System-Level Exploration of In-Package Wireless Communication for Multi-Chiplet Platforms

Nikhil Suri — https://github.com/nikhil21268

Email — nikhil21268@iiitd.ac.in

Section 1: The Big-Picture

Objective

The primary objective of this project was to replicate and analyze the study conducted in *System-Level Exploration of In-Package Wireless Communication for Multi-Chiplet Platforms* by Medina et al., 2023, which explores the feasibility and performance benefits of using in-package wireless communication for multi-chiplet architectures. The focus was to investigate how these wireless solutions can alleviate inter-chiplet communication constraints and enhance system performance, particularly for different workloads including deep neural networks (DNNs).

Introduction

Chiplet-based platforms are emerging as a prominent solution for designing highly integrated systems. While these architectures mitigate manufacturing and design challenges associated with monolithic chips, they introduce complexities in interconnects and power delivery. The traditional reliance on physical wiring for chiplet communication imposes limitations on bandwidth and flexibility, thereby constraining system performance.

The study by Medina et al. proposed leveraging in-package wireless communication as an alternative to wired interconnects. This approach allows more package resources to be allocated for power delivery and enables seamless integration of heterogeneous components, which is particularly valuable for applications requiring high parallelism, such as artificial intelligence workloads.

Key Goals

1. Understand System-Level Design:
    - Analyze the interaction between wireless interconnect configurations, chiplet architecture, and application performance.
    - Explore the role of Medium Access Control (MAC) protocols in optimizing communication.
2. Replication of Original Study:
    - Recreate the simulation environment using the gem5-X simulator, as described in the original study.
    - Model wireless communication systems and evaluate their performance under various configurations and workloads.
3. Performance Metrics:
    - Compare wireless and wired interconnects in terms of speedup and efficiency for different application profiles.

Methodology

Simulation Setup

- **Simulator**: gem5-X platform was utilized to model multi-chiplet systems and simulate wireless communication using nanoantennas.
- **Architectures**:
  - Per-core wireless connectivity: Each processing core equipped with dedicated antennas.
  - Per-cluster wireless connectivity: Groups of cores sharing a single antenna.
- **Applications**: AI workloads, including DNNs, were used to assess the runtime impact of wireless solutions (by the authors in the original paper [1]). Although, in our project, we restricted ourselves to traditional computations from the Stream and Splash2 benchmarks.

Wireless Communication Modeling

- **Physical Layer**:
  - Data modulation/demodulation and collision detection mechanisms were implemented.
- **MAC Protocols**:
  - Different protocols were tested, including token-passing schemes, which proved most effective in the original study. In our implementation as well, token passing was overall the best protocol as per simulation time, and other metrics.
  - This is because exponential backoff has a high penalty every time the contention window doubles due to packet losses.
  - Token passing does not suffer from this sort of a problem since the algorithm by itself ensures that a single node is transmitting at a given point in time.

Workload Profiles

Benchmarked workloads ranged from general-purpose tasks from the Stream and Splash2 benchmarks. This diversity enabled a comprehensive assessment of the wireless interconnect's performance.
For Splash2, we ran OceanCP, Radiosity and Raytrace benchmarks - the same ones that were run and reported in the original paper [1].

Performance Evaluation

Metrics included execution time, speedup and average latency. Speedups were computed relative to wired interconnect systems. All computations were performed relative to an ideal wired interconnect (defined as a system with CPI=1).

Results and Analysis

Comparison of Wired and Wireless Interconnects

Wireless solutions demonstrated notable advantages in flexibility and performance:

- A **Best speedup** was observed for OceanCP workloads in a four-cluster configuration.
- Freed microbumps in the package substrate allowed improved power delivery which also enhances system stability [1].

Impact of MAC Protocols

Token-passing schemes consistently outperformed exponential back-off for reasons described previously, offering reduced collision rates and better bandwidth utilization.

Granularity of Wireless Interconnects

Based on empirical observations, per-cluster configurations provided a balanced tradeoff between performance and collision rates, making them ideal for scalable systems.

Scalability

The wireless interconnect effectively scaled with the increasing number of cores and clusters, demonstrating its viability for future multi-chiplet platforms.
Not only that, we also found out (in line with the original paper's [1] findings, that at speeds >= 500 GBps, the benefits of using a wireless interconnect outweigh its potential overheads - making it the best choice at higher bandwidths.

Challenges Faced

1. Accurate replication of the gem5-X extensions and their integration required significant effort due to the complexity of the original simulation environment.
2. Fine-tuning MAC protocols and workload parameters to replicate results proved time-intensive.

Section 2: Fine-Grain Details

Benchmark Selection and Justification

For this project, we utilized a suite of Splash2 benchmarks as our primary performance indicators. Splash2 is a widely recognized benchmark suite designed to evaluate the performance of parallel applications in multicore and multiprocessor environments. The specific applications selected include:

- **Ocean**:
  - *contiguous_partitions*
  - *non-contiguous_partitions*
- **Radiosity**
- **Raytrace**

These applications were chosen due to their diverse computational and communication patterns, providing a comprehensive assessment of the simulated system's performance under varied workloads.

Environment Setup and Configuration

Acquisition and Preparation of Splash2 Benchmarks

1. **Downloading the Splash2 Suite**:
   - Obtained the `splash2.tar.xz` archive from an online repository, such as this:

     `https://www.capsl.udel.edu/splash/Download.html`

2. **Extraction**:
   - Extracted the downloaded archive to the working directory.
   - Command:

     ```
     tar -xf splash2.tar.xz
     ```

Building the Benchmarks

1. **Initial Build Process**:
   - Navigated to the extracted Splash2 directory.
   - Initiated the build process using the provided Makefile.
   - Encountered several Makefile errors due to outdated configurations.
2. **Resolving Makefile Errors**:
   - Followed the `old.gem5.splash2` tutorial to identify and rectify Makefile discrepancies.
   - Adjusted compiler flags and library paths to align with the current system's environment.
   - Applied patches as recommended in the tutorial to ensure compatibility.
3. **Further Makefile Adjustments**:
   - Consulted an additional online tutorial, as well as referred heavily to stackoverflow to understand compilation-related errors, to fine-tune Makefile attributes specific to our simulation requirements.
   - Ensured that all benchmark binaries were correctly compiled for full system simulation.

Integration with gem5X-on-Chip-Wireless

1. **Shared Directory Configuration**:
   - Established a shared directory to facilitate the simulation of `.o` files.
   - Mounted the compiled `.o` files onto the shared directory using the `mount.sh` script.
2. **Simulation Execution**:
   - Utilized the gem5X-On-Chip-Wireless simulator to execute the benchmarks.
   - Employed identical procedures as used for the Stream benchmark to maintain consistency.
3. **Simulation Commands**:

```
build/ARM/gem5.fast
--remote-gdb-port=0
-d /home/divyasuri2011/On-Chip-Wireless/gem5-X-wireless/m5out/system_a
configs/example/fs_wirelessExample.py
--kernel=/home/divyasuri2011/On-Chip-Wireless/gem5-X-
wireless/full_system_images/binaries/vmlinux_wa
--disk-image=/home/divyasuri2011/On-Chip-Wireless/gem5-X-wireless/full_system_images
/disks/gem5_ubuntu16.img
--machine-type=VExpress_GEM5_V1
--dtb-file=/home/divyasuri2011/On-Chip-Wireless/gem5-X-wireless/system
/arm/dt/armv8_gem5_v1_16cpu.dtb
--cpu-clock=2GHz --sys-clock=1600MHz -r 2 -n 16 --mem-size=4GB
--mem-type=DDR4_2400_4x16 --mem-ranks=4 --caches --l2cache
--l1i_size=32kB --l1d_size=32kB --l2_size=1MB --l2_assoc=2
--l2_cluster_size=4 --membus-wireless --wireless-bandwidth=500GB/s
--mac-protocol=exp_backoff
--workload-automation-vio=/home/divyasuri2011/On-Chip-Wireless/benchmarks/splash2/codes
```

This is how all of the commands were structured. The only variation would be the parameters for MAC protocol and L2 cache cluster size - in order to simulate all the cases.

Execution Process

Mounting Shared Directories

For each simulation instance, the following steps were executed in a separate terminal to mount the necessary directories:

```
m5term localhost 3456
su root
./mount.sh /home/user/gem5-shared/Stream
cd /mnt
./stream-arm-c > stream-arm-c-output.txt
```

Note that the directory that you're deciding to mount would change based on which benchmark you're trying to run.

Benchmark Execution

Executed the following benchmarks, ensuring all necessary data files were appropriately decompressed and prepared (although, for this project, I've only added data for OceanCP, Raytrace and Radiosity - since these were the benchmarks followed in the original paper [1]):

```
./BARNES < input
./FMM < inputs/input.16384
./OCEAN -p16
./RADIOSITY -batch -p16
./RAYTRACE -m64 inputs/car.env
./VOLREND 1 inputs/head
./WATER-NSQUARED < input
./WATER-SPATIAL < input
```

Experimental Setup Context

The benchmarking process was conducted within the framework of the study titled "System-Level Exploration of In-Package Wireless Communication for Multi-Chiplet Platforms" by Medina et al. This study investigates the viability and performance implications of integrating in-package wireless communication in multi-chiplet systems. The key objectives aligned with our benchmarking efforts include:

- Performance Assessment: Evaluating how in-package wireless communication impacts the execution of complex workloads.
- Configuration Variations: Testing different MAC protocols to determine their suitability under varying network loads.
- Application Diversity: Employing benchmarks with diverse runtime profiles to capture a broad spectrum of performance metrics.

Analysis of Token Passing and Exponential Backoff Protocols

Based on the data obtained from computational benchmarks, this section talks about the comparative performance analysis between the token passing and exponential backoff protocols. The benchmarks considered include raytrace, radiosity, oceancp, and stream.

Token Passing Protocol

- The token passing protocol consistently achieves lower latencies across all bandwidth scenarios, which includes both communication-intensive and less demanding environments.
- This protocol shows superior performance particularly at lower bandwidths, maintaining reduced latencies even under conditions of high congestion.
- It notably provides a higher percentage of the ideal interconnect speed, especially in scenarios where there is frequent and simultaneous communication between nodes, as observed in the raytrace and radiosity benchmarks.

Exponential Backoff Protocol

- The exponential backoff protocol tends to perform relatively poorer in environments where there is frequent and concurrent communication. This is attributed to high collision rates and subsequent retransmissions.
- However, it achieves better performance compared to the token passing protocol at very high bandwidths, specifically noted in the oceancp benchmark [1]. However, in our simulations, token passing was still marginally better than exponential backoff - even though exponential backoff closed the gap really fast.
- There could be a few reasons for this variation - one of the obvious ones is that the authors used their custom python scripts to simulate Splash2 benchmarks, whereas we went ahead with a stock Splash2 benchmark compilation - via editing makefiles and compiling them for ARM architecture.
- Nevertheless, exponential backoff by-far performs the best on OceanCP benchmark.
- The improvement in performance at higher bandwidths suggests a reduction in collisions and retransmissions, which correspondingly reduces latency and enhances throughput.

Comparative Analysis Across Benchmarks

- In the raytrace and radiosity benchmarks, the token passing protocol exhibits clear advantages in terms of latency reduction and overall speedup compared to the exponential backoff, at nearly all bandwidth levels.
- In the oceancp benchmark, the performance of token passing and exponential backoff protocols is more closely matched, with exponential backoff slightly outperforming token passing at very high bandwidths. This may be due to better handling of less frequent but larger data transmissions.
- For the stream benchmark, token passing consistently delivers higher speedup, confirming its speed in handling parallel data streams even at lower bandwidths.

Impact of Bandwidth on Protocol Performance

- As bandwidth increases, the performance gap between the two protocols narrows, particularly in benchmarks that are less communication-intensive.
- The exponential backoff protocol benefits more significantly from higher bandwidths due to the reduced collision rates. However, the token passing protocol remains competitive, particularly in scenarios where rapid, frequent communications are essential.

In conclusion, the token passing protocol is generally more efficient in environments characterized by high communication needs and low to moderate bandwidths. On the other hand, the exponential backoff protocol might be more suited for conditions where communications are less frequent and bandwidth availability is higher, allowing it to manage collisions and retransmissions more effectively.

Plots

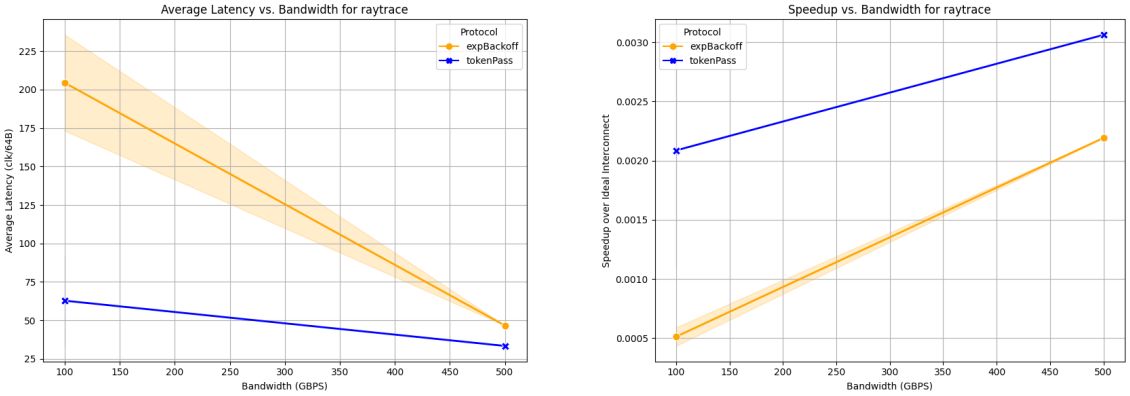

Figure 1: Raytrace

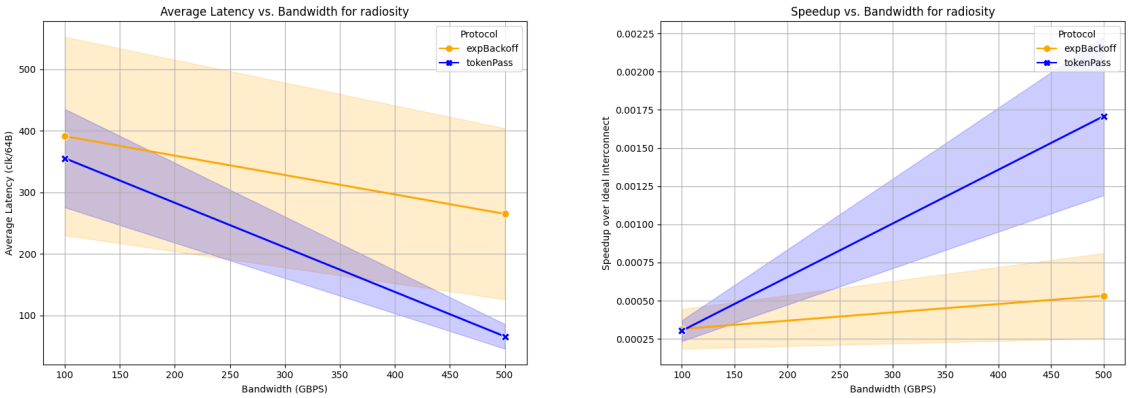

Figure 2: Radiosity

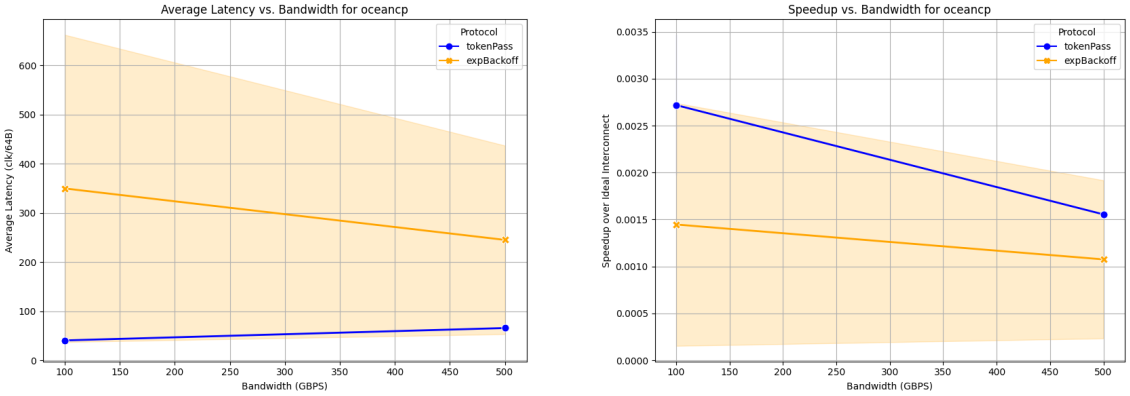

Figure 3: OceanCP

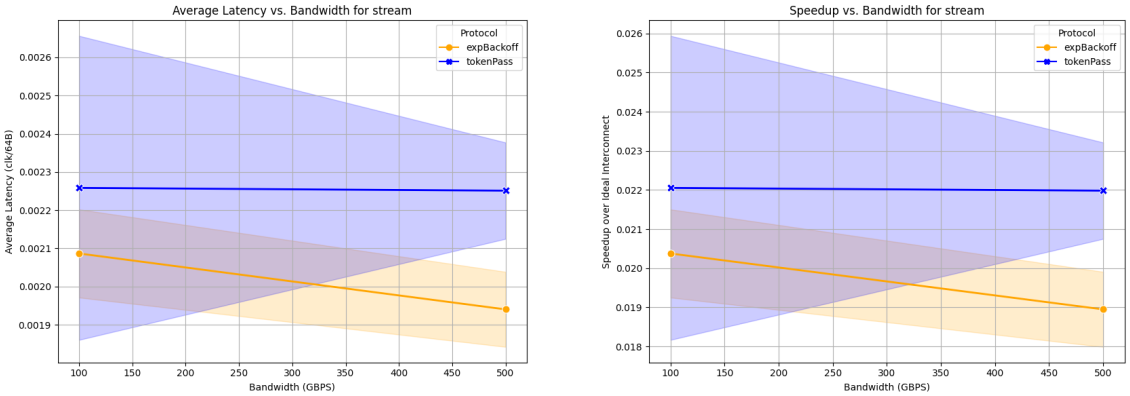

Figure 4: Stream

Conclusion

This project successfully replicated the findings of the original study, validating the potential of in-package wireless communication for multi-chiplet architectures. Wireless interconnects potentially offers a promising path forward for integrating heterogeneous systems, improving performance, and addressing the challenges of power delivery and interconnect bandwidth.

The insights gained from this project emphasize the need for continued exploration of wireless solutions, particularly for large-scale multi-chiplet systems.

Future Work

1. Explore the impact of emerging technologies, such as graphene-based nanoantennas, on bandwidth and system performance.
2. Investigate hybrid interconnect architectures combining wireless and wired solutions for optimized performance.
3. Conduct further studies on MAC protocol innovations to reduce latency and improve system responsiveness.

References

1. Medina, R., Kein, J., Ansaloni, G., Zapater, M., Abadal, S., Alarcón, E., & Atienza, D. (2023). *System-Level Exploration of In-Package Wireless Communication for Multi-Chiplet Platforms.* ASPDAC '23, ACM, New York. DOI: `https://doi.org/10.1145/3566097.3567952`.