EuroHPC RISC-V activities at Chalmers

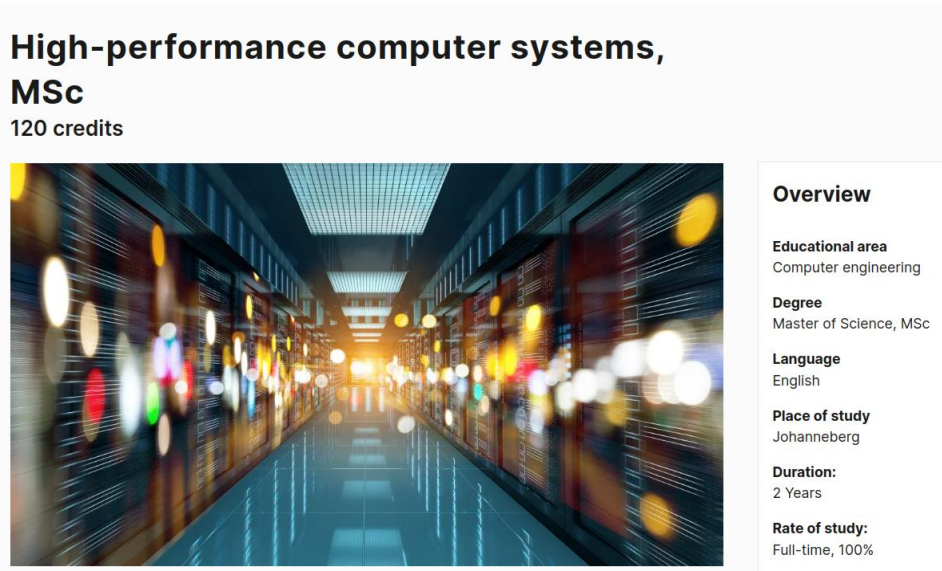
Miquel Pericas and Bhavishya Goel
Dept. Computer Science and Engineering
Chalmers University of Technology

Gbg RISC-V meetup #3, Feb 20, 2024

Who we are

- CHART: Chalmers Heterogeneous Architectures and Runtimes Team

- Research Goals: Performance, Efficiency
  - Focus: High Performance Computing, Supercomputing, + some AI & embedded
- Master Program in High Performance Computer Systems
  - https://www.chalmers.se/en/education/find-masters-programme/high-performance-computer-systems-msc/



Why is the EU interested in RISC-V?

- EU has interest in:
  - development of autochthonous HPC industry
  - need to increase HPC capacity + development of skills
  - open-source hardware, to complete the stack (below the kernel still closed)
- Why RISC-V?
  - Only option for Europe to try to regain position in computing
    - x86, ARM are under control of non-EU entities
  - RISC-V as "language" for academia to communicate ideas to industry
- Note that RISC-V is an Open Standard, not same as Open Source

Panel on EU & RISC-V in RISC-V Summit Europe 2023
https://www.youtube.com/watch?app=desktop&v=7cLaO-MFY2s&ab_channel=RISC-VInternational

EuroHPC and RISC-V strategy

- EuroHPC Joint Undertaking https://eurohpc-ju.europa.eu/index_en
  - *"EuroHPC JU is a joint initiative between the EU, European countries and private partners to develop a World Class Supercomputing Ecosystem in Europe."*
- Funds big supercomputers, open for use by researchers
  - e.g. LUMI in Finland, Top #5 supercomputer 530 PetaFLOPS (peak)
- EuroHPC R&I projects in which Chalmers participates
  - **European Processor Initiative** (2019-2026)
    - Goal: building ARM host processor and RISC-V based accelerator (EPAC)
  - Development of pilots: **EUPILOT** and EUPEX (2021-)
    - EUPILOT focus on RISC-V accelerator for HPC (focus of this presentation)
  - **eProcessor** (2021-)
    - Developing RISC-V processor with extensions for HPC and Bioinformatics

European Processor Initiative

THE EUPILOT

eProcessor

Pilot using Independent Local & Open Technologies

The European PILOT

HiPEAC – Project Overview

Based on slides by Carlos Puchol – Barcelona Supercomputing Center – January 18, 2024

The European PILOT project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No.101034126. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Italy, Switzerland, Germany, France, Greece, Sweden, Croatia and Turkey.

www.eupilot.eu

19 Partners

Academia + Research Institutions

Industrial Partners

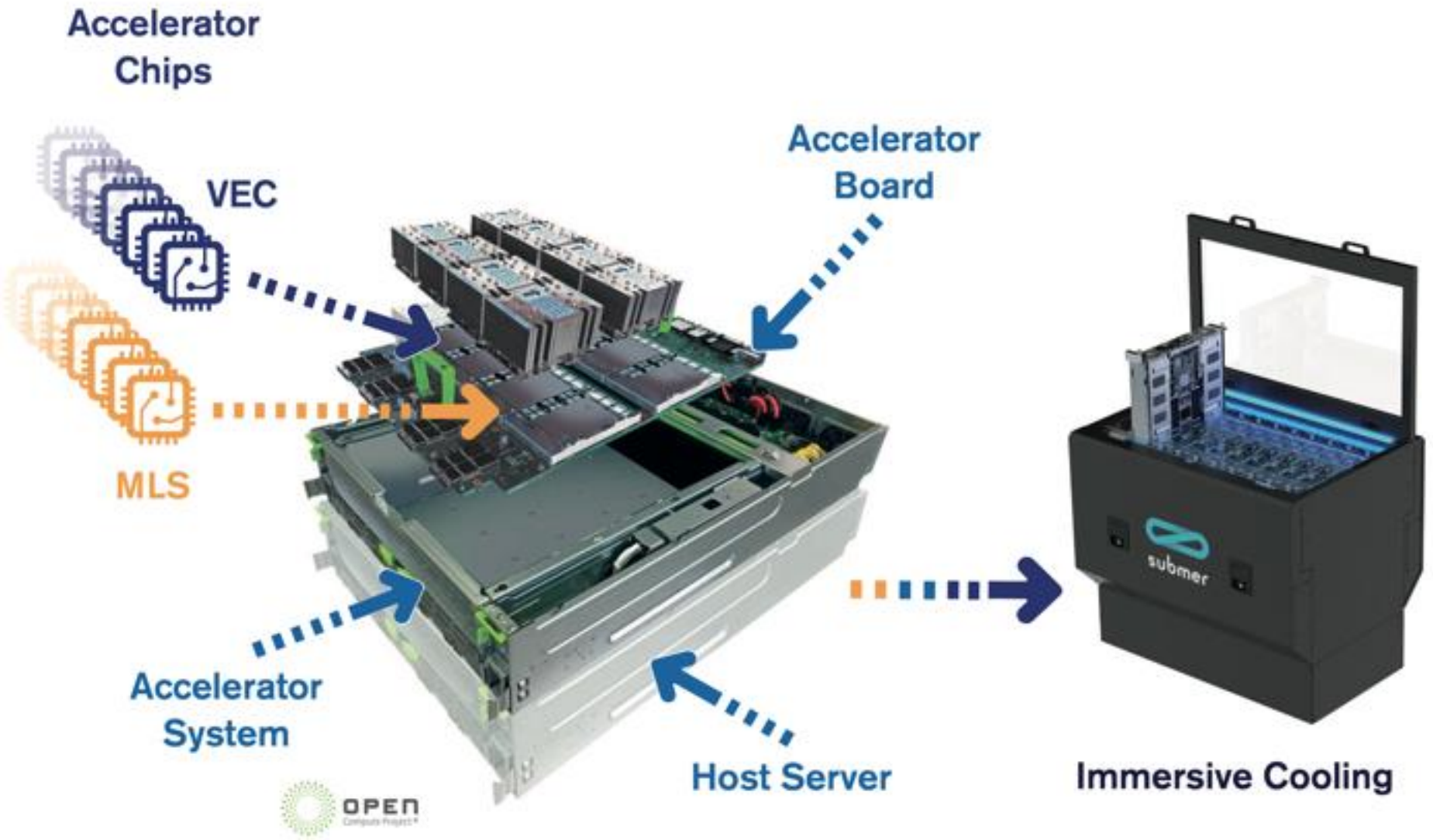
Objectives

- **Demonstrate a European pre-exascale accelerator platform**

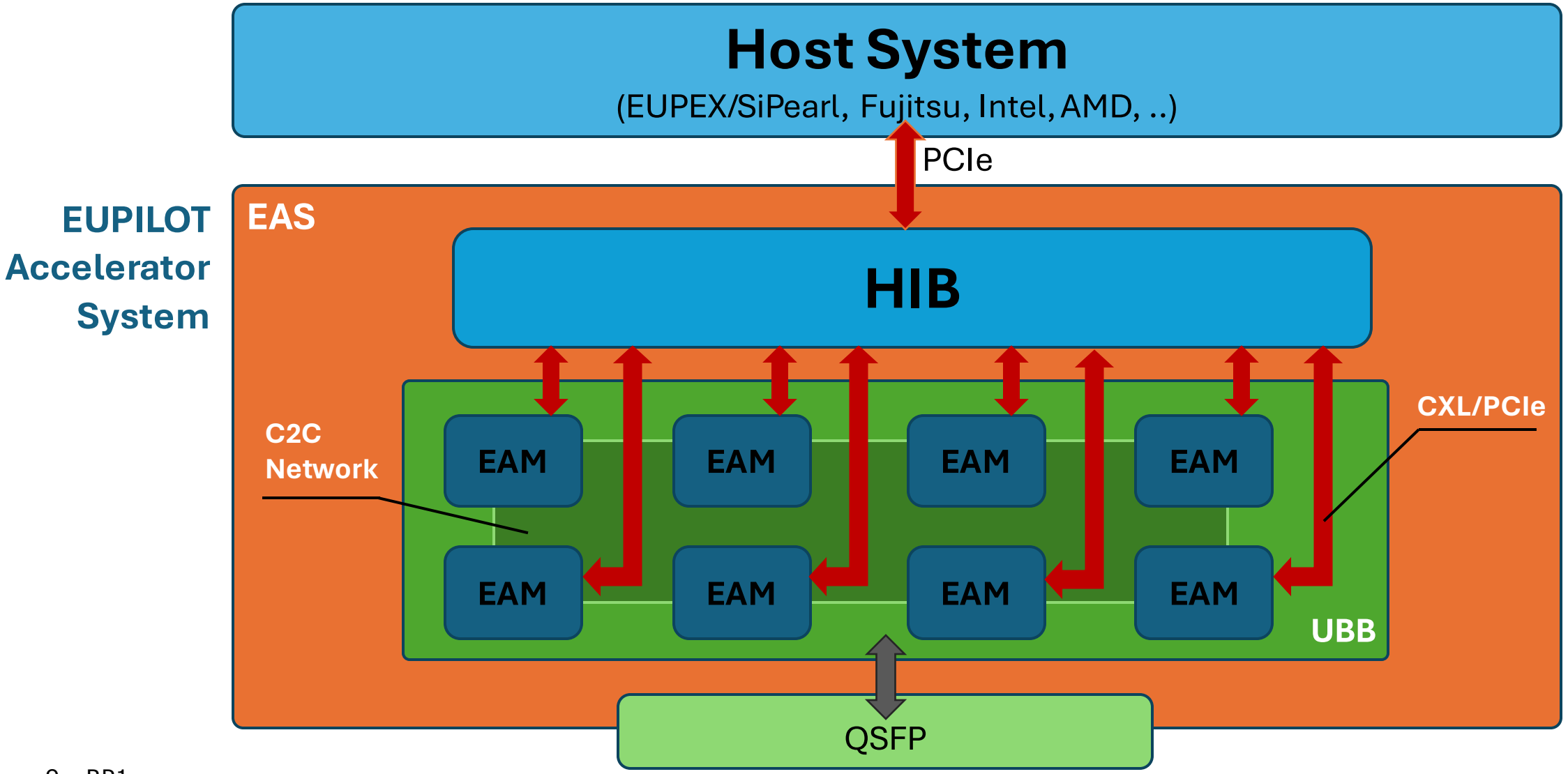
- Design, validate, build and deploy a fully-integrated accelerator platform
- Maximize use of European technology & assets
- Stimulate European collaboration, enable future exascale systems
- SW/HW co-design for improved performance and energy efficiency
- Further extend open source into hardware for HPC
- Leverage open source and the RISC-V ISA

- **Strengthen European digital autonomy and supply chain**

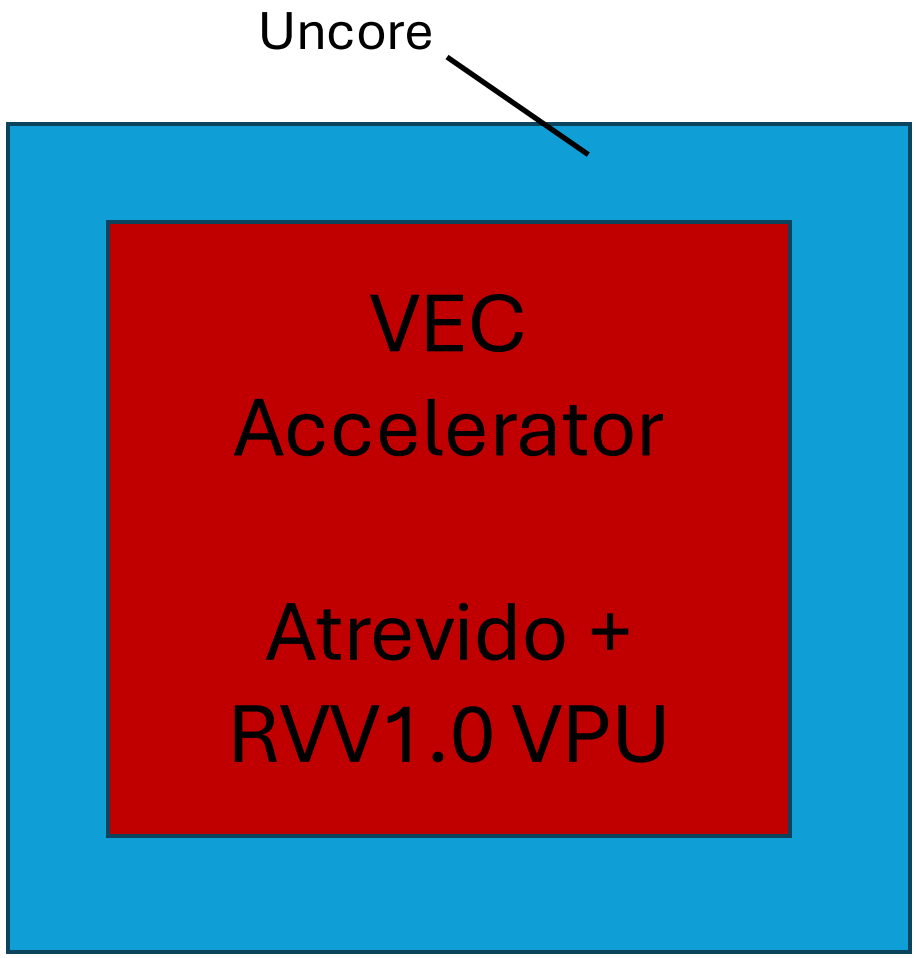
Top Level View



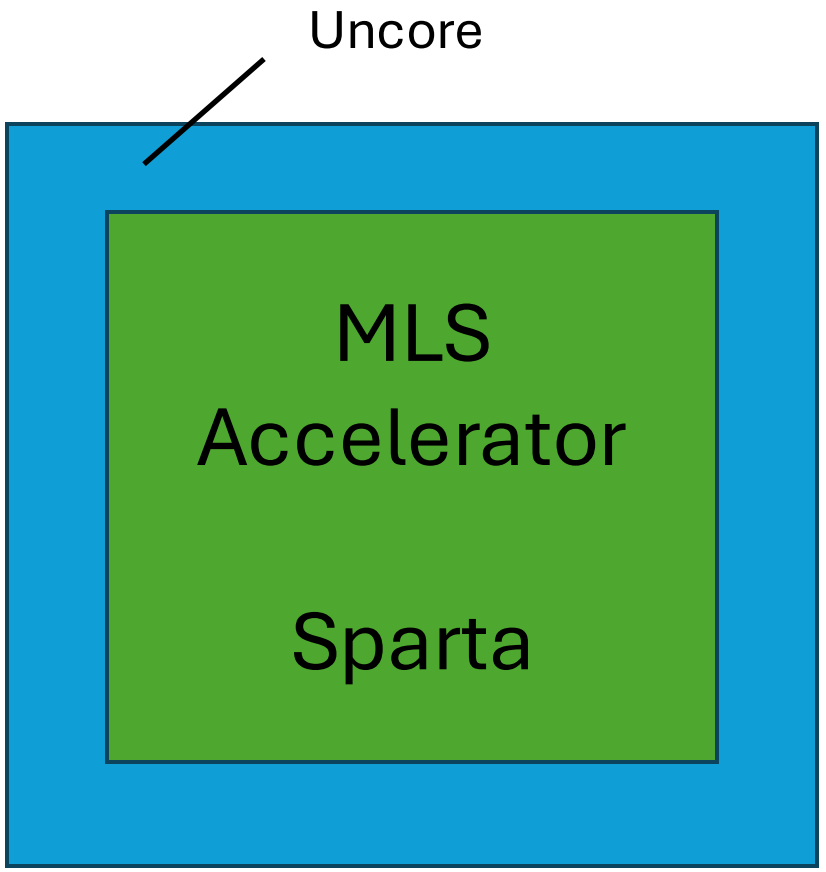
EUPILOT System Architecture



EUPILOT Accelerator Chips



45~50* mm²

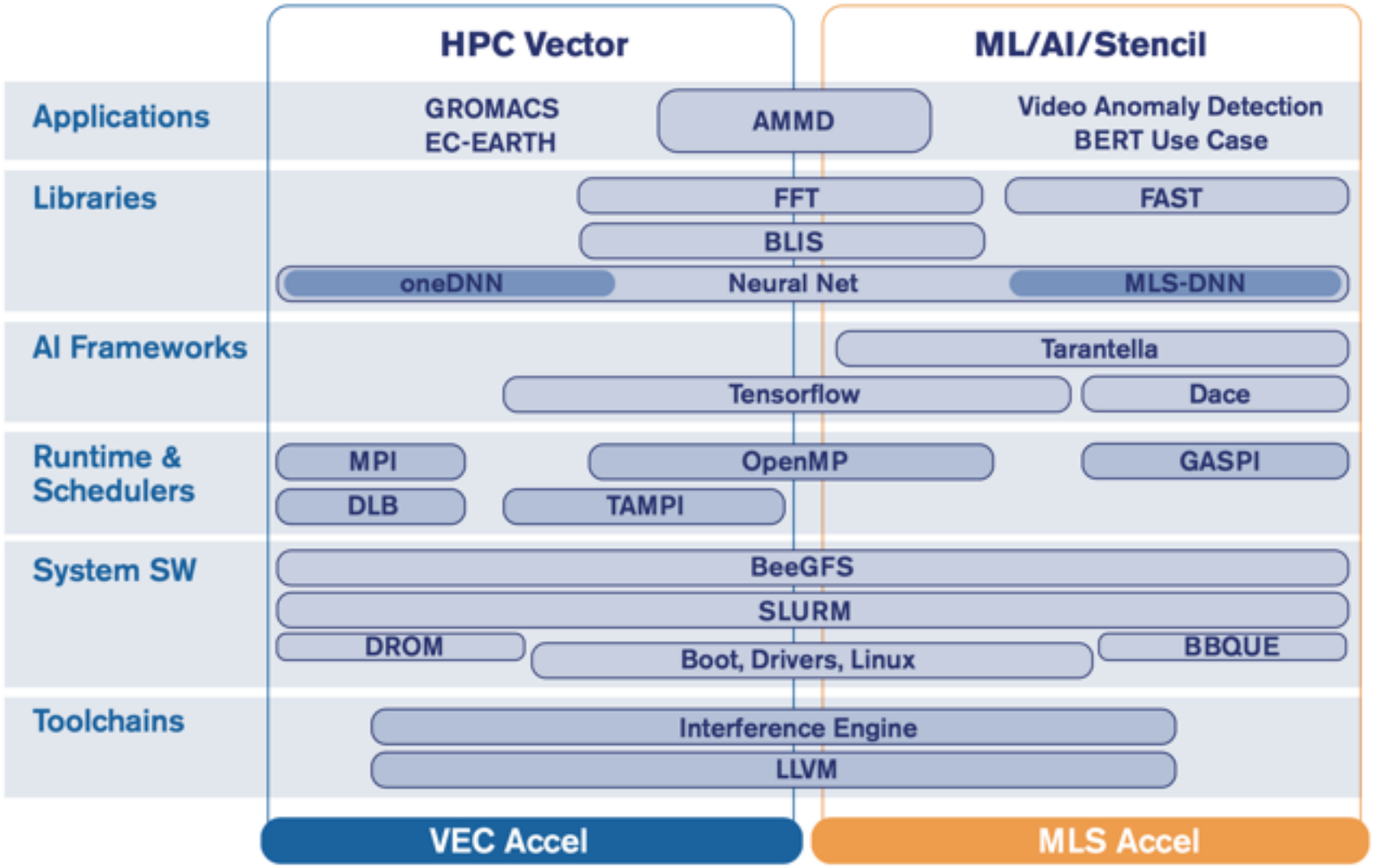


~40* mm²

12nm GF 12LPP process

* subject to budget constraints

Software Stack



EUPILOT Peak Performance Projections*

VEC

- 16-tile area budget of ~46mm^2
- Each VPU has 8x FPUs
- 2x FLOPs for FMA
- 64-bit ops
- Target at 1.5GHz
- 16x8x2x1.5 = 384 GFLOPs/chip
- w/RISC-V cores: 432 GFLOPs/chip

~3.5 TFLOPs/EAS (64-bit)

MLS

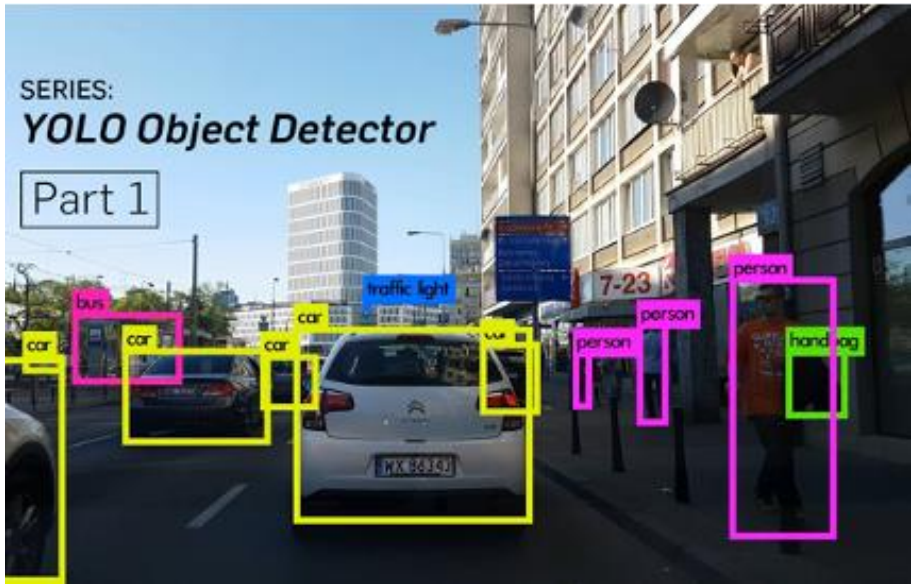
- Mid-area budget of ~35mm^2
- 2 Groups, 6 clusters
- Of (8+1) cores = total 108x FPUs
- 8-bit FP ops
- Target at 1GHz
- ~768 GFLOPs/chip

~6.1 TFLOPs/EAS (8-bit)

Activities related to EUPILOT

- **Our focus is on the VEC chip**
- **We will highlight three activities related to EUPILOT:**
- **Software**
  - Inference on VEC: fast vectorized convolutions
  - OpenMP runtime for VEC: vectorization of barriers and reductions
- **Hardware**
  - Cache coherence, both intra-chip, and inter-chip, based on AMBA CHI

Inference on VEC: CNNs



Inference via Convolutional Neural Network (CNNs) require high throughput and low latency

Image credit: https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/

Vector processors can offer

- low latency
- high performance
- energy efficiency

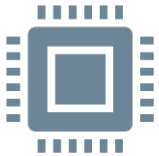
- Can we use long vector architectures (eg RISC-VV)?

Sonia Rani Gupta, Nikela Papadopoulou, and Miquel Pericàs. 2023. *Challenges and Opportunities in the Co-design of Convolutions and RISC-V Vector Processors*. In Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W '23).

Objective

Algorithmic Optimizations:

- Utilize the vector unit and vector registers effectively.
- Vectorize the **Winograd** algorithm by leveraging the available EPI RISC-V intrinsics



Hardware Parameters Tuning

- Vector unit: how long should vector lengths be?
- Caches: how large should caches be for different vector lengths?

Motivation: RISC-V and Vector Spec

- Why vector?
  - Decode 1 instruction, execute N operations
  - Higher Performance and Energy Efficiency
- RISC-V Vector Extension (RISC-VV) supports Vector Length Agnostic (VLA) programming style
  - Vector ISA is decoupled from the Vector Implementation
  - Unlike e.g. x86 where ISA (AVX2, AVX512) determines HW vector length (256b, 512b)
  - RISC-VV supports very long vectors (up to 16384b)
- Can we use Long Vector RISC-VV accelerators as an alternative to GPGPUs?
- Compiling applications to use vectors is a challenge:
  - Intrinsics, autovectorization, #pragma omp simd, potentially SIMT approaches

Tools

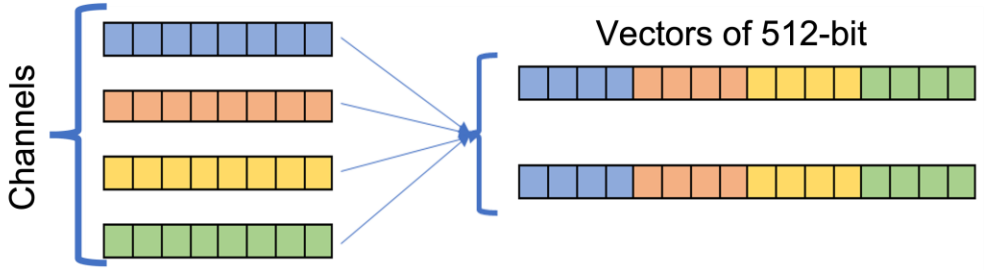
- Network models:
  - YOLOv3: 75 convolutional layers out of 107.
  - **VGG16: 13 convolutional layers out of 16**
  - Implemented in Darknet framework
- Algorithmic implementation
  - NNPACK library for Winograd implementation
- Hardware Exploration:
  - RISC-V Vector Extension: Gem5 Simulator*
- Compiler
  - RISC-V LLVM/Clang toolchain from the European Processor Initiative (EPI)

*Gem5 Simulator – plctlab. 2022. plct-gem5 (https://github.com/plctlab/plct-gem5/), supports v1.0 “V” extension with max VL of 4096 bits

Winograd: initial ARM-SVE implementation

Transformations:

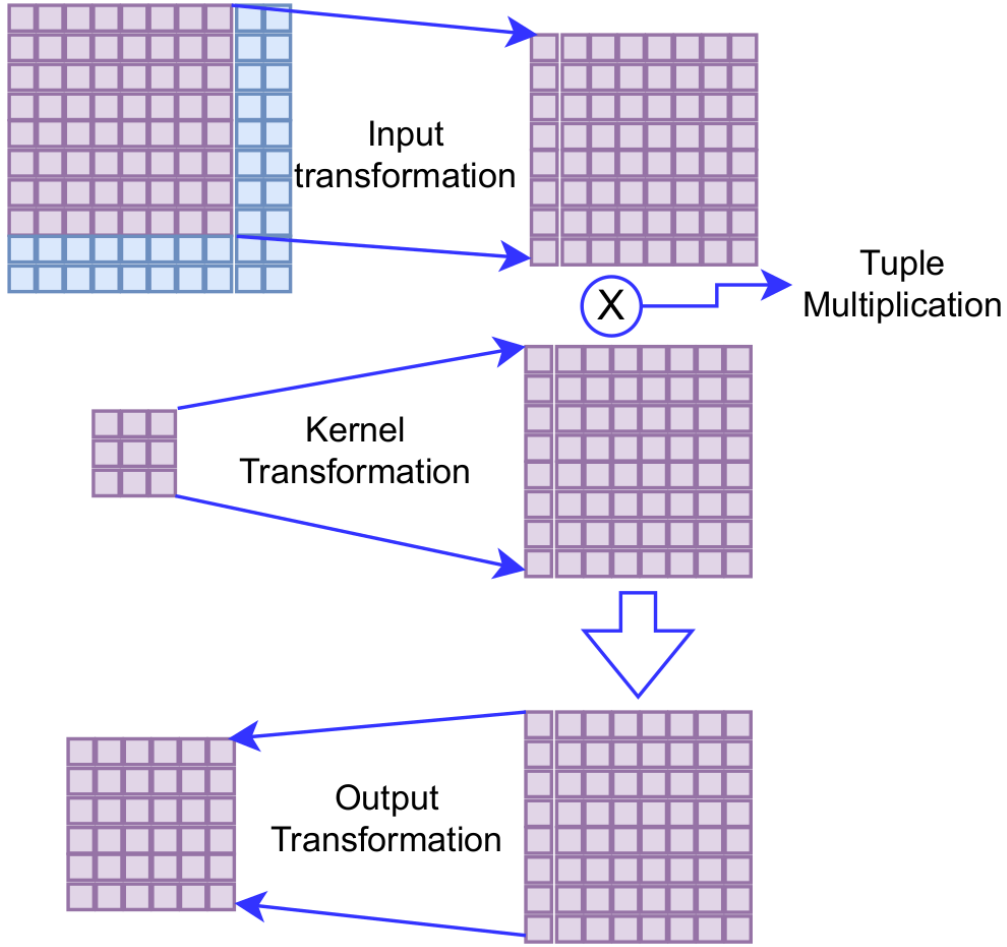
- 8x8 tile from one channel (NNPACK)
- Inter-tile Parallelism across the channels**
- Similarly, 32 channels to utilize 4096-bit VL



1 row of 8x8 tile from 4 channels

Tuple multiplication

- Increase tuple size to 32 with 4 elements in each block to utilize longer vector length.

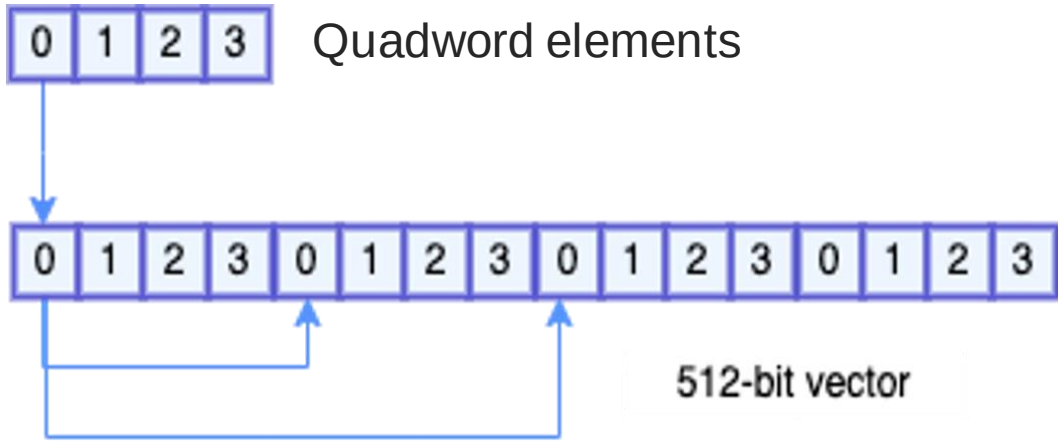


F(6x6,3x3) -> m+r-1 x m+r-1 tile
[m= output, r =kernel]
6x6 output and 3x3 kernel size = 8x8 Tile

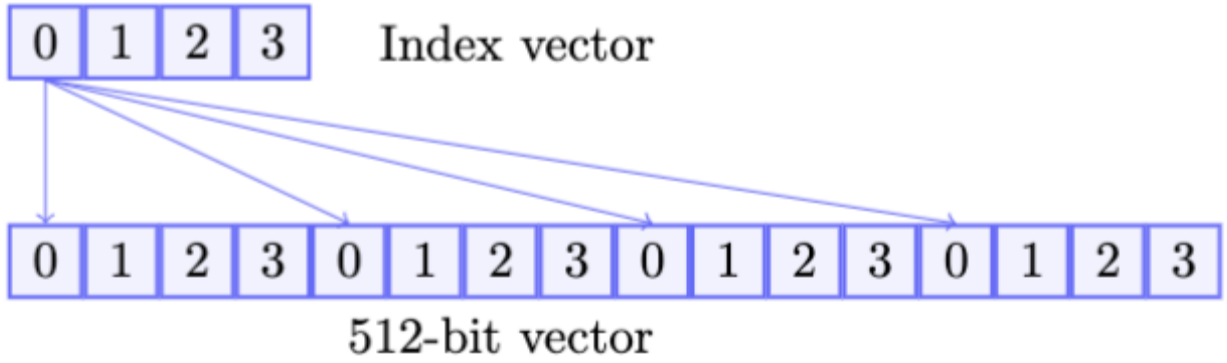
**Sonia Rani Gupta, Nikela Papadopoulou, and Miquel Pericas. 2023. Accelerating CNN inference on long vector architectures via co-design. In 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 145–155.

Challenge #1: Tuple Multiplication

- Operation: Load Quadword elements in a vector and replicate:
  - No specialized RISC-VV Instruction
- We test two alternatives
  - **Implementation 1: Indexed Load**
  - **Implementation 2: Slideup instructions**



Implementation 2



Implementation 1

Implementation 2 with slideup is ~2.3X faster than implementation 1 with indexed load.
Having specialized instruction likely to be faster, and reduce register pressure.

Challenge #2: Transformations – Transpose four vectors

- **Operation:** Transpose of 4 vectors in all transformations
  - Again, no RISC-VV instruction is available.
  - **EPI custom extension provides transpose with 2 vectors.**
  - **We tested two alternatives:**
    - **Implementation 1**: unit-strided store followed by Indexed load
    - **Implementation 2:** Strided store followed by unit-strided load

Example for transposing 4 vector registers having elements from 1 channel

No significant difference in performance with both implementations

Potential RISC-VV extension: vector transpose of 4 vectors, eliminates need for extra memory operations

Challenge #3: Transformations - Calling Conventions

Problem: Cannot pass references to vector registers as parameters to a procedure

- require intermediate vector registers to store the intermediate vector data.
- +require ~30 lines of code at 6 places in the input transformation kernel.
- Problems:
  - Register spilling
  - Less Programmability
- **Potential Workaround: Macros** can improve programmability, but it will still be required to have **intermediate registers. Hence the problem of register spilling will remain**

Being able to pass references to vector registers would improve programmability and reduces the chances of register spilling

VGG16: Analysis

VGG16:

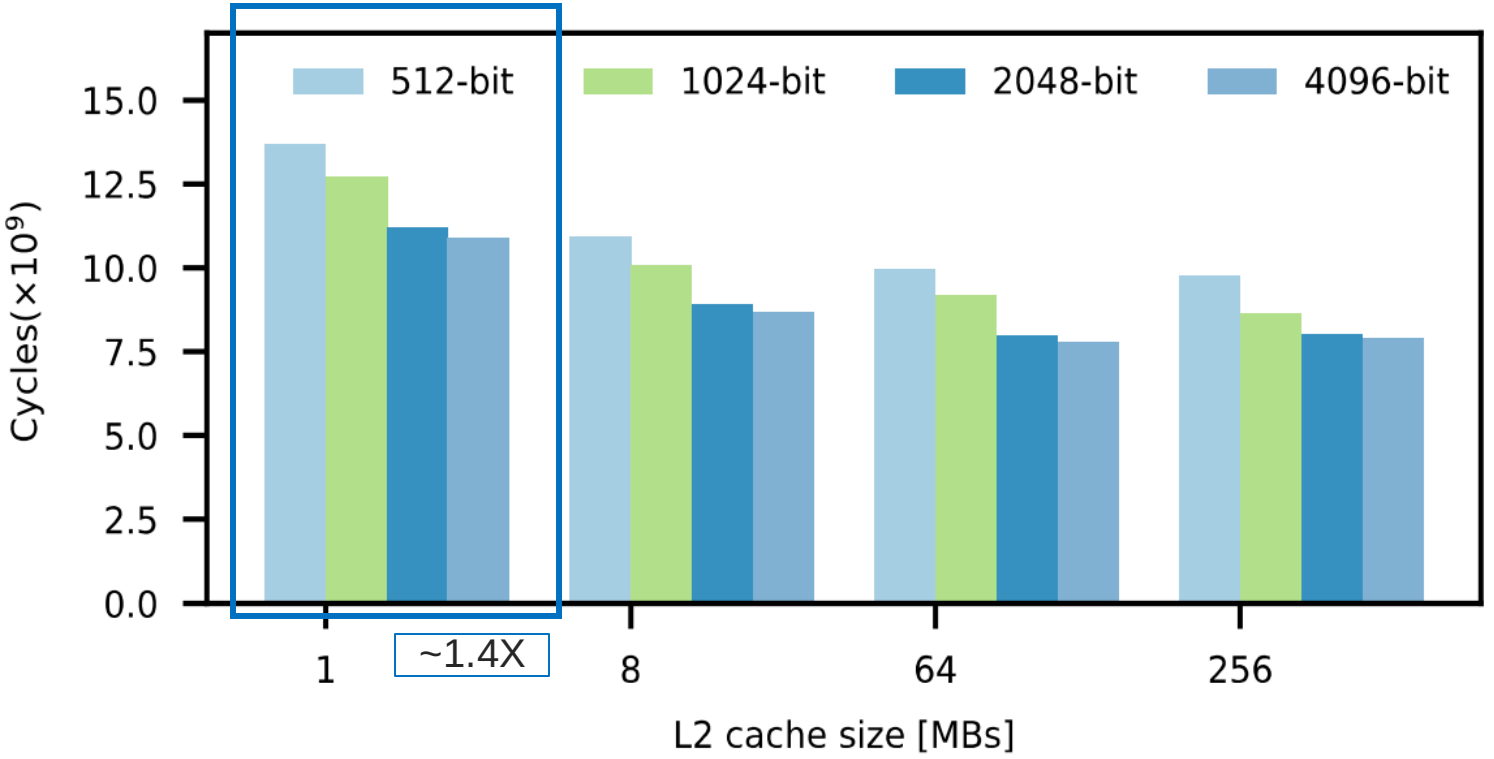
- 3x3 kernel size with stride 1: **Winograd**
- All the layers use Winograd algorithmic optimizations

Comparison with im2col+GEMM:

- 2048 bits VL and an L2 cache of 1MB modeled with gem5
- **1.2x performance improvement** compared to the **pure im2col+GEMM** approach.
- Similar performance compared to our optimized ARM-SVE implementation (on gem5)

HW Design Space: VGG16

Impact of vector lengths and L2 cache size with Winograd on RISC-VV@gem5 for VGG16.



Impact of Vector lengths:

- **No scalability beyond 2048-bit.**
- No significant difference in the number of instructions from 2048-bit to 4096-bit vector lengths

Impact of L2 caches from 1MB to 64MB:

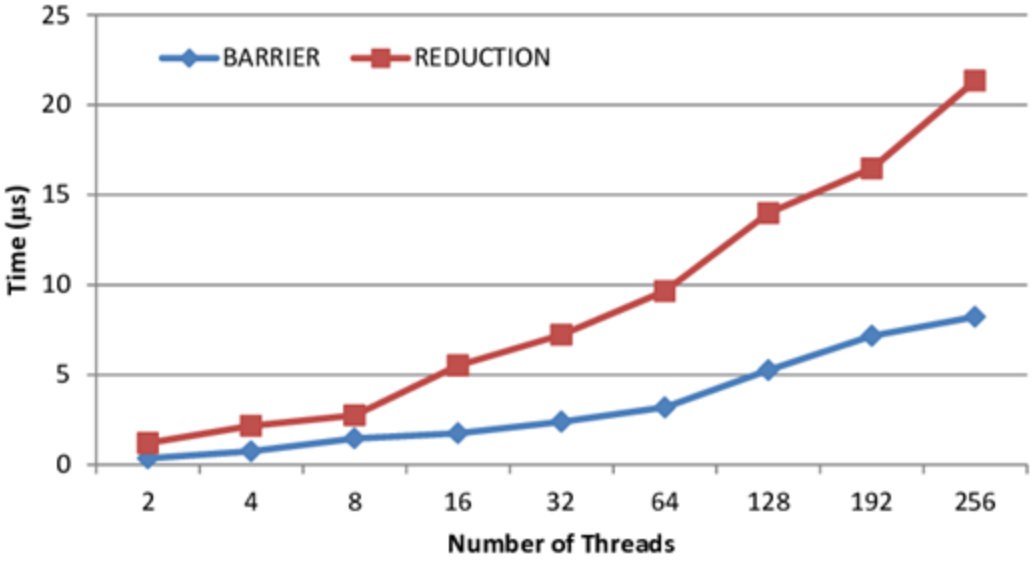
- ~1.3X performance improvement
- **No performance improvement beyond 64MB L2 cache**

2K vector length with 64MB caches can provide up to ~1.8x speedup

Current work: Integration of RISC-VV Winograd, im2col and Direct Conv into oneDNN. Potential contribution to EUPILOT Software Stack.

Vectorized Barrier and Reduction in LLVM OpenMP Runtime

- Barrier and Reduction commonly used operations in parallel programs
- Challenges
  - Resources wait idle
  - Overhead increases with increasing # of threads
  - # of cores/node are expected to increase at exascale
- Goals
  - Low overhead barrier and reduction in OpenMP
  - Utilize vector/simd unit to reduce overhead



LLVM OpenMP barrier and reduction overheads using the EPCC benchmark on Intel KNL

Muhammad Nufail Farooqi and Miquel Pericàs. 2021. Vectorized Barrier and Reduction in LLVM OpenMP Runtime. In OpenMP: Enabling Massive Node-Level Parallelism: 17th International Workshop on OpenMP, IWOMP 2021, Bristol, UK, September 14–16, 2021

Vectorized Barrier in LLVM

- **Data structure**
  - Shared array for each team of threads
  - Shared array is initialized to 1
  - Gather/Release -> see pseudocode
  - Tree pattern with configurable branching
    - shared array for each tree level
- **Implemented using Intrinsics**
  - RISC-V Vector Extension
  - Arm-SVE
  - Intel AVX
- Similar approach for Reductions, except that we also need to modify clang

Gather phase:

```
vec_array[thread_id]=0;
if(primary_thread){
  do {
    tmp_vec[1:VLEN]=0;
    for (i=0;i<nThreads;i+=VLEN) {
        tmp_vec |= vector_load(&vec_array[i],VLEN);}

  } while (tmp_vec);
```

Release phase:

```
if(primary_thread){
    vec_array[1:nThreads]=1;
} else {
     while(!vec_array[thread_id]);
  }
```

- VLEN is length of the vector unit
- nThreads are number of threads in a team

Evaluation

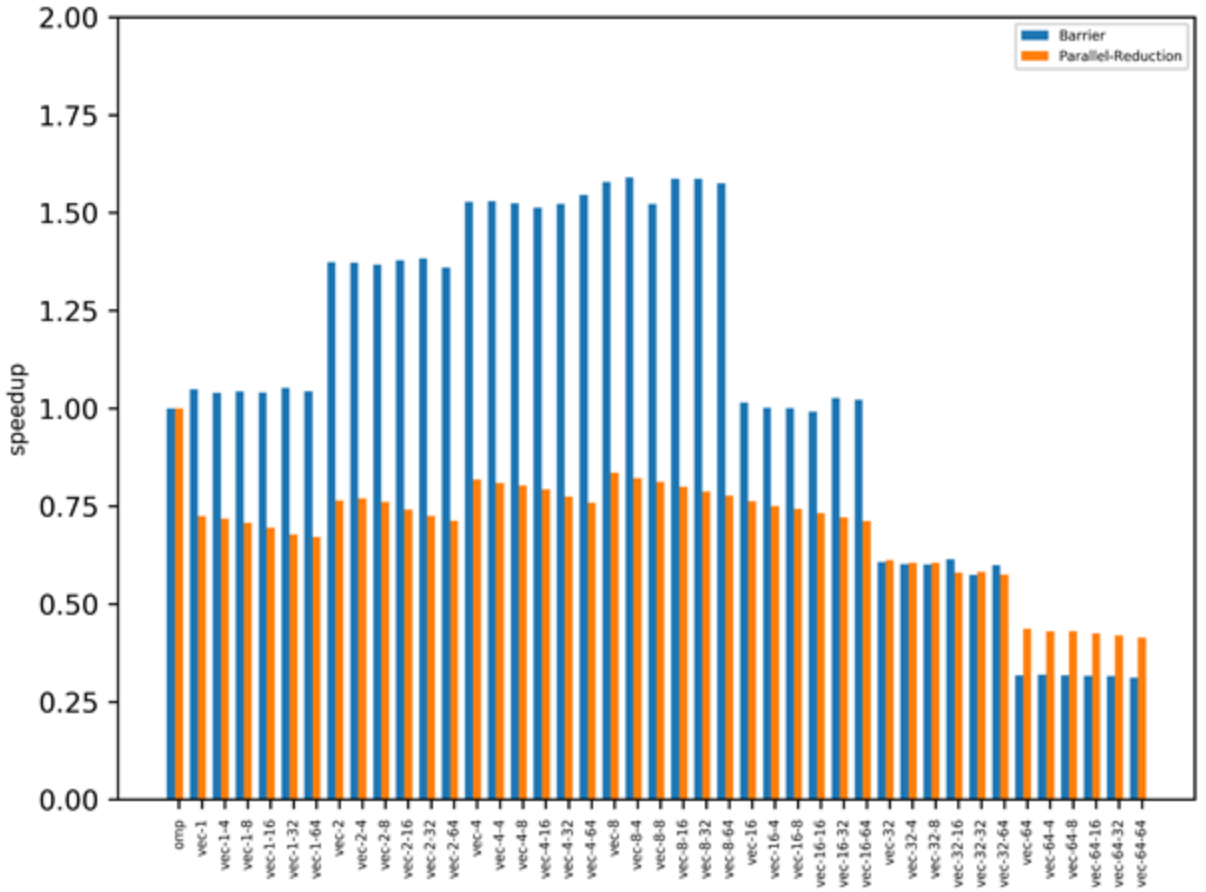
- OpenMP *parallel*, *barrier* and *reduction* microbenchmarks
- Baseline is LLVM OpenMP's default barrier and reduction
- Performance studies using Intel KNL and Fujitsu A64FX
- For RISC-V Vector Extensions, validated functional correctness using qemu+vehave (RISC-V vector emulator)

Table 1. Machine Specifications.

	Intel KNL	Fujitsu A64FX
Cores	68	48
L1	32 KB	64 KB
L2	34 MB (private)	32 MB (shared)
Memory	192 GB	32 GB
Bandwidth	90 GB/s	1 TB/s

Performance Results

- EPI FPGA VEC emulator did not support multiple cores (in 2021)
- Use Intel KNL as proxy: supports 512-bit vectors, 64 cores w/ 4x HW multithreading
- Performance: trade-off between false sharing and on-chip memory traffic
- Best performance achieved when max branching factor is used (=linear pattern).
- Max 2.2x speedup on Intel KNL for 128 threads
- Next: evaluate barriers and reductions performance on EUPILOT prototype

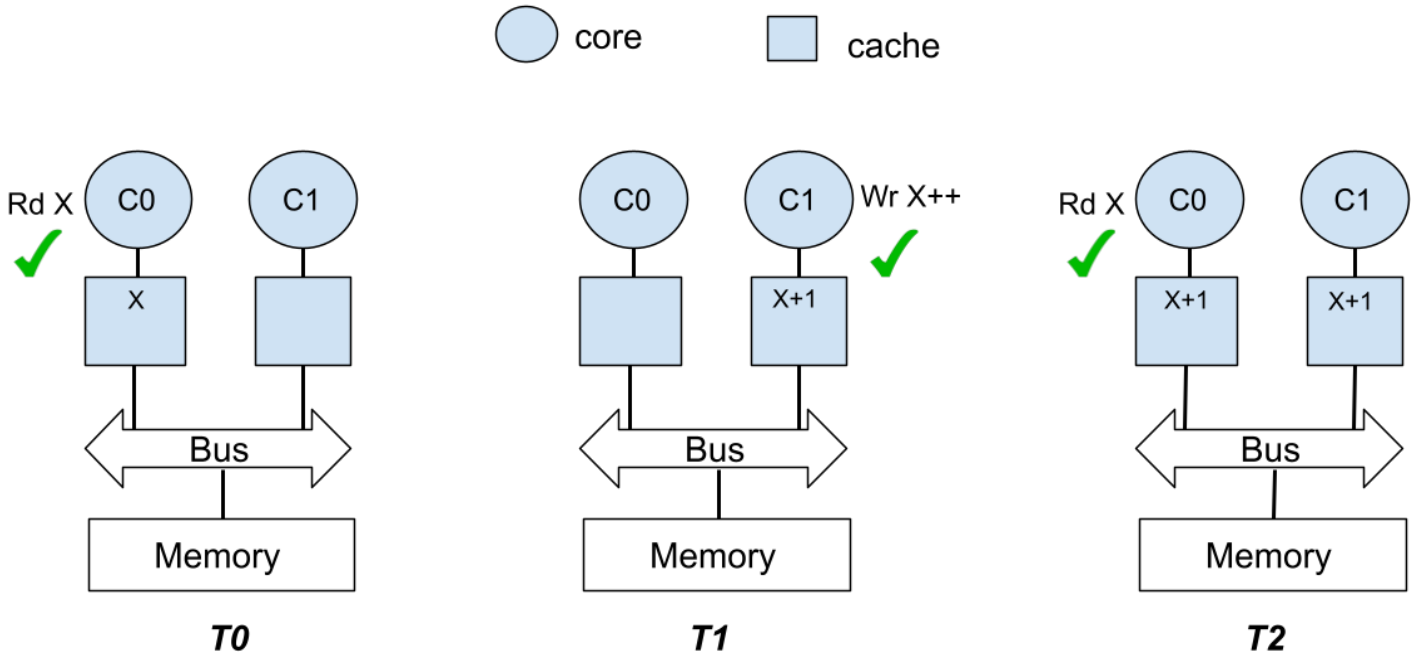


Performance of vectorized barrier and reduction with varying padding for barrier flags and reduction array (256 Threads on Intel KNL with snc4 cluster mode)

Cache Coherence Implementation in EuroHPC Projects

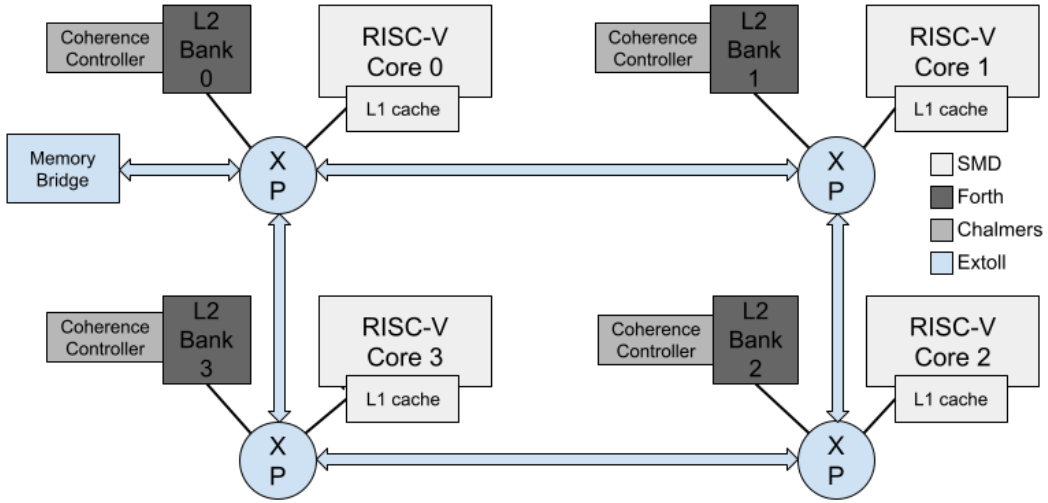
Cache Coherence Definition

“A memory system is coherent if the value returned on a LOAD instruction is always the value given by the latest STORE instruction for the same address”



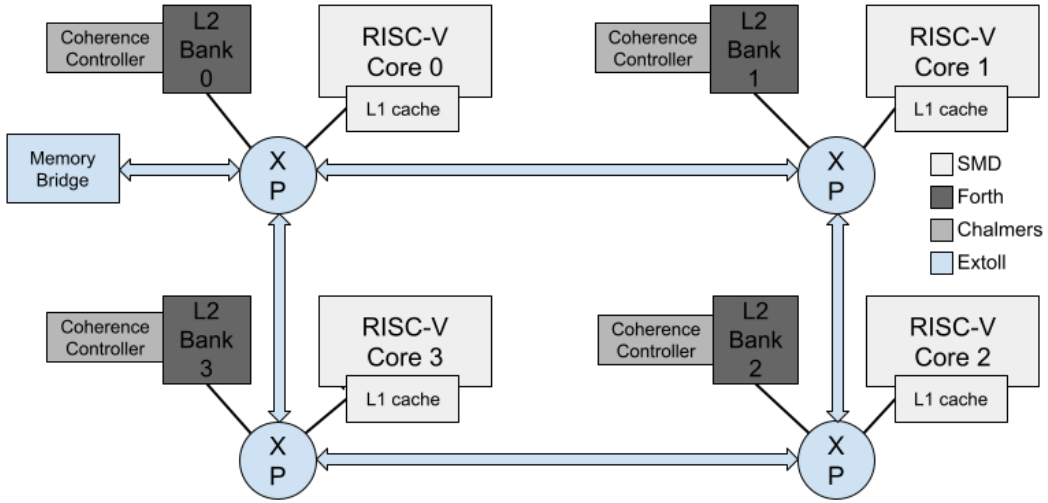
Cache Coherence in EPAC 1.0

- Coherent private L1 Caches and inclusive shared L2 Cache
- Distributed directory-based cache coherence controller co-located with address interleaved L2 banks
- AMBA CHI compliant coherent Network-on-chip



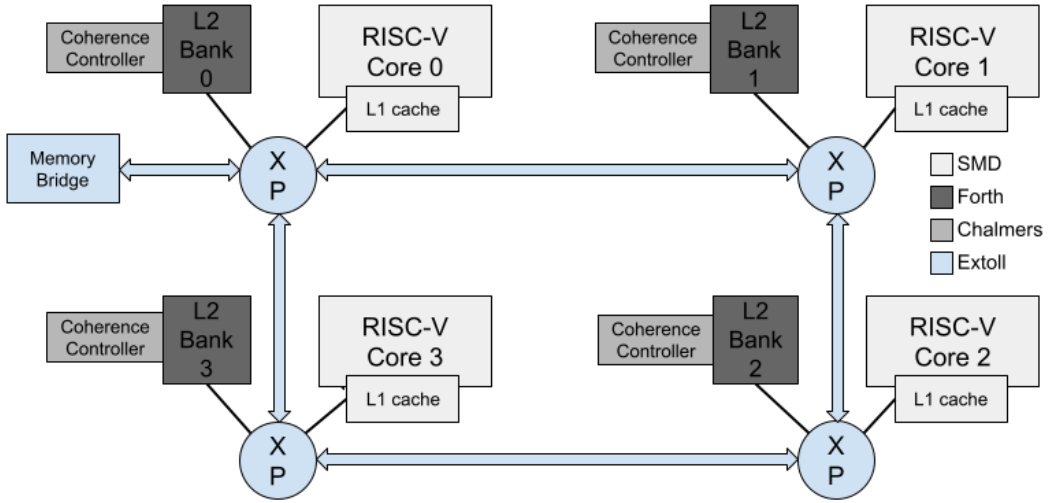
Cache Coherence in EPAC 1.0

- Two coherence modes:
  - Full coherence - for data cache requests
  - IO coherence - for instruction cache and vector processor requests



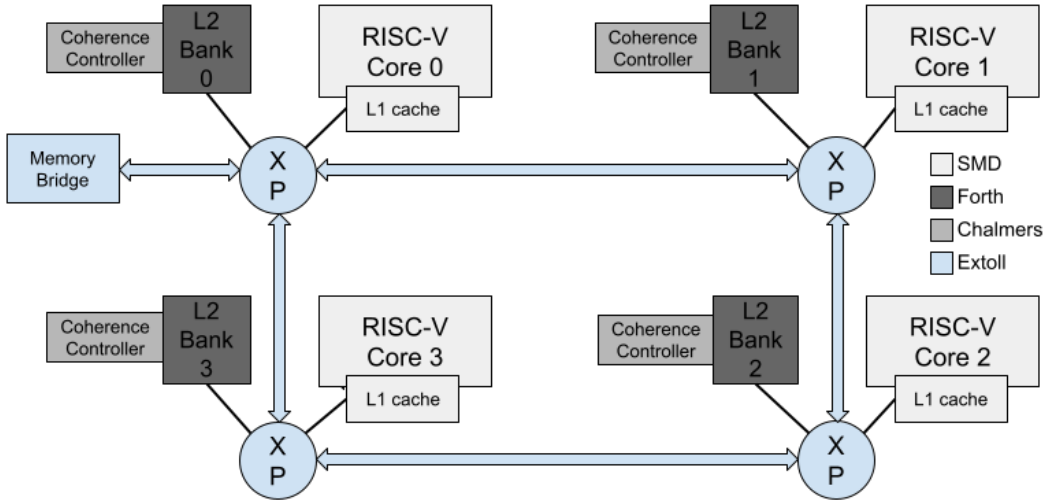
Cache Coherence in EPAC 1.0

- Synchronization support
  - Exclusive requests - for load-link/store-conditional
  - Atomics
    - AtomicLoad
    - AtomicStore
    - AtomicSwap
    - AtomicCompare



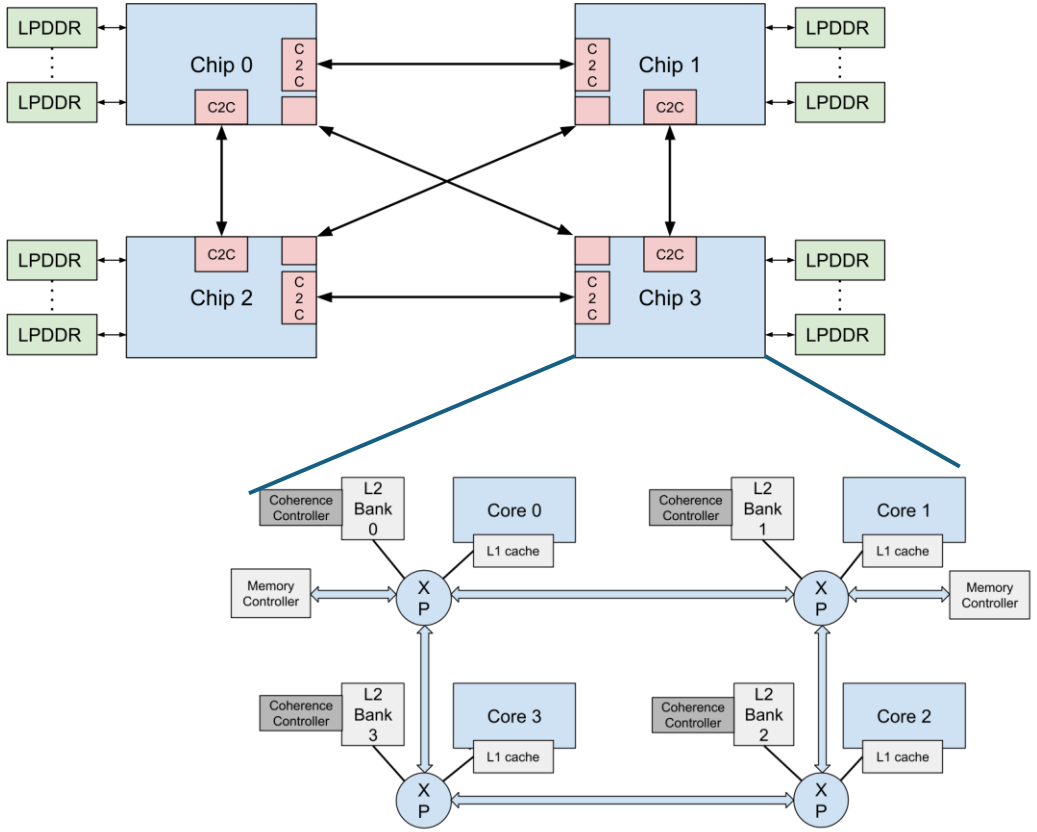
Cache Coherence in EPAC 1.0

- Cache Management Operations
  - Write dirty data back to the next level
  - Discard
  - Flush



Cache Coherence in EUPILOT/EPAC 2.0

- Intra-chip coherence
  - Up to 16 cores per chip
  - Support for cache-to-cache transfers and Direct Memory (Read/Write) transfers
  - Optimizations to reduce snoop traffic and latency
- Inter-chip coherence
  - Up to 8 chips - connected point to point
  - IO coherence
  - Full coherence
  - Mode selection at boot time



Conclusions

- Still some work to be done both on RISC-V SW and HW to reach a stable and competitive HPC ecosystem.
- RISC-V Vector extension has the potential to provide high performance, but we need:
  1. More boards to play with, and better performance tools
  2. Better compilers to extract long vectors
  3. Reworked algorithms to expose more vector parallelism
- RISC-V is a vibrant community, please join the effort!

Contact

Reach out for potential collaborations or other RISC-V adventures :)

- Bhavishya Goel <goelb@chalmers.se>
- Miquel Pericas <miquelp@chalmers.se>

CHALMERS
UNIVERSITY OF TECHNOLOGY

THANK YOU

CHALMERS