RVipc: Low Latency Multi-core Communication Protocol for Trusted Environments

Kaustubh Khulbe, Sanjeevi Sengottuvel
University of Illinois at Urbana-Champaign
{kkhulbe2,ss152}@illinois.edu

Abstract

With the collapse of Denard scaling and Moore's Law, architects are transitioning towards multi-core systems. This paradigm shifts allows multiple cores on a single die, connected by high-bandwidth interconnects.

These interconnects are typically designed for high bandwidth but are elaborate NoCs, which often lead to nondeterministic latencies. Moreover, they are software-transparent, meaning application programmers cannot directly control these fabrics. Application programmers are left with shared memory as the primary means of communication.

However, shared memory suffers from nondeterministic latencies due to the cache hierarchy, TLB flushes, and cache pollution. This is especially problematic for latency-critical applications such as autonomous vehicles, drones, robotics, and augmented/virtual reality applications.

In this paper, we present RVipc, a low-latency inter-core communication protocol for real-time systems. RVipc is designed for trusted environments, such as embedded systems. It provides a mechanism to synchronize and communicate between cores with low latency and predictable latency metrics.

1 Introduction

Multi-core systems are becoming increasingly fundamental to modern computing, from servers down to embedded systems and edge devices. This paradigm shift is driven by the need for higher compute power, as traditional scaling methods have plateued.

Multi-core systems incorporate multiple cores on a single die, often tied together by high-bandwidth interconnects. For example, AMD utilizes Infinity Fabric, a proprietary interconnect technology, to connect multiple chips and cores.

There are several challenges with these forms of interconnects. First, they are usually designed for high bandwidth, and often cause nondeterministic latencies. This could be due to the routing algorithms or contention. More importantly, these interconnects are software-transparent, meaning application programmers cannot directly control how traffic flows across them. This limitation motivates the use of shared memory as the primary means of communication.

Shared memory suffers from nondeterministic latencies due to the cache hierarchy, TLB flushes, and cache pollution.

This is problematic for latency-critical applications such as autonomous vehicles, drones, robotics, and augmented/virtual reality applications. The success and safety of these applications directly depends on low-latency synchronization and communication between cores. A failure to do so significantly lowers the quality and feel of these products.

This motivates the need for a low-latency inter-core communication protocol. In this paper, we present RVipc, a low-latency inter-core communication protocol designed for trusted environments. Our contributions are as follows:

1. Deterministic latencies in communication between cores
2. Low-latency communication between cores
3. Software tooling to automatically communicate between cores in the most efficient manner

2 Design

In order to achieve deterministic latencies, we need to ensure both hardware and software usage is deterministic.

2.1 Hardware Design

To do so, we allow cores to sync and set up a dedicated FIFO buffer between them.

This FIFO buffer is used to pipe data between cores. Since it is a dedicated resource, there is no contention for it and will provide highly predicatable latencies in hardware.

The design then is a pool of FIFO buffers with a hardware unit to allocate buffers to cores. This hardware unit allows dedicated FIFOs to be set. Therefore, the only source of nondeterminsm is initializing the FIFO buffers.

Once initialized, there is a fixed latency guarantee to send and recieve data between cores.

2.2 Software Design

To avoid nondeterministic latencies in the software stack, we need to minimize the use of the kernel. This means we cannot invoke system calls and kernel resources, as they utilize the trap, I/O, and scheduling mechanisms out of the application programmer's control.

We can do this by providing a set of user-space instructions that are exposed to the application programmer. These instructions can be used to configure FIFO buffers, send and receive data, and tear down FIFO buffers.

We chose to use a polling mechanism to configure and tear down FIFO buffers. This is because polling, as opposed to interrupt-based, does not need the kernel and the programmer has full control over the frequency of polling.

We modified the RISC-V toolchain and ISA to add the following instructions:

1. `fconn`: Connect to a FIFO buffer
2. `fcreate`: Create a FIFO buffer
3. `fsend`: Write to a FIFO buffer
4. `frecv`: Read from a FIFO buffer
5. `fclose`: Close a FIFO buffer
6. `fstatus`: Check the status of a FIFO buffer

We provide a modified toolchain and compiler to be able to support these instructions natively.

3 Related Work

The related work of your project [1].

4 Conclusion

This project is awesome.

5 Metadata

The presentation of the project can be found at:

https://zoom/cloud/link/

The code/data of the project can be found at:

https://github.com/you/repo

References

[1] Dijkstra, E. W. The Structure of the "THE" Multiprogramming System. In *Proceedings of the 1st ACM Symposium on Operating System Principles (SOSP'67)* (Oct. 1967).