GRVI Phalanx

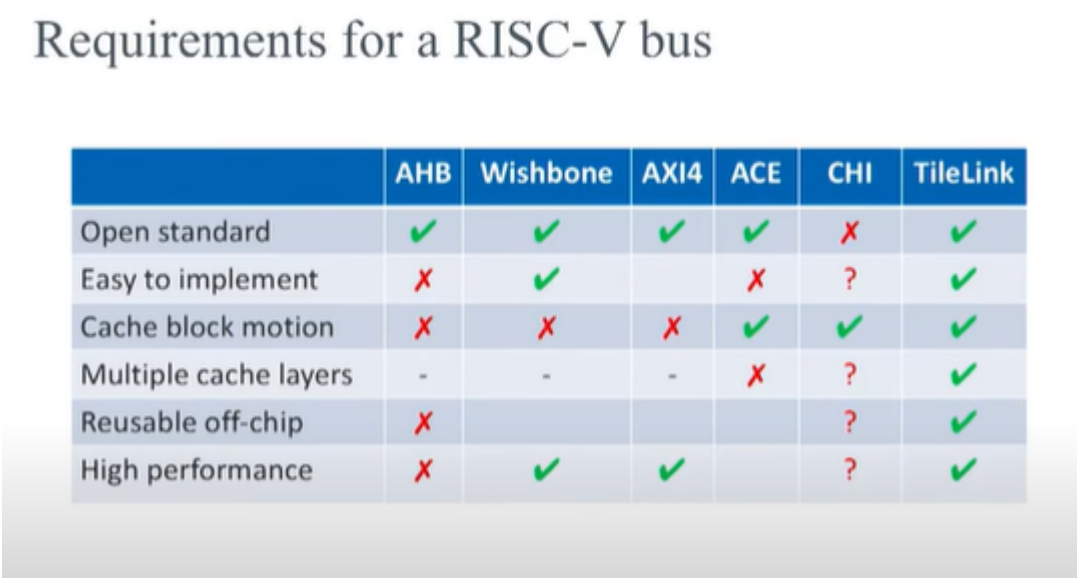
- Not general purpose, will probably just run a c/c++ kernel
- Each 8-core GRVI "cluster"
  - In-cluster interconnection: 4×4 cross bar for the cores to access the cluster memory
  - No caches, all cores(in a cluster) see the same memory
- Hoplite 2d router to connect all the clusters (memories+cores)
- It is a 2d Uni-direction Torus
- 32 grvi phalanx(400 cores, 50 clusters 8 cores each) fit on artix35t

Core extensions: RV32I (default)
NOC: cross bar in-cluster, Unidirectional torus inter-cluster
SW: na

Ref: https://www.youtube.com/watch?v=828oMNFGSjg

Tilelink

- A cache coherent riscv bus **protocol**, master-slave, point-to-point
- The rocket chip generator uses tilelink by default.
- Preferred requirements:



- Tilelink buses have three levels of complexity to choose from,
- TL-UL - Tilelink uncached lite
- TL - UH - Tilelink uncached heavy
- TL - C - Tilelink cached

TileLink Protocol Conformance Levels

	TL-UL	TL-UH	TL-C
Read/Write Operations	✔	✔	✔
Multibeat Messages		✔	✔
Atomic Operations		✔	✔
Hint (Prefetch) Operations		✔	✔
Cache Block Transfers			✔
Priorities B+C+E			✔

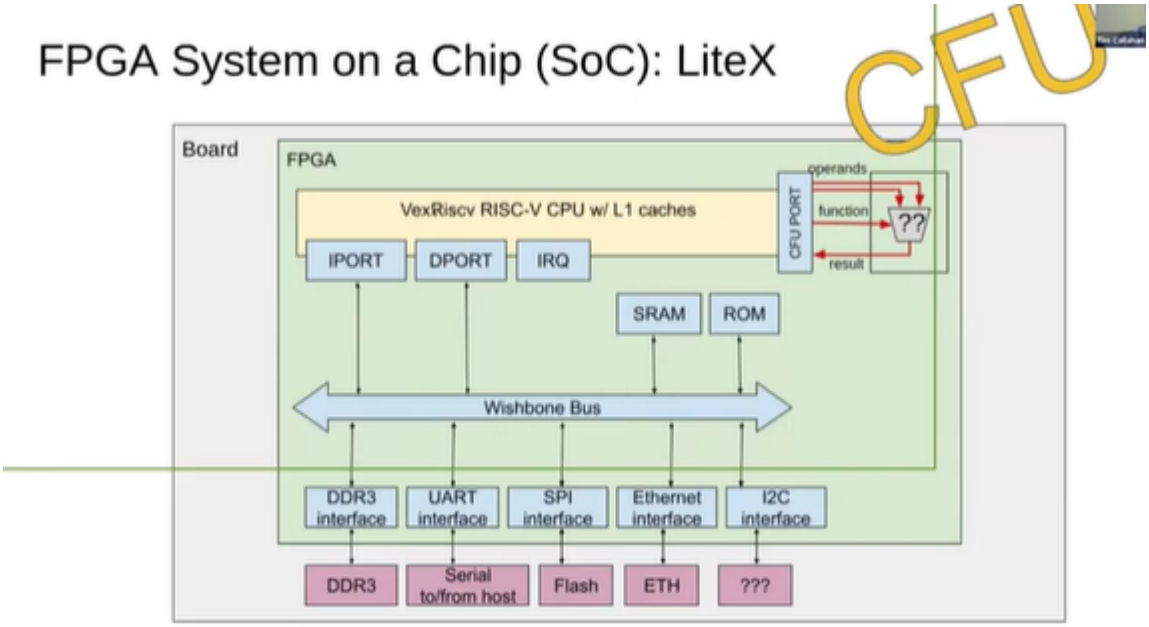
- Messages have "strict priorities" so that lower priority messages never block higher priority messages. This prevents deadlock during cache block motion.

Ref: https://www.youtube.com/watch?v=EVITxp-SEp4

CFU playground

- "Custom functional units" let you start with small custom hardware and build up iteratively to a full custom accelator
- The CFU and memory hierarchy are not directly connected, but the CFU can have its own memory and registers
- Uses vexRiscv
- LiteX is a migen/misoc based core builder
  (migen is a toolbox to create fpga designs in python, also provides bios, drivers, and lowlevel software libraries)
  Migen
  LiteX

- Only provide the cfu.v part, the cfu playground builds the rest



- Simulations facilitated by LiteX:
  - Software emulation of the CFU, on the FPGA board
    - *Checks algorithm restructuring*
  - Renode - ISA simulation (CPU), can be extended for CFU
    - *Useful in continuous integration (CI)*
  - Verilator - RTL simulation of CPU+CFU (waveforms!)
    - *Checks CFU implementation*
  - Debug bridge allows attaching GDB to the SoC running on the board
  - In progress: Renode+Verilator cosimulation
  - Not started: LiteScope (on-FPGA recording of selected signals, like Xilinx's ChipScope)

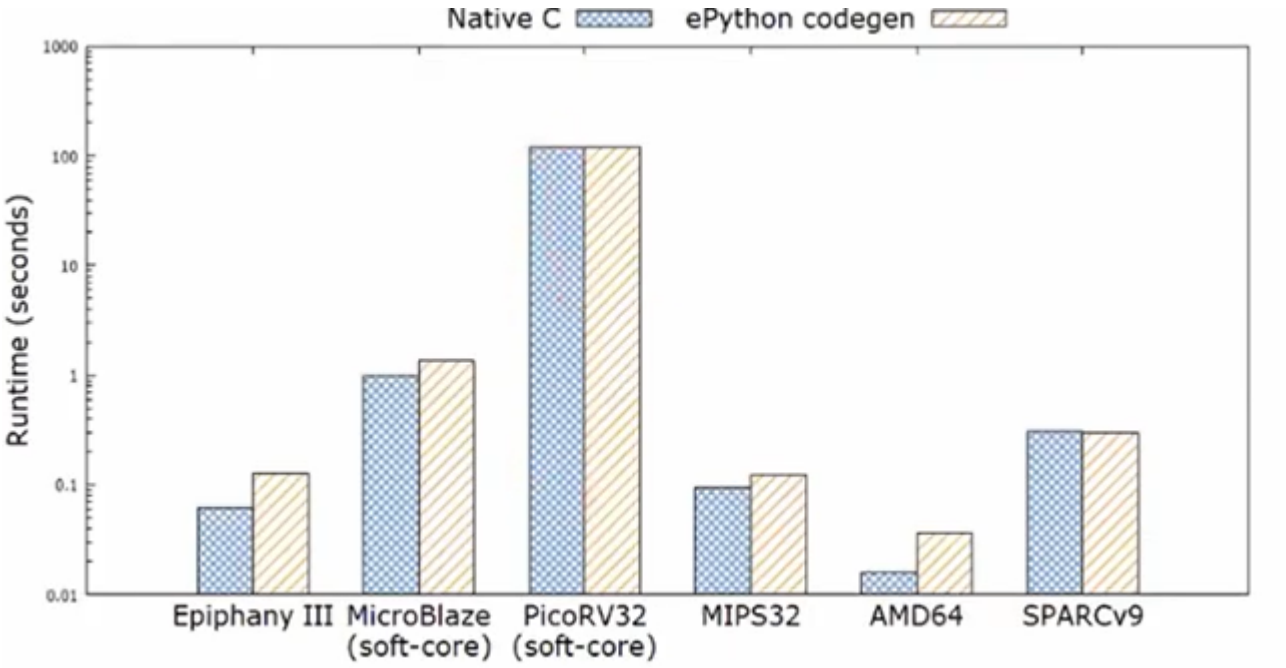
Core extensions: VexRISCV based, RV32I[M][A][F[D]][C] instruction set
NOC: na, uses wishbone
SW: Migen based "Linux on Vex" - LiteX

Ref: https://www.youtube.com/watch?v=_1yrxrl61o4

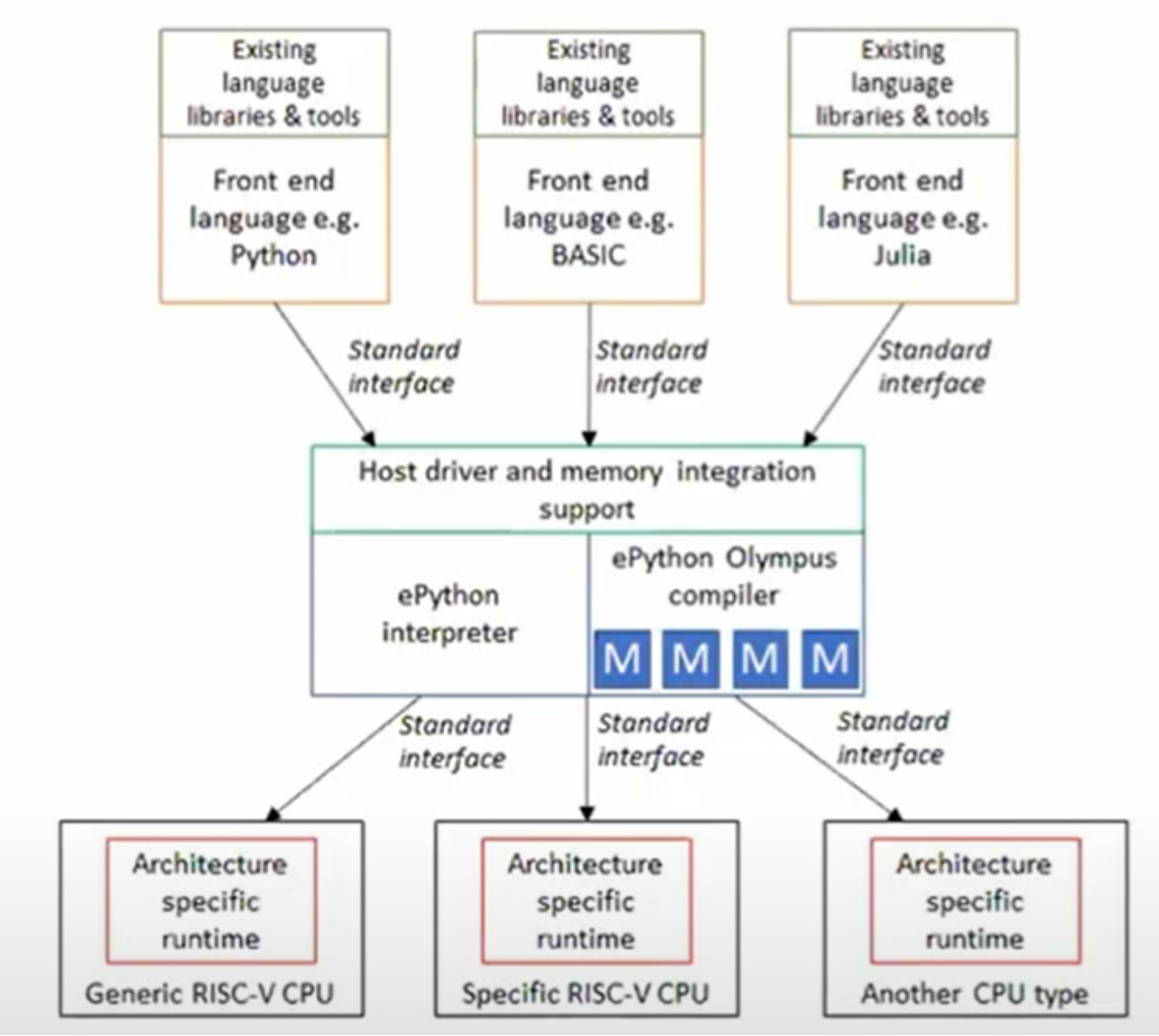
Epython

- Tiny implementation of python, only for a 24kb memory
- Run on a PicoRV32 (16 core softcore) on Zynq
- Using an interpreter directly - too slow

- Their compilation approach is translating python code to C and then compiling it on the RISCV gcc,
- To translate python to C use the Olympus compiler, compile it on an abstract machine which generates macros(?)
- https://dl.acm.org/doi/10.1145/3446804.3446853
- A case to use python for programming fpgas, claim to have performance close to the native c implementations



- General model



Core extensions: PicoRV32 based, RV32IMC Instruction Set
NOC: na
SW: Python translated to C by their "Olympus", C run on RISCV's gcc

Ref: https://www.youtube.com/watch?v=o8sjOhOmMjg

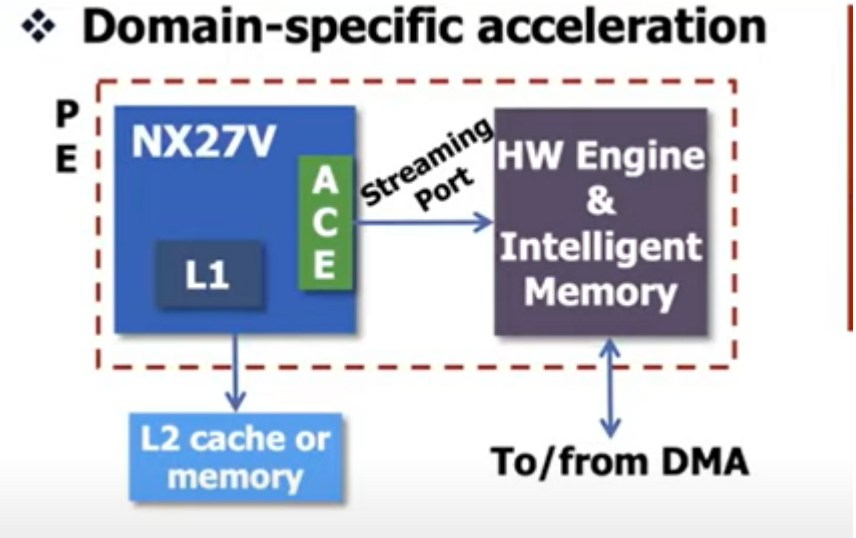
NX27V

The real challenge for RISCV vector processors is data movement - to increase bandwidth without additional power while maintaining high frequency.
The CPU approach to solve this problem would be including larger caches, but high frequency is impossible.

Ref: https://www.youtube.com/watch?v=kLM8lHqQzRI

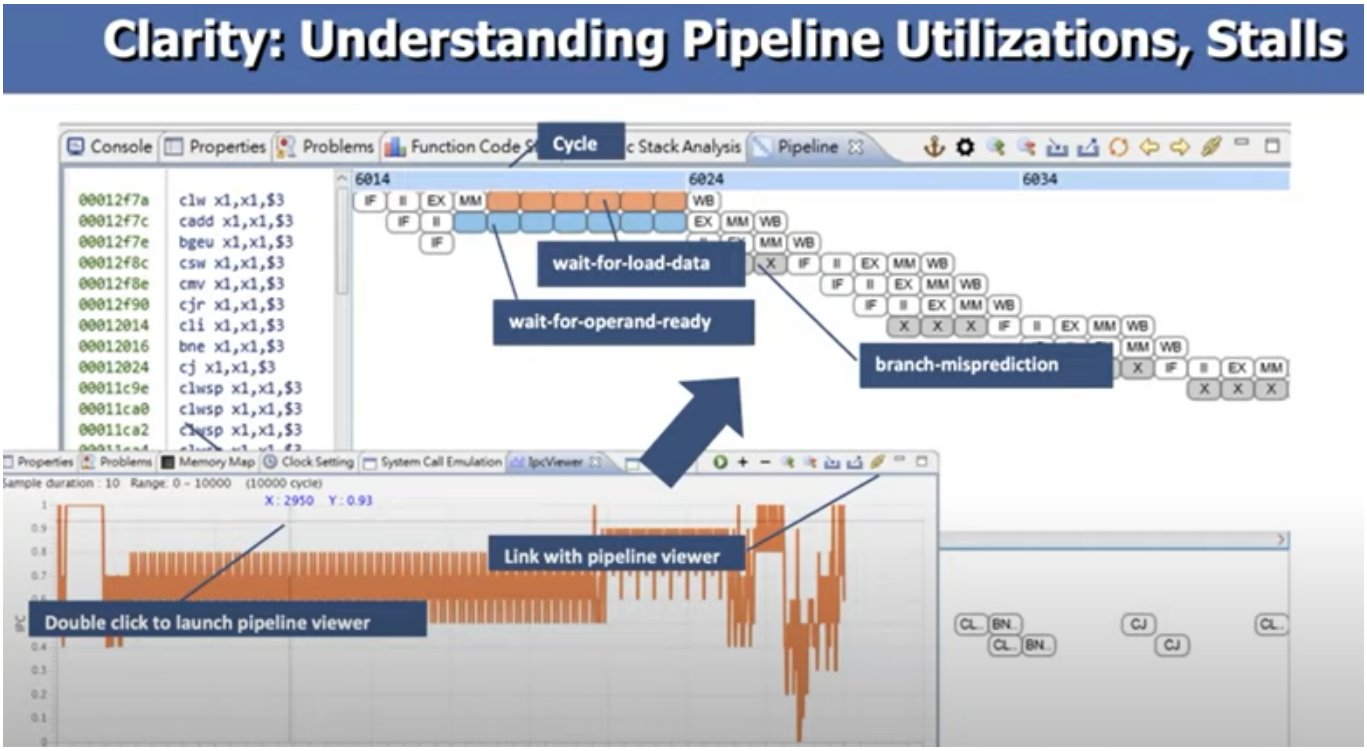
How they built NX27V and extended it to manycore

They added **custom instructions for streaming ports**, so that there is independent memory access via RISCV vector load/store(the main VPU) and the ACE(Andes custom extension) load/store.



They used the Andesight IDE for GUI, chip simulation, gcc, debug.

Clarity for pipeline and stalls simulation



They also used a virtual platform based software dev - imperas for (instruction-accurate) simulation and to run workload on the virtual platform even before the hardware is ready.

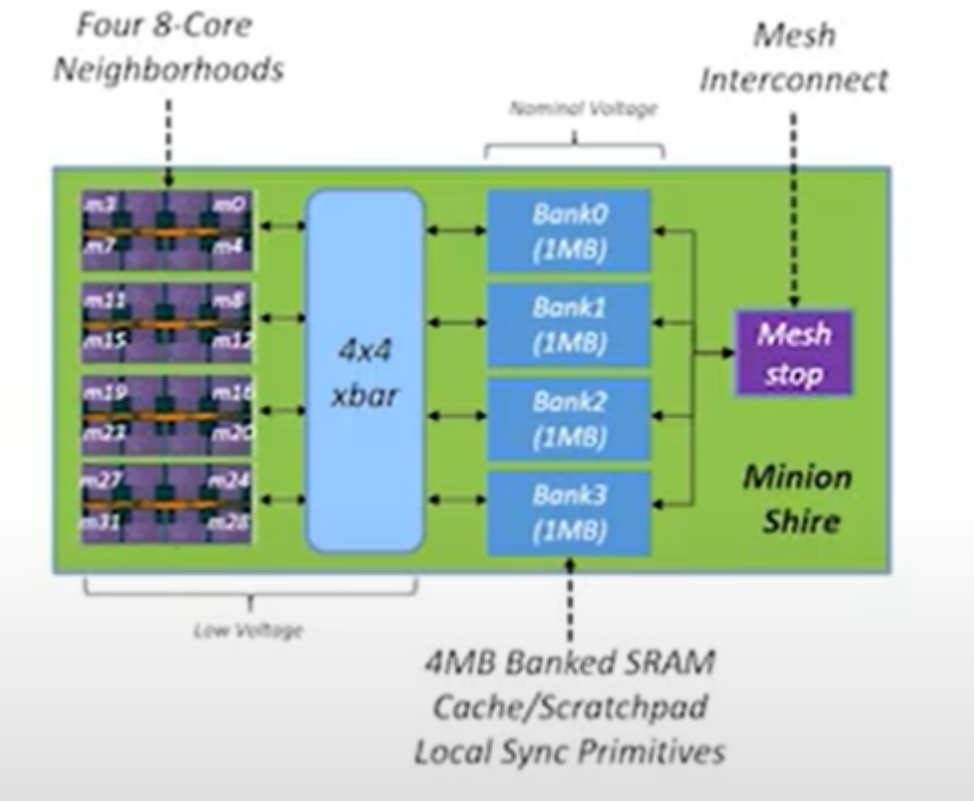
Core extensions: RV64GCN+andes V5 extensions, RV vector extension
NOC: na
SW: 1. openCL, 2. the Andesight IDE for the manycore, 3. Imperas OVP: profiling tools to analyze custom instruction addition + virtual platform

---

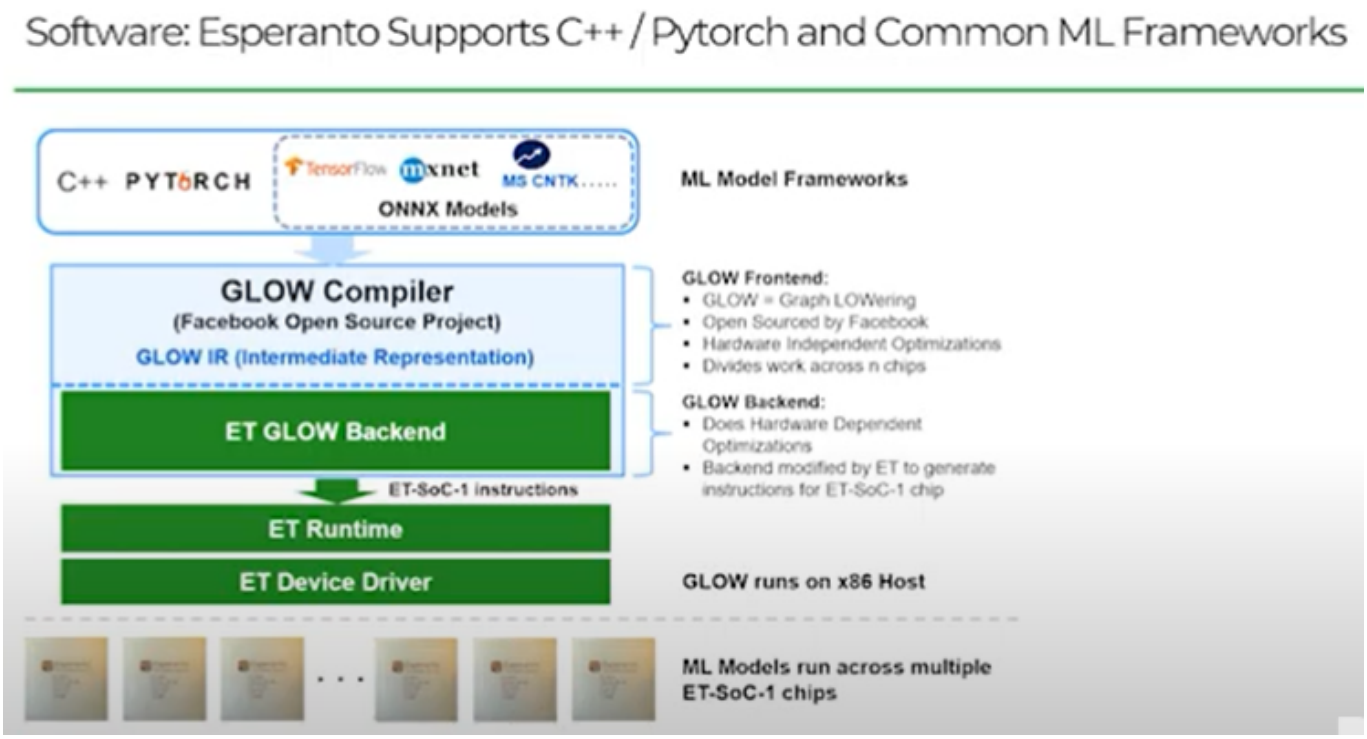
Enabling open programming models in RISC V for AI and HPChards is about codeplay's products: computeaorta and computecpp and An Introduction to RISC-V Vector Programming with C Intrinsics - Chih-Mao Chen, Andes Technology is RISCV Vector programming syntax intro.

Esperanto

- The ET-soc 1 1000+ cores, for data interferencing.
- According to emulation results, it has superior performance, 100x energy efficiency (better inferences per watt), future proof
- **ET minion** is the main core, it's a 64 bit riscv processor with custom vector/tensor instructions.
    - 256 bit wide floating point per cycle,
      16 32-bit single precision operations per cycle,
      32 16-bit half precision operations per cycle
      512 wide integer per cycle
      128 8bit int operations per cycle
      vector transcendental instructions
- A **minion shire** - 32 ET minions and 4MB mem
- The shires are mesh connected
- The core-SRAM connection is a 4×4 crossbar



- ET maxion is outoforder, can run linux,, can be placed with the ET minions
- Glow by facebook - "Compiler for Neural Network hardware accelerators"
- [Glow's github]



Core extensions: "custom vector/tensor isntructions"
NOC: 4×4 cross bar in-shire, mesh connect accross shires
SW: pytorch on Glow

Ref: https://www.youtube.com/watch?v=7UfmW9Ea2Kk
