Evaluating Chiplet-based Large-Scale Interconnection Networks via Cycle-Accurate Packet-Parallel Simulation

Yinxiao Feng and Yuchen Wei, *Institute for Interdisciplinary Information Sciences, Tsinghua University;* Dong Xiang, *School of Software, Tsinghua University;* Kaisheng Ma, *Institute for Interdisciplinary Information Sciences, Tsinghua University*

https://www.usenix.org/conference/atc24/presentation/feng-yinxiao

This paper is included in the Proceedings of the 2024 USENIX Annual Technical Conference.

July 10–12, 2024 • Santa Clara, CA, USA

978-1-939133-41-0

Open access to the Proceedings of the 2024 USENIX Annual Technical Conference is sponsored by

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

Evaluating Chiplet-based Large-Scale Interconnection Networks via Cycle-Accurate Packet-Parallel Simulation

Yinxiao Feng[1], Yuchen Wei[1], Dong Xiang[2], Kaisheng Ma[1]*
[1]*Institute for Interdisciplinary Information Sciences,* [2]*School of Software, Tsinghua University*

Abstract

The *Chiplet* architecture has achieved great success in recent years. However, chiplet-based networks are significantly different from traditional networks, thus presenting new challenges in evaluation. On the one hand, on-chiplet and off-chiplet networks are tightly coupled; therefore, the entire heterogeneous network must be designed and evaluated jointly rather than separately. On the other hand, existing network simulators cannot efficiently evaluate large-scale chiplet-based networks with cycle-accurate accuracy.

In this paper, we present the design and implementation of the *Chiplet Network Simulator (CNSim)*, a cycle-accurate packet-parallel simulator supporting efficient simulation for large-scale chiplet-based (shared-memory) networks. In *CNSim*, a packet-centric simulation architecture and an atomic-based hyper-threading mechanism are adopted, accelerating simulation speed by $11\times \sim 14\times$ compared with existing cycle-accurate simulators. Besides, we implement the heterogeneous router/link microarchitecture and many other features, including hierarchical topologies, adaptive routing, and real workload traces integration. Based on *CNSim*, two typical chiplet-based networks, which cannot be efficiently simulated by existing simulators, are systematically evaluated. The advantages and limitations of chiplet-based networks are revealed through systematical cycle-accurate simulations. The simulator and evaluation framework are open-sourced to the community[§].

1 Introduction

Chiplet is an emerging architecture that has achieved great success in modern computing systems. By integrating multiple silicon dies in a package, the entire computing density and efficiency are greatly improved [34,56]. However, chiplet-based systems are significantly different from traditional systems, and the chiplet interconnection architecture has not been thoroughly evaluated. Critical design issues to be evaluated include but are not limited to: **1)** hierarchical topologies of the on-chiplet and off-chiplet networks, **2)** routing algorithms on the heterogeneous networks [30], **3)** impacts of the heterogeneous chiplet-to-chiplet interfaces and heterogeneous routers [29], and **4)** overall performance of various topologies under various workloads. There are two main challenges in evaluating chiplet-based interconnection networks.

Challenge 1. *Unified simulations on the entire network are necessary for evaluating chiplet interconnection architecture.* Traditional on-chip and off-chip networks are usually separately designed. On-chip networks are connected to a switch, and the off-chip network is constructed among multiple switches regardless of the on-chip network architecture. However, the *Chiplet* architecture integrates multiple silicon dies with ultra-high density and connectivity, breaking the boundary between on-chip and off-chip networks. Multiple on-chip routers of chiplets are directly connected by low-latency off-chip links, forming large-scale heterogeneous networks and leading to potential problems: **1)** Networks-of-chiplet cannot be separately evaluated since they are tightly coupled with on-chiplet networks [60]. **2)** The scale of chiplet-based networks can be much larger than a traditional on-chip network [23]. **3)** The on/off-chip links and routers are heterogeneous in mechanism, bandwidth, and latency, which are not uniformly and accurately modeled in existing tools [29].

Challenge 2. *Existing network simulators are inefficient for evaluting large-scale chiplet-based shared-memory networks.* On-chip and off-chip networks have been two distinct areas, each with abundant existing evaluation tools. Due to the latency of traditional off-chip links and switches, *e.g. Ethernet* and *InfiniBand*, is usually in the order of microsecond (*us*) [42,53], a coarse-grained event-based simulator is sufficient for evaluating traditional off-chip networks. However, low-latency on-chip routers in multi-chiplet systems are directly connected by low-latency links of nanoseconds (*ns*), *e.g. UCIe* [6]. Besides, circuit-level microarchitecture features, including buffer, link, switch, and flow-control, significantly impact the performance of chiplet-based systems. Therefore,

*Corresponding author
§https://github.com/Yinxiao-Feng/chiplet-network-sim

it is necessary to use fine-grained cycle-accurate simulators to evaluate chiplet-based networks. However, existing cycle-accurate tools are inefficient for large-scale chiplet-based networks because they are designed for small-scale on-chip networks [9, 13, 21, 40], whose scale is no more than tens of routers. Existing parallel network simulators are designed for MPI-based distributed-memory systems [20, 52]; thus, they are also unsuitable for chiplet-based shared-memory systems [15].

Existing cycle-accurate network simulators are slow for two main reasons: **1)** The simulator models the entire RTL router, including many noncritical circuit behaviors, *e.g.* handshake and the *round-robin* allocation. **2)** Complex resource dependency and competition (thread synchronization) make network simulation challenging to parallelize. Therefore, for the first reason, we are motivated to compromise some unnecessary modeling to accelerate simulation speed while verifying necessary microarchitectures and maintaining cycle accuracy. For the second reason, we adopted an atomic-based hyper-threading mechanism to realize efficient packet-parallel simulation with little inconsistency. As a result, we present *Chiplet Network Simulator (CNSim)*, a cycle-accurate packet-parallel network simulator that helps us to evaluate the chiplet interconnection architecture systematically and efficiently. The contributions of this paper are summarized as follows:

- Unlike existing small-scale on-chip or large-scale distributed network simulators, *CNSim* is designed for large-scale chiplet-based (shared-memory) networks.
- A packet-centric simulation architecture and an atomic-based hyper-threading mechanism are adopted. Simulation speed is $11\times \sim 14\times$ faster than existing tools while verifying necessary microarchitectures and maintaining cycle accuracy.
- Various features for evaluating chiplet-based networks are supported, including heterogeneous router/link, hierarchical topology, adaptive routing, and real workload traces integration. The simulator and the evaluation framework are open-sourced to the community.
- Based on *CNSim*, two typical chiplet-based networks, which cannot be efficiently simulated by existing simulators, are systematically evaluated. Extensive evaluations reveal the advantages and limitations of the chiplet interconnection architecture.

2 Background & Motivation

2.1 Chiplet Interconnection Architecture

The traditional chip is implemented on a single silicon die, whose area is very limited due to manufacturing and yield issues [3]. Besides, each chip has a limited number of I/O ports

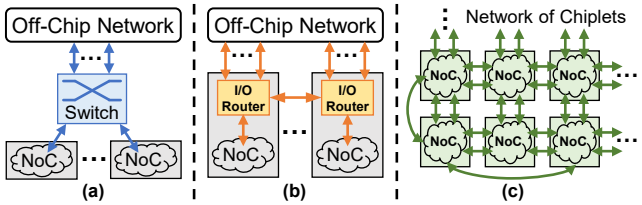

Figure 1: Comparison of different interconnection architectures. (a) On/off-chip networks are isolated by switches. (b) On/off-chip networks are isolated by I/O routers. (c) On/off-chip networks in chiplet-based systems are tightly coupled.

with limited bandwidth, which leads to low off-chip connectivity. In recent years, advanced packaging and high-speed wireline technologies have significantly progressed, thus allowing chips to be integrated at ultra-high scale and density [27]. For example, by using *Integrated-Fan-Out-System-on-Wafer (InFO-SoW)* [24], the *Telsa DOJO* integrates 25 645 mm^2 *D1* chips on a wafer [32], and the total off-chip bandwidth reaches 1576 Tb/s (576 lanes of 112G SerDes per chip) [54, 55]. The *Chiplet* technologies significantly impact the interconnection architecture of large-scale systems.

Relation between on-chip and off-chip networks: In traditional interconnection systems, on-chip and off-chip networks are usually decoupled. As shown in Figure 1(a), on-chip networks are connected to a switch, and the off-chip network is constructed among multiple switches. Most switch-based datacenter networks, including *Fat-tree* [12, 59], *Slingshot* [26, 43], and *NVIDIA HGX* [39], adopt this architecture; therefore, the overall network performance can be measured among the network interface controllers (NICs) regardless of the network-on-chip (NoC). Besides, some high-performance computing (HPC) interconnection networks adopt direct switch-less topologies, *e.g. Torus*. As shown in Figure 1(b), most of them, including *TPUv4* [41] and *TofuD* [10], implement an I/O router to centralize all interconnection channels, thus also isolating on-chip and off-chip networks. However, emerging advanced packaging technologies allow ultra-high-density integration and connectivity of multiple chips, thus breaking the on/off-chip boundary. As shown in Figure 1(c), multiple on-chiplet networks are tightly connected by numerous physical channels. As a result, the on-chip and off-chip networks of chiplet-based systems must be designed and evaluated jointly rather than separately.

Throughput: An obvious limitation of traditional interconnection architectures is that the NICs or I/O routers are the bottleneck of the network. The traffic that can be injected into the off-chip network is much less than the total throughput of the on-chip network. For example, two servers (processors) are connected to a 64-port 400G switch, with a total switching bandwidth of 25.6Tb/s; however, the communication bandwidth between the two servers is only 400G. For some workloads (*e.g.* AI), the local throughput is more important than the global throughput [38]. By advanced packaging

and high-speed wireline technologies, the chip itself can provide communication bandwidth no weaker than a regular switch [32, 38]; thus, the injection/ejection bandwidth of each chiplet is no longer bounded by the centralized NIC or I/O router, significantly improving the network scalability and throughput.

Latency: The latency of chiplet components is much lower than the traditional off-chip switches and links, *e.g.* Ethernet [53] and InfiniBand [42], whose latency are more than hundreds of nanoseconds (*us*-level). However, the latency of typical on-chip routers and chiplet-to-chiplet interfaces, *e.g.* UCIe [6], is only a few nanoseconds (cycle-level). As a result, the microarchitecture of on-chip routers can significantly affect the overall performance of chiplet-based systems. On the other hand, the low-latency die-to-die links make replacing a costly switch-to-switch hop with multiple low-cost chiplet-to-chiplet hops possible, which makes the chiplet-based large-scale network have a diameter of numerous hops. Compared with traditional switch-based high-radix networks, chiplet-based networks require cycle-accurate modeling to evaluate the overall performance; however, such fine-grained simulations of large-scale networks can be slow.

Shared-memory vs. distributed-memory: In a shared-memory architecture, multiple processors or cores share a common address space, while in distributed-memory architecture, each processor has its own private memory space. Shared-memory architecture is programming-friendly, but the hardware-based coherence protocol has high requirements for network performance; thus, current large-scale systems are usually distributed-memory-based. However, the high connectivity and low latency of chiplet architecture make large-scale chiplet-based systems adopting the shared-memory architecture possible [34]. As a result, the MPI-based parallelism mechanism, used in most distributed-memory systems, is unsuitable for chiplet-based systems. The shared-memory architecture presents simulation challenges for large-scale chiplet-based networks due to the lack of a parallel simulation framework.

2.2 Network Simulators

Various existing network simulators can be categorized into *cycle-based* and *discrete-event* according to the simulation mechanism. The **cycle-based simulator** updates the states of all components in each cycle based on the states of the last cycle. The popular *BookSim* [40] and other cycle-accurate simulators [21, 46] adopt such a design because it is highly compatible to the actual implementation of routers. Cycle-based simulators are capable of modeling fine-grained microarchitectures but are not qualified for evaluating large-scale distributed systems without a synchronous clock. The **discrete-event simulator** is a more mainstream approach for evaluation large-scale networks [4, 9, 20, 50, 52]. Any change in discrete-event simulators is identified as an event, and the simulator processes every event along the timeline. The discrete-event simulator can also achieve cycle-accurate [13] by generating events at cycle-granularity, but the simulation can be slow due to the serial processing of the large event queue.

Parallel Simulation: Since traditional on-chip networks are scale-limited, existing cycle-accurate simulators pay little attention to simulation performance. In fact, they are already quite time-consuming for on-chip networks (*e.g.*, it takes tens of hours to simulate the entire *PARSEC* traces [36, 37]), let alone for large-scale networks. For large-scale distributed networks, almost all simulators are based on parallel discrete-event simulation (PDES) [33]. The system is partitioned into separate simulation objects, each with its own event queue and *Logical Process (LP)*. Necessary synchronizations are performed among different LPs to guarantee the distributed events are processed appropriately. In most existing parallel discrete-event simulators, including *SST* [52], *ROSS/CODES* [20, 49], *NS-3* [50], and *OMNeT++* [4], systems are partitioned into MPI ranks/nodes, and the synchronization is achieved by MPI communication. However, the distributed-memory-based clusters are unavailable to most researchers [15], and the MPI-based mechanism is unsuitable for chiplet-based shared-memory systems. Therefore, hyper-threading is more suitable for parallel-simulating large-scale chiplet-based networks. As with all multithreaded programs, the hyper-threading bottleneck is the program's parallelizability and the synchronization overhead between threads. Packets and resources in a network are tightly dependent and associated, thus challenging to parallelize.

Parallelism strategy: Numerous components in a network can be parallelized in different ways. The *network parallelism*, in which multiple threads concurrently process multiple sub-networks/routers, is commonly used [46]. Although *network parallelism* is straightforward, we observe two major limitations. First, each router (thread) has resource dependency and contention with all adjacent routers, which leads to fierce thread competition. Second, multiple physical/virtual channels with multiple packets in a router are serially processed, which limits the parallelism for high-radix and large-buffer routers. Therefore, we are motivated to propose a new strategy, called *packet parallelism*, promising to achieve more efficient parallel simulation.

2.3 Motivations from Profiling BookSim

For real circuit-level routers, every pipeline stage is finished in one cycle, no matter whether it is critical or trivial. But for simulators, the runtime of different stages can vary greatly. The simulation model of an unimportant circuit stage can cost a lot of time. Besides, the computer's memory capacity for running the simulation is finite; therefore, simulators can't fully replicate the hardware mechanism, *e.g.* the routing table, which leads to potential simulation overheads. As a result, the

Table 1: Profiling results, measured by cycles of function calls, of the *BookSim* on the 2D-mesh and dragonfly topologies.

Cycle Estimation (%)	2D-Mesh	Dragonfly
`IQRouter::_internalStep()`	92.18	71.92
Call: `_VCAllocEvaluate()`	48.08	17.65
Call: `SparseAllocator::Clear()`	28.86	17.93
Call: `_SWAllocEvaluate()`	5.71	13.43
Call: `_SWAllocUpdate()`	4.59	11.26

simulation runtime and circuit runtime are disproportionate. To fully demonstrate why existing cycle-accurate simulators are slow, we profile the *BookSim* as an example. The *Valgrind Callgrind* framework [5], which can count the execution cycles of function calls, is used for profiling.

As shown in Table 1, the router status update function `IQRouter::_InternalStep()`, which is called every simulation-cycle by every router, takes up the majority of the runtime. Further decomposition shows that most of the runtime is spent processing resource allocation, including the virtual channel and switch. In the process, a request mapping table `vector<vector<sRequest>>` for "requester-provider" pairs is maintained to determine the allocation of resources in a *round-robin* way. The frequent indexing, erasing, and inserting of the table significantly affect the simulation performance. Specifically, more than 70% of the time is spent evaluating the allocation behavior, which is an already thoroughly discussed issue. As a matter of fact, different allocation policies, including *round-robin* and *first-come-first-serve (FCFS)*, have advantages and disadvantages, but they do not seriously affect the overall performance because either policy fully utilizes the physical channel [7, 31].

Since allocation policy is not a significant issue to be evaluated, saving related simulation times for evaluating large-scale chiplet-based networks is necessary. Therefore, we are motivated to use an implementation-friendly allocation policy and a more efficient simulation mechanism. In brief, the simulation speed can be significantly improved if the available resources can be immediately allocated to the requester without gathering and managing all the requests. If all the requests are processed in the coming order, the *first-come-first-serve* policy is naturally achieved. Maintaining numerous coming-order queues is difficult, but we can easily maintain a packet queue based on the injection order, *i.e.* the *first-inject-first-serve* policy.

3 CNSim: A Cycle-Accurate Packet-Parallel Simulator for Chiplet-based Networks

The *Chiplet Network Simulator (CNSim)* is a cycle-accurate simulator designed for large-scale chiplet-based (shared-memory) interconnection networks but also applies to traditional on/off-chip networks. This paper mainly focuses on the novel features of *CNSim*; other features can be found in the source code and appendix.

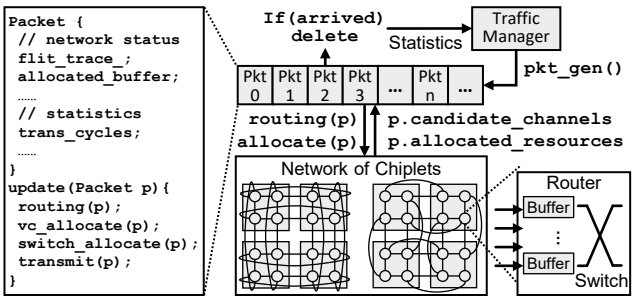

Figure 2: Overview of the packet-centric *CNSim*. The key status values are stored in `Packet` rather than in the network.

Algorithm 1 PROCESSING PACKET

Input: The packet p;

```
1:  if p.candidate_channels is EMPTY then
2:      ROUTING(p);
3:  else if p.allocated_buffer is NULL then
4:      VC_ALLOCATION(p);
5:  else if p.switch_allocated is FALSE then
6:      SWITCH_ALLOCATION(p);
7:  end if
8:  if p.switch_allocated is TRUE then
9:      LINK_TRAVERSAL(p.HEAD);
10: end if
11: Transmit all body flits one step forward.
```

3.1 Packet-Centric Simulation

In cycle-based simulators, all state elements of the network, including buffer/link usages, requests, and routing/allocation results, are updated in each cycle. Such a mechanism is compatible with the circuit implementation and implies a natural clock-level synchronization. In contrast, an event queue is maintained in discrete-event simulators, and the simulator processes discrete events along the timeline, which is more general and flexible. As shown in Figure 2, we present a new *packet-centric* architecture, combining the advantages of cycle-based and discrete-event simulators. A packet queue is maintained based on the injection time, and all packets are updated every cycle sequentially. The status values stored in each packet include the routing results, the allocated buffer, the switch allocation status, the trace of each flit, and necessary statistical information. As shown in Algorithm 1, each packet is updated once per cycle according to the network status. Each update is equivalent to processing a series of events in the discrete-event simulator, including routing, VC allocation, switch allocation, link traversal, and flit transmission.

As introduced in § 2.3, the allocation modeling is supposed to be improved. The *packet-centric* architecture can directly and efficiently achieve the *first-inject-first-serve (FIFS)* pol-

icy. As shown in Figure 2, packets injected by the *traffic manager* are appended to the end of the packet container. If a packet reaches the destination, it is removed without affecting the order of the remaining packets. Since the single-thread simulator processes the packet queue sequentially in each cycle, if the available resources are directly allocated to the request, *FIFS* allocation policy is naturally achieved. With such a mechanism, the costly management of the complicated request-resource mapping is eliminated, and the simulation speed can be significantly improved. Besides the *FIFS* policy, *FCFS* can also be easily implemented by maintaining a packet queue per router.

However, utterly real-time resource management can lead to process-order-induced deviations. For example, suppose an earlier packet directly releases the resources it occupies (*e.g.* physical link). In that case, the later packet may successfully acquire the resource in the same cycle, which violates the hardware mechanism. Therefore, we still record the critical resources (*e.g.* physical link) and update the status of these resources after all packets have been processed. Such a scheme can be regarded as cycle-level synchronization, which is necessary to ensure the simulation accuracy. The packet-centric architecture and the cycle-level synchronization are compatible with the hyper-threading simulation.

3.2 Packet Parallelism Simulation

Concurrency can significantly accelerate the simulation; however, it is not easy for simulators to achieve efficient parallel simulation. As discussed in § 2.2, the commonly-used *network-parallelism* has limitations; thus, we present a new *packet-parallelism* to process packets concurrently.

As shown in Figure 3, the packet queue is regarded as a workload pool shared by all worker threads. Each worker thread sequentially fetches one or a few packets from the queue for processing. After a round of parallel processing, status updates have been completed for most of the packets, but some packets that modify critical resources, including buffers and links, are tagged with special status. Then, the main thread handles all necessary synchronization, including updating the link status and removing finished packets. One major advantage of *packet parallelism* is that a packet is less likely to compete for resources with other packets. For example, only the head packet of an input queue competes with head packets of other input queues in the same router. Therefore, the overheads and deviations caused by hyper-threading can be negligible.

Atomic-based parallel programming. Synchronization is the most important but performance-costly operation for multithreading programs. Traditional parallel simulators [46] manually handle thread contentions using *locks* in any critical section. For example, if *packet parallel* is used, the link/switch allocation must wait until all packet requests of all threads have been collected to determine the final result. Such synchronization seriously affects the performance of parallel simulation. Therefore, we adopt an atomic-based mechanism that sacrifices some allocation consistency to avoid frequent synchronization and accelerate parallel simulation. Shared resources, including buffers and links, are set to atomic variables, and the first packet of a thread that successfully modifies the atomic variable wins the contention. We let the thread contention decide the allocation of resources instead of the manual handling logic. As shown in Figure 3, the switch allocation status is an atomic variable and is directly allocated to the first thread acquiring regardless of the requests of other threads.

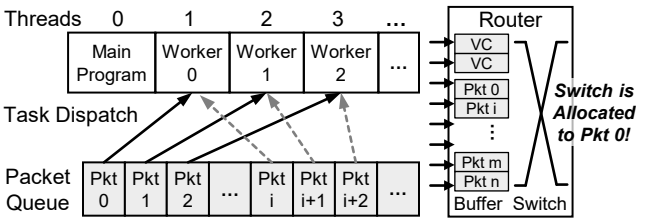

Figure 3: The packet-parallel scheme. Each worker fetches a packet from the queue for processing sequentially. The earlier processed packet directly gets the allocation regardless of later requests, *i.e.* approximate *first-inject-first-serve*.

Multi-thread inconsistency. Compared with the single-thread simulation, the multi-thread program no longer follow the strict first-injected-first-served policy but an approximate one with multi-thread randomness that later-injected packets may win the allocation in the competition with earlier-injected packets. Besides, the uncertain acquiring/releasing order for the buffer can also lead to one-flit wasted buffer space. However, since all packets are dispatched to workers (threads) in injection order, early packets are still more likely to win the contention (detailed probability estimation is presented in Appendix D). Validation experiments in § 3.5.3 also show negligible inconsistency after parallelism. *Negligible* is defined as the average packet latency deviations caused by multi-threading are smaller than the deviations of different random seeds for the traffic pattern, which is measured at 0.5%. Another potential problem is that such a mechanism may lead to starvation and affect performance. However, permanent starvation is impossible because the thread contention is random.

The *packet parallelism* can be combined with the *network parallelism*. For the *first-come-first-serve* allocation policy, each router maintains a packet queue, and workers can process packets in different queues concurrently. When too many threads are in parallel, competition among workers for the next workload (packet) in each queue becomes intense. To prevent the head-packet competition from becoming the bottleneck, each worker can fetch multiple packets from the queue every time. However, such a "multiple-issue" scheme may lead to worse allocation inconsistency (also discussed in Appendix D); thus, we make validation experiments § 3.5.3

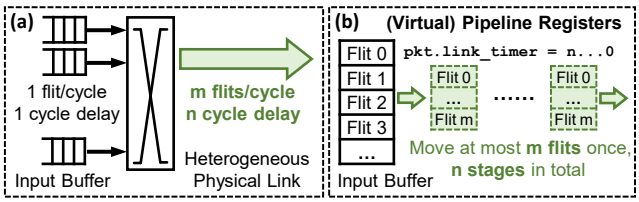

Figure 4: Unified modeling of the heterogeneous routers and links.

to verify the efficiency-accuracy balance.

3.3 Heterogeneous Router & Link

In traditional on-chip networks, all links are uniform (one flit/cycle); therefore, all routers and links in existing cycle-accurate network simulators are uniformly configured with the same latency and bandwidth. However, the inter-chiplet links are different from the on-chip links, and a chiplet can have multiple types of off-chiplet interfaces [29]. Besides, due to the heterogeneity of links, the routers on the chiplet are also heterogeneous. Heterogeneity is a significant feature that all existing cycle-accurate simulators lack modeling.

In *CNSim*, all different kinds of routers and links are built by a uniform model, and objects are configured individually in the simulation. As shown in Figure 4, the *multi-width FIFO* and *virtual pipeline stages* are implemented to model heterogeneous routers and links uniformly. For a link of m flits/cycle bandwidth and n cycle latency, at most m flits at the head of the input buffer can be transmitted to the next router. It takes n virtual pipeline stages in total to finish the transmission. Non-integer delays are directly approximated as integers, and non-integer bandwidths can be achieved by alternately passing integer numbers of flits, *e.g.* 1.5 flits/cycle bandwidth can be achieved by passing 2 flit in the first cycle and 1 flits in the next cycle. Besides, *CNSim* also compatible to the two newly proposed heterogeneous interface implementations, *hetero-PHY* and *hetero-channel* [29].

3.4 Other Notable Features & Compromises

Eliminate unimportant pipeline stages: As shown in Figure 5, a typical cycle-accurate network simulator usually models all circuit-level pipelines of routers, including input queuing (IQ), routing computation (RC), virtual channel allocation (VA), switch allocation (SA), switch traversal (ST), and link traversal (LT). Among these stages, what really affects the overall performance are those competing for resources and causing pipeline stalls. For example, the switch allocation to one input port stalls other input ports requesting the same output port. To accelerate the simulation, we focus only on four core stages: RC, VA, SA, and LT. Other router stages are eliminated because they don't cause pipeline stalls. For example, a packet allocated to the output port is immediately transmitted through the link rather than going through switch traversal

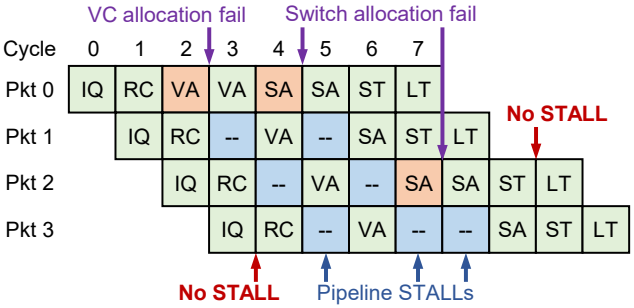

Figure 5: Some pipeline stages of a router don't cause stalls and thus can be eliminated for accelerating simulation.

and output queuing. Smooth transmission is guaranteed by the VC/switch/link allocation.

Cache the results of repetitive routing computations: Routing computation is a time-consuming stage in simulation, especially for adaptive routing and complex hierarchical topologies. Using a complete routing table is impossible because the table size of an n nodes network can be $O(n^2)$, which is too large for memory. As a compromise, we cache some results of repetitive computations. Taking the *Dragonfly* topology as an example, we cache the global channel between any two groups and the local channel between any two routers in the same group. By caching these results, though the memory usage is increased, the simulation speed can be significantly improved. Validation experiments in § 3.5.2 show that the memory usage overhead of *partial caching* is not significant.

Integration with real workload traces: *CNSim* supports various traffic patterns, including the permutation patterns and AllReduce patterns. To better analyze the network performance under real workloads, *CNSim* also supports two typical traces. First is the shared-memory workload traces with cycle-accurate timestamps. The *Netrace* are collected from a 64-core M5 simulation system [18] executing multithreaded applications from the PARSEC v2.1 suite [17,36,37]. Each trace item records the `inject_cycle`, `bus_name`, `src_port_id`, `dst_port_id`, and `msg_type` (message size). By using cycle-accurate traces, the evaluation results are more persuasive. *CNSim* also supports the distributed-memory workload traces without timestamps. The traces include a list of messages (source-destination) by order of generation [11]. The performance of distributed-memory traces can be evaluated by adjusting the injection rate.

3.5 Validation

3.5.1 Implementation & Performance Simulation

CNSim is implemented in C++ and validated by comparison with three typical cycle-accurate simulators: *BookSim* [40], *Garnet* [9], and *OMNET++* [4]. The release versions of these simulators in the experiments are the *BookSim* 2.0, *HNOCS* [13] based on OMNeT++ 5.7.1, and Garnet 3.0 [16]

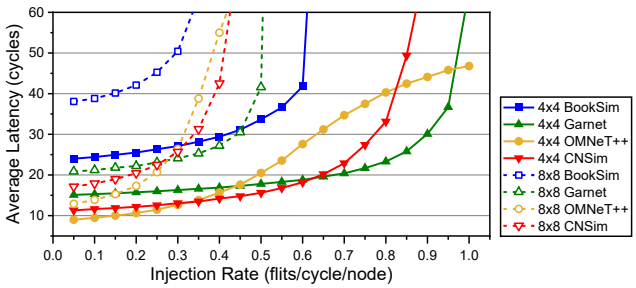

Figure 6: The latency-injection curves of different cycle-accurate simulators for uniform traffic on 2D-mesh.

based on Gem5 23.0. The *HNOCS* is compiled by gcc/g++ 9.4, and all other simulators are compiled by gcc/g++ 11.4. Most compilation options are left at default, and the "O2/O3" optimizations are turned on when possible in comparing runtime. All the experiments are performed on an Intel Core I9-13900K CPU with 32 GB two-channel memory.

The router microarchitecture we implement in the *CNSim* is the typical input-queued virtual channel router, similar to most other cycle-accurate simulators. However, implementation details can vary: *e.g.*, Garnet is two-stage pipelined; BookSim is four-stage pipelined; OMNeT++ is event-driven and thus does not have a cycle-based pipeline; *CNSim* can be one-stage or multi-stage by different configurations. The performance of the networks in different simulators is evaluated through latency-injection curves. Dimensional order (*i.e.* XY) routing on 2D-mesh topology is adopted, and the network scale is 4×4 and 8×8. Critical parameters, including the packet size, buffer size, and VC number, are set to 5-flits, 4-packets, and 2-VC; other parameters remain default. The evaluated traffic is uniformly random, and all experiments are simulated for 10K cycles.

All these cycle-accurate simulators can be used for evaluating microarchitecture and overall performance, but the performance results of different simulators are different due to the diverse modeling. As shown in Figure 6, the simulated performance of *CNSim* is close to *Garnet* and better than the *BookSim* because *BookSim* is four-stage pipelined, and it strictly simulates the circuit implementation. *CNSim* and *Garnet* use a more efficient router design but do not reflect some circuit-level behaviors. The OMNeT-based *HNOCS* is cycle-accurate but not cycle-based; as a result, it behaves differently from other simulators.

3.5.2 Simulation Speed & Memory Usage

One of the significant advantages of *CNSim* is the simulation speed. We compare the simulation performance of *CNSim* with other simulators. Each experiment uses uniform traffic, continues for 100K cycles, and the runtime is measured. As shown in Figure 7(a), the single thread *CNSim* is over $14\times$ faster than existing cycle-accurate simulators for the 4×4 2D-mesh topology. For small-scale networks, hyper-threading is

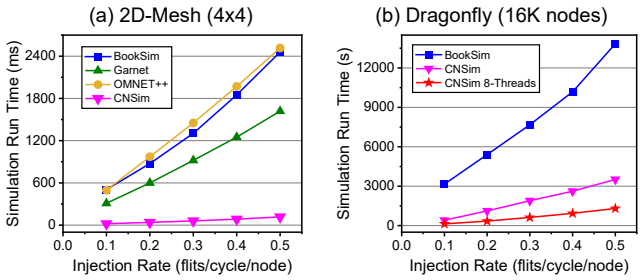

Figure 7: Simulation run time comparison for different simulators and topologies.

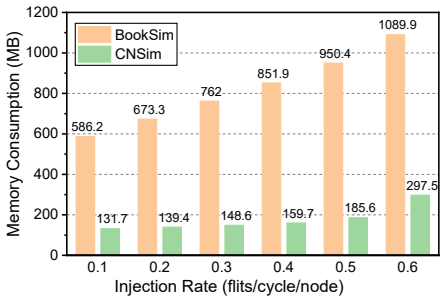

Figure 8: Comparison of heap memory consumption at different injection rates.

not necessary. As shown in Figure 7(b), single thread *CNSim* is $\sim 4\times$ faster than *BookSim* for the large-scale *Dragonfly* of 16K nodes. The 8-threads *CNSim* achieves $3\times$ hyper-threading speedup and is $\sim 11\times$ faster than the *BoosSim*. The runtime comparisons show that *CNSim* is much more efficient than existing cycle-accurate simulators under various conditions.

The computation complexity can be estimated by $O(nIT\widetilde{L})$, where n is the network scale (router number), I is the injection rate (flits/cycle/router), T is the simulation time (cycles), and $\widetilde{L}$ is the average latency (cycles) of packets. As a result, the speedup can be fewer when the network scale is ultra-large and heavily congested since $\widetilde{L}$ is large (more than hundreds of cycles) in such simulations. However, *CNSim* can still be much faster than other simulators by using enabling hyper-threading. More evaluation of parallelism is presented in the next subsection.

Another factor affecting a simulator's scalability is the memory overhead, especially for hyper-threading simulators running on a single machine. We profile the heap memory usage of *CNSim* using *Valgrind Massif* [5]. Uniform traffic pattern on 16K nodes *Dragonfly* is simulated. As shown in Figure 8, *CNSim* consumes 131.7 MB to 297.5 MB heap memory at 0.1 to 0.6 flits/cycle/node injection rates. For comparison, the *BookSim* consumes 586.2 MB to 1.1 GB heap memory at 0.1 to 0.6 injection rates. Since *CNSim* is packet-centric, the `Packet` objects, tracing records, and routing results consume much memory, and the consumption is proportional to the packet number. Besides, the network status and the cached partial routing tables also take up some fixed-size memory.

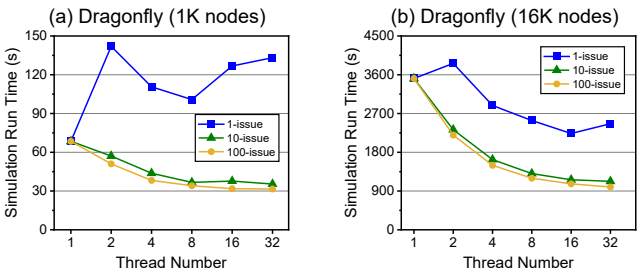

Figure 9: Speedup of parallel simulations with different number of threads.

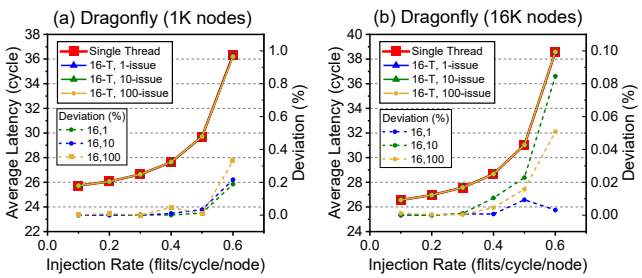

Figure 10: Average latency deviations of hyper-threading and multiple-issue.

When the network is heavily congested (0.6 flits/cycle/node in Figure 8), the memory usage of *CNSim* will be higher because the packet number is significant. Even so, the memory usage of *CNSim* is still much lower than *BookSim* because *CNSim* does not maintain request mapping tables. In conclusion, the packet-centric architecture and routing caching technology do not bring memory overhead, and *CNSim* is scalable for large-scale networks.

3.5.3 Concurrency & Inconsistency

CNSim achieves parallel (hyper-threading) simulation on any multicore CPU without additional programming overheads. We evaluate the parallel speedup of *CNSim* on two dragonfly topologies with 1K and 16K nodes. The injection rate is set to 0.5 flits/cycle/node, and the simulation continues for 100K cycles. Each thread is issued 1/10/100 packets from the packet queue every time. As shown in Figure 9, the parallelism is insignificant for small-scale networks. About 50% run time is saved using 32 100-issue threads for the 1K nodes *Dragonfly*. However, multi-threading achieves significant speedup for larger-scale networks. The 4-thread and 32-thread 100-issue *CNSim* are about 3× and 5× faster than the single-thread simulator for the 16K nodes *Dragonfly*. Compared with the single-issue, the multiple-issue scheme reduces the thread contention among multiple workers, thus achieving better speedup. For the dragonfly topology, issuing a few packets each time is enough, and a larger issue number has limited improvement. If the processing runtime of one packet is smaller (*e.g.*, by using simpler topologies and routing), the issue number should be increased.

As discussed in § 3.2, hyper-threading and multi-issue without strict synchronization can lead to inconsistency. Therefore, we evaluate the parallel inconsistency of *CNSim* on the same two *Dragonfly* networks. The average latency of single-thread simulation and multi-thread simulation is measured and compared. All experiments use the same random number seed (*i.e.*, and all packets are injected in the same order and timing) to eliminate random bias. As shown in Figure 10, the latency deviation of multithreading of most conditions is less than 0.2%. When the traffic is heavy, the inconsistency is slightly higher, but the deviation is still less than 0.5%. For the 1K nodes *Dragonfly*, a larger issue number may lead to worse inconsistency, but for the 16K nodes *Dragonfly*, the inconsistency is not significantly affected by the issue number. That is because packets are broadly distributed in large-scale networks, and the possibility of inverted allocation is low (Appendix D). In most cases, 10-issue and 16-thread simulations can achieve considerable speedup while maintaining high consistency.

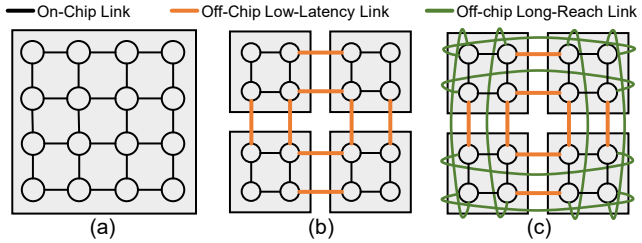

Figure 11: Hetero-link-based chiplet architecture. (a) Traditional 2D-mesh on a single chip. (b) Chiplet-based 2D-mesh topology. (c) Hetero-link-based 2D-torus topology.

4 Evaluation of Chiplet-based Networks

In our previous practices, we have encountered two significant real problems. **1)** Current simulators cannot simulate the heterogeneity of the chiplet-based networks. **2)** The simulation speed is too slow to evaluate large-scale chiplet-based networks, *e.g.* dragonfly. *CNSim* addresses the limitations of existing tools, making efficient and accurate evaluations possible. In this section, extensive evaluations of two typical chiplet-based networks are presented. **1)** Heterogeneous-link-based 2D-mesh/torus with billions of *PARSEC* traces. **2)** Large-scale chiplet-based dragonfly network with tens of thousands of nodes. Excluding the wasted and unshown experiments, about 500 billion cycles and 20 billion packets are simulated for the presented evaluations. The total simulation runtime is more than 20 hours, which is expected to take more than 200 hours if using other simulators.

4.1 Heterogeneous-Link-based Networks

4.1.1 Setup

As shown in Figure 11(b)(c), 2D-mesh and 2D-torus are two typical topologies for chiplet-based networks. All links are

Table 2: Default Parameters

Parameter	Value
Packet Length	4 flits
Input Buffer Size	32 flits for on-chip buffers
Virtual Channel Number	2 channels/link
Simulation Time	10000 cycles (1000 cycles warming up)

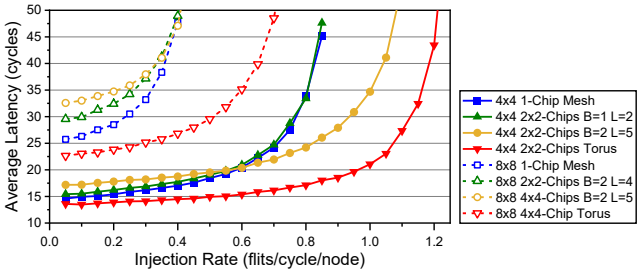

Figure 12: Evaluation results of the uniform traffic patterns.

configured by two parameters: bandwidth (B) and latency (L). The bandwidth of the on-chip link is 1 flit/cycle, and the latency is 1 cycle. In the evaluation, two die-to-die configurations are used for chiplet-based networks: the low-latency parallel link (B=1, L=2) and the high-bandwidth serial link (B=2, L=4). Two routing algorithms are used for 2D-mesh: dimensional order (XY) routing and Duato-protocol-based negative-first adaptive routing (NFR) [28]; and the *CLUE* [47] routing algorithm is used for 2D-torus. Hetero-link is used in the chiplet-based 2D-torus topology, *i.e.*, the adjacent die-to-die links are configured with (B=2, L=4), and the wraparound die-to-die links are configured with (B=2, L=4).

Uniform pattern and *PARSEC* traces are evaluated, and the default parameters used in the simulations are shown in Table 2. For the uniform traffic pattern, the average latency of packets is measured at different injection rates. For the *PARSEC* traces, 16 Bytes link width is used(*i.e.*, the largest packet has 5 flits), and all messages are injected according to the trace timestamps, even if queuing occurs.

4.1.2 Evaluation Results

Uniform traffic pattern. XY routing is used for evaluating uniform traffic on 2D-mesh. As shown in Figure 12, small-scale (4×4 nodes) and large-scale (8×8 nodes) networks are evaluated. According to the results, the latency of the die-to-die links directly determines the average packet latency at low traffic but does not significantly affect the saturation throughput. The bandwidth of the links in 2D-mesh affects the throughput for the small-scale network, but the impact is negligible for the large-scale network. That is because, for small networks, a certain percentage of random messages are adjacent at chip edges; thus, the bandwidth of the die-to-die links is fully used. However, most random messages in large networks are long-distance messages that go across chiplets and are congested at the on-chip links; thus, the higher-bandwidth

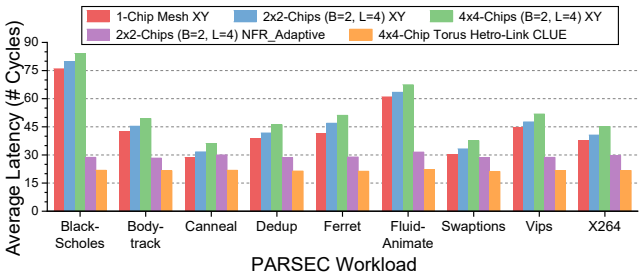

Figure 13: Evaluation results of PARSEC traces on chiplet-based networks.

serial link is not better than the low-latency parallel link in the middle of the 2D-mesh network. The real advantage of the serial link is that the long transmission distance makes wraparound links possible. The hetero-link-based 2D-torus achieves the best latency and throughput for both the small-scale and large-scale networks because the diameter of the 2D-torus is smaller than 2D-mesh, and the bisection bandwidth is also larger. All the results are consistent with the theoretical analysis, and the simulation results give cycle-accurate quantitative validations.

PARSEC traces. As shown in Figure 13, large-scale networks (8×8 nodes) with different configurations are evaluated. According to the results, the higher-bandwidth serial link is not beneficial since the bottleneck is on the on-chip links, which is consistent with the uniform traffic pattern. The deterministic XY routing algorithm for some workloads (*e.g. Black–Scholes* and *Fluidanimate*) is severely congested, and the average latency is much higher than the static network diameter. If the adaptive routing algorithm is used, the average latency of all workloads can be significantly reduced. The hetero-link-based 2D-torus with *CLUE* routing algorithm achieves the lowest average latency for all workloads and almost completely eliminates network bottlenecks. The entire *PARSEC* traces include over 100 billion cycles and 3 billion messages. Each round of experiments (some not presented in the paper) is run for $\sim$ 2 hours, which is quite efficient.

4.2 Chiplet-based Dragonfly

4.2.1 Topology Description

The traditional *Dragonfly* topology achieves high scalability by using numerous high-radix switches. However, high-bandwidth and high-radix switches are expensive and introduce additional latency and power consumption. Besides, the single physical channel connecting the terminal and the switch severely limits the local throughput. Using advanced packaging technologies (*e.g.* InFO-SoW [24]), multiple chips can be integrated at a large scale and high density.

As shown in Figure 14, the *Dragonfly* topology can be implemented in a switch-less way. Instead of using switches connecting nodes (processors), the chiplet-based dragonfly connects nodes (chiplets) by a 2D-mesh-on-package, forming

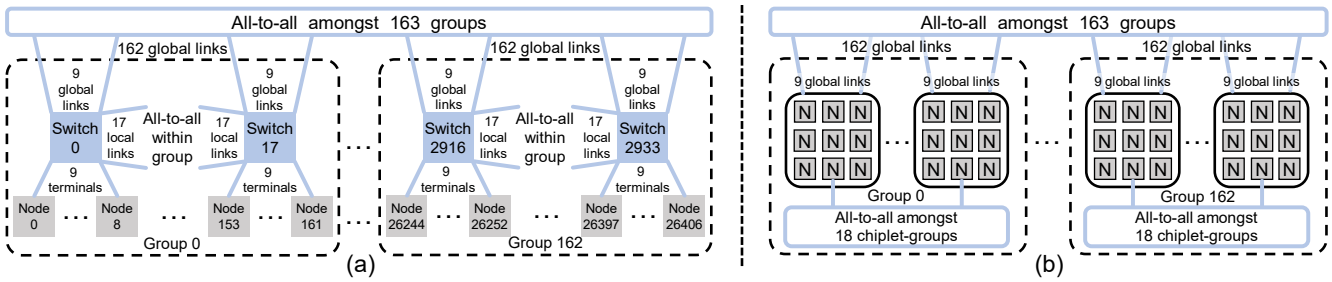

Figure 14: Dragonfly interconnection networks. (a) Traditional switch-based dragonfly: nodes (processors) are connected to the switches, switches in each group are fully connected, and all groups are also fully connected. (b) Chiplet-based switch-less dragonfly: nodes (chiplets) are connected by a 2D-mesh, forming chiplet-groups; chiplet-groups are connected just as the switch in traditional switch-based dragonfly.

chiplet-groups. Each chiplet-group is equivalent to the *Dragonfly* switch, and all chiplet-groups are connected just as the switch in the traditional switch-based dragonfly. In this way, the chiplet-based dragonfly is constructed without additional switches.

4.2.2 Analysis

In traditional switch-based *Dragonfly*, both the local throughput and global throughput are bounded by 1 flit/cycle/chip, which is the bandwidth of each switch port. However, in the chiplet-based dragonfly, chips are connected by multiple physical channels, breaking through the switch bottleneck. If there are n physical channels on each chiplet edge in an $m \times m$ chiplet-group, the throughput can be estimated by [25]

$$T_{\text{cg}} < \frac{2B_C}{N} = \frac{nm \times \times 2}{m^2} = \frac{4n}{m}\ [\text{flits/cycle/chip}], \quad (1)$$

where B_C is the total bisection bandwidth, and N is the total chiplet number. For appropriate configurations (*e.g.*, $n = 3$ and $m = 3$), the throughput within the chiplet-group can be much better than the throughput through switches.

At the same time, for local performance within a dragonfly group, the throughput

$$T_{\text{local}} < \frac{(a/2)^2 \times 2 \times 2}{am^2} = \frac{2mn}{m^2} = \frac{2n}{m}\ [\text{flits/cycle/chip}], \quad (2)$$

where $a = 4mn/2$ is the chiplet-group number in a dragonfly group. Therefore, the local throughput of the chiplet-based dragonfly is also much larger than the traditional switch-based dragonfly.

A potential bottleneck of the chiplet-based dragonfly is the total bisection bandwidth of the chiplet group. The total full-duplex bisection bandwidth of a k-port switch is k flits/cycle; however, the 2D-mesh-based chiplet-group only has $k/2$ bisection bandwidth. As a result, when the network is simultaneously under heavy global traffic and local traffic, the overall network performance can be poor. A simple way to eliminate the bottleneck is to increase the bandwidth in the 2D-meshbased chiplet-group. For example, the UCIe die-to-die interface can provide 1317 GB/s/mm die edge density (947 GB/s/mm^2 area density). That is to say, high bandwidth in the chiplet-group is easy to achieve by using advanced packaging and high-speed wireline technologies.

However, theoretical analysis is not convincing enough. To further validate the chiplet-based dragonfly architecture, large-scale systematical simulations must be performed.

4.2.3 Setup

The baseline in the evaluations is the standard switch-based dragonfly. The terminal, local, and global ports of a switch are configured at $4:7:5$ for radix-16 and $8:15:9$ for radix-32. As a result, the total group number and chip number are $(41, 1312)$ for radix-16 and $(145, 18560)$ for radix-32. In the chiplet-based dragonfly, the local and global ports are configured as the same number as the switch-based dragonfly. For real systems, the latency of switch-to-switch hops is far larger than the chiplet-to-chiplet hops; but in the simulations, external link latency is configured to maintain similar diameters. The default parameters used in the simulations are shown in Table 2, and the configuration details can be referred to in the source code. Only minimal routing is used since the misrouting and other details of the chiplet-based *Dragonfly* are beyond the scope of this paper.

Workloads. The evaluations use two kinds of network workloads: **(1) Unicast traffic patterns.** The *uniform* and other permutation patterns [25] are evaluated. **(2) Collective traffic patterns.** We evaluate the *bidirectional-ring-based AllReduce* traffic pattern, where each chip (process) i sends the $1/2N$ segment to chip $(i-1) \bmod N$ and chip $(i+1) \bmod N$ [38].

4.2.4 Evaluation Results

Local (intra-switch) performance: The throughput of the switch-based dragonfly is bounded by the single physical channel (1 flit/cycle/chip) connecting the terminal and the

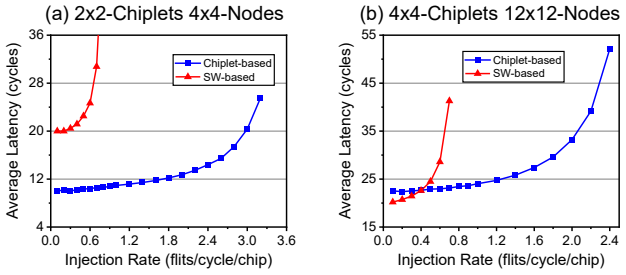

Figure 15: Local performance comparison: intra-chiplet-group *vs.* intra-switch (a) 4 chiplets forming a 4x4-nodes 2D-mesh; (b) 16 chiplets forming a 12x12-nodes 2D-mesh

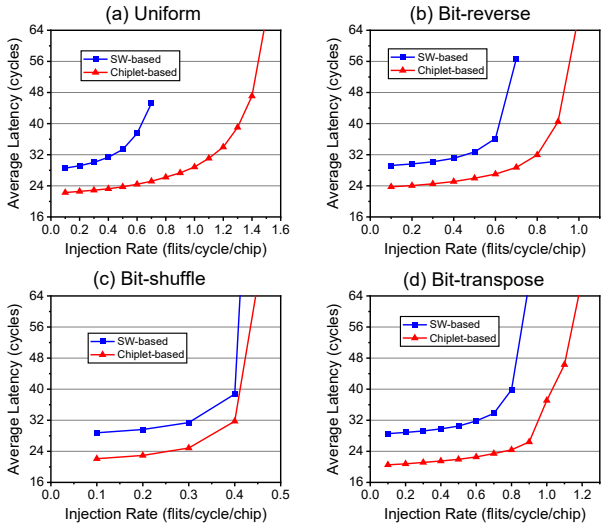

Figure 16: Intra-group performance comparison for different traffic patterns.

switch. However, in the chiplet-based dragonfly, chips are connected by multiple physical channels, breaking through the switch bottleneck. As shown in Figure 15, the local throughput of the chiplet-based dragonfly is much better than the switch-based dragonfly. For the 2x2-chiplets 4x4-nodes 2D-mesh, the intra-c-group throughput is more than 3 flits/cycle/chip, much higher than the switch-based dragonfly. For the 4x4-chiplets 12x12-nodes 2D-mesh, the throughput is also more than 2× higher than the switch-based dragonfly. In the evaluation, the latency of the dragonfly channels is set to 8 cycles. The static latency of the chiplet mesh increases with the scale since the diameter becomes larger. However, for real system, the cost of switch-to-switch hops is far more expensive than the cost of chiplet-to-chiplet hops. In conclusion, the chiplet-based dragonfly can achieve better intra-switch performance by eliminating the switch bottleneck.

Intra-group performance: We also evaluate the intra-group performance of the 41-groups *Dragonfly*. As shown in Fig. 16, except for the bit-shuffle pattern, the saturation injection rates of the chiplet-based networks are larger than the switch-based dragonfly. The injection/ejection bandwidth is no longer bounded by the single physical channel connecting

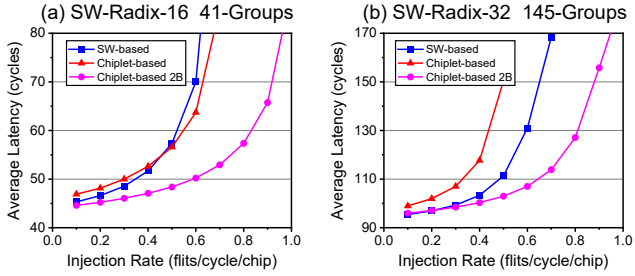

Figure 17: Global performance comparison for different network scales.

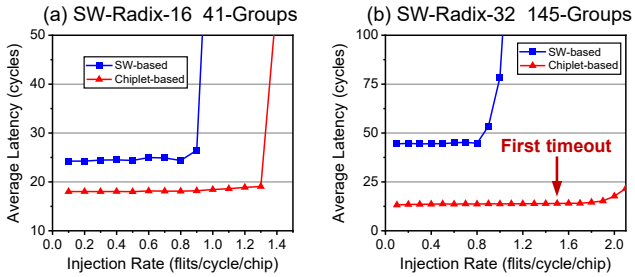

Figure 18: AllReduce performance comparison for different network scales.

the terminal and the switch, achieving higher performance. Especially for the uniform and bit-transpose pattern, the saturation injection rates are over 1. However, if the bottleneck is not at the switch, the chiplet-based dragonfly performs similarly to the switch-based dragonfly (*e.g.*, bit-shuffle shown in Fig. 16(c)).

Global performance: As shown in Figure 17, we evaluate the global performance on two networks with different scales. For the small-scale (41-groups) network, the chiplet-based dragonfly performs similarly to the switch-based dragonfly without changing the internal bandwidth. However, for the large-scale (145-groups) network, as discussed in § 4.2.2, the large 2D-mesh-based chiplet-group becomes the bottleneck; as a result, the chiplet-based dragonfly performs poorly. However, the bottleneck is eliminated if the bandwidth inside the chiplet-group is increased to 2 flits/cycle/chip. In real systems, increasing the bandwidth of chiplet-to-chiplet links is much easier than switch-to-switch links; therefore, it is a reasonable practice. Ten thousand simulated cycles and 0.5 injection rate for the large-scale dragonfly means 23 million 4-flit packets. Each round of experiments (from low-traffic to saturation) is run for about 15 minutes, which is quite efficient.

AllReduce performance: The ring-based AllRdeuce traffic is not heavy since each chip (process) only sends traffic to adjacent chips. As shown in Figure 18, the throughput of the switch-based dragonfly is bounded by the switch (1 flit/cycle/chip). For the chiplet-based dragonfly, the throughput can reach much higher since there is no switch bottleneck. The injection rates for the first occurrence of timeout packets are 1.3 and 1.5 flits/cycle/chip for small-scale and large-scale networks, respectively.

Simulator	Language	Parallelism	Scalability [# Nodes]*	Special Features
BookSim [40]	C/C++	No	10,000 [14]	Cycle-accurate, validated against RTL, widely used.
Noxim [21]	SystemC	No	256 [22]	Cycle-accurate, wireless NoC, implement-friendly.
DARSIM [46]	C/C++	Yes	64	5× speedup by 8 threads.
Garnet [9, 16]	C/C++	No	64 [16]	Gem5-based, full-system simulation, heterogeneous clock-domain.
HNOCS [13]	C/C++	Yes†	64 [13]	Cycle-accurate, based on OMNET++, heterogeneous link.
This Work	C/C++	Yes	≥ 279,040	Cycle-accurate, non-MPI-based parallel simulation.
OMNeT++ [4]	C/C++	Yes†	17,000 [15]	Extensive functionality, heterogeneous link.
NS-3 [50]	C/C++	Yes†	5,000 [51]	Extensive functionality, heterogeneous link
SST [52]	C/C++	Yes†	110,592 [35]	Modular, scalable, MPI-based packet-parallel simulation.
ROSS/CODES [19, 20, 49]	C/C++	Yes†	50,000,000 [48]	Scalable, MPI-based parallel simulation.

Table 3: Comparison of existing network simulators. *"Scalability" is the largest scale of network that the simulator supports, or we found in the literature. †Parallel simulation is achieved by manual partitioning and MPI communication protocol.

5 Related Work

Table 3 lists some typical network simulators. In general, they can be categorized as shared-memory design (*BookSim*, *Noxim*, *DARSIM*, and *Garnet*) and distributed-memory design (*OMNeT++*, *NS-3*, *SST*, and *ROSS*).

BookSim [40] is a cycle-accurate network-on-chip simulator validated against RTL implementations of real routers. It is accurate and highly modular, thus widely used in evaluating networks of various architectures and scales. The largest network scale that we found in the literature is 10,000 nodes [14]. However, *BookSim* neither supports distributed nor hyper-threading simulations; thus, the simulation speed is slow for large-scale networks.

Noxim [21] is a cycle-accurate simulator based on the SystemC library. It is designed for wireless networks-on-chip (WiNoC), but it can also be used for traditional on-chip networks. The largest network scale that we found in the literature is 256 nodes [22]. However, the speed of the simulation is slower since *Noxim* is implemented based on SystemC and does not support parallel simulation.

DARSIM [46] is a cycle-accurate simulator that achieves multi-threading using divided tiles and periodic synchronization. However, the topologies, network scales, and simulation speedup of *DARSIM* are limited and deprecated.

Garnet [9, 16] is a cycle-accurate network simulator based on the Gem5 simulator [2]. The network scale supported by *Garnet* is hard to exceed 256 because the Gem5 memory subsystem cannot instantiate more directories [44].

OMNeT++ [4] is an extensible, modular, and component-based discrete-event simulator that naturally supports heterogeneous link configurations. The **HNOCS** [13] is a cycle-accurate implementation based on *OMNeT++*. The largest network scale we found in the literature is 17,000 nodes [15]. *OMNeT++* supports parallel simulation; however, the network must be partitioned manually, and the partitions run concurrently in separate processes communicating over MPI, which is not available for shared-memory designs.

NS-3 [50] is a discrete-event simulator that similar to *OMNeT++*. The largest network scale we found in the literature is 5,000 nodes [51]. Parallel simulations are also achieved by manual partitioning and MPI communication protocol.

SST [52] is a modular discrete-event simulator that supports parallel simulation. It enables the co-design of large-scale architectures by simulating diverse hardware and software aspects. The largest network scale we found in the literature is 110,592 nodes [35]. *SST* also supports MPI-based parallel simulation.

ROSS [20] is a high-performance, low-memory overhead, massively parallel discrete-event simulator. **CODES** [19, 49] is an MPI-based simulation toolkit running on *ROSS* that provides a higher-level modeling API and high-fidelity models for HPC network and storage systems. The largest network scale we found in the literature is 50,000,000 nodes [48].

6 Summary

This paper presents *CNSim*, a cycle-accurate parallel network simulator designed for large-scale chiplet-based (shared-memory) networks. By using a packet-centric architecture and a novel atomic-based packet-parallel scheme, *CNSim* achieves high simulation speed and scalability, about $11\times \sim 14\times$ faster than other cycle-accurate simulators. Based on *CNSim* extensive evaluations on hetero-link-based networks and chiplet-based dragonfly networks are presented. The *CNSim* and the evaluation framework are open-sourced to the community.

7 Acknowledgments

This work is partially supported by the Wafer-Scale Silicon-Optic Interconnected System (2022YFB2804100), the National Natural Science Foundation of China (20211710187), the research projects from ZTE on advanced techniques for networks-on-chip, and the collective communication project

from Lenovo and Ant group. In addition, we would like to thank Sibin Mohan, the shepherd from The George Washington University, for his contribution to the revision of the paper.

References

[1] Fix performance bottlenecks with intel® vtune™ profiler. https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html.

[2] Gem5: The gem5 simulator system. https://www.gem5.org/.

[3] Mask / reticle - wikichip. https://en.wikichip.org/wiki/mask.

[4] Omnet++ discrete event simulator. https://omnetpp.org/.

[5] Valgrind home. https://valgrind.org/.

[6] Universal chiplet interconnect express (ucie) specification revision 1.1, July 2023.

[7] A. Agarwal and R. Shankar. Survey of network on chip (noc) architectures & contributions. 2009.

[8] Anant Agarwal. Waiting algorithms for synchronization in large-scale multiprocessors. *ACM Transactions on Computer Systems*, 11(3), 1993.

[9] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 33–42, Boston, MA, USA, April 2009. IEEE.

[10] Yuichiro Ajima, Takahiro Kawashima, Takayuki Okamoto, Naoyuki Shida, Kouichi Hirai, Toshiyuki Shimizu, Shinya Hiramoto, Yoshiro Ikeda, Takahide Yoshikawa, Kenji Uchida, and Tomohiro Inoue. The tofu interconnect d. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 646–654, Belfast, September 2018. IEEE.

[11] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. On the complexity of traffic traces and implications. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(1):1–29, May 2020.

[12] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. *The Datacenter as a Computer: Designing Warehouse-Scale Machines, Third Edition*, volume 13. October 2018.

[13] Yaniv Ben-Itzhak, Eitan Zahavi, Israel Cidon, and Avinoam Kolodny. Hnocs: Modular open-source simulator for heterogeneous nocs. In *2012 International Conference on Embedded Computer Systems (SAMOS)*, pages 51–57, Samos, Greece, July 2012. IEEE.

[14] Maciej Besta and Torsten Hoefler. Slim fly: A cost effective low-diameter network topology. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 348–359, New Orleans, LA, USA, November 2014. IEEE.

[15] Maciej Besta, Marcel Schneider, Salvatore Di Girolamo, Ankit Singla, and Torsten Hoefler. Towards million-server network simulations on just a laptop, May 2021.

[16] Srikant Bharadwaj, Jieming Yin, Bradford Beckmann, and Tushar Krishna. Kite: A family of heterogeneous interposer topologies enabled via accurate interconnect modeling. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, San Francisco, CA, USA, July 2020. IEEE.

[17] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, pages 72–81, Toronto Ontario Canada, October 2008. ACM.

[18] N.L. Binkert, R.G. Dreslinski, L.R. Hsu, K.T. Lim, A.G. Saidi, and S.K. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52–60, July 2006.

[19] Christopher Carothers and Argonne National Laboratory (ANL). Enabling co-design of multi-layer exascale storage architectures. Technical Report DOE-RPI–4875-1, 1311761, August 2015.

[20] Christopher D Carothers, David Bauer, and Shawn Pearce. Ross: A high-performance, low-memory, modular time warp system. *J. Parallel Distrib. Comput.*, 2002.

[21] Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. Noxim: An open, extensible and cycle-accurate network on chip simulator. In *2015 IEEE 26th International Conference on Application-Specific Systems, Architectures and Processors (ASAP)*, pages 162–163, Toronto, ON, Canada, July 2015. IEEE.

[22] Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. Cycle-accurate network on chip simulation with noxim. *ACM Transactions on Modeling and Computer Simulation*, 27(1):1–25, January 2016.

[23] Bill Chang, Rajiv Kurian, Doug Williams, and Eric Quinnell. Dojo: Super-compute system scaling for ml training. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–45, Cupertino, CA, USA, August 2022. IEEE.

[24] Shu-Rong Chun, Tin-Hao Kuo, Hao-Yi Tsai, Chung-Shi Liu, Chuei-Tang Wang, Jeng-Shien Hsieh, Tsung-Shu Lin, Terry Ku, and Douglas Yu. Info_sow (system-on-wafer) for high performance computing. In *2020 IEEE 70th Electronic Components and Technology Conference (ECTC)*, pages 1–6, Orlando, FL, USA, June 2020. IEEE.

[25] William J. Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers, Amsterdam ; San Francisco, 2004.

[26] Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler. An in-depth analysis of the slingshot interconnect. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14, Atlanta, GA, USA, November 2020. IEEE.

[27] Douglas Yu. Tsmc packaging technologies for chiplets and 3d. In *Proceedings of the 2021 IEEE Hot Chips (HCS)*, 2021.

[28] José Duato, Sudhakar Yalamanchili, and Lionel M. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann, San Francisco, CA, rev. printing edition, 2003.

[29] Yinxiao Feng, Dong Xiang, and Kaisheng Ma. Heterogeneous die-to-die interfaces: Enabling more flexible chiplet interconnection systems. In *56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 930–943, Toronto ON Canada, October 2023. ACM.

[30] Yinxiao Feng, Dong Xiang, and Kaisheng Ma. A scalable methodology for designing efficient interconnection network of chiplets. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1059–1071, Montreal, QC, Canada, February 2023. IEEE.

[31] Erik Fischer and Gerhard P. Fettweis. An accurate and scalable analytic model for round-robin arbitration in network-on-chip. In *2013 Seventh IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*, pages 1–8, Tempe, AZ, USA, April 2013. IEEE.

[32] Tim C. Fischer, Anantha Kumar Nivarti, Raghuvir Ramachandran, Ram Bharti, Derek Carson, Anton Lawrendra, Vineet Mudgal, Vivek Santhosh, Sunil Shukla, and Te-Chen Tsai. 9.1 d1: A 7nm ml training processor with wave clock distribution. In *2023 IEEE International Solid- State Circuits Conference (ISSCC)*, pages 8–10, San Francisco, CA, USA, February 2023. IEEE.

[33] Richard M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[34] Wilfred Gomes, Altug Koker, Pat Stover, Doug Ingerly, Scott Siers, Srikrishnan Venkataraman, Chris Pelto, Tejas Shah, Amreesh Rao, Frank O'Mahony, Eric Karl, Lance Cheney, Iqbal Rajwani, Hemant Jain, Ryan Cortez, Arun Chandrasekhar, Basavaraj Kanthi, and Raja Koduri. Ponte vecchio: A multi-tile 3d stacked processor for exascale computing. In *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, pages 42–44, San Francisco, CA, USA, February 2022. IEEE.

[35] Taylor Groves, Ryan E. Grant, Scott Hemmer, Simon Hammond, Michael Levenhagen, and Dorian C. Arnold. (sai) stalled, active and idle: Characterizing power and performance of large-scale dragonfly networks. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 50–59, Taipei, Taiwan, September 2016. IEEE.

[36] Joel Hestness, Boris Grot, and Stephen W. Keckler. Netrace: Dependency-driven trace-based network-on-chip simulation. In *Proceedings of the Third International Workshop on Network on Chip Architectures*, pages 31–36, Atlanta Georgia USA, December 2010. ACM.

[37] Joel Hestness and Stephen W Keckler. Netrace: Dependency-tracking traces for efficient network-on-chip experimentation. Technical report.

[38] Torsten Hoefler, Tommaso Bonato, Daniele De Sensi, Salvatore Di Girolamo, Shigang Li, Marco Heddes, Jon Belk, Deepak Goel, Miguel Castro, and Steve Scott. Hammingmesh: A network topology for large-scale deep learning. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–18, Dallas, TX, USA, November 2022. IEEE.

[39] Alexander Ishii and Ryan Wells. The nvlink-network switch: Nvidia's switch chip for high communication-bandwidth superpods. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–23, Cupertino, CA, USA, August 2022. IEEE.

[40] Nan Jiang, James Balfour, Daniel U. Becker, Brian Towles, William J. Dally, George Michelogiannakis, and John Kim. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE International Symposium on Performance Analysis of Systems*

and Software (ISPASS), pages 86–96, Austin, TX, USA, April 2013. IEEE.

[41] Norman P Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Andy Swing, Brian Towles, Cliff Young, Xiang Zhou, Zongwei Zhou, and David Patterson. Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *2023 ACM/IEEE 50th Annual International Symposium on Computer Architecture (ISCA)*, 2023.

[42] M. R. Siavash Katebzadeh, Paolo Costa, and Boris Grot. Evaluation of an infiniband switch: Choose latency or bandwidth, but not both. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 180–191, Boston, MA, USA, August 2020. IEEE.

[43] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *2008 International Symposium on Computer Architecture*, pages 77–88, Beijing, China, June 2008. IEEE.

[44] Tushar Krishna. Garnet2.0: A detailed on-chip network model inside a full-system simulator, 2017.

[45] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi, and K.I. Farkas. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings. 31st Annual International Symposium on Computer Architecture, 2004.*, pages 64–75, Munchen, Germany, 2004. IEEE.

[46] Mieszko Lis, Keun Sup Shim, Myong Hyon Cho, Pengju Ren, Omer Khan, and Srinivas Devadas. Darsim: A parallel cycle-level noc simulator. 2010.

[47] Wei Luo and Dong Xiang. An efficient adaptive deadlock-free routing algorithm for torus networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(5):800–808, January 2012.

[48] Misbah Mubarak, Christopher D. Carothers, Robert Ross, and Philip Carns. Modeling a million-node dragonfly network using massively parallel discrete-event simulation. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 366–376, Salt Lake City, UT, November 2012. IEEE.

[49] Misbah Mubarak, Christopher D. Carothers, Robert B. Ross, and Philip Carns. Enabling parallel simulation of large-scale hpc network systems. *IEEE Transactions on Parallel and Distributed Systems*, 28(1):87–100, January 2017.

[50] nsnam. Ns-3 network simulator. https://www.nsnam.org/.

[51] Joshua Pelkey and George Riley. Distributed simulation with mpi in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, Barcelona, Spain, 2011. ACM.

[52] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob. The structural simulation toolkit. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):37–42, March 2011.

[53] Omer S. Sella, Andrew W. Moore, and Noa Zilberman. Fec killed the cut-through switch. In *Proceedings of the 2018 Workshop on Networking for Emerging Applications and Technologies*, pages 15–20, Budapest Hungary, August 2018. ACM.

[54] Emil Talpes, Debjit Das Sarma, Doug Williams, Sahil Arora, Thomas Kunjan, Benjamin Floering, Ankit Jalote, Christopher Hsiong, Chandrasekhar Poorna, Vaidehi Samant, John Sicilia, Anantha Kumar Nivarti, Raghuvir Ramachandran, Tim Fischer, Ben Herzberg, Bill McGee, Ganesh Venkataramanan, and Pete Banon. The microarchitecture of dojo, tesla's exa-scale computer. *IEEE Micro*, 43(3):31–39, May 2023.

[55] Emil Talpes, Douglas Williams, and Debjit Das Sarma. Dojo: The microarchitecture of tesla's exa-scale computer. In *2022 IEEE Hot Chips 34 Symposium (HCS)*, pages 1–28, Cupertino, CA, USA, August 2022. IEEE.

[56] Kai Troester and Ravi Bhargava. Amd next generation "zen 4" core and 4th gen amd epyc™ 9004 server cpu. In *2023 IEEE Hot Chips 35 Symposium (HCS)*, pages 1–25, Palo Alto, CA, USA, August 2023. IEEE.

[57] Charlotte Truchet. Estimating parallel runtimes for randomized algorithms in constraint solving. *Journal of Heuristics*, 2016.

[58] Wemke van der Weij, Sandjai Bhulai, and Rob van der Mei. Dynamic thread assignment in web server performance optimization. *Performance Evaluation*, 66(6):301–310, June 2009.

[59] Wenfeng Xia, Peng Zhao, Yonggang Wen, and Haiyong Xie. A survey on data center networking (dcn): Infrastructure and operations. *IEEE Communications Surveys & Tutorials*, 19(1):640–656, 2017.

[60] Jieming Yin, Zhifeng Lin, Onur Kayiran, Matthew Poremba, Muhammad Shoaib Bin Altaf, Natalie Enright Jerger, and Gabriel H. Loh. Modular routing design for chiplet-based systems. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 726–738, Los Angeles, CA, June 2018. IEEE.

Appendices

A Profiling of CNSim

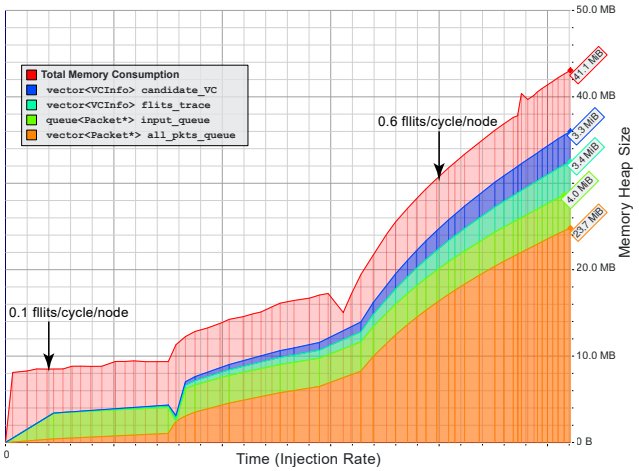

Figure 19: Detailed profiling of the heap memory consumption.

Valgrind Massif can give more detailed memory consumption profiling results. As shown in Figure 19, the heap memory usage of *CNSim* is mainly from: **1)** the candidate VCs provided to each packet by routing computation; **2)** the traces (location information) of each packet flit; **3)** the input buffer (queue) of each VC channel that maintains the packet arriving order **4)** the global packet queue that maintains the packet injection order.

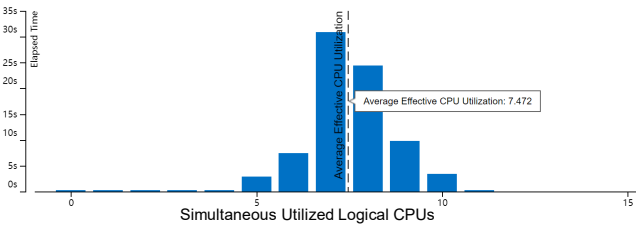

Figure 20: Effective CPU utilization histogram of the 16-threads *CNSim*.

We also use *Intel Vtune* [1] to analyze the CPU utilization. As shown in Figure 20, the effective CPU utilization measured by *Intel Vtune* is 7.5 for the 16-threads *CNSim*, which implies that there is still room for further improvement.

B PARSEC Benchmark Traces

As shown in Table 4, the entire *PARSEC* benchmark traces provided by *netrace* [36, 37] include over 100 billion simulated cycles and 3 billion messages, which is time-consuming for simulation.

Table 4: Simulated cycle and packet counts of the *PARSEC* benchmark traces.

Benchmark	Cycles	Packets	Benchmark	Cycles	Packets
Blackscholes	5.83B	114M	Fluidanimate	10.2B	188M
Bodytrack	4.58B	386M	Swaptions	1.75B	310M
Canneal	23.1B	372M	Vips	5.43B	335M
Dedup	5.45B	431M	x264	43.0B	585M
Ferret	8.26B	287M	**Total**	**108B**	**3B**

C Artifact

Abstract

Chiplet Network Simulation (CNSim) is an open-sourced cross-platform cycle-accurate parallel network simulator designed for large-scale chiplet-based networks. CNSim supports various network topologies, routing algorithms, interface configurations, and traffic patterns. The artifact helps the community reproduce the major results of this paper and make better use of CNSim.

Contents & Scope

The artifact includes the source code of CNSim, configuration files, and instructions, allowing the reproduction of major evaluation results (running speed) and demonstrating the scalability and functionality of CNSim.

Hosting

GitHub: chiplet-network-sim (branch atc24_artifact)
URL: https://github.com/Yinxiao-Feng/chiplet-network-sim/tree/atc24_artifact

Requirements

CNSim is a C/C++ program developed by Visual Studio 2022 and is verified on both Windows and Linux platforms. We recommend using an Ubuntu 22.04 machine (at least 8C/16T) in the evaluation. Please refer to the README for software requirements.

D Inconsistency Estimation

This section computes the bound of the parallel inconsistency of the simulator by a probabilistic model. The model gives the estimation of the probability of inverted allocations happening in the parallel simulation. An *inverted allocation* is defined as that a later-injected packet wins the allocation in the competition with an earlier-injected packet. Intuitively, since all the packets are dispatched to workers in injection order (iterate along the queue), the larger the distance (index difference) between two packets in the queue, the lower the probability of inverted allocation (inconsistency).

D.1 Problem Specification

Suppose M workers (threads) are working in parallel to compute the resource allocation of all the packets. As shown in Figure 3, the packets are in injection order (denoted as p_0, p_1, ... p_n, ... p_{n+k}, ...) and each worker will successively fetch a packet from the queue and compute its allocation. After one computation finishes, the worker will fetch another unprocessed packet from the queue, until the queue is empty. We assume that the computation time of each packet follows the Exponential Distribution $Exp(\lambda)$ so that queue pop is a Poisson Process with rate $M\lambda$ [8,45,57,58]. If one resource is requested by two packets p_n and $p_{n+k}, k \in \mathbb{N}_+$ in the same cycle, we are going to estimate the probability of p_n being allocated after p_{n+k}, which violates the first-injected-first-serve policy. We also give the estimation and assessment of k, which indicates the distance (index difference) between two competing packets in the queue that is maintained with injection order.

D.2 Probability Estimation

We assume that packet p_n is fetched by worker w_n from the queue at time S_n and the allocation happens at time $t_n \in [S_n, S_n + X_n]$, where $X_n \sim Exp(\lambda)$ is the total calculation time of p_n. We also assume that worker $i = 1, ..., M$ is computing packet $p_{a_{i,n}}$ at time S_n, and the computation start time of these packets is $S_{a_{i,n}}$. Since $t_n \leq S_n + X_n$ and $t_{n+k} \geq S_{n+k}$, the probability bound of packet p_n being allocated after p_{n+k} can be estimated as

$$P(t_n > t_{n+k}) \leq P(S_n + X_n > S_{n+k})$$

In other words, packet p_n occupies the worker w_n all the way, and any packet $p_{n+i}, 1 \leq i \leq k$, can not be dispatched to worker w_n before p_n completes computation. Therefore, $w_{n+i} \neq w_n$ must hold for all $1 \leq i \leq k$, and

$$P(t_n > t_{n+k}) \leq P(w_{n+i} \neq w_n, \forall 1 \leq i \leq k)$$

Now consider the process after time S_n, packet $p_{a_{i,n}}$ is being processed by worker i at time S_n and $S_{a_{i,n}} + X_{a_{i,n}} \geq S_n$. $Y_{i,n} = X_{a_{i,n}} - (S_n - S_{a_{i,n}})$ denotes the time that the worker i still needs to compute packet $p_{a_{i,n}}$ after S_n. Since the Poisson Process is memoryless, $Y_{i,n}$ still follows the distribution $Exp(\lambda)$ and is independent for $i = 1, 2, \ldots, M$. Thus, the probability of any worker completing its current packet computation first is $\frac{1}{M}$, which means w_{n+1} has a uniform distribution on $\{1, 2, \ldots, M\}$; therefore, the probability of $w_{n+1} \neq w_n$ is $(1 - \frac{1}{M})$.

Similar for $Y_{i,n+1}$ and we can conclude that w_{n+2} also has a uniform distribution, and is independent to w_{n+1} (since memoryless guarantees that events before S_{n+1} is independent to events after it). Then consider $Y_{i,n+2}$ and w_{n+3} and so on. With the same argument, we can conclude that all w_{n+i} are independent and follows a uniform distribution on $\{1, 2, \ldots, M\}$.

$$P(t_n > t_{n+k}) \leq (1 - \frac{1}{M})^k \leq e^{-\frac{k}{M}}. \quad (3)$$

If considering the multi-issue width of c, the probability

$$P(t_n > t_{n+k}) \leq e^{-\lfloor \frac{k}{c} \rfloor \frac{1}{M}}. \quad (4)$$

D.3 Distance Estimation

From Equation 3, we can see that when k is a few times larger than M (e.g., $k = 200, c = 5, M = 8$), the probability of inverted allocation is small ($P(t_n > t_{n+k})\big|_{\lfloor \frac{k}{c} \rfloor > 5M} < 1\%$). The final inverted allocation probability can be estimated as

$$P_{\text{inverted allocation}} \leq \sum_i P(k = i) e^{-\lfloor \frac{i}{c} \rfloor \frac{1}{M}}. \quad (5)$$

If the two packets are injected $x > 0$ cycles apart, initially, the distance $k(x)$ can be estimated as xnI, where n is the network scale (router number), I is the injection rate (flits/cycle/router). As the simulation continues, many packets are moved out of the queue and the distance becomes smaller. For example, if the network scale is $1K$, the injection rate is 0.1 packets/cycle/router, and the maximum latency is 100 cycles, then $0 < k < 10000$, which is expected to be quite large.

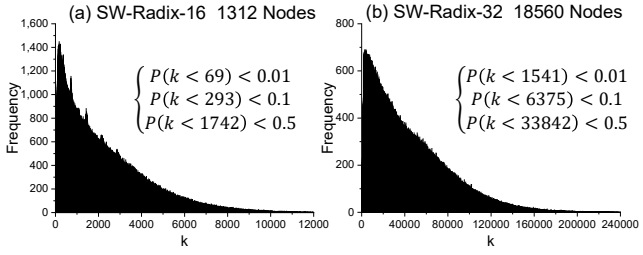

Figure 21: The frequency distribution of the distance (k) between two competing packets in the queue.

We count the frequency distribution of the distance k between all the competing packets during the simulation. The simulation time is set to 10000 cycles, and the injection rate is set to 0.5 flits/cycle/router. As shown in Figure 21, for the small-scale network, most of the competing packets are about hundreds to thousands of packets apart; for the large-scale network, the distance is about thousands to tens of thousands of packets apart. By Equation5, for 4-thread 5-issue simulation on the small-scale network and 16-thread 10-issue simulation on the large-scale network, $P_{\text{inverted allocation}} < 10^{-3}$.