Enabling Scalable Heterogeneous Hardware Integration with Socket IPC

Richard Yan, Zekai Lin
CS 262A Fall 2023

Overview

- A socket library for seamless communication between diverse hardware simulations.
- Simplify integration of heterogeneous hardware blocks, ensuring performance and scalability.
- Features a modular, flexible socket-based IPC interface, supporting both blocking and non-blocking remote procedure calls.

Motivation

- Increasing heterogeneity in hardware landscape
- Difficult to integrate and evaluate new external IPs
- Simulation not scalable to increasing design size and difficult to parallelize execution

Design

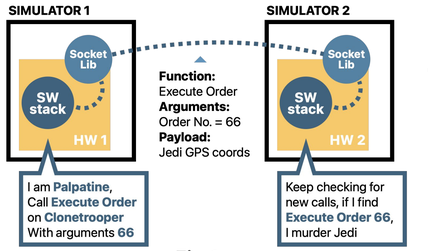


Fig 1

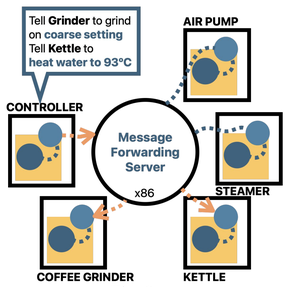


Fig 2

- *Lightweight.* A static C++ library of around 300 lines; or C version with no library dependencies for baremetal
- *Intuitive Function Call Interface.* (figure 1)
  - *Sending:* provide target ID, function ID, custom arguments, and optional data payload.
  - *Receiving:* provide function ID, blocking/non-blocking
- *Flexible Communication.* P2P or central server. (figure 2)
- *Efficient.* Low overhead for coarse-grained and acceptable overhead for fine-grained message communication

Microbenchmarks

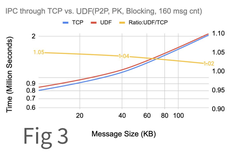


Fig 3

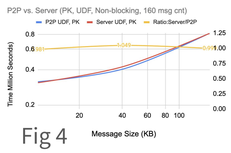


Fig 4

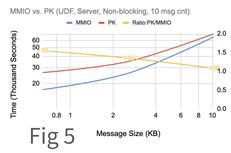


Fig 5

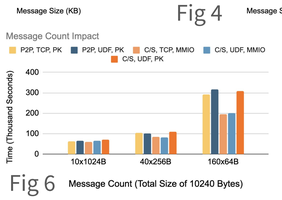


Fig 6

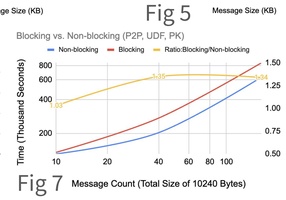


Fig 7

- **IPC Channel: TCP vs. UDF** (figure 3)
  - UDF has slightly more overhead (~1.02-1.05x)
  - The difference goes down as message size increases
- **Communication Architecture: P2P vs. Server-Client** (figure 4)
  - Server-Client has ~1.05x more overhead
- **Data Transfer: MMIO vs. PK** (figure 5)
  - PK has very large (2x) overhead when message size is small
  - Overhead goes down to 1.08x as message size increases
- **More small messages vs. Less large messages** (figure 6)
  - High message count (low msg size) gives more overhead
- **Blocking call latency** (figure 7)
  - Same number of calls and payload, blocking is ~130% slower

Case Study: CPU-Vortex GPU (Coarse-grained)

Background & Motivation

- Vortex is an OpenCL compatible RISC-V GPGPU developed at Georgia Tech
- A full SoC integration of an external GPU and a Rocket CPU in Chipyard is challenging, but socket-based modular simulation simplifies the integration
- RTL simulation of CPU and GPU is slow. Hardware integration with IPC enables functional and RTL co-simulation, which considerably reduces simulation time

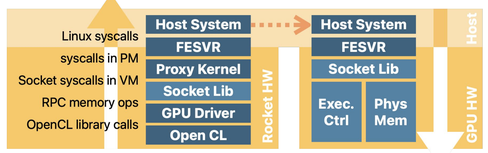
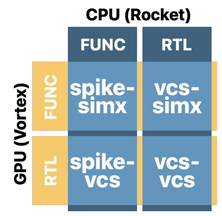


Fig 8

Design

- **CPU and GPU Co-simulation:**
  - Two independent processes that one runs CPU simulation and another runs GPU simulation. CPU dispatches the work to GPU using the IPC interface of the socket library (figure 8)
  - Simulations of CPU and GPU can be either RTL or functional. Spike, a RISC-V ISA simulator, and Simx, a simulator developed by the Vortex team, are used for functional simulation of CPU and GPU respectively (figure 9)
  - Monolithic CPU-GPU simulation is modeled as a sum of two individual hardware simulations, which is compared as a baseline.
- **GPU Driver & Softmax Workload:**
  - OpenCL GPU driver connects to GPU simulator during device initialization
  - The driver supports data transfer and starting execution with socket API
  - Softmax kernel with different input vector lengths are used for performance evaluation

Fig 9



Benchmarks

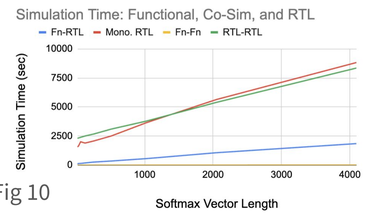


Fig 10

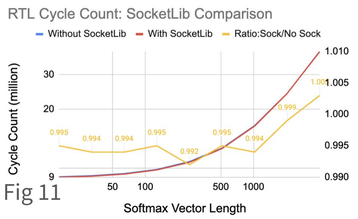


Fig 11

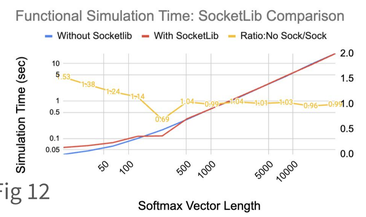


Fig 12

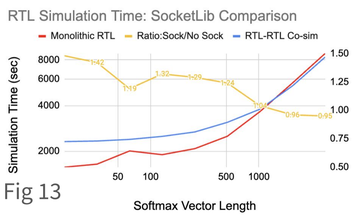


Fig 13

- **RTL-RTL Co-simulation:**
  - RTL-RTL co-simulation with socket library is more scalable than monolithic RTL simulation, as the design size of individual simulations is smaller (figure 13)
  - Cycle numbers obtained using sockets is a slight overestimation with long vector lengths, but overall predicts true cycle count very well (figure 11)
- **Functional-RTL Co-simulation**
  - Functional-RTL co-sim reduces simulation time by a factor of 5 (figure 10)
- **Functional-Functional Co-simulation**
  - Socket library introduces minimal overhead compared to the native functional integration (x86-Simx) developed by the Vortex team (figure 12)

Case Study: Many Accelerators (Fine-grained)

Background & Motivation

- Gemmini is Berkeley's Machine Learning hardware accelerator
- It has not been possible to integrate multiple Gemminis for a single CPU, making it difficult to parallelize large ML workloads like LLM inference
- Simulation time with large workloads is extremely slow, and must resort to FPGA simulations which are not easily-debuggable (if at all)

Design

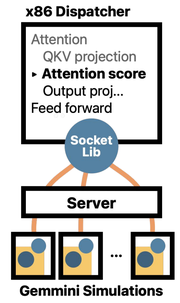


Fig 14

- **Hardware & supporting software:** (figure 14)
  - Many Gemmini+Rocket simulations in independent processes, each running a "headless" binary that exposes Gemmini library functions to the IPC interface, special routines for memory transfers
  - Each "worker" connects as client to central socket server, an x86 binary, also connected as client, creates and dispatches workloads to workers
- **ML Workload**
  - Different sized transformer encoder layers
  - Multithreaded (parallel) versions cut up large matrix multiplication into equal sized chunks for each worker, send to & wait for each worker, then copy memory back

Benchmarks

Test case variables (figure 15):

- *Model size*
- *Serial vs. parallel execution*,
- *Number of accelerators*,
- *Functional vs RTL* sim,
- *Native* (baseline monolithic integration) vs *Socket* (our work)

Model Sizes	Small	Compact	Medium	Large	Bert
# of fp32 weights	28,032	111,456	444,096	1,772,928	7,084,800
Hidden dim	48	96	192	384	768
Sequence length	32	48	64	128	512
Expansion dim	192	384	768	1536	3072
Num heads	2	4	4	8	12
Memory usage (MiB)	0.183	0.742	2.725	13.138	69.026

Fig 15

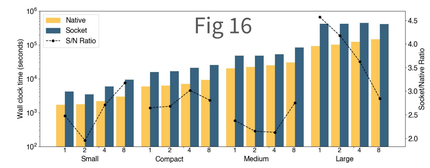


Fig 16

Serial RTL Simulation (figure 16):

- About 2-4x overhead with socket
- Scales similarly to native in terms of # of workers (ratio trendline)

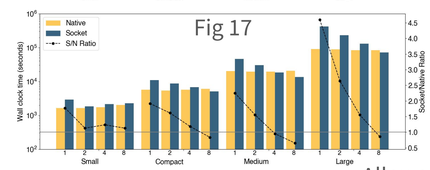


Fig 17

Parallel RTL Simulation (figure 17):

- Some overhead in cycles/real time
- Cycle number by default not predictive of true cycle count in monolithic integration

Advantages:

- Allows for true parallelism & correct execution, vs. emulated numbers for native
- Socket: constant simulation speed scaling, vs. Native: simulation speed scales linearly to design size) (figure 18)
- Overhead negated by parallel workers
- Net simulation time gain for large workloads

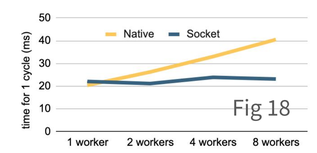


Fig 18

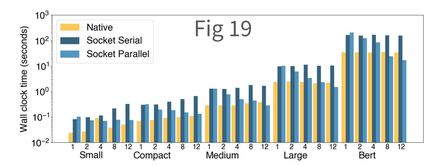


Fig 19

Functional Simulation (figure 19):

- Similar to VCS simulation but on larger scale and with more workers
- Parallel simulation also allows for correct behavior and time gain

Predicting true cycle numbers (figure 20):

- Can fit linear model to log(cycle numbers) to predict true cycle no. from socket cycle no. and no. of workers; accuracy 94.9%, R^2=0.996

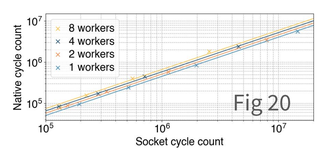


Fig 20