2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 14-15, 2025

Improving Flexibility in Hardware-Software Co-Development with Remote Virtual Prototypes

Przemysław Mikluszka, Patryk Górniak

Imagination Technologies

Imagination

accellera SYSTEMS INITIATIVE

Context & Motivation

Imagination at a Glance

- **Imagination Technologies** – provider of GPU IP and related technologies
- Expertise in **graphics, compute, and AI acceleration** for SoCs
- Key markets: **automotive, mobile, and data centre compute**

Our HW-SW Co-Development Approach

- Virtual Prototypes (VPs) for early driver and software validation
- Multiple VP models at different abstraction levels
  - e.g. C/C++, SystemC, FPGA emulators
- Models vary in accuracy and availability during the development cycle
- Integration with full-system simulators
  - QEMU, Gem5, TLM2-based frameworks
- Unified integration through a common internal API
  - Same front-end also used for emulator support

Possible Areas for Improvement

- Developer machines may lack sufficient resources to run compute-intensive simulation components
- Remote execution could offload heavy tasks to better-equipped machines
- Repetitive setup for each developer or VP model change
- Some VP implementations require extra resources without possibility of setup/teardown automation

Our Goal

- Enable **remote integration of GPU VPs** into full-system simulations
  - Similar solution previously implemented for our NNA IP
  - Work focused on QEMU solution only
- Maintain **a unified API** for seamless switching between VP implementations
- Reduce **setup overhead** and simplify environment configuration
- Support **consistent validation methodology** across all VP variants

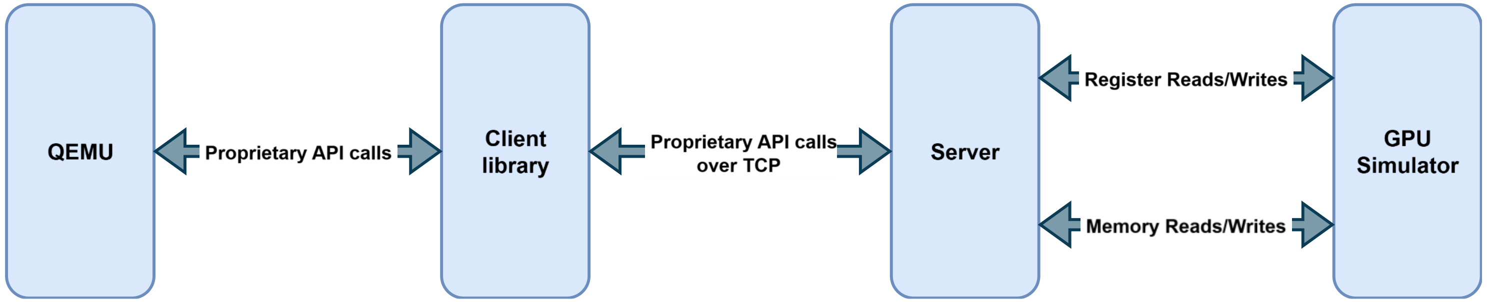
Architecture & Implementation

Requirements

- Maintain conformance with internal API
  - Device has a defined, stateful life cycle
  - Device memory managed outside the model
  - Requires device memory accessors to be supplied during setup
- Support multiple client connections

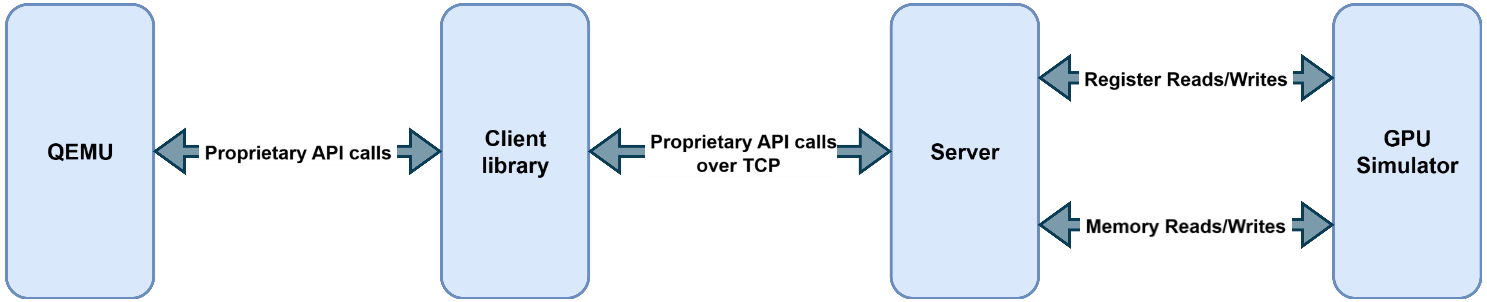
Initial architecture (1)

- Client library conformant with internal API
- Device memory model implemented on the client side
- Forward API calls to the server over TCP
- Server provides memory accessors to the GPU model



Initial architecture (2)

- All memory and register transactions forwarded to the client
- Communication via Protocol Buffers over ZeroMQ for bidirectional messaging
- Dealer–Router pattern used for message routing



Initial results

- Initial tests with client and server on the same machine
- Promising results: performance penalty of 281%–347% vs. baseline
- Real-world scenario (60 ms latency) proved infeasible
    - Simple workloads (baseline runtime of few seconds) failed to complete within hours

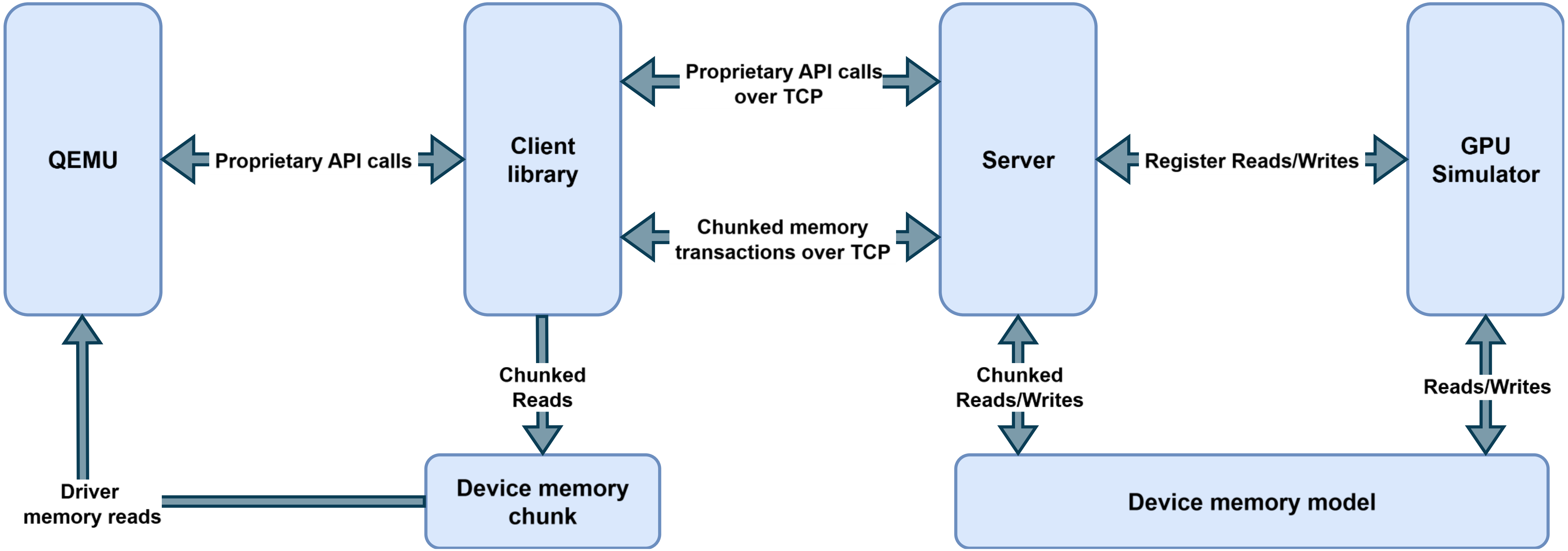
Optimisation ideas (1)

- Bottleneck caused by data transfer latency
- Reduce latency impact by minimizing the number of client-server transactions
- API used overly narrow data containers
- Move memory model to the server side
  - Intercept all device memory transactions from the driver and forward to server
  - GPU model now performs accesses to its memory locally
- Defer driver write requests and send them in batches

Optimisation ideas (2)

- Send memory in chunks for read operations
  - Divide device memory into equal-sized chunks
  - Subsequent reads reuse data from the same chunk if possible
  - Chunk content has limited validity
  - Chunk size is configurable
- Driver performs validation of data written
  - Track contiguous memory regions written by the driver
  - If the first address of a region is accessed, send a chunk matching that region's size

Final architecture



2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 14-15, 2025

Experimental Evaluation

accellera SYSTEMS INITIATIVE

Experimental Evaluation (1)

- Measured impact of optimizations on memory transaction count
- Evaluated progressively with each approach change
- Tests performed using a simple 3D application
- Significant reduction in number of transactions

Approach version	Number of transactions	Portion of baseline
Initial (baseline)	95409	100 %
Server-side memory	937876	983 %
Batched writes	126448	132 %
Chunked reads	2430	2.54 %
Dynamic chunk sizing	2233	2.34 %

Experimental Evaluation (2)

- Measured performance impact of the proposed solution
- Server and client on the same virtual local network (60 ms latency)
- Three application types tested
  - Simple 3D app using OpenGL API (**OGL**)
  - Simple 3D app using OpenGL ES with shader compilation (**OGLES**)
  - Compute app using OpenCL for FFT calculations (**IMGFFT**)

Experimental Evaluation (3)

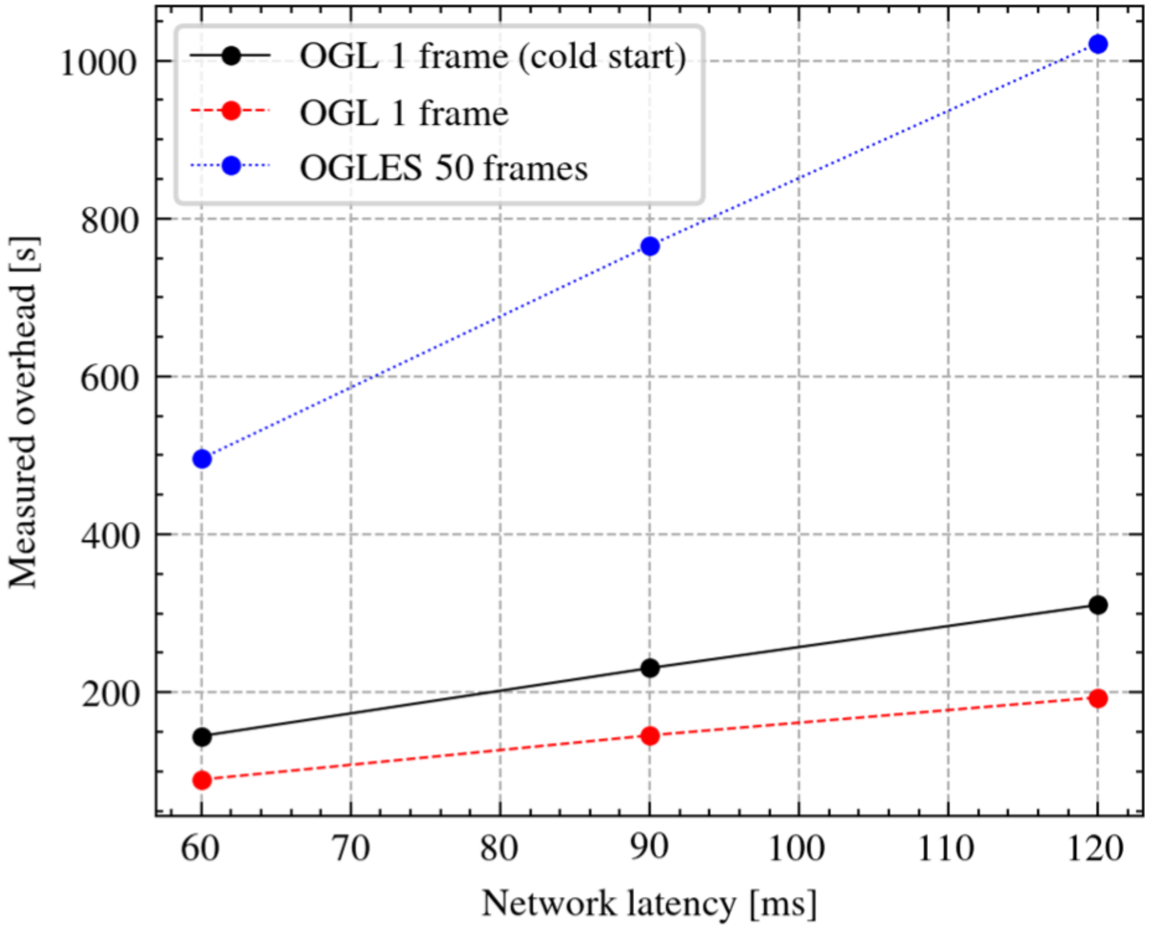
Application	Non-remote baseline	Min chunk size			
		512 bits	2048 bits	*4096 bits*	*8192 bits*
OGL 1 frame (cold start)	100%	7867%	5500%	**5033%**	5300%
OGL 1 frame	100%	4800%	4800%	**4700%**	4750%
OGL 10 frames	100%	595%	586%	**581%**	595%
OGL 50 frames	100%	246%	236%	**232%**	417%
OGLES 1 frame	100%	12800%	12875%	**12650%**	12725%
OGLES 10 frames	100%	2567%	2552%	**2524%**	2538%
OGLES 50 frames	100%	663%	642%	**630%**	640%
IMGFFT	100%	143%	127%	**120%**	132%

Experimental Evaluation (4)

- Minimum chunk size must be chosen carefully
  - Small size → increases number of transactions
  - Large size → can saturate network bandwidth
- Compute workload showed proportionally lower overhead
- Solution is better suited for compute-bound workloads

Experimental Evaluation (5)

- Conducted additional experiments to measure impact of network latency
- Re-ran tests with latency set to 90 ms and 120 ms using traffic control mechanisms
- Observed overhead increase is approximately linear with latency



Conclusion & Future Work

Conclusion & Future Work (1)

- Current solution shows potential but needs refinement for interactive and graphics-heavy applications
- Best suited for compute-bound workloads with lower relative overhead
- Less effective for simple debugging applications due to high memory transaction volume in relation to compute
- Main bottleneck: frequency and volume of memory synchronization between client and server

Conclusion & Future Work (2)

- Future work: develop more efficient device memory synchronization mechanisms
- Key challenge: maintain compatibility with internal API while improving performance
- Balancing API conformance and optimization is critical for real-world adoption

2025
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
EUROPE
MUNICH, GERMANY
OCTOBER 14-15, 2025

Questions

accellera
SYSTEMS INITIATIVE

2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

MUNICH, GERMANY
OCTOBER 14-15, 2025

Improving Flexibility in Hardware-Software Co-Development with Remote Virtual Prototypes

Przemysław Mikluszka, Patryk Górniak

Imagination Technologies

Imagination

accellera SYSTEMS INITIATIVE