Disaggregated & Vendor-Generic APUs with CXL

Victor Fritz Trost
Advisor: Nicolò Carpentieri, Anatole Lefort
Chair of Computer Systems
https://dse.in.tum.de/

Motivation: Memory bottleneck in GPUs

- **Increased VRAM demand**
  - AI models, data sets keep getting bigger, **can't always fit in VRAM**

- **Inefficient *copy-then-execute* model**
  - Requires workload-specific data splitting and partitioning

- **Inefficiency** of Unified Virtual Memory (**UVM**):
  - Assisted through software

Which path to a **more efficient UVM**?

Motivation: APUs (Accelerated Processing Units)

APU: CPU-GPU tightly integrated on a single die

- Provides a **“hardware UVM”**: efficient and convenient solution for GPGPU
  - Developed to make GPU memory more efficient

- Currently **limited to a single platform**, or single chips
  - AMD Instinct MI300a: Single chip with integrated memory
  - Nvidia GH200, NVLink C2C: Separate chips, but single (proprietary) platform
  - NVLink Fusion: Nvidia + multiple OEMs, but not yet available

- APU-like design cannot be applied to arbitrary CPU, GPU pair

Could we **couple any CPU, GPU** into a ***logical* APU?**

Research Gap: CPU-GPU coherent memory with CXL

- **CXL** is promising for a generic APU design
  - Enables **cache-coherent** CPU-GPU **memory sharing**
  - **External protocol:** can connect discrete (off-the-shelf) CPU, GPU devices
  - Open standard

<u>But:</u>

- No existing CXL CPU-GPU coherent hardware
- No simulation models

We don’t know what performance to expect from a CXL-based APU

Research Statement

Can an APU-like CPU/GPU interface be efficient with CXL?

Proposal

We need to:

- Model performance of CXL-based APUs:
  - estimate **slowdowns of remote memory** for GPU kernels
  - estimate **gains with CPU-GPU cache coherence**

Thus we propose:

A simulation model for APUs with CXL memory

- **Goal #1 Programmability:** Single address space
- **Goal #2 Efficiency:** Hardware cache coherence
- **Goal #3 Genericity:** Platform agnostic

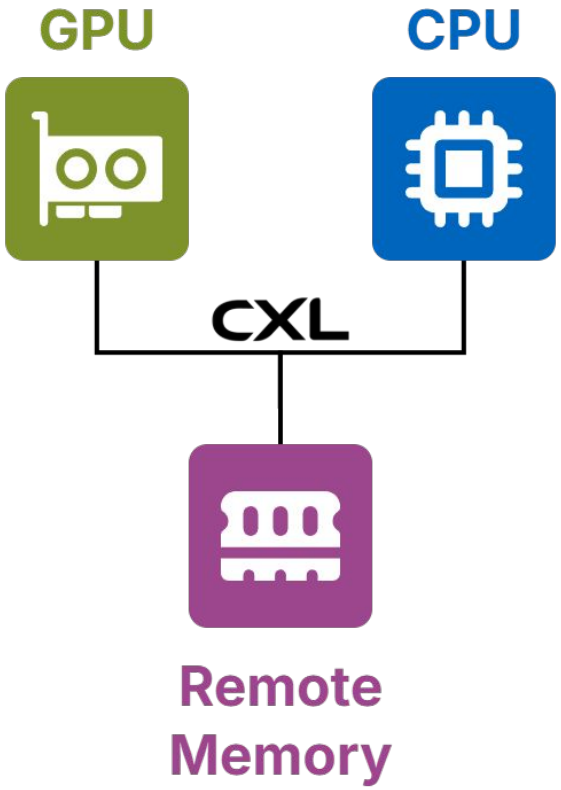
Design

- **Efficient UVM:**
  hardware based

- **Unified Memory Pool:**
  scalable beyond VRAM limits

- **Platform independent:**
  not locked into a specific vendor

- **CXL-based**
  disaggregation-ready



Background

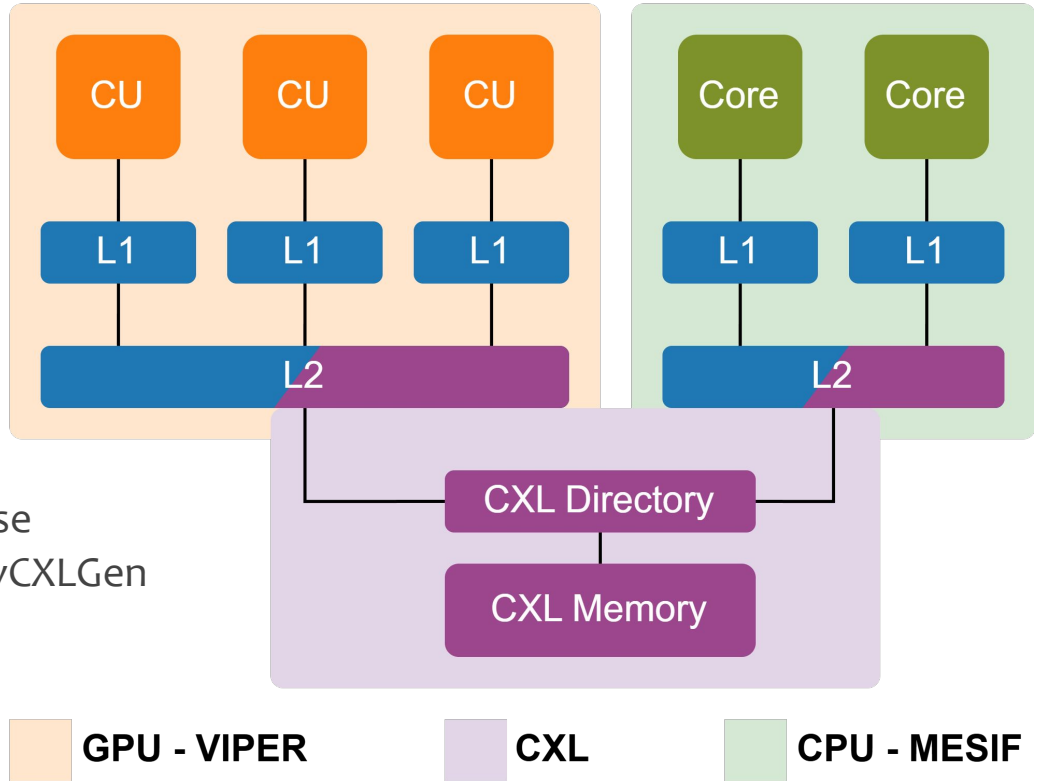
State-of-the-Art GPU simulations:

- **GPGPU-Sim:**
  - Simulates NVIDIA GPUs, limited CPU/UMA simulation, focus only on GPU kernels
- **Accel-Sim:**
  - Based on GPGPU-Sim, but trace-based, similar drawbacks
- ***gem5*'s GPU models:**
  - Simulates AMD GCN, **out-of-the-box support for APU**

Quick proof-of-concept by extending *gem5*'s APU model

Implementation - Overview

- **CPU-side: MESIF** protocol
  - Generated using Synapse
- **GPU-side: VIPER** protocol
  - Already in gem5, extended
- Both protocols **bridged** to CXL
  - CXL domain generated with Synapse
  - MESIF-CXL bridge generated with vCXLGen
  - VIPER-CXL bridge hand-written



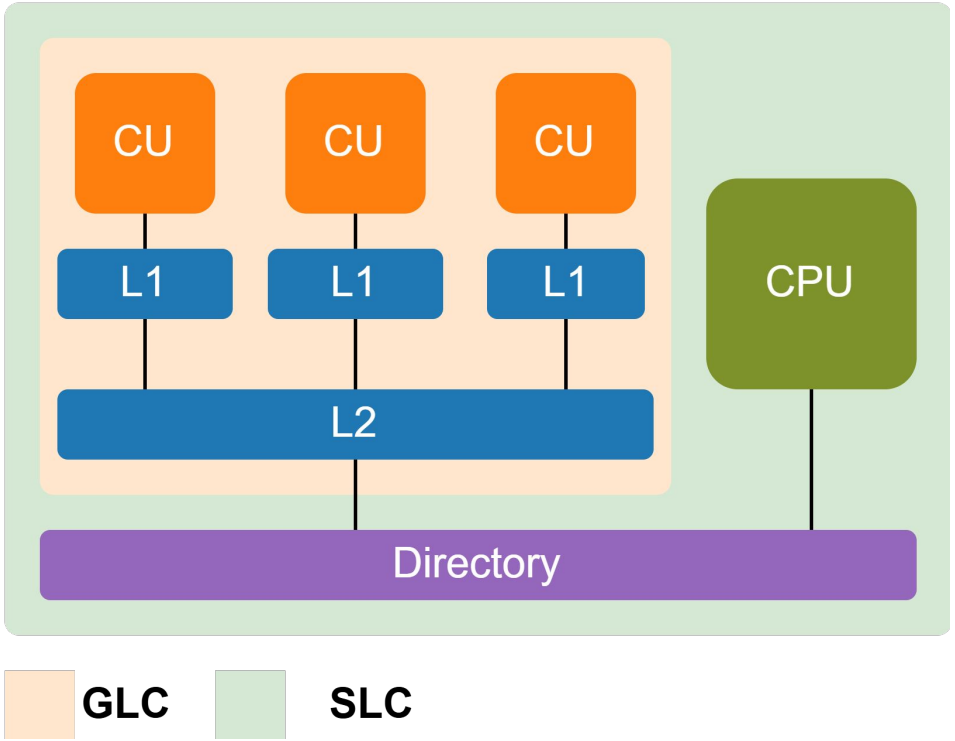
CPU Implementation - MESIF

Why MESIF?

- **Forwarding (F):** cache-to-cache sharing to minimize latency
- Intel uses MESIF-like protocol in multi-socket (NUMA) servers
- We use **directory-based** approach
  - Simpler to generate bridge, better scalability
- Gem5 doesn't have MESIF
  - We write MESIF specifications and generate it with Synapse PCC-to-SLICC compiler

GPU Implementation - VIPER

- VIPER provided by *gem5*
- **Adapt** only **L2** to CXL-bridge
- CXL bridge implementation:
  VIPER is a scoped protocol,
  L2 data consistency changes wrt. access scope:
  - SLC: system-wide (incl. CPU)
  - GLC: only GPU-wide
- Uses **SLC** and **GLC** scopes
  - Introduce a new incoherent P state in CXL to model GPU-wide coherent data (GLC)



Evaluation - Setup

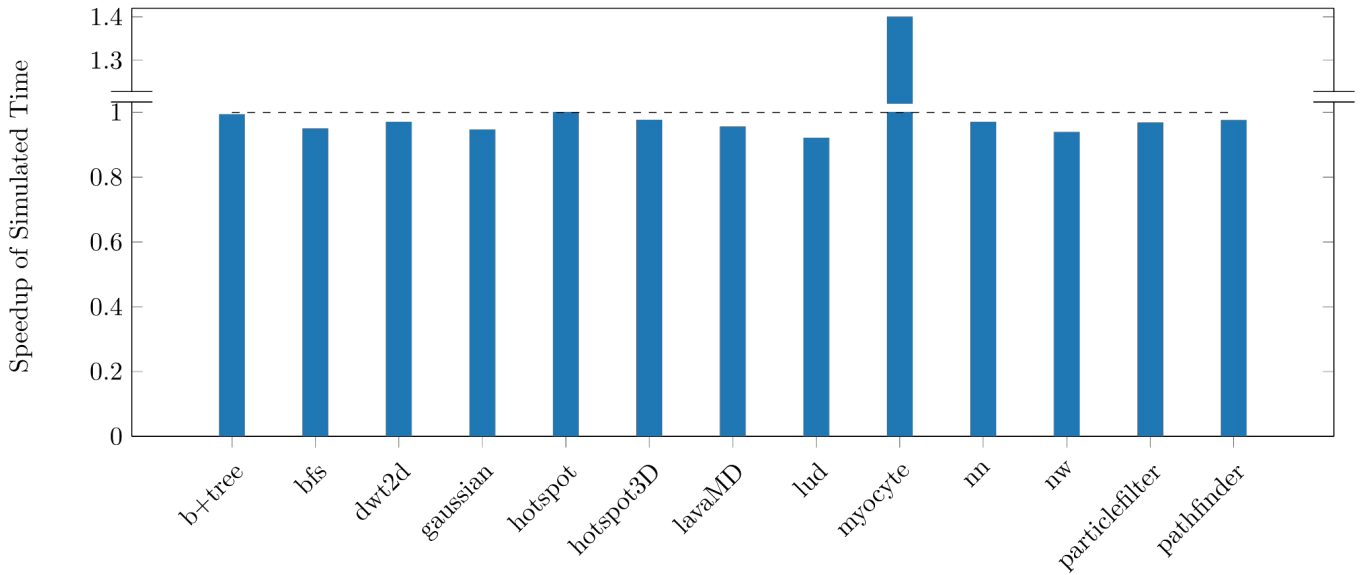
How does the CXL-based disaggregated APU compare to the integrated APU chip?

- Use *Rodinia* benchmark suite
  - Heterogeneous benchmark suite focusing on GPUs
  - Cross platform and domain
  - Implemented in CUDA
  - We translate to HIP (AMD GPGPU stack) with the `hipify` tool

- Use original *gem5* model as baseline (AMD VIPER APU)
  - Original GPU latency as CXL latency (60ns)
  - Original model uses local DRAM -> faster CPU accesses than with CXL memory

- 4 CPU cores (MESIF), 4 GPU CUs (VIPER)

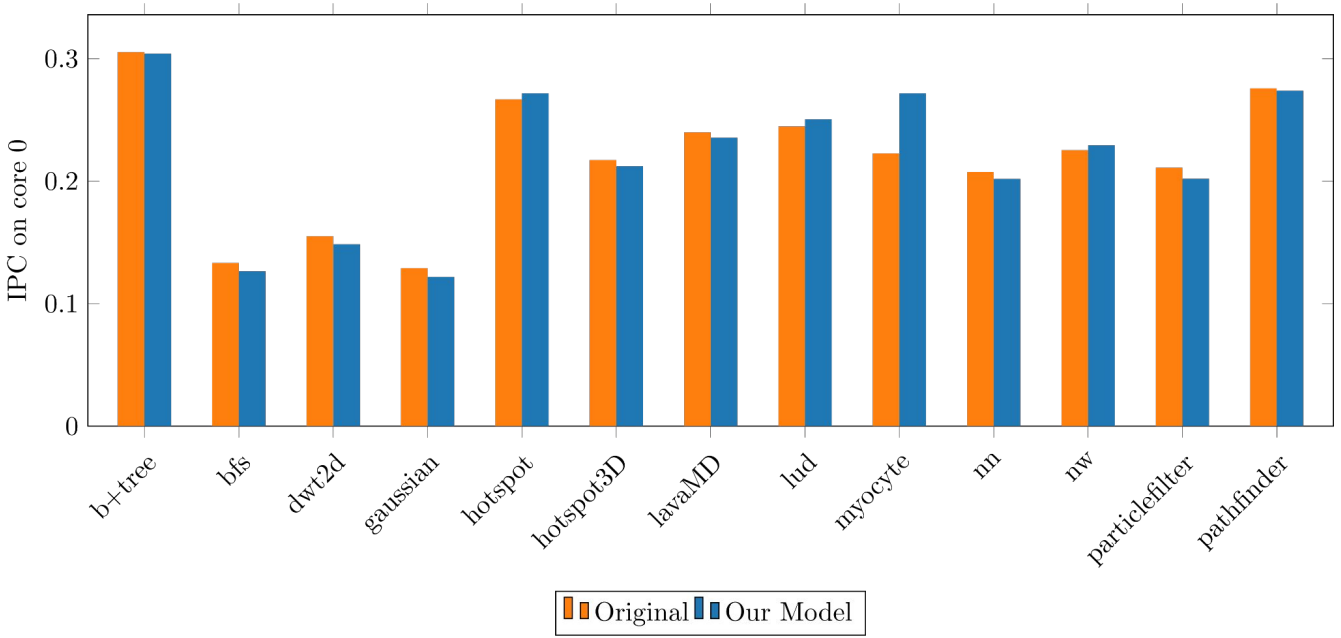
Evaluation - Overall



At similar latencies to original runtime

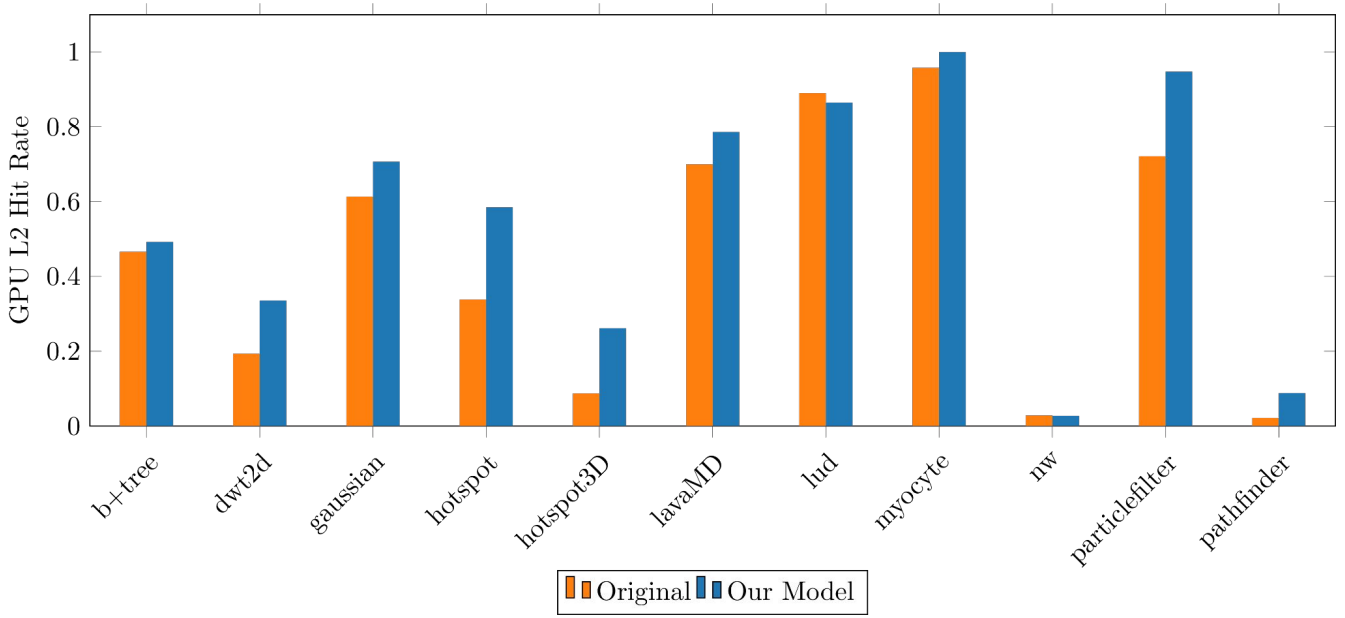
→ Comparable performance to original VIPER implementation

Evaluation - CPU



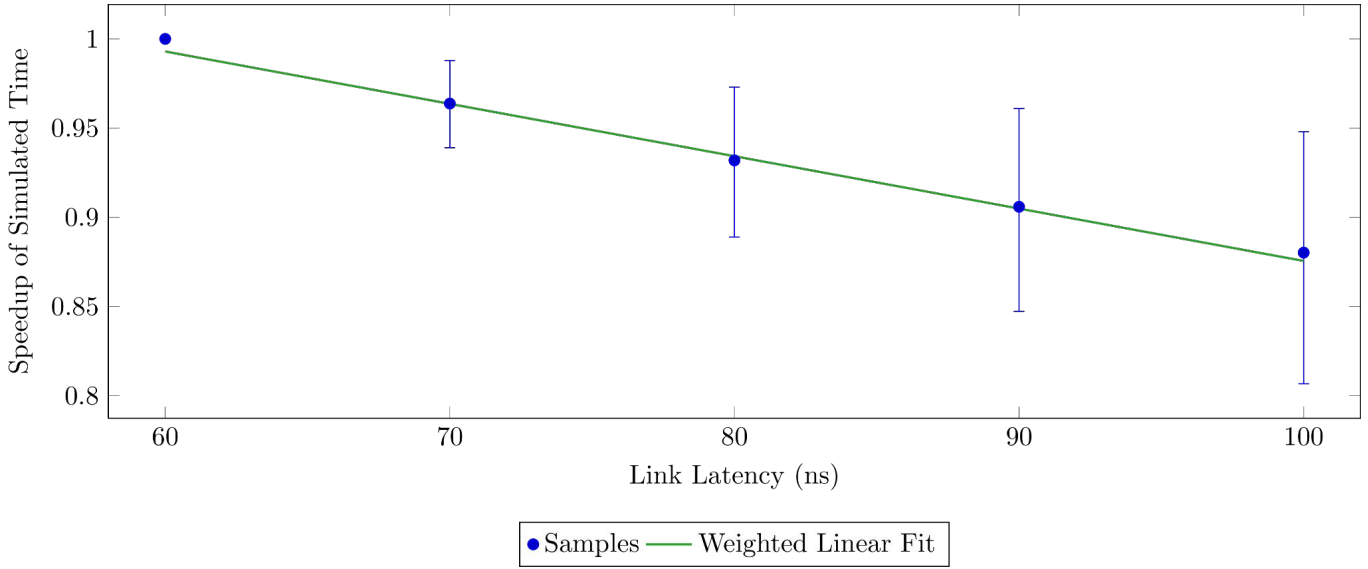
- Slightly worse IPC w/ CXL
- Due to higher memory latency with CXL (from CPU), original uses a different topology
- Original “bridges” GPU’s L2 directly to CPU’s LLC

Evaluation - GPU



- Overall higher hit rate
- Expected due to write-through nature of original VIPER

Evaluation - Link Latency



- Linear slowdown with increasing CXL link latency
- Performance degradation comprised between 5-20% for 1.5x link latency

Conclusion

We achieve:

- A generic APU using CXL shared memory
- With well-scaling performance, similar to *gem5*'s original APU

Try it!
github.com/TrostV/cxl-apu

CXL-based APUs are as efficient as integrated APUs

CXL emerges as a valuable option in the VRAM limited AI-era.