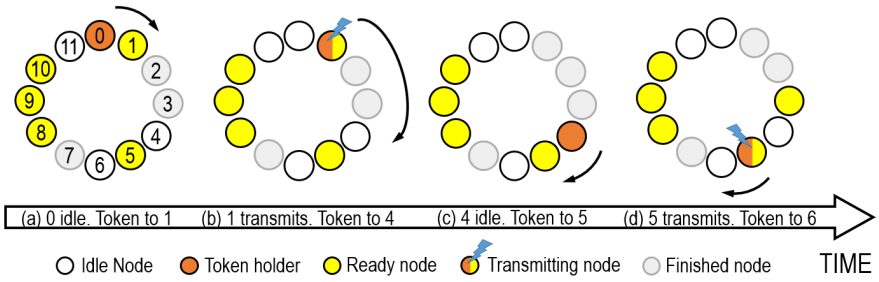


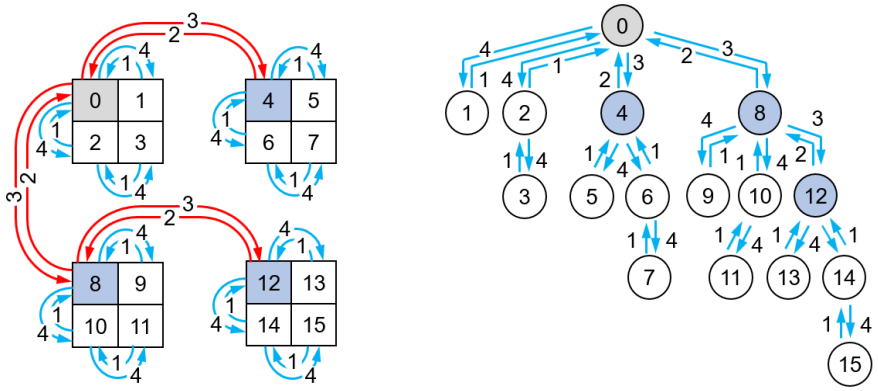
Figure 5.4: Example of the adaptive ring used in *WiPackage*

5.5 COLLECTIVE PRIMITIVES IN *WIPACKAGE*

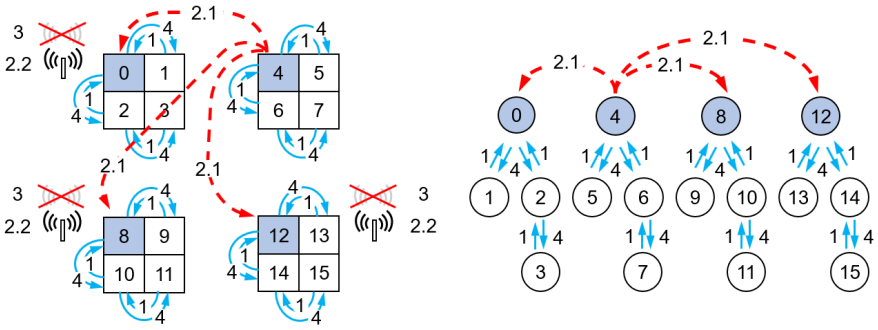
Collective communication is a mode of communication in which all processes in an MPI communicator participate. Although message passing collectives can be performed as send/receive operations, they usually consist of an entirely different, efficient implementation. This section describes a chiplet-based implementation for both *Baseline* and *WiPackage* architectures, of the most commonly used MPI collective operations: *Barrier*, *Broadcast*, *Scatter*, *Gather*, and *Reduce*. They are key building blocks of other operations, such as *Allgather*, *Allreduce*, and *Alltoall*. For simplicity and illustrative purposes, all examples in this section use a 4-chiplet and 4-core/chiplet setting, with all cores participating in the same primitive. The same ideas apply to higher scale-out designs, as well as to multiple co-existing MPI communicator groups.

5.5.1 Barrier

A *Barrier* is a primitive for synchronization. Any process that calls it is blocked until the entire group has called it as well. We choose to implement the MCS tree-based barrier algorithm [196], as is also used in Open MPI. This algorithm assigns each rank to a tree node. Each tree node contains an array of flags to track which of its children have reached the barrier. After a node and all its children have reached the barrier (*i.e.* all flags in the node are set), this node then notifies its parent. Leaf nodes start with their flag already set. Once the root node has reached the barrier and its array of flags is all set (*i.e.* all ranks have reached the barrier), it notifies all its children so they can proceed past the barrier.



(a) *Baseline*



(b) *WiPackage*

Figure 5.5: Propagation of data in an `MPI_Barrier` for both *Baseline* and *WiPackage* architectures. Solid lines indicate wired communication, whereas dashed lines are wireless. Blue lines refer to intra-chiplet traffic, and red lines to inter-chiplet. Root process is shaded in gray, and *master cores* shaded in blue.

Baseline implementation. Figure 5.5a shows the propagation of data in the baseline chiplet-based implementation of the MCS barrier algorithm, broken down into 4 phases. In phase 1 all processes within each chiplet notify their barrier arrival to the *master core* of their chiplet (shaded in blue). This communication takes place through the wired on-chip network (solid blue lines). In phase 2, as each master core also reaches the barrier and finds its array of flags all set, it uses the inter-chiplet on-package network to propagate their arrival information (solid red lines) to the root (shaded in gray). Once the root has arrived to the barrier and received the message from all its children, it follows the same procedure but in the opposite direction; that is, in phase 3 the root sends a departure notification to all master cores in the other chiplets, and in phase 4 each master core propagates the departure notification in its chiplet. All four phases are broken down into multiple steps, potentially requiring multiple inter-/intra-chiplet hops.

***WiPackage* implementation.** The *WiPackage* implementation of the barrier reuses the concept of master cores and wired intra-chiplet arrival/departure notifications from the *Baseline* (Section

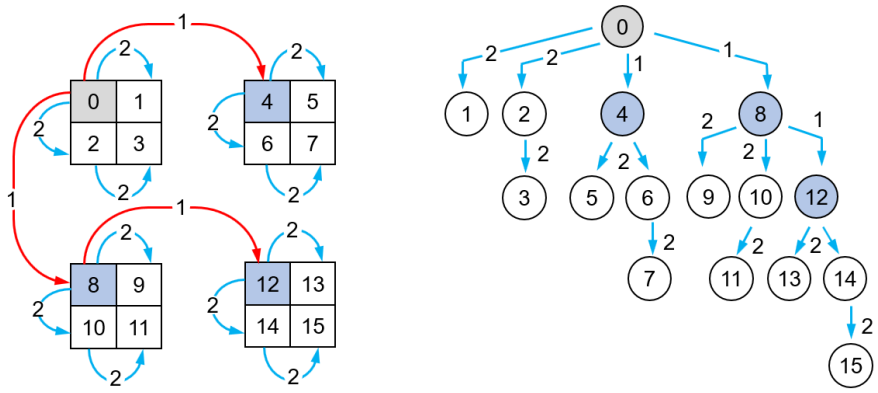
5.5.1), but changes the way master cores communicate their arrival/departure to the barrier with the root. Due to the chiplet nature of the architecture, the inter-chiplet communication between the master cores and the root is actually the segment of the algorithm that suffers the highest latency (see Section 5.2.2). To overcome high inter-chiplet latency, *WiPackage* leverages the tone channel described in Section 5.4.2. As shown in Figure 5.5b, the master core of the first chiplet that completes phase 1 (which is the same as in the baseline), broadcasts a message to all other chiplets (phase 2.1) through the wireless network (dashed red line). This triggers the activation of all other chiplets' tone channel (phase 2.2). Then, in phase 3, as the master core of each chiplet also arrives to the barrier with its array of flags all set to *true*, it turns off its tone channel and waits until all others do the same (no energy in the tone channel). When that happens, in phase 4, the master cores of each chiplet then propagates the departure message to all other cores of its chiplet through the wired on-chip network.

5.5.2 Broadcast

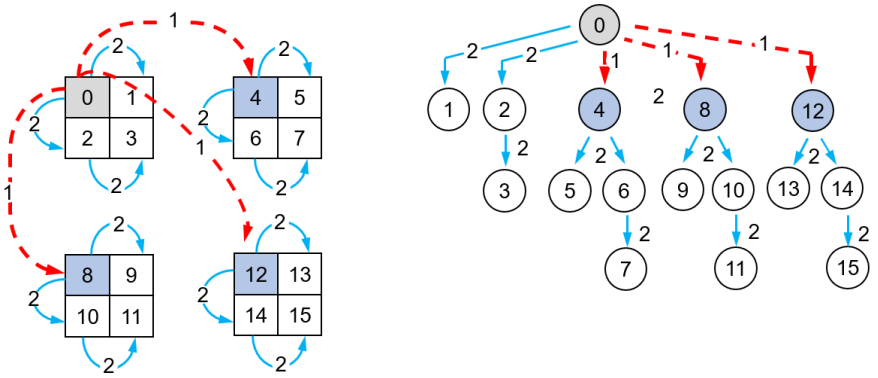
The broadcast operation forwards data from one process to all other processes in the group. Following MPICH, we implement the broadcast primitive with the binomial tree algorithm [197]. Using this algorithm, the root starts by transmitting all data in the buffer to its partner process ($root + p/2$), where p is the total number of processors in the group. Then, both the root and this partner process operate as new roots within their own sub-trees, continuing the algorithm in a recursive and parallel manner. It takes $\lceil log_2 p \rceil$ steps for the data to reach the farthest process from the root.

Baseline implementation. The chiplet-based implementation of this algorithm consists of 2 phases. In phase 1, the root sends the data to the master core of all other chiplets through the wired on-package network. Notice that for the farthest chiplets, this transaction takes multiple long-latency hops. Phase 2 consists of an intra-chiplet propagation, where the master core of each chiplet initiates a propagation of the data to all other processes within its chiplet, through the wired on-chip network. The transactions in the second phase might also take multiple on-chip hops to reach their destination.

***WiPackage* implementation.** In *WiPackage*, phase 1 of the broadcast implementation is performed wirelessly with a single packet and hop (Figure 5.6b), instead of the multiple individual packets through multiple long-latency inter-chiplet hops that the *Baseline* requires. This is a result of the natural broadcast capabilities of wireless, and is key to the performance improvement obtained by *WiPackage*.



(a) *Baseline*



(b) *WiPackage*

Figure 5.6: Propagation of data in an `MPI_Bcast` for both *Baseline* and *WiPackage* architectures. Solid lines indicate wired communication, whereas dashed lines are wireless. Blue lines refer to intra-chiplet traffic, and red lines to inter-chiplet. Root process is shaded in gray, and *master cores* shaded in blue.

5.5.3 Scatter

The *Scatter* primitive serves a similar purpose as the *Broadcast* routine described in Section 5.5.2, in the sense that it is used when one process possesses data that is required by a group of other processes. However, while in the *Broadcast* primitive the designated root sends the same data to all processes, in the *Scatter* the root sends different data to each process in the communicator. Following Open MPI and MPICH, we choose to implement the *Scatter* primitive also with the binomial tree algorithm [197], described in Section 5.5.2.

Baseline implementation. The propagation of data of the *Scatter* in a chiplet-based system is analogous to the one shown in Figure 5.6a, with the difference that in each phase the root (of either the whole tree or a sub-tree) must send half of all its data to its partner process, so that each can then spread it in parallel. Consequently, the messages are bigger in the first steps than in the last

ones, and the buffers for the intermediate processes must be sized not only to accommodate for their corresponding data, but also for the partners they will communicate with in the subsequent steps.

***WiPackage* implementation.** Similarly as in the *WiPackage* implementation for the *Broadcast* primitive (Section 5.5.2), the first phase of the *Scatter* can be transmitted wirelessly to bypass the long-latency inter-chiplet hops of the wired on-package network. However, unlike in the *Broadcast* primitive, each chiplet must receive a different chunk of data from the root's sending buffer. Therefore, in the first step of the *WiPackage* implementation, the root broadcasts the entire buffer wirelessly, and the master core of each chiplet picks its corresponding chunk of data by applying a mask to it. Analogously as in the *Broadcast* primitive (Figure 5.6), in the second phase, the master core propagates this chunk within its chiplet, through the wired on-chip network. Although again, the sub-chunks sent to each partner are different and smaller at each step.

5.5.4 Gather

The *Gather* collective primitive aggregates elements from all processes in a communicator group into a single process. Because a *Gather* can be viewed as a *Scatter* performed backwards, a *Gather* is also implemented with the binomial tree algorithm (Section 5.5.2), but with the direction of the data flow inverted – from the leaves to the root.

Baseline implementation. The *Baseline* implementation for the *Gather* follows the same 2-phase approach as the *Scatter* (Section 5.5.3), but reversed: phase 1 and 2 take place in the opposite order, and the direction of the data flow is also inverted. The number of steps, message sizes, and intermediate buffer requirements are however the same as in the *Scatter*. Unlike *Scatter*, each tree node in the *Gather* cannot proceed communicating with its parent until it has received all data from its children, *i.e.* the master core of one chiplet must wait until all cores within its chiplet have sent their data to it, before forwarding that data to the root chiplet. This restriction does not occur in the *Scatter*, since a parent always has its data ready for all its children.

***WiPackage* implementation.** Similarly as in the *Baseline*, in *WiPackage*, the data gathering of all processes within a chiplet to the master core is also done through the wired on-chip network. However, the second phase, where all master cores send their aggregated data to the root, is done wirelessly. As in the *WiPackage* implementation of the other aforementioned primitives, this has the advantage of bypassing the long-latency inter-chiplet wired links, and having the communication time to be distance-independent. Because the gathering of data from all master cores to the

root follows an *n-to-1* communication pattern, if the communication is not organized properly, it could lead to bottlenecks in the wireless medium. This is why in *WiPackage* we make use of the adaptive ring scheme described in Section 5.4.3, so that each master core only transmits on its assigned slot. The organized transmission of data with a token-ring overcomes the chaotic generation of collisions by alternative random-access methods like BRS [140], and guarantees a time by which data will be transmitted. Adding the adaptive ring sizing scheme further minimizes the wait time of each master core before they are allowed to transmit.

5.5.5 Reduce

Global reductions and combine operations are two of the most common collective operations. A global reduction uses a simple function like *sum*, *max*, or *min* to combine partial results from each process in the communicator group, and deliver the result to the root node. Commonly, the algorithm used for the *Reduce* is very similar to that of the *Gather*, with the difference that the amount of data being transmitted is the same at each step of each of the two phases. Therefore, both *Baseline* and *WiPackage* architectures implement the *Reduce* with the binomial tree algorithm described in Section 5.5.4.

Baseline implementation. As already stated, the *Baseline* implementation of the *Reduce* is similar to that of the *Gather*, in the sense that it consists of 2 phases, the first one being a propagation within each chiplet, and the second one being a propagation across chiplets to the root. However, the main difference here is that with *Reduce*, at the end of each communication step, each process applies a reduction operation to the received data and its own, instead of aggregating both, before sending the result to its parent.

***WiPackage* implementation.** Because as we mentioned in Section 5.5.5 the *Reduce* implementation is somewhat similar to that of the *Gather*, in *WiPackage* we can also apply the adaptive ring technique that we used in Section 5.5.4 and described in Section 5.4.3 to the *Reduce* implementation. Similarly as described in Section 5.5.5 however, the difference between the *Gather* and the *Reduce* implementation is that the inter-chiplet communication is used to transmit the reduced values of each chiplet, instead of the aggregated data of all its processes, and therefore for a given message size, the overall wireless transmission time is lower in the case of the *Reduce*.

5.6 EVALUATION METHODOLOGY

We use the SST-macro [198] simulation framework together with the integrated SST-core [142] as a discrete event simulator to model a server architecture with up to 64 chiplets and up to 16

Table 5.1: Architecture modeled.

Package	
Number of Chiplets	16 (4×4), 32 (8×4), and 64 (8×8, default)
NoP Interconnect	Wired 2D-torus + Wireless
Wired NoP	20 ns/Hop [177], 100 Gb/s/Chiplet
Chiplet	
Number of Cores	4 (2×2), 8 (4×2, default), and 16 (4×4)
NoC Interconnect	Wired 2D-mesh
Wired NoC	10 ns/Hop [177], 100 Gb/s/Chiplet
General Parameters	
Clock Frequency	2.1 GHz
Node Memory	1 GHz, 31 ns access latency, 16 GB/s/channel [194]
***WiPackage* Parameters**	
Data Wireless Channels	30/60/90/120 GHz, 25 Gb/s [181, 199, 200, 201, 202]
Tone Wireless Channel	240GHz, 1Gb/s [203]
Transceiver	At 65nm: 0.25mm^2, 30mW [77, 78]
Data Converter	At 65nm: 0.03mm^2, 0.72mW [79]
Serializer/Deserializer	At 65nm: 0.04mm^2, 10.8mW [80]
Antenna	At 65nm: 0.08mm^2 [81]
All RF Circuit in Node (average, per channel)	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)

cores/chiplet. The architecture parameters as well as the energy and area of the wireless components are shown in Table 5.1. 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 10 ns/hop. We augment the simulator to model wireless transmissions in detail. The wireless network has four data channels of 25 Gb/s each, residing at four different central frequencies, as well as a tone channel of 1 Gb/s. The frequencies are picked as multiples of a common denominator, *i.e.* 30 GHz, as this minimizes the amount of bulky oscillators required to generate the carrier signals [181]. In Table 5.1, area and power consumption figures are given on average, per channel, based on experimentally validated transceiver and antenna designs [79, 80, 81, 181, 199, 200, 201, 202, 203].

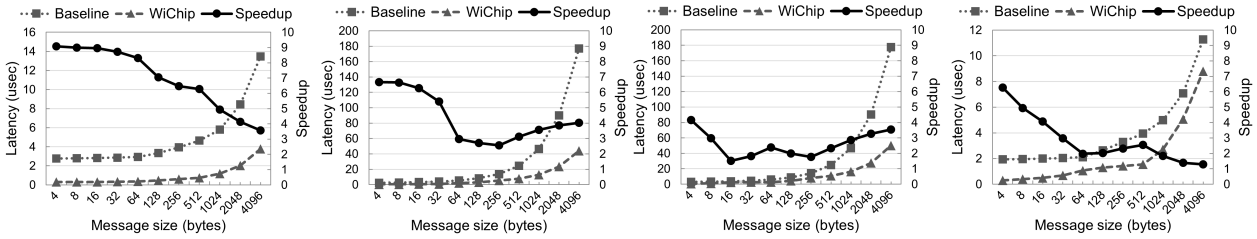
Table 5.2 presents the applications we use for the evaluation, what they do, and the MPI Collectives used in them. We used the compiler macros provided by SST-macro to compile the benchmarks, and used manual instrumentation to break the runtimes down to the components.

5.7 RESULTS

In this section we first provide a performance and scaling analysis of each of the collective primitives described in Section 5.5, we then evaluate the performance of 10 MPI applications in *WiPackage* and compare it with that from the *Baseline* architecture. Finally, we show an exploration of the impact that the different architectural parameters in both wired and wireless networks

Table 5.2: Benchmark details

Name	Description	MPI Collectives
bfs	Breadth First Search on a random graph	`MPI_Bcast`, `MPI_Gather`
sssp	Single Source Shortest Path on a random graph	`MPI_Bcast`, `MPI_Gather`
matvec	Matrix Vector Multiplication	`MPI_Bcast`, `MPI_Scatter`, `MPI_Gather`
sobel	Sobel edge detection on a random bitmap [204]	`MPI_Scatter`, `MPI_Gather`
kmeans	Partitions random input points into k clusters [205]	`MPI_Bcast`, `MPI_Scatter`, `MPI_Reduce`
blackscholes	Computes the prices of a portfolio of options	`MPI_Scatter`, `MPI_Gather`
hist	Computes a histogram from a random list [206]	`MPI_Scatter`, `MPI_Reduce`
quad	Approximate an integral using a quadrature rule [207]	`MPI_Bcast`, `MPI_Reduce`
satisfy	Exhaustive search for solutions of the circuit satisfy problem [208]	`MPI_Bcast`, `MPI_Reduce`
tightloop	Adding a list of elements synchronized by a barrier[45]	`MPI_Barrier`



(a) Broadcast (b) Scatter (c) Gather (d) Reduce

Figure 5.7: Latency of MPI *bcast*, *scatter*, *gather*, and *reduce* collectives, over different message sizes, for *WiPackage* and *Baseline*.

have in the overall execution time of the primitives and applications studied. We will use XX/YY to refer to a configuration with XX chiplets and YY cores per chiplet.

5.7.1 Characterization of Collective Primitives

Figure 5.7 shows the latency of *Broadcast*, *Scatter*, *Gather*, and *Reduce* MPI collective primitives in *Baseline* and *WiPackage* as we increase the size of the messages. The figures also show the speedup generated by *WiPackage*.

Broadcast. As demonstrated in Figure 5.7a, the *1-to-n* pattern exhibited by the broadcast primitive proves very efficient in *WiPackage*. This is because *WiPackage* uses a single wireless message and hop to bypass the long-latency multi-hop inter-chiplet communication associated with the *Baseline*. This advantage from *WiPackage* is more significant at smaller message sizes, where the transmission time of the data is comparable to the latency overhead posed by the die-to-die wired links.

Scatter. While the *Broadcast* and the *Scatter* primitives are implemented using a similar algorithm,

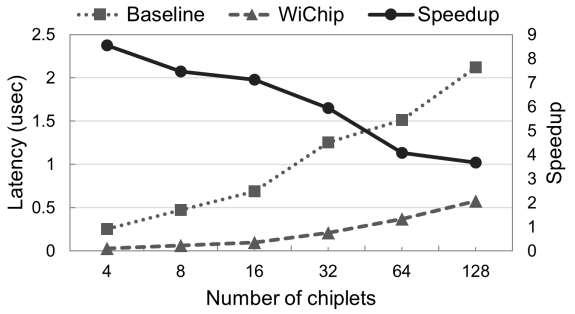


Figure 5.8: Latency of MPI *Barrier* for different number of chiplets with 8 cores/chiplet, for *WiPackage* and *Baseline*.

for any given message size m and p processes, the *Scatter* sends $m \times p$ data, whereas the *Broadcast* sends m. This *(p-1)* difference factor is translated into the speedups of *WiPackage* shown in Figure 5.7b.

Gather. As shown in Figure 5.7c, the *Baseline* implementation of the *Gather* suffers a very similar latency than the *Scatter*, since one is the inverse of the other. However, for *WiPackage* the implementation is sightly different, since the *Gather* makes use of the wireless adaptive ring, which makes a slightly less efficient usage of the wireless medium than the *Scatter* (in the *Scatter* the root always has data to transmit through the wireless medium, whereas in the *Gather* the master cores not always have their data ready to be sent to the root). Because in wireless the *n-to-1* communication pattern of the *Gather* is slightly less efficient than the *1-to-n* pattern of the *Scatter*, the speedups exhibited by *WiPackage* are lower than in the *Scatter*.

Reduce. The *Baseline* implementation of the *Reduce* has a similar latency as the *Broadcast* (Figure 5.7d). This is because from the communication perspective, for the wired *Baseline* it is basically the same process but inverted. For *WiPackage*, the *n-to-1* communication pattern of the *Reduce* is optimized by the wireless adaptive ring, which proves even more efficient than in the *Gather*, since for a given message size, the total size of the buffers being sent are smaller.

Barrier. As shown in Figure 5.8, *WiPackage* speeds up the *Barrier* primitive by a significant factor, for all chiplet counts studied.

5.7.2 Application Workloads

Figure 5.9 shows the total runtime of the benchmarks in *Baseline* and *WiPackage*, normalized by the *Baseline* runtime. The graph shows the runtimes for a 64 chiplet system. For each benchmark we show the runtimes for a system with 4, 8, and 16 cores per chiplet. The first bar in each configuration is the *Baseline* runtime and the second bar is the *WiPackage* runtime. Each bar is broken down into the runtime spent on computations (*Compute*), and the MPI collectives. Final set of bars

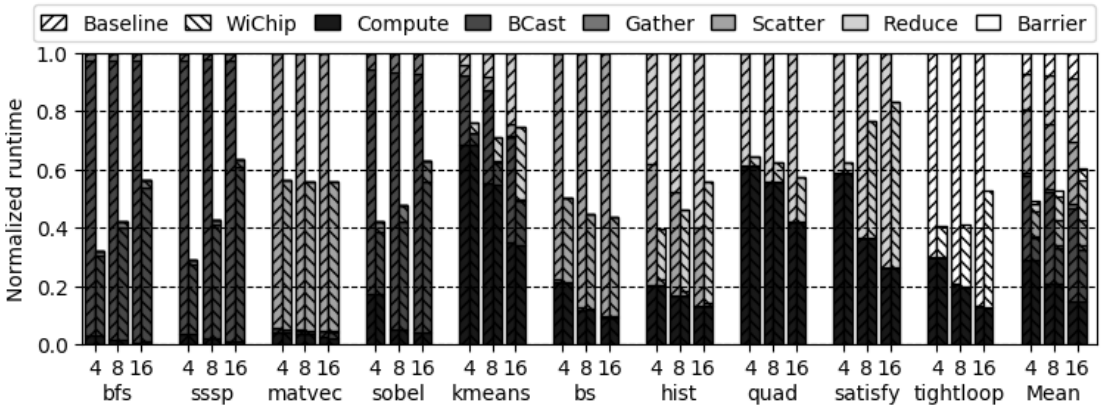


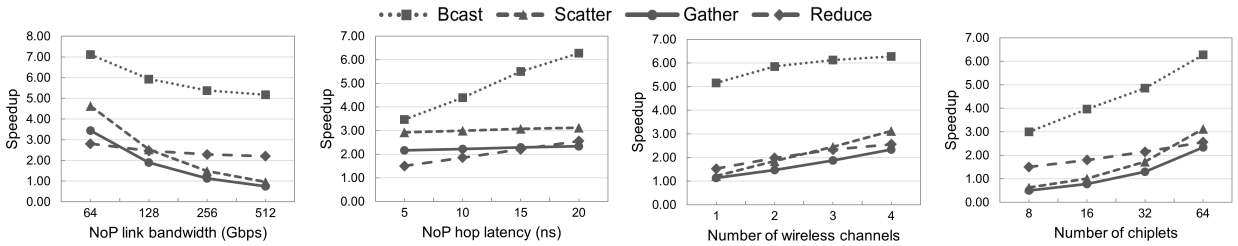
Figure 5.9: Execution time in *WiPackage* and *Baseline*, normalized to *Baseline* for 64 chiplets and 4/8/16 cores/chiplet. The bars are broken down based on execution time of MPI collectives.

in the figure shows the average runtimes for all benchmarks. arch produces an average speedup of 2.05x for the system with 64/8 configuration. Largest speedups were observed for programs that include a significant amount of global communication (2.42x for `tightloop` followed by 2.3x for `bfs` and `sssp`).

For the 64/8 configuration, *WiPackage* sped up broadcasts by 2.54x, gather by 1.19x, scatters by 1.85x, and reduces by 1.10x. `tightloop` benchmark shows the benefits of *WiPackage* for MPI barriers. *WiPackage* sped up the time processes spent waiting for all processes to arrive at the barrier 3.69x.

As the figure shows average speedups decrease when increasing the number of cores per chiplet: from 2.26x for a 64/4 configuration to 1.63x for 16 cores per chiplet. This is due to more time being spent in the second phase of the collectives where data is shared internally.

5.7.3 Sensitivity to Architectural Parameters



(a) NoP link bandwidth (b) NoP hop latency (c) Number of wireless channels (d) Number of chiplets

Figure 5.10: Speedup of *WiPackage* over Baseline for different MPI collectives with message size 512 Bytes, over different parameters.

Figure 5.10 shows the speedup of *WiPackage* over Baseline for the MPI primitives studied, for different architectural parameters.

Sensitivity to NoP link bandwidth. In Figure 5.10a we see how *Broadcast* and *Reduce* are the most robust primitives against a faster *Baseline* on-package wired network, whereas *Scatter* and *Gather* fall behind.

Sensitivity to NoP hop latency. Figure 5.10b shows how the primitives that benefit the most from *WiPackage*, such as *Broadcast*, are also the ones that lose the most advantage when the latency of the *Baseline* die-to-die links goes down.

Sensitivity to number of wireless channels. While having multiple wireless channels allows for multiple groups to communicate in parallel, it also allows for a single group to communicate faster if it can make use of multiple channels at the same time. Figure 5.10c shows that primitives like *Scatter*, *Gather*, and *Reduce* benefit more from hoarding multiple channels, whereas the higher advantage posed by the *Broadcast* pattern is more robust to lower aggregated bandwidths.

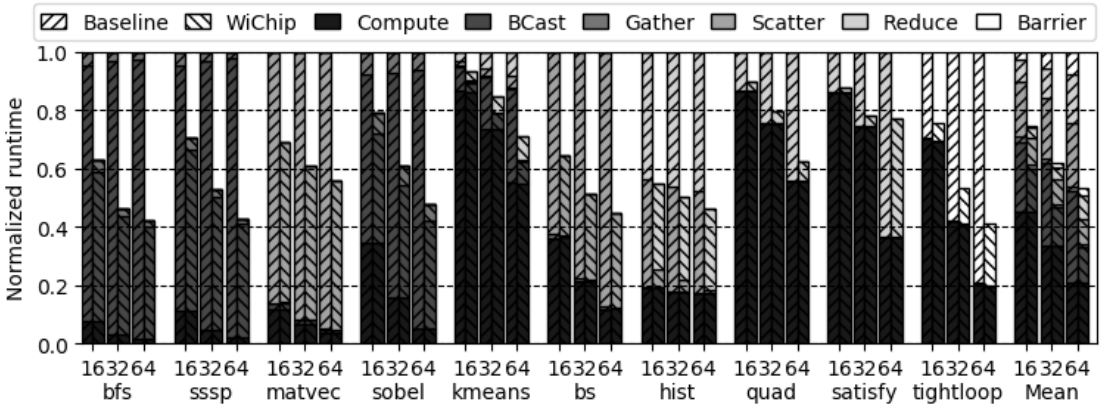


Figure 5.11: Execution time in *WiPackage* and *Baseline* over an increasing number of chiplets. Normalized to *Baseline*, for 8 cores/chiplet. The bars are broken down based on execution time of MPI collectives.

Sensitivity to number of chiplets. Figure 5.11 shows the runtimes for our benchmarks for 16, 32, and 64 chiplet configurations with 8 cores per chiplet. *WiPackage* produced average speed ups of 1.32x, 1.68x, and 2.05x for the 16/8, 32/8, and 64/8 configurations. These results show that as the number of chiplets increase the speedups from *WiPackage* increase with it. This trend is also confirmed by the pattern shown in the speedups of the primitives in Figure 5.10d.

Sensitivity to input size. Figure 5.12 shows the runtimes for the benchmarks when we increase the input size. We collected the runtimes for multiple inputs for each benchmark. We increased the input size 2x, 3x, and 4x. We observe that the speedups remain stable for communication-heavy benchmarks. However, for benchmarks with reductions (*quad*, *satisfy*) and the *tightloop*

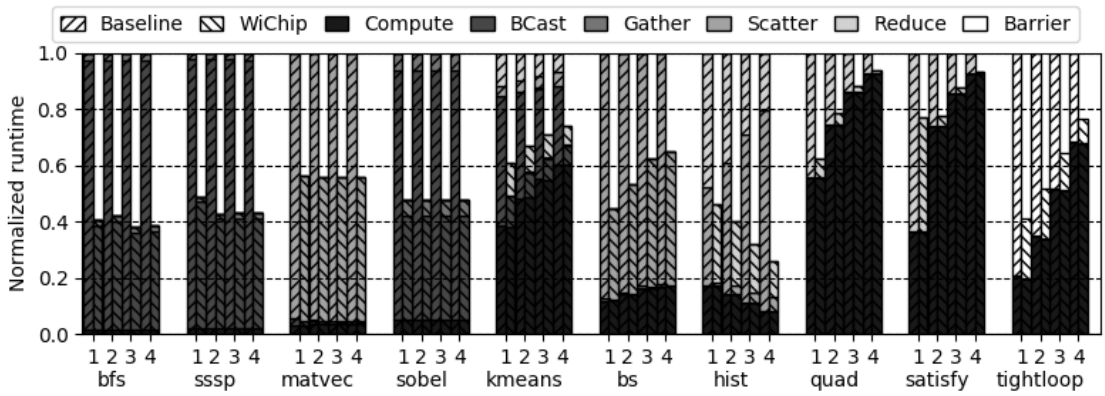


Figure 5.12: How speedups scale as input sizes increase for a 64 chiplet system with 8 cores/chiplet

benchmark, the communication volume does not grow as the number of input points increase, leading to lower speedups in bigger inputs.

5.8 RELATED WORK

Chiplet-Based Architectures. The chiplet approach has been shown to scale performance while improving yield in multi-CPU [169, 209], multi-GPU [170, 210], and multi-FPGA [211] architectures. The work by Fotouhi *et al.* explores the NUMA implications of coherence across multiple chiplets and how to adress them with a multi-chip bus [186]. All these approaches, however, suffer from scalability issues when increasing the chiplet count due to the interconnect limitations, which *WiPackage* aims to break with wireless technology.

Others have investigated how to scale out DNN accelerators with chiplets. NVIDIA demonstrated Simba, a weight stationary dataflow accelerator with up to 36 chiplets in a MCM mesh [177]. Tan *et al.* analyzed how the number of chiplets and their size affects the performance of Simba-like accelerators [212]. Further research on how to improve over Simba has proposed the use of wireless technology up to 16 chiplets with data compression [213] and 64 chiplets with adaptive dataflows [180]. The low-latency broadcast allows to speed up multicast data distribution. However, unlike *WiPackage* that targets general-purpose supercomputing broadly, the gains of [180, 213] are limited to the DNN accelerator domain only.

Wireless Architectures. We described WiSync [45], Choi *et al.* [132], WIENNA [180] and WiDir [179] in Section 5.1. Other works use a WNoC to accelerate the communication of certain applications [63, 147, 148]. These differ from *WiPackage* in that all architectures (except WIENNA) are single-chip and use a single wireless channel, and that only a set of applications are optimized (except WiDir). *WiPackage*, for the first time, exploits the benefits of multi-channel wireless in-

package communications to enable scalable and general-purpose supercomputing on a package.

Network-on-Package (NoP). Most multi-chip architectures count on either a fully connected NoP, e.g. Infinity Fabric [176], or a mesh NoP [170, 177]. These NoPs can already pose correctness challenges in heterogeneous architectures, which can be addressed at routing as illustrated in [210]. Other works have proposed novel topologies for multichip systems with passive or active interposers [168, 172, 214, 215], aiming at improving scalability while minimizing pin usage and interposer area.

The wireless NoP approach has been investigated in very few works. In [193], antennas communicate without distinction both within and across chips, creating a seamless interconnect across the system. In [189], the wireless NoP is designed to prioritize one-to-many transfers from memory to processor chiplets over one-to-one or many-to-few communication flows. In both cases, one channel at 16 Gb/s, which is insufficient in modern multi-chip processors. In WIENNA [180], the wireless network operates at 64 or 128 Gb/s with a unidirectional wireless network that does not require collision handling. *WiPackage* is different in that four channels of 25 Gb/s each are dynamically shared across the chiplets via an intelligent MAC protocol optimized for the MPI primitives.

Other technologies. There are two emerging technologies that can also offer inter-chiplet broadcast capabilities and low latency: nanophotonics [186, 216, 217] and integrated RF transmission lines [164, 218, 219]. Although these technologies are more efficient and provide more bandwidth than wireless communication, they are (i) more complicated as they require laying out an extra and overprovisioned network through the package, and (ii) less scalable as they are still limited by constraints related to pin scarcity, fanout, or laser power requirements.

5.9 CONCLUSION

To alleviate the pressure of directory-based coherence in chiplet-based manycores, and to bypass the high hop count and long latency associated with chiplet-to-chiplet communication, this work proposed a hierarchical memory and network architecture, called *WiPackage*. *WiPackage* used on-package wireless network technology to augment the standard implementation of common message-passing collective primitives to send data across chiplets. *WiPackage* carefully integrates wired and wireless operations in a seamless manner. In this work, we described the architecture and the implementation of these primitives in detail. Further, and evaluation showed that *WiPackage* substantially reduces the execution time of most common MPI collective primitives. For 64-chiplet runs with 8 cores per chiplet, *WiPackage* reduced the execution time of applications by an average of 2.05x compared to the *Baseline* implementation. Moreover, *WiPackage* was shown to be more scalable than the *Baseline*. These benefits were obtained with a very modest power cost.

CHAPTER 6: CONCLUSIONS

This thesis presented several applications of on-chip wireless technology for hundred-core systems and beyond. In diverse ways, the proposed systems take advantage of the properties of wireless to overcome the scalability challenges left unmet by wired on-chip networks in manycore processors.

First, this thesis presented the design and implementation of *Replica*, a manycore architecture that uses wireless communication for synchronization and communication-intensive ordinary data. In *Replica*, we also proposed an adaptive wireless protocol with selective message dropping to offer improved performance. The adaptive wireless protocol presented and employed in *Replica* was subsequently expanded and researched in much greater depth in the second study, *Fuzzy-Token*, which is a medium access control protocol specifically tuned to the unique characteristics of wireless on-chip networks. We showed that with a set of simple rules, *Fuzzy-Token* achieved the low latency of contention-based protocols at low loads, and the high throughput of fair collision-free protocols, such as token passing, at high loads. We evaluated the *Fuzzy-Token* protocol in a variety of synthetic traffic patterns and with real application traces, demonstrating a severalfold speedup with respect to other state-of-the-art protocols.

The third effort augmented a traditional directory-based invalidation cache coherence protocol with wireless communication to increase the programmability of the system provided in the first work. The resulting protocol, dubbed *WiDir*, transitions between wired and wireless coherence transactions for the same data in a programmer-transparent manner depending on access patterns. In this thesis we described the protocol in detail, and 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.

Lastly, in the fourth contribution of this thesis, we focused on the novel chiplet paradigm, and its scalability concerns beyond a handful of dies. To overcome the pressure of coherence traffic, and to bypass the long latency associated with chiplet-to-chiplet communication, we presented *WiPackage*. To transport data between chiplets, *WiPackage* used on-package wireless network technologies and enhanced the conventional implementation of common message-passing collective primitives. *WiPackage* meticulously blended wired and wireless functions for a seamless experience. Our evaluation showed that *WiPackage* significantly decreases the execution time of the majority of MPI collective primitives. When compared to a traditional wired implementation, *WiPackage* decreased application execution time by an average of 2.05x for 64-chiplet runs.

Our contributions, together with other advances in the field of on-chip scale wireless communication, pave the way for scalable and efficient manycore processors for general-purpose computing, machine learning acceleration, and high-end server processors.

REFERENCES

[1] J. Lowe-Power, A. Basu, J. Gu, S. Puthoor, B. M. Beckmann, M. D. Hill, S. K. Reinhardt, and D. A. Wood, "Heterogeneous system coherence for integrated cpu-gpu systems," in *Proceedings of the MICRO-46*, Dec 2013, pp. 457–467.

[2] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, Dec 2012, pp. 449–460.

[3] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-out processors," in *2012 39th Annual International Symposium on Computer Architecture (ISCA)*, June 2012, pp. 500–511.

[4] B. Bohnenstiehl, A. Stillmaker, J. J. Pimentel, T. Andreas, B. Liu, A. T. Tran, E. Adeagbo, and B. M. Baas, "KiloCore: A 32-nm 1000-Processor Computational Array," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 891–902, 2017.

[5] 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.

[6] G. Nychis, C. Fallin, and T. Moscibroda, "On-chip networks from a networking perspective: congestion and scalability in many-core interconnects," in *SIGCOMM*, 2012.

[7] H. Kwon, A. Samajdar, and T. Krishna, "A Communication-centric Approach for Designing Flexible DNN Accelerators," *IEEE Micro*, vol. 38, no. 6, pp. 25–35, 2018.

[8] G. Chen, M. A. Anders, H. Kaul, S. K. Satpathy, S. K. Mathew, S. K. Hsu, A. Agarwal, R. K. Krishnamurthy, V. De, and S. Borkar, "A 340 mV-to-0.9 v 20.2 Tb/s source-synchronous hybrid packet/circuit-switched 16 16 network-on-chip in 22 nm tri-gate CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 59–67, 2015.

[9] B. 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 *Proceedings of the ISCA-41*, 2014, pp. 25–36.

[10] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar, "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, 2008.

[11] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. F. Brown III, and A. Agarwal, "On-chip interconnection architecture of the tile processor," *IEEE Micro*, vol. 27, no. 5, pp. 15–31, 2007.

[12] T. Krishna, L.-S. Peh, B. Beckmann, and S. K. Reinhardt, "Towards the ideal on-chip fabric for 1-to-many and many-to-1 communication," in *Proceedings of the MICRO-44*, 2011, pp. 71–82.

[13] X. Xiang, S. Ghose, O. Mutlu, and N.-F. Tzeng, "A model for application slowdown estimation in on-chip networks and its use for improving system fairness and performance," in *Proceedings of the ICCD '16*, 2016.

[14] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, "Kilo-noc: a heterogeneous network-on-chip architecture for scalability and service guarantees," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2011, pp. 401–412.

[15] B. S. Feero and P. P. Pande, "Networks-on-Chip in a Three-Dimensional Environment: A Performance Evaluation," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 32–45, 2009.

[16] K. C. Chen, S. Y. Lin, H. S. Hung, and A. Y. A. Wu, "Topology-aware adaptive routing for nonstationary irregular mesh in throttled 3D NoC systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 10, pp. 2109–2120, 2013.

[17] S. Das, J. R. Doppa, D. H. Kim, P. P. Pande, and K. Chakrabarty, "Optimizing 3d noc design for energy efficiency: A machine learning approach," in *Proceedings of the ICCAD '15*, 2015.

[18] 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, 2012.

[19] R. Morris, E. Jolley, and A. K. Kodi, "Extending the Performance and Energy-Efficiency of Shared Memory Multicores with Nanophotonic Technology," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 1, pp. 83–92, 2014.

[20] J. L. Abellán, C. Chen, and A. Joshi, "Electro-Photonic NoC Designs for Kilocore Systems," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 2, p. Art. 24, 2016.

[21] 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, 2012.

[22] S. Laha, S. Kaya, D. W. Matolak, W. Rayess, D. DiTomaso, and A. Kodi, "A New Frontier in Ultralow Power Wireless Links: Network-on-Chip and Chip-to-Chip Interconnects," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, 2015.

[23] R. S. Narde, J. Venkataraman, A. Ganguly, and I. Puchades, "Intra-and Inter-Chip Transmission of Millimeter-Wave Interconnects in NoC-based Multi-Chip Systems," *IEEE Access*, vol. PP, pp. 1–1, 2019.

[24] J. Kim, K. Choi, and G. Loh, "Exploiting new interconnect technologies in on-chip communication," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 2, pp. 124–136, 2012.

[25] S. Abadal, A. Mestres, E. Alarcón, M. Nemirovsky, A. González, H. Lee, and A. Cabellos-Aparicio, "Scalability of Broadcast Performance in Wireless Network-on-Chip," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3631–3645, 2016.

[26] V. Fernando, A. Franques, S. Abadal, S. Misailovic, and J. Torrellas, "Replica: A Wireless Manycore for Communication-Intensive and Approximate Data," in *Proceedings of the ASPLOS '19*, 2019.

[27] T. S. Rappaport, J. N. Murdock, and F. Gutierrez, "State of the Art in 60-GHz Integrated Circuits and Systems for Wireless Communications," *Proceedings of the IEEE*, vol. 99, no. 8, 2011.

[28] E. Seok, D. Shim, C. Mao, R. Han, S. Sankaran, C. Cao, W. Knap, and K. K. O, "Progress and challenges towards terahertz CMOS integrated circuits," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 8, 2010.

[29] D. 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, 2012.

[30] H. M. Cheema and A. Shamim, "The last barrier: On-chip antennas," *IEEE Microwave Magazine*, vol. 14, no. 1, pp. 79–91, 2013.

[31] 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.

[32] O. Markish, B. Sheinman, O. Katz, D. Corcos, and D. Elad, "On-chip mmWave Antennas and Transceivers," in *NoCS*, 2015, p. Art. 11.

[33] S. V. Thyagarajan, S. Kang, and A. M. Niknejad, "A 240 GHz Fully Integrated Wideband QPSK Receiver in 65 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 10, pp. 2268–2280, 2015.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] D. 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, 2012.

[40] 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.

[41] 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.

[42] Q. J. Gu, "THz interconnect: The last centimeter communication," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 206–215, 2015.

[43] 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.

[44] 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.

[45] S. Abadal, A. Cabellos-Aparicio, E. Alarcón, and J. Torrellas, "WiSync: An architecture for fast synchronization through on-chip wireless communication," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016, pp. 3–17.

[46] V. Fernando, A. Franques, S. Abadal, S. Misailovic, and J. Torrellas, "Replica: A wireless manycore for communication-intensive and approximate data," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019, pp. 849–863.

[47] 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 *Proceedings of the 31st IEEE International System-on-Chip Conference (SOCC)*, 2018.

[48] 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.

[49] 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.

[50] I. El Masri, T. Le Gouguec, P.-M. Martin, R. Allanic, and C. Quendo, "Electromagnetic characterization of the intra-chip propagation channel in Ka and V bands," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 9, no. 10, pp. 1931–1941, 2019.

[51] S. Abadal, E. Alarcón, A. Cabellos-Aparicio, and J. Torrellas, "WiSync: An Architecture for Fast Synchronization through On-Chip Wireless Communication," in *ASPLOS*, 2016.

[52] Cray Research Inc., "CRAY T3D System Architecture Overview," 1993.

[53] S. Scott, "Synchronization and Communication in the T3E Multiprocessor," in *ASPLOS*, 1996.

[54] A. Gara, M. A. Blumrich, D. Chen, G. L.-T. Chiu, P. Coteus, M. E. Giampapa, R. A. Haring, P. Heidelberger, D. Hoenicke, G. V. Kopcsay, T. A. Liebsch, M. Ohmacht, B. D. Steinmacher-Burow, T. Takken, and P. Vranas, "Overview of the Blue Gene/L System Architecture," in *IBM Journal of Research and Development*, March/May 2005.

[55] J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," in *ISCA*, June 1997.

[56] Y. Fu, T. M. Nguyen, and D. Wentzlaff, "Coherence Domain Restriction on Large Scale Systems," in *MICRO*, 2015.

[57] P. Stenstrom, M. Brorsson, and L. Sandberg, "An Adaptive Cache Coherence Protocol Optimized for Migratory Sharing," in *ISCA*, 1993.

[58] 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 *PACT*, 2013.

[59] 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 *MICRO*, 2006.

[60] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. Beausoleil, and J. Ahn, "Corona: System Implications of Emerging Nanophotonic Technology," in *ISCA*, 2008.

[61] C.-K. Liang and Milos Prvulovic, "MiSAR: Minimalistic Synchronization Accelerator with Resource Overflow Management," in *ISCA*, 2015.

[62] W. Zhu, V. C. Sreedhar, Z. Hu, and G. R. Gao, "Synchronization State Buffer: Supporting Efficient Fine-grain Synchronization on Many-core Architectures," in *ISCA*, 2007.

[63] K. Duraisamy, H. Lu, P. P. Pande, and A. Kalyanaraman, "High-Performance and Energy-Efficient Network-on-Chip Architectures for Graph Analytics," *ACM Trans. Embed. Comput. Syst*, vol. 15, no. 26, 2016.

[64] K. Duraisamy, H. Lu, P. P. Pande, and Aananth Kalyanaraman, "Accelerating Graph Community Detection with Approximate Updates via an Energy-Efficient NoC," in *DAC*, 2017.

[65] S. Abadal, A. Mestres, J. Torrellas, E. Alarcón, and A. Cabellos-Aparicio, "Medium access control in wireless network-on-chip: A context analysis," *IEEE Communications Magazine*, vol. 56, no. 6, 2018.

[66] 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 *Proceedings of the 9th International Workshop on Network on Chip Architectures*, 2016.

[67] D. Clark, K. Pogran, and D. Reed, "An introduction to local area networks," *Proceedings of the IEEE*, vol. 66, no. 11, 1978.

[68] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *ISCA*, 1995.

[69] M. Ahmad, F. Hijaz, Q. Shi, and O. Khan, "CRONO: A benchmark suite for multithreaded graph algorithms executing on futuristic multicores," in *IISWC*, 2015.

[70] "Stanford Network Analysis Project `snap.stanford.edu`," 2018.

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

[72] R. Ubal, P. Mistry, D. Schaa, H. Ave, and D. Kaeli, "Multi2Sim: A Simulation Framework for CPU-GPU Computing," in *PACT*, 2012.

[73] N. Barrow-Williams, C. Fensch, and S. Moore, "A communication characterisation of SPLASH-2 and PARSEC," in *IISWC*, 2009.

[74] 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 *MICRO*, 2009.

[75] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[76] 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 *NoCS*, 2012.

[77] 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.

[78] 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.

[79] 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.

[80] 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.

[81] 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.

[82] H. K. Mondal, S. Kaushik, S. H. Gade, and S. Deb, "Energy-Efficient Transceiver for Wireless NoC," in *VLSID*, 2017.

[83] 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, 2015.

[84] 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 *HPCA*, 2008.

[85] C. Sun, M. T. Wade, Y. Lee, J. S. Orcutt, L. Alloatti, M. S. Georgas, A. S. Waterman, J. M. Shainline, R. R. Avizienis, S. Lin, B. R. Moss, R. Kumar, F. Pavanello, A. H. Atabaki, H. M. Cook, A. J. Ou, J. C. Leu, Y.-H. Chen, K. Asanović, R. J. Ram, M. A. Popović, and V. M. Stojanović, "Single-chip microprocessor that communicates directly using light," *Nature*, vol. 528, no. 7583, 2015.

[86] G. Kurian, J. Miller, J. Psota, J. Eastep, J. Liu, J. Michel, L. Kimerling, and A. Agarwal, "ATAC: A 1000-Core Cache-Coherent Processor with On-Chip Optical Network," in *PACT*, 2010.

[87] A. Carpenter, J. Hu, J. Xu, M. Huang, and H. Wu, "A case for globally shared-medium on-chip interconnect," in *ISCA*, 2011.

[88] A. Carpenter, J. Hu, O. Kocabas, M. Huang, and H. Wu, "Enhancing effective throughput for transmission line-based bus," in *ISCA*, 2012.

[89] E. Socher and M.-C. F. Chang, "Can RF Help CMOS Processors?[Topics in Circuits for Communications]," *IEEE Communications Magazine*, vol. 45, no. 8, 2007.

[90] G. Sun, S.-H. Weng, C.-K. Cheng, B. Lin, and L. Zeng, "An on-chip global broadcast network design with equalized transmission lines in the 1024-core era," in *Proceedings of the International Workshop on System Level Interconnect Prediction*, 2012.

[91] J. Oh, M. Prvulovic, and A. Zajic, "Tlsync: support for multiple fast barriers using on-chip transmission lines," in *ISCA*, 2011.

[92] B. M. Beckmann and D. A. Wood, "TLC: Transmission Line Caches," in *MICRO*, 2003.

[93] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: A design alternative for cache on-chip memory in embedded systems," in *CODES*, 2002.

[94] R. Boyapati, J. Huang, P. Majumder, K. H. Yum, and E. J. Kim, "APPROX-NoC: A Data Approximation Framework for Network-On-Chip Architectures," in *ISCA*, 2017.

[95] B. K. Daya, L.-s. Peh, and A. P. Chandrakasan, "Quest for High-Performance Bufferless NoCs with Single-Cycle Express Paths and Self-Learning Throttling," in *DAC*, 2016.

[96] M. Rinard, "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks," in *SC*, 2006.

[97] M. C. Rinard, "Using early phase termination to eliminate load imbalances at barrier synchronization points," in *OOPSLA*, 2007.

[98] J. Meng, S. Chakradhar, and A. Raghunathan, "Best-effort parallel execution framework for recognition and mining applications," in *IPDPS*, 2009.

[99] L. Renganarayana, V. Srinivasan, R. Nair, and D. Prener, "Programming with relaxed synchronization," in *Relax*, 2012.

[100] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke, "Paraprox: pattern-based approximation for data parallel applications," in *ASPLOS*, 2014.

[101] S. Misailovic, D. Kim, and M. Rinard, "Parallelizing sequential programs with statistical accuracy tests," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, p. 88, 2013.

[102] M. Rinard, "Parallel synchronization-free approximate data structure construction," in *HotPar*, 2013.

[103] S. Misailovic, S. Sidiroglou, and M. C. Rinard, "Dancing with uncertainty," in *RACES*, 2012.

[104] S. Campanoni, G. Holloway, G.-Y. Wei, and D. Brooks, "Helix-up: Relaxing program semantics to unleash parallelization," in *CGO*, 2015.

[105] A. Udupa, K. Rajan, and W. Thies, "Alter: Exploiting breakable dependences for parallelization," in *PLDI*, 2011.

[106] E. A. Deiana, V. St-Amour, P. A. Dinda, N. Hardavellas, and S. Campanoni, "Unconventional parallelization of nondeterministic applications," in *ASPLOS*, 2018.

[107] S. K. Khatamifard, I. Akturk, and U. R. Karpuzcu, "On approximate speculative lock elision," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 2, 2018.

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

[109] *AMD Epyc 7742 Processor*, 2019 (accessed April 16, 2020). [Online]. Available: https://www.amd.com/en/products/cpu/amd-epyc-7742

[110] *Intel Xeon Platinum 9282 Processor*, 2019 (accessed April 16, 2020). [Online]. Available: https://www.intel.com/content/www/us/en/products/processors/xeon/scalable/platinum-processors/platinum-9282.html

[111] 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, 2018.

[112] S.-B. Lee, S.-W. Tam, I. Pefkianakis, S. Lu, M.-C. F. Chang, C. Guo, G. Reinman, C. Peng, M. Naik, L. Zhang, and J. Cong, "A scalable micro wireless interconnect structure for CMPs," in *Proceedings of the MOBICOM '09*, 2009, p. 217.

[113] H. Mondal, S. Gade, M. Shamim, S. Deb, and A. Ganguly, "Interference-Aware Wireless Network-on-Chip Architecture using Directional Antennas," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 3, no. 3, pp. 193–205, 2017.

[114] S. Abadal, A. Mestres, and J. Torrellas, *et al.*, "Medium Access Control in Wireless Network-on-Chip: A Context Analysis," *IEEE Communications Magazine*, vol. 56, no. 6, pp. 172–178, 2018.

[115] N. Mansoor and A. Ganguly, "Reconfigurable Wireless Network-on-Chip with a Dynamic Medium Access Mechanism," in *Proceedings of the NoCS '15*, 2015, p. Article 13.

[116] N. Mansoor, S. Shamim, and A. Ganguly, "A Demand-Aware Predictive Dynamic Bandwidth Allocation Mechanism for Wireless Network-on-Chip," in *Proceedings of the SLIP '16*, 2016.

[117] K. Terplan and P. A. Morreale, *The Telecommunications Handbook*. CrC Press, 2018.

[118] V. Soteriou, H. Wang, and L. Peh, "A Statistical Traffic Model for On-Chip Interconnection Networks," in *Proceedings of MASCOTS '06*, 2006, pp. 104–116.

[119] 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," *arXiv preprint arXiv:1901.04291*, 2018. [Online]. Available: https://arxiv.org/pdf/1901.04291.pdf

[120] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of Ethernet traffic," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, 1994.

[121] K. Duraisamy, R. G. Kim, and P. P. Pande, "Enhancing Performance of Wireless NoCs with Distributed MAC Protocols," in *Proceedings of the ISQED '15*, 2015, pp. 406–11.

[122] 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 *Proceedings of the NoCArc '16*, 2016, pp. 21–26.

[123] N. Mansoor, A. Vashist, M. M. Ahmed, M. S. Shamim, S. A. Mamun, and A. Ganguly, "A traffic-aware medium access control mechanism for energy-efficient wireless network-on-chip architectures," *arXiv preprint arXiv:1809.07862*, 2018.

[124] V. D. Kapsalis, S. A. Koubias, and H. C. Haralabidis, "New hybrid mac-layer protocol for real-time bus networks," *IEE Proceedings - Communications*, vol. 141, no. 5, pp. 325–333, Oct 1994.

[125] Tsern-Huei Lee, "Performance analysis of a class of hybrid token-csma/cd protocols," in *Proceedings of the TENCON '91*, Aug 1991.

[126] A. Ephremides and O. Mowafi, "Analysis of a Hybrid Access Scheme for Buffered Users-Probabilistic Time Division," *IEEE Transactions on Software Engineering*, vol. SE-8, no. 1, pp. 52–61, 1982.

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

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

[129] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1988.

[130] A. Gupta, W.-D. Weber, and T. Mowry, "Reducing memory and traffic requirements for scalable directory-based cache coherence schemes," in *Proceedings of the International Conference on Parallel Processing (ICPP)*, 1990.

[131] 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.

[132] 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.

[133] L. M. Censier and P. Feautrier, "A new solution to coherence problems in multicache systems," *IEEE Transactions on Computers*, 1978.

[134] D. Chaiken, J. Kubiatowicz, and A. Agarwal, "LimitLESS directories: A scalable cache coherence scheme," in *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1991.

[135] D. Sanchez and C. Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *Proceedings of the 18th International Symposium on High Performance Computer Architecture (HPCA)*, 2012.

[136] S. J. Eggers and R. H. Katz, "Evaluating the performance of four snooping cache coherency protocols," in *Proceedings of the 16th International Symposium on Computer Architecture (ISCA)*, 1989, pp. 2–15.

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

[138] A. Gupta and W.-D. Weber, "Cache invalidation patterns in shared-memory multiprocessors," *IEEE Transactions on Computers*, 1992.

[139] J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA highly scalable server," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1997.

[140] 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 *Proceedings of the 9th International Workshop on Network on Chip Architectures*, 2016, pp. 21–26.

[141] J. Oh, M. Prvulovic, and A. Zajic, "TLSync: support for multiple fast barriers using on-chip transmission lines," in *Proceedings of the International Symposium on Computer Architecture*, 2011, pp. 105–115.

[142] 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.

[143] 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.

[144] 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.

[145] 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*. IEEE, 2011, pp. 277–285.

[146] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros, "Splash-3: A properly synchronized benchmark suite for contemporary research," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2016, pp. 101–111.

[147] 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.

[148] 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.

[149] 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.

[150] 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.

[151] 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 *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000, pp. 25–36.

[152] M. Martin, M. D. Hill, and D. A. Wood, "Token coherence: Decoupling performance and correctness," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2003, pp. 182–193.

[153] K. Strauss, X. Shen, and J. Torrellas, "Uncorq: Unconstrained snoop request delivery in embedded-ring multiprocessors," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 327–342.

[154] N. Agarwal, L.-S. Peh, and N. K. Jha, "In-Network Snoop Ordering (INSO): Snoopy coherence on unordered interconnects," in *Proceedings of the IEEE 15th International Symposium on High Performance Computer Architecture (HPCA)*, 2009, pp. 67–78.

[155] 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 *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2014, pp. 25–36.

[156] W.-C. Kwon and L.-S. Peh, "A universal ordered NoC design platform for shared-memory MPSoC," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 697–704.

[157] 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 *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017.

[158] D. Vantrease, M. H. Lipasti, and N. Binkert, "Atomic coherence: Leveraging nanophotonics to build race-free cache coherence protocols," in *Proceedings of the IEEE 17th International Symposium on High Performance Computer Architecture (HPCA)*, 2011, pp. 132–143.

[159] 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 *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2010, pp. 477–488.

[160] 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.

[161] A. Carpenter, J. Hu, J. Xu, M. Huang, and H. Wu, "A case for globally shared-medium on-chip interconnect," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2011, pp. 271–282.

[162] A. Carpenter, J. Hu, O. Kocabas, M. Huang, and H. Wu, "Enhancing effective throughput for transmission line-based bus," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2012, pp. 165–176.

[163] 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 *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2013, pp. 309–318.

[164] 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.

[165] B. M. Beckmann and D. A. Wood, "TLC: Transmission line caches," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* IEEE, 2003, pp. 43–54.

[166] 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.

[167] Y. Fu, T. M. Nguyen, and D. Wentzlaff, "Coherence domain restriction on large scale systems," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015, pp. 686–698.

[168] A. Kannan, N. E. Jerger, and G. H. Loh, "Enabling interposer-based disintegration of multi-core processors," in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2015, pp. 546–558.

[169] S. Naffziger, N. Beck, T. Burd, K. Lepak, G. H. Loh, M. Subramony, and S. White, "Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[170] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-chip-module GPUs for continued performance scalability," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 320–332, 2017.

[171] N. E. Jerger, A. Kannan, Z. Li, and G. H. Loh, "Noc architectures for silicon interposer systems: Why pay for more wires when you can get them (from your interposer) for free?" in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014, pp. 458–470.

[172] A. Kannan, N. E. Jerger, and G. H. Loh, "Exploiting interposer technologies to disintegrate and reintegrate multicore processors," *Ieee Micro*, vol. 36, no. 3, pp. 84–93, 2016.

[173] G. H. Loh, N. E. Jerger, A. Kannan, and Y. Eckert, "Interconnect-memory challenges for multi-chip, silicon interposer systems," in *Proceedings of the 2015 international symposium on Memory Systems*, 2015, pp. 3–10.

[174] H. Braunisch, A. Aleksov, S. Lotz, and J. Swan, "High-speed performance of silicon bridge die-to-die interconnects," in *2011 IEEE 20th Conference on Electrical Performance of Electronic Packaging and Systems*. IEEE, 2011, pp. 95–98.

[175] R. Mahajan, R. Sankman, N. Patel, D.-W. Kim, K. Aygun, Z. Qian, Y. Mekonnen, I. Salama, S. Sharan, D. Iyengar, and D. Mallik, "Embedded multi-die interconnect bridge (EMIB)–a high density, high bandwidth packaging interconnect," in *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*. IEEE, 2016, pp. 557–565.

[176] *AMD Infinity Architecture Technology*, 2019 (accessed July 12, 2021). [Online]. Available: https://www.amd.com/en/technologies/infinity-architecture

[177] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. Tell, Y. Zhang, W. Dally, J. Emer, C. Gray, B. Khailany, and S. Keckler, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[178] *Intel Ponte Vecchio Xe-HPC GPGPU*, 2019 (accessed July 12, 2021). [Online]. Available: https://newsroom.intel.com/news-releases/intel-unveils-new-gpu-architecture-optimized-for-hpc-ai-oneapi/

[179] A. Franques, A. Kokolis, S. Abadal, V. Fernando, S. Misailovic, and J. Torrellas, "WiDir: A wireless-enabled directory cache coherence protocol," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 304–317.

[180] R. Guirado, H. Kwon, S. Abadal, E. Alarcón, and T. Krishna, "Dataflow-architecture co-design for 2.5D DNN accelerators using wireless network-on-package," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2021, pp. 806–812.

[181] X. Yu, J. Baylon, P. Wettin, D. Heo, P. P. Pande, and S. Mirabbasi, "Architecture and design of multichannel millimeter-wave wireless NoC," *IEEE Design & Test*, vol. 31, no. 6, pp. 19–28, 2014.

[182] K. K. Tokgoz, S. Maki, J. Pang, N. Nagashima, I. Abdo, S. Kawai, T. Fujimura, Y. Kawano, T. Suzuki, T. Iwai et al., "A 120Gb/s 16QAM CMOS millimeter-wave wireless transceiver," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 168–170.

[183] *Thin & Light & High Performance Graphics*, Hot Chips 2018 (accessed August 12, 2021). [Online]. Available: https://old.hotchips.org/hc30/1conf/1.04_Intel_Thin_Light_Gaming_HotChips_SC_Final.pdf

[184] B. Zimmer, R. Venkatesan, Y. S. Shao, J. Clemons, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. Tell, Y. Zhang, W. Dally, J. Emer, C. Gray, S. Keckler, and B. Khailany, "A 0.32–128 TOPS, scalable multi-chip-module-based deep neural network inference accelerator with ground-referenced signaling in 16 nm," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 4, pp. 920–932, 2020.

[185] S. S. Iyer, "Heterogeneous integration for performance and scaling," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 6, no. 7, pp. 973–982, 2016.

[186] P. Fotouhi, S. Werner, J. Lowe-Power, and S. B. Yoo, "Enabling scalable chiplet-based uniform memory architectures with silicon photonics," in *Proceedings of the International Symposium on Memory Systems*, 2019, pp. 222–334.

[187] J. Kim, G. Murali, H. Park, E. Qin, H. Kwon, V. C. K. Chekuri, N. M. Rahman, N. Dasari, A. Singh, M. Lee, H. Torun, K. Roy, M. Swaminathan, S. Mukhopadhyay, T. Krishna, and S. Lim, "Architecture, chip, and package codesign flow for interposer-based 2.5-D chiplet integration enabling heterogeneous IP reuse," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 11, pp. 2424–2437, 2020.

[188] C. Liu, J. Botimer, and Z. Zhang, "A 256Gb/s/mm-shoreline AIB-Compatible 16nm FinFET CMOS Chiplet for 2.5 D Integration with Stratix 10 FPGA on EMIB and Tiling on Silicon Interposer," in *2021 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2021, pp. 1–2.

[189] M. Ahmed, N. Mansoor, and A. Ganguly, "An asymmetric, one-to-many traffic-aware mm-wave wireless interconnection architecture for multichip systems," *IEEE Transactions on Emerging Topics in Computing*, 2020.

[190] A. Gupta, W.-D. Weber, and T. Mowry, "Reducing memory and traffic requirements for scalable directory-based cache coherence schemes," in *Scalable shared memory multiprocessors*. Springer, 1992, pp. 167–192.

[191] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos, "A tagless coherence directory," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009, pp. 423–434.

[192] W. Gropp, "Mpi 3 and beyond: Why mpi is successful and what challenges it faces," in *Recent Advances in the Message Passing Interface*, J. L. Träff, S. Benkner, and J. J. Dongarra, Eds., 2012.

[193] M. S. Shamim, N. Mansoor, R. S. Narde, V. Kothandapani, A. Ganguly, and J. Venkataraman, "A wireless interconnection framework for seamless inter and intra-chip communication in multichip systems," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 389–402, 2017.

[194] A. Prodromou, M. Meswani, N. Jayasena, G. Loh, and D. M. Tullsen, "Mempod: A clustered architecture for efficient and scalable migration in flat address space multi-level memories," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 433–444.

[195] N. Weissman, S. Jameson, and E. Socher, "An F-band dual band 12+12Gbps packaged CMOS bi-directional transceiver," in *2016 IEEE MTT-S International Microwave Symposium (IMS)*. IEEE, 2016, pp. 1–4.

[196] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for scalable synchronization on shared-memory multiprocessors," *ACM Transactions on Computer Systems (TOCS)*, vol. 9, no. 1, pp. 21–65, 1991.

[197] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[198] G. Hendry, J. J. Wilke, J. P. Kenny, K. Sargsyan, M. Drohmann, and B. A. Allan, "Sst/macro: A coarse-grained simulation workflow." Sandia National Lab.(SNL-CA), Livermore, CA (United States), Tech. Rep., 2013.

[199] S. Ooms and P. Reynaert, "A 120-GHz wireless link using 3D-printed lens with flexible dielectric fiber feed and 28-nm CMOS transceiver," *IEEE Solid-State Circuits Letters*, vol. 3, pp. 142–145, 2020.

[200] S. H. Kim, T. H. Jang, D. M. Kang, C. J. Lee, H. S. Son, and C. S. Park, "A 120 GHz wireless radio link for high-speed chip-to-chip communication," in *2019 IEEE Asia-Pacific Microwave Conference (APMC)*. IEEE, 2019, pp. 375–377.

[201] A. Townley, N. Baniasadi, S. Krishnamurthy, C. Sideris, A. Hajimiri, E. Alon, and A. Niknejad, "A fully integrated, dual channel, flip chip packaged 113 GHz transceiver in 28nm CMOS supporting an 80 Gb/s wireless link," in *2020 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2020.

[202] Y. Kim, B. Hu, Y. Du, A. Tang, H.-N. Chen, C. Jou, J. Cong, T. Itoh, and M.-C. F. Chang, "A 20Gb/s 79.5 mW 127GHz CMOS transceiver with digitally pre-distorted PAM-4 modulation for contactless communications," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 278–280.

[203] Z. Wang, P. Y. Chiang, P. Nazari, C. C. Wang, Z. Chen, and P. Heydari, "A CMOS 210-GHz fundamental transceiver with OOK modulation," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 3, pp. 564–580, 2014.

[204] C. Jianxun, S. Chauvin, and T. El-Ghazawi, "Mpi-based sobel edge detection," 2000, https://upc.lbl.gov/download/dist/upc-tests/benchmarks/gwu_bench/sobel/MPI/.

[205] "Mpi-k-means-clustering," https://github.com/rexdwyer/MPI-K-means-clustering.

[206] "mpi-histogram," 2016, https://github.com/retrohacker/mpi-histogram.

[207] J. Burkardt, "C code which approximates an integral using a quadrature rule." 2010, https://people.sc.fsu.edu/ jburkardt/c_src/quad_mpi/quad_mpi.html.

[208] J. Burkardt, "Circuit satisfiability using mpi," 2016, https://people.sc.fsu.edu/ jburkardt/c_src/satisfy_mpi/satisfy_mpi.html.

[209] P. Vivet, E. Guthmuller, Y. Thonnart, G. Pillonnet, G. Moritz, I. Miro-Panades, C. Fuguet, J. Durupt, C. Bernard, D. Varreau, J. Pontes, S. Thuries, D. Coriat, M. Harrand, D. Dutoit, D. Lattard, L. Arnaud, J. Charbonnier, P. Coudrain, A. Garnier, F. Berger, A. Gueugnot, A. Greiner, Q. Meunier, A. Farcy, A. Arriordaz, S. CHeramy, and F. Clermidy, "A 220GOPS 96-Core Processor with 6 Chiplets 3D-Stacked on an Active Interposer Offering 0.6 ns/mm Latency, 3Tb/s/mm 2 Inter-Chiplet Interconnects and 156mW/mm 2@ 82%-Peak-Efficiency DC-DC Converters," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 46–48.

[210] J. Yin, Z. Lin, O. Kayiran, M. Poremba, M. S. B. Altaf, N. E. Jerger, and G. H. Loh, "Modular routing design for chiplet-based systems," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 726–738.

[211] D. Greenhill, R. Ho, D. Lewis, H. Schmit, K. H. Chan, A. Tong, S. Atsatt, D. How, P. McElheny, K. Duwel, J. Schulz, D. Daulkner, G. Iyer, G. Chen, H. Kong Phoon, H. Wooi Lim, W.-Y. Koay, and T. Garibay, "A 14nm 1GHz FPGA with 2.5 D transceiver integration," in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 2017, pp. 54–55.

[212] Z. Tan, H. Cai, R. Dong, and K. Ma, "NN-Baton: DNN workload orchestration and chiplet granularity exploration for multichip accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1013–1026.

[213] G. Ascia, V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Improving inference latency and energy of DNNs through wireless enabled multi-chip-module-based architectures and model parameters compression," in *2020 14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2020, pp. 1–6.

[214] S. Bharadwaj, J. Yin, B. Beckmann, and T. Krishna, "Kite: a family of heterogeneous interposer topologies enabled via accurate interconnect modeling," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[215] D. Stow, I. Akgun, and Y. Xie, "Investigation of cost-optimal network-on-chip for passive and active interposer systems," in *2019 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*. IEEE, 2019, pp. 1–8.

[216] M. Wade, E. Anderson, S. Ardalan, P. Bhargava, S. Buchbinder, M. L. Davenport, J. Fini, H. Lu, C. Li, R. Meade, M. Ramamurthy, Chandruand Rust, F. Sedgwick, V. Stojanovic, D. Van Orden, C. Zhang, C. Sun, S. Shumarayev, C. O'Keeffe, T. Hoang, D. Kehlet, R. Mahajan, M. Guzy, C. Allen, and T. Tran, "TeraPHY: A chiplet technology for low-power, high-bandwidth in-package optical I/O," *IEEE Micro*, vol. 40, no. 2, pp. 63–71, 2020.

[217] N. C. Abrams, Q. Cheng, M. Glick, M. Jezzini, P. Morrissey, P. O'Brien, and K. Bergman, "Silicon photonic 2.5 d multi-chip module transceiver for high-performance data centers," *Journal of Lightwave Technology*, vol. 38, no. 13, pp. 3346–3357, 2020.

[218] B. Yu, Y. Ye, X. Ding, Y. Liu, Z. Xu, X. Liu, and Q. J. Gu, "Ortho-mode sub-thz interconnect channel for planar chip-to-chip communications," *IEEE Transactions on Microwave Theory and Techniques*, vol. 66, no. 4, pp. 1864–1873, 2017.

[219] J. W. Holloway, L. Boglione, T. M. Hancock, and R. Han, "A fully integrated broadband sub-mmwave chip-to-chip interconnect," *IEEE Transactions on Microwave Theory and Techniques*, vol. 65, no. 7, pp. 2373–2386, 2017.