ESP4ML

Platform-Based Design of System-on-Chip for Embedded Machine Learning

Davide Giri
Kuan-Lin Chiu
Giuseppe di Guglielmo
Paolo Mantovani
Luca P. Carloni

DATE 2020

COLUMBIA UNIVERSITY
IN THE CITY OF NEW YORK

CS @CU COMPUTER SCIENCE

ESP4ML

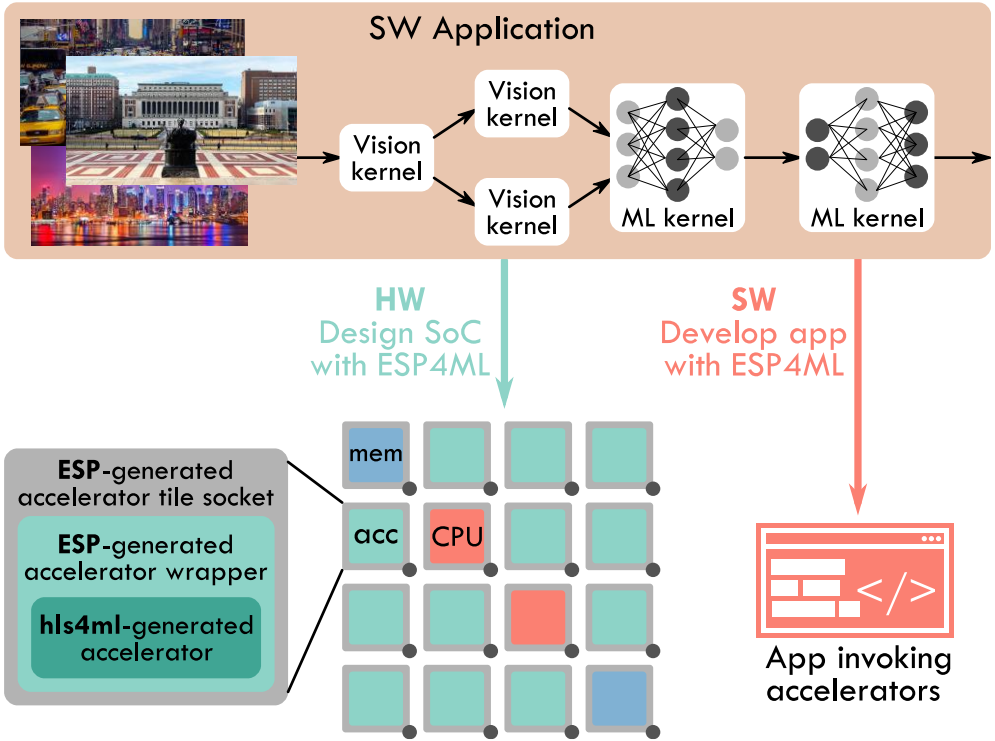
Open-source design flow to build and program SoCs for ML applications.

Combines ESP and hls4ml

- **ESP** is a platform for heterogeneous SoC design
- **hls4ml** automatically generates accelerators from ML models

Main **contributions to ESP**:

- Automated integration of hls4ml accelerators
- Accelerator-accelerator communication
- Accelerator invocation API



hls4ml

- Open-source tool developed by **Fast ML Lab**
- Translates ML algorithms into HLS-able accelerator specifications
  - Targets Xilinx **Vivado HLS** (i.e. FPGA only)
  - ASIC support is in the works
- Born for high-energy physics (small and ultra-low latency networks)
  - Now has broad applicability

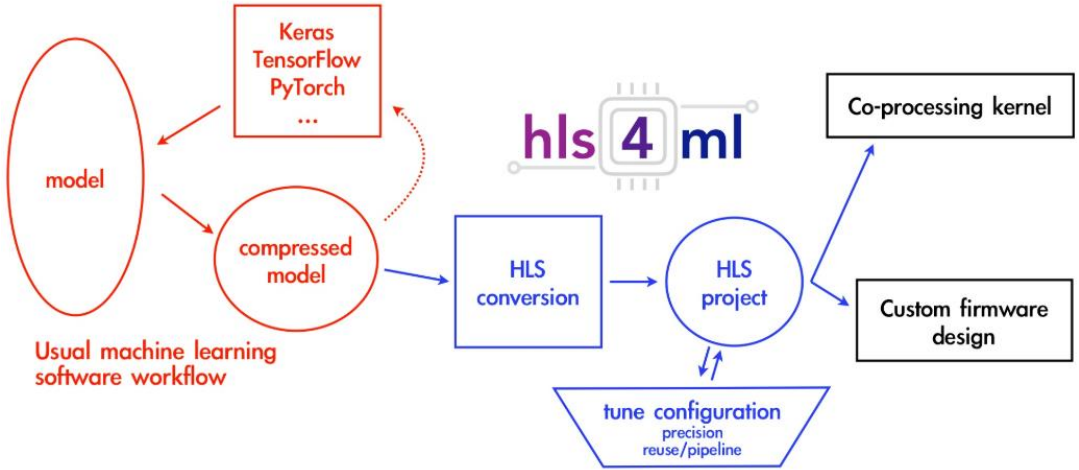
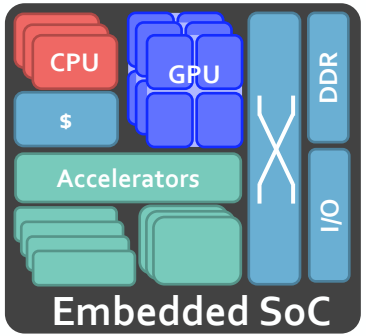
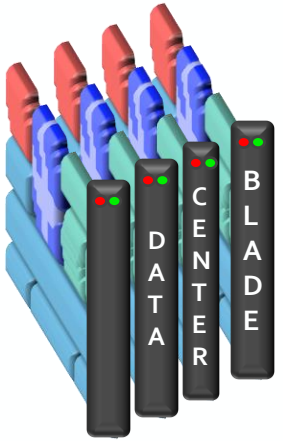


Image from https://fastmachinelearning.org/hls4ml/

ESP motivation





Heterogeneous systems are pervasive

Integrating **accelerators** into a SoC is hard

Doing so in a **scalable** way is very hard

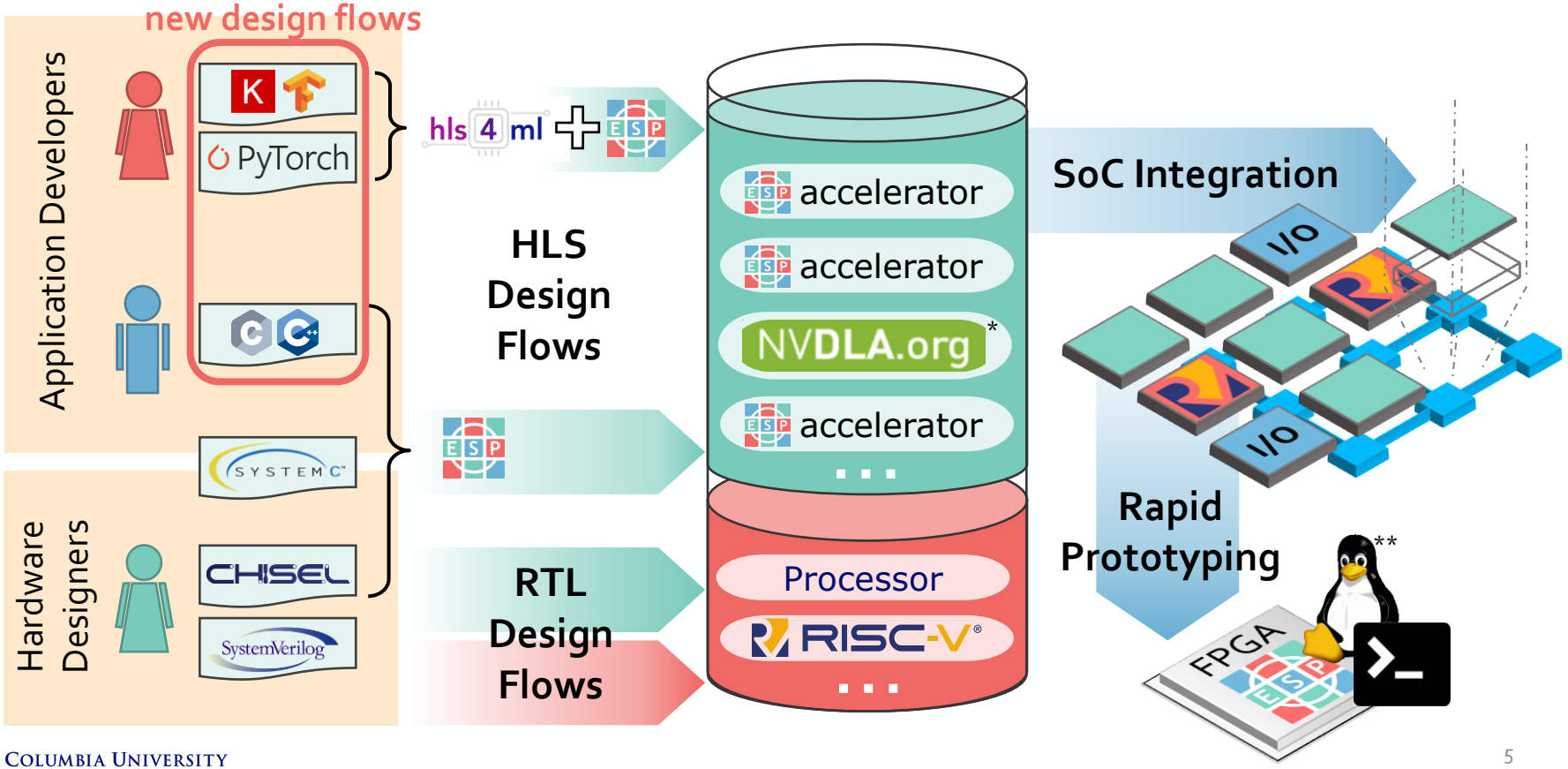
Keeping the system **simple to program** while doing so is even harder

ESP makes it easy

ESP combines a **scalable architecture** with a **flexible methodology**

ESP enables **several accelerator design flows**
and takes care of the hardware and software integration

ESP overview

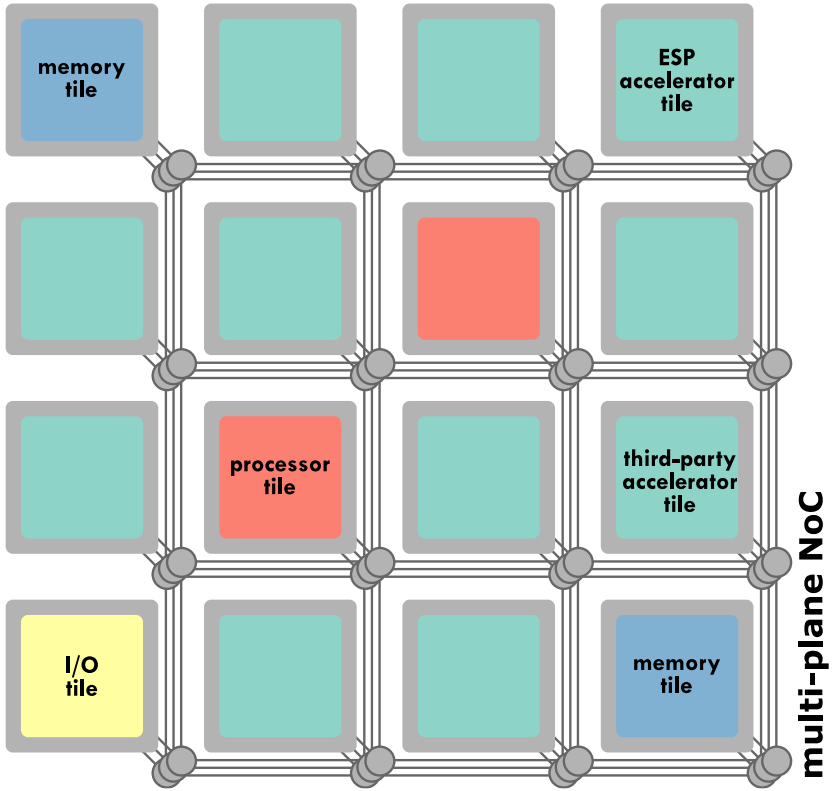


* By Nvidia Corporation
** By lewing@isc.tamu.edu Larry Ewing and The GIMP

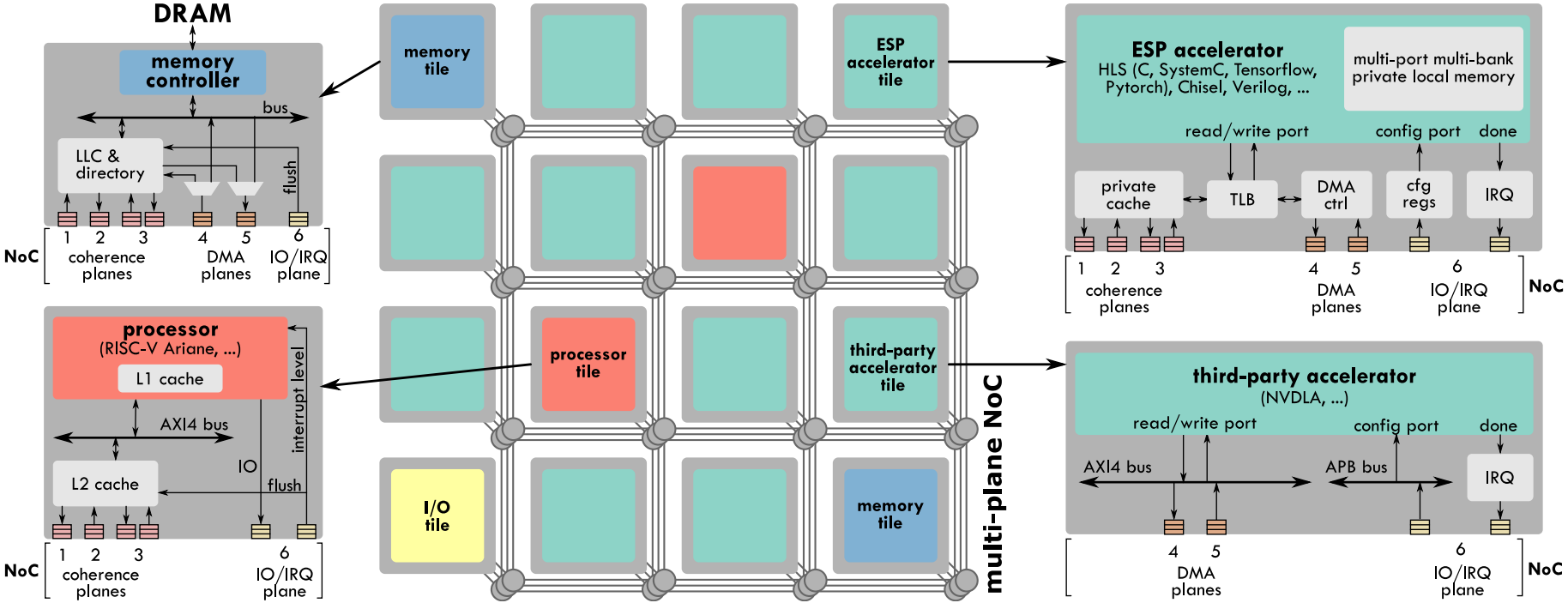
ESP architecture

- Multi-Processors
- Many-Accelerator
- Distributed Memory
- Multi-Plane NoC

The ESP architecture implements a **distributed** system, which is **scalable**, **modular** and **heterogeneous**, giving processors and accelerators similar weight in the SoC



ESP architecture: the tiles



ESP methodology in practice

automated
interactive
manual (opt.)
manual

Accelerator Flow

Generate accelerator

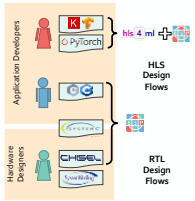
Specialize accelerator
(not required by hls4ml flow)

Test behavior

Generate RTL

Test RTL

Optimize accelerator



SoC Flow

Generate sockets

Configure SoC

Compile bare-metal

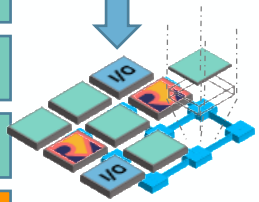
Simulate system

Implement for FGPA

Design runtime apps

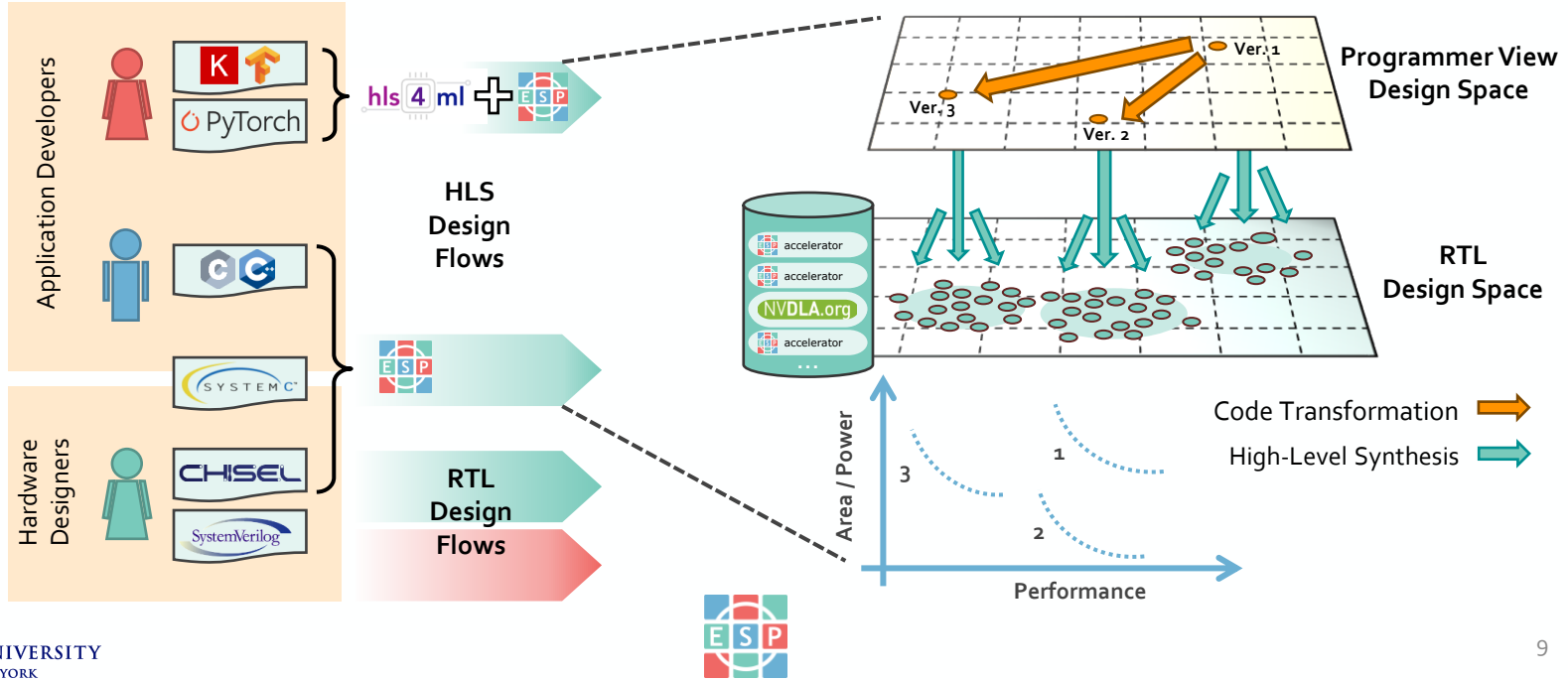
Compile Linux

Deploy prototype

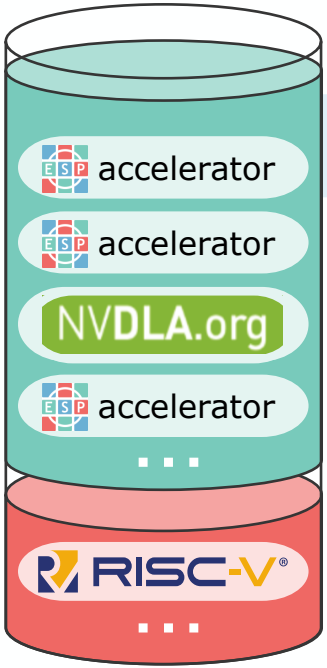


ESP accelerator flow

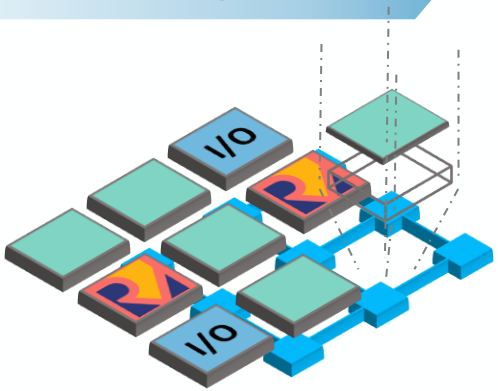
Developers focus on the **high-level specification**, **decoupled** from memory access, system communication, hardware/software interface

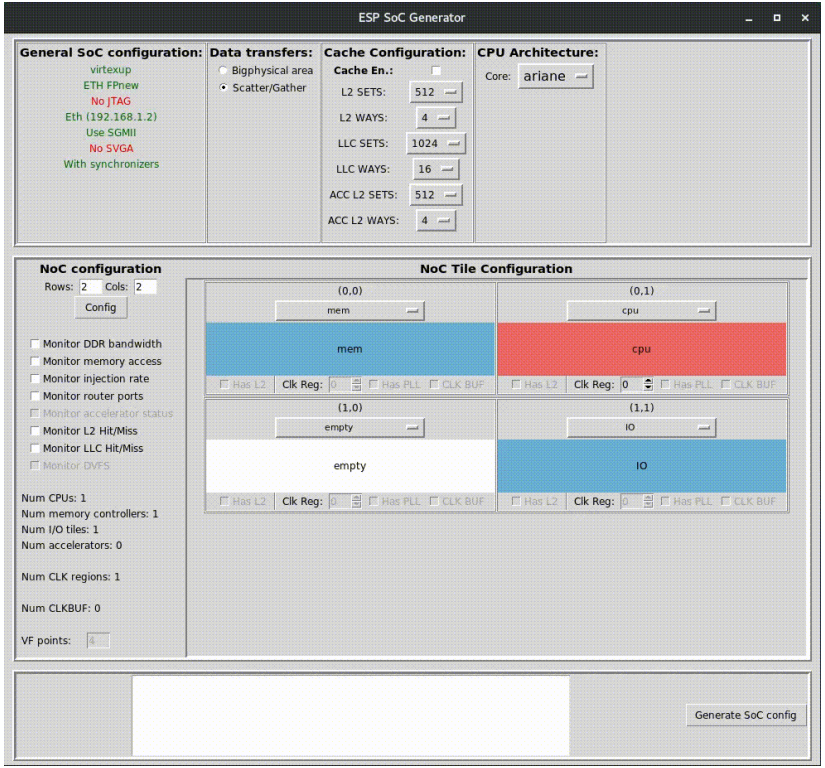


ESP Interactive SoC Flow



SoC Integration





New ESP features

- New accelerator design flows (C/C++, Keras/Pytorch/ONNX)
- Accelerator-to-accelerator communication
- Accelerator invocation API

New accelerator design flows

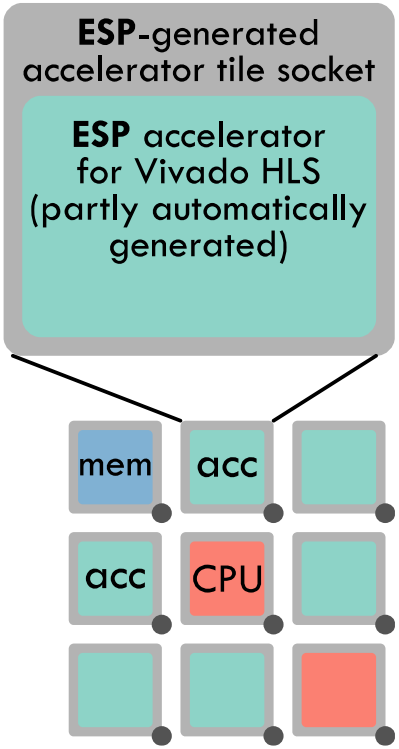
C/C++ accelerators with *Vivado HLS*

- Generate the accelerator skeleton with ESP
  - Takes care of communication with the ESP tile socket
- Implement the computation part of the accelerator

```
void top(dma_t *out, dma_t *in1, unsigned cfg_size,
         dma_info_t *load_ctrl, dma_info_t *store_ctrl)
{
    for (unsigned i = 0; i < cfg_size; i++) {
        word_t _inbuff[IN_BUF_SIZE];
        word_t _outbuff[OUT_BUF_SIZE];

        load(_inbuff, in1, i, load_ctrl, 0);
        compute(_inbuff, _outbuff);
        store(_outbuff, out, i, store_ctrl, cfg_size);
    }
}
```

Example of top level function of ESP accelerator for Vivado HLS

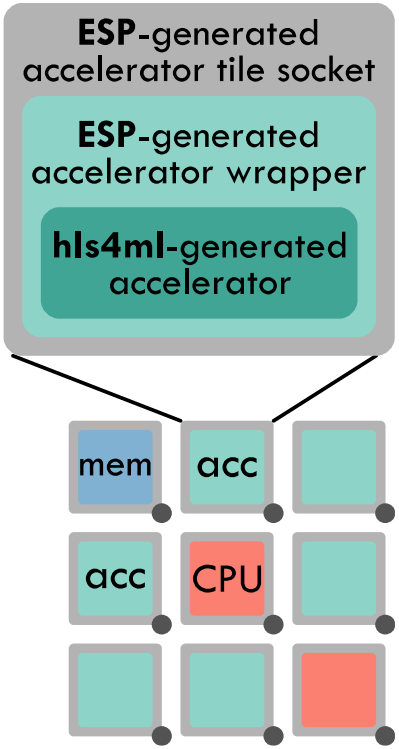


New accelerator design flows

Keras/Pytorch/ONNX accelerators with *hls4ml*

Completely automated integration in *ESP*:

- Generate an accelerator with *hls4ml*
- Generate the accelerator wrapper with ESP



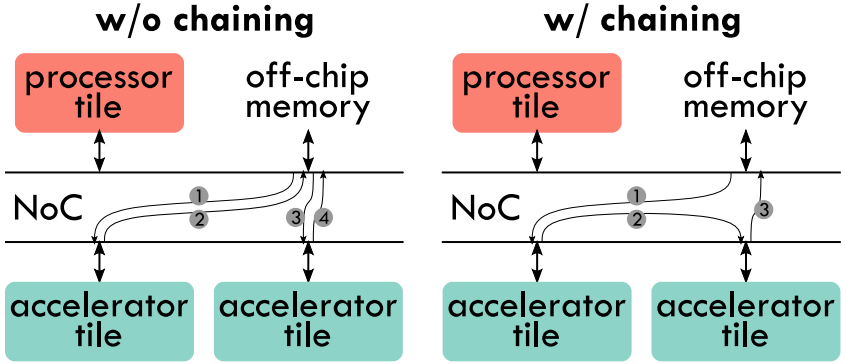
Accelerator-to-accelerator communication

Accelerators can exchange data with:

- Shared memory
- Other accelerators **(new!)**

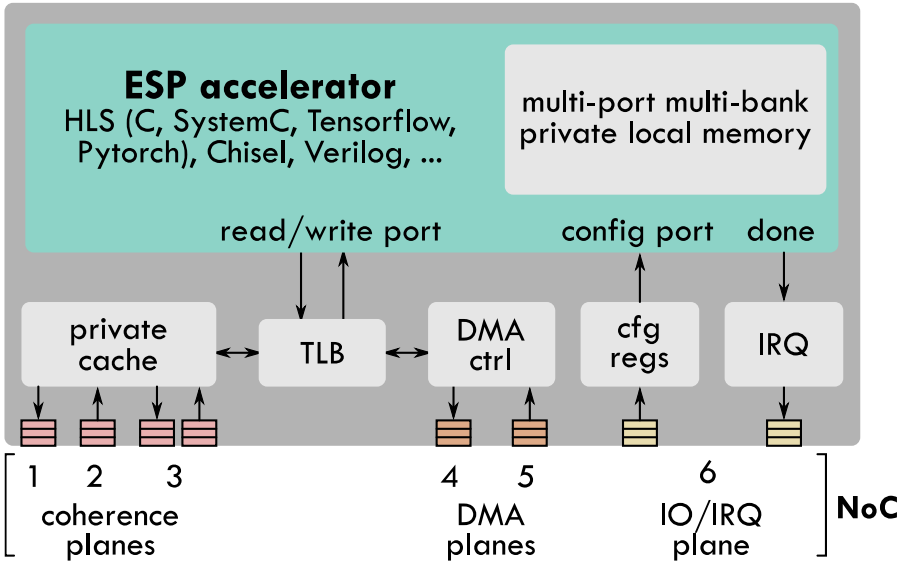
Benefits

- Avoid roundtrips to shared memory
- Fine-grained accelerators synchronization
  - Higher throughput
  - Lower invocation and data pre- or post-processing overheads



Accelerator-to-accelerator communication

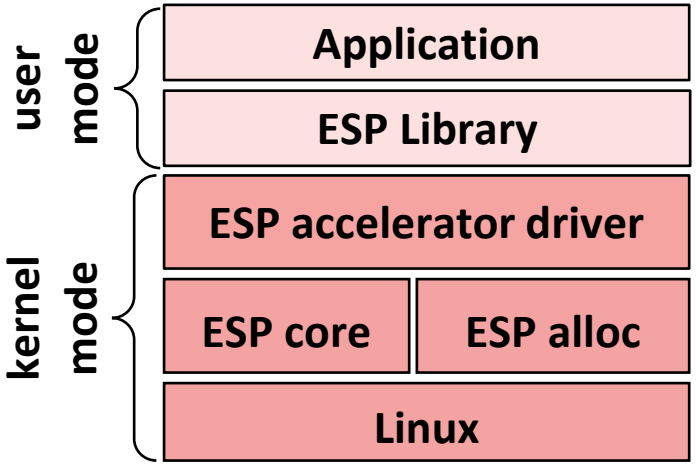
- No need for additional queues or NoC channels
- Communication configured at invocation time
- Accelerators can pull data from other accelerators, not push



Accelerator invocation API

API for the invocation of accelerators from a user application

- Exposes only 3 functions to the programmer

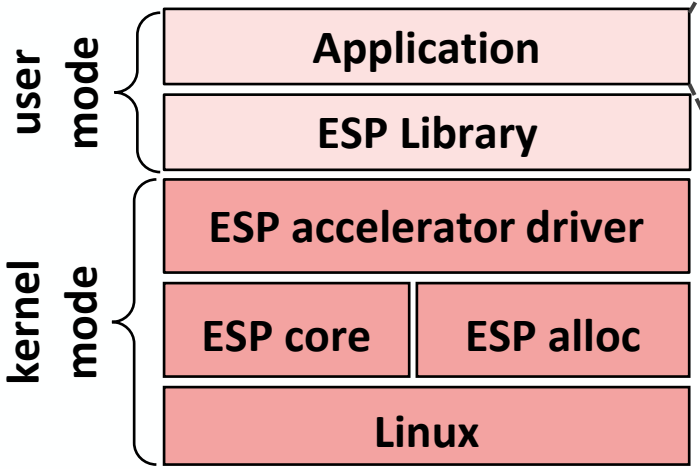


- Invokes accelerators through Linux device drivers
  - ESP automatically generates the device drivers
- Enables shared memory between processors and accelerators
  - No data copies
- Can be targeted by existing applications with minimal modifications
- Can be targeted to automatically map tasks to accelerators

Accelerator invocation API

API for the invocation of accelerators from a user application

- Exposes only 3 functions to the programmer




```
/*
 * Example of existing C application
 * with ESP accelerators that replace
 * software kernels 2, 3 and 5
 */
{
  int *buffer = esp_alloc(size);

  for (...) {

    kernel_1(buffer,...); // existing software

    esp_run(cfg_k2);      // run accelerator(s)
    esp_run(cfg_k3);

    kernel_4(buffer,...); // existing software

    esp_run(cfg_k5);
  }

  validate(buffer);       // existing checks

  esp_cleanup();          // memory free
}
```

Accelerator API

Configuration example:

- Invoke accelerators *k1* and *k2*
- Enable point-to-point communication between them

```
/* Example of double-accelerator config */
esp_thread_info_t cfg_k12[] =
{
  {
  .devname = "k1.0",
  .type = k1,
  /* accelerator configuration */
  .desc.k1_desc.nbursts = 8,
  /* p2p configuration */
  .desc.k1_desc.esp.p2p_store = true,
  .desc.k1_desc.esp.p2p_nsrcs = 0,
  .desc.k1_desc.esp.p2p_srcs = {"","","",""},
  },
 {
  .devname = "k2.0",
  .type = k2,
  /* accelerator configuration */
  .desc.k2_desc.nbursts = 8,
  /* p2p configuration */
  .desc.k2_desc.esp.p2p_store = false,
  .desc.k2_desc.esp.p2p_nsrcs = 1,
  .desc.k2_desc.esp.p2p_srcs = {"k1.0","","",""},
  },
};
```

Evaluation

Experimental setup

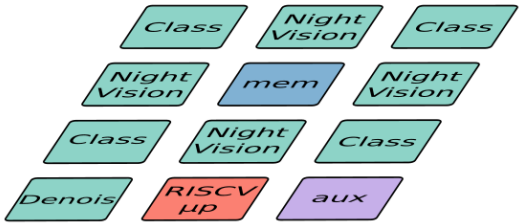
- We deploy two multi-accelerator SoCs on FPGA (Xilinx VCU118)
- We execute applications with accelerator chaining and parallelism opportunities
- We compare the our SoCs against:
  - Intel i7 8700K processor
  - NVIDIA Jetson TX1
    - 256-core NVIDIA Maxwell GPU
    - Quad-core ARM Cortex A57

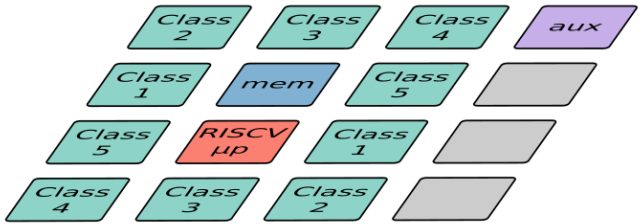
Featured accelerators:

- **Image classifier** (hls4ml)
  - Street View House Numbers (SVHN) dataset from Google
- **Denoiser** (hls4ml)
  - Implemented as an autoencoder
- **Night-vision** (Stratus HLS)
  - Noise filtering, histogram, histogram equalization

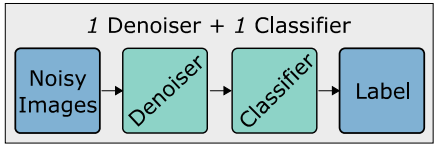
Case studies

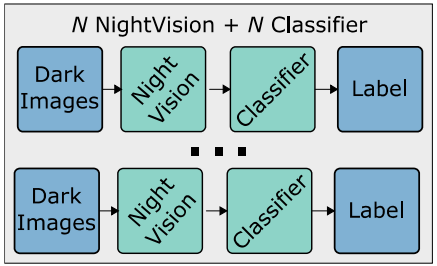
SoCs

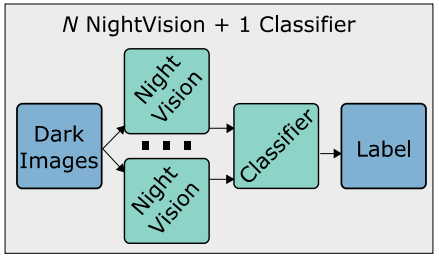


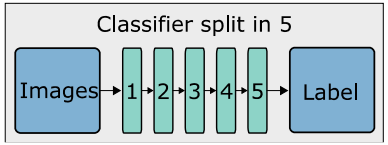


Applications





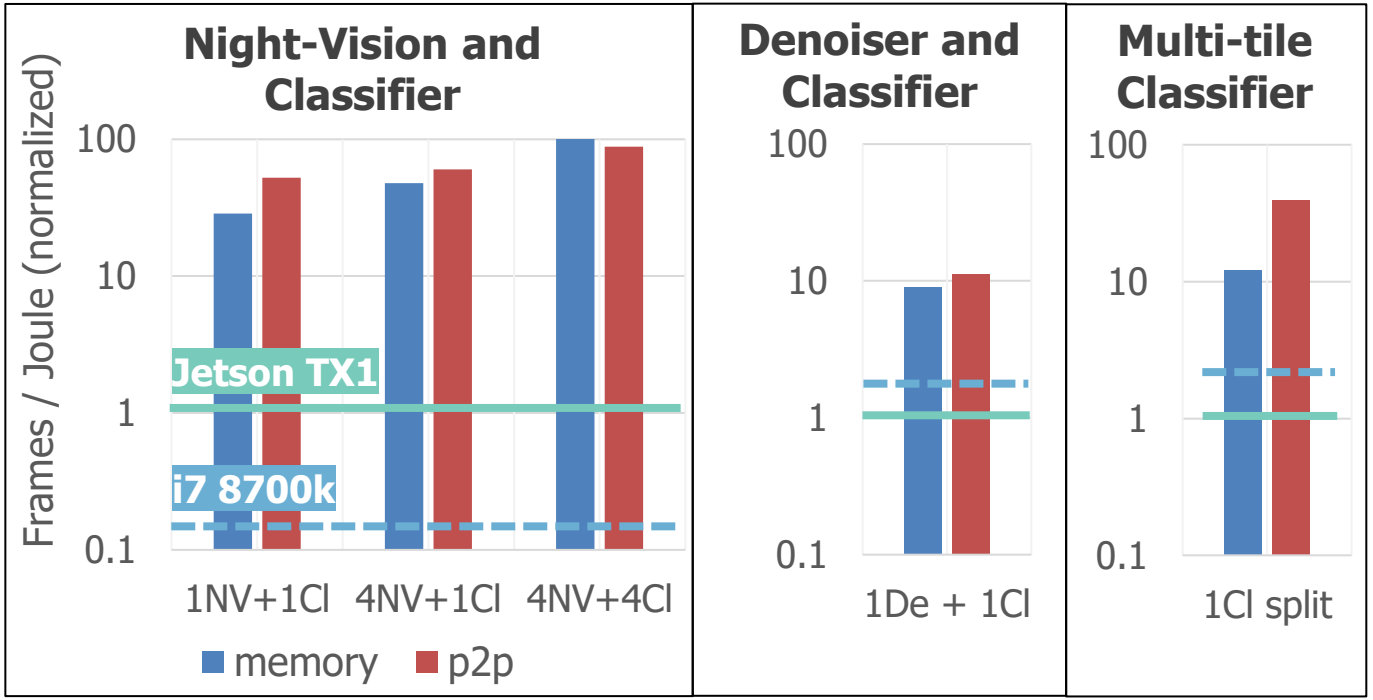




Efficiency

Chaining accelerators brings energy savings.

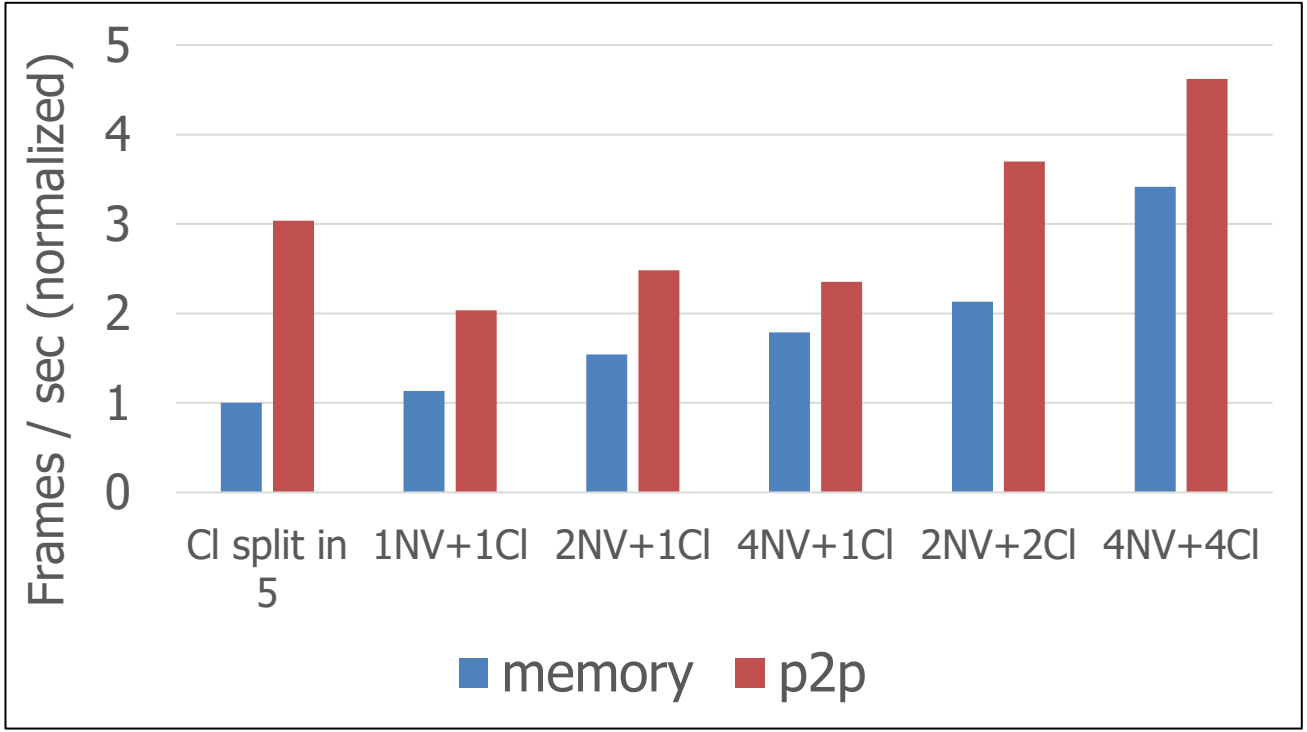
Our SoCs achieve better energy efficiency than Jetson and i7.



Performance

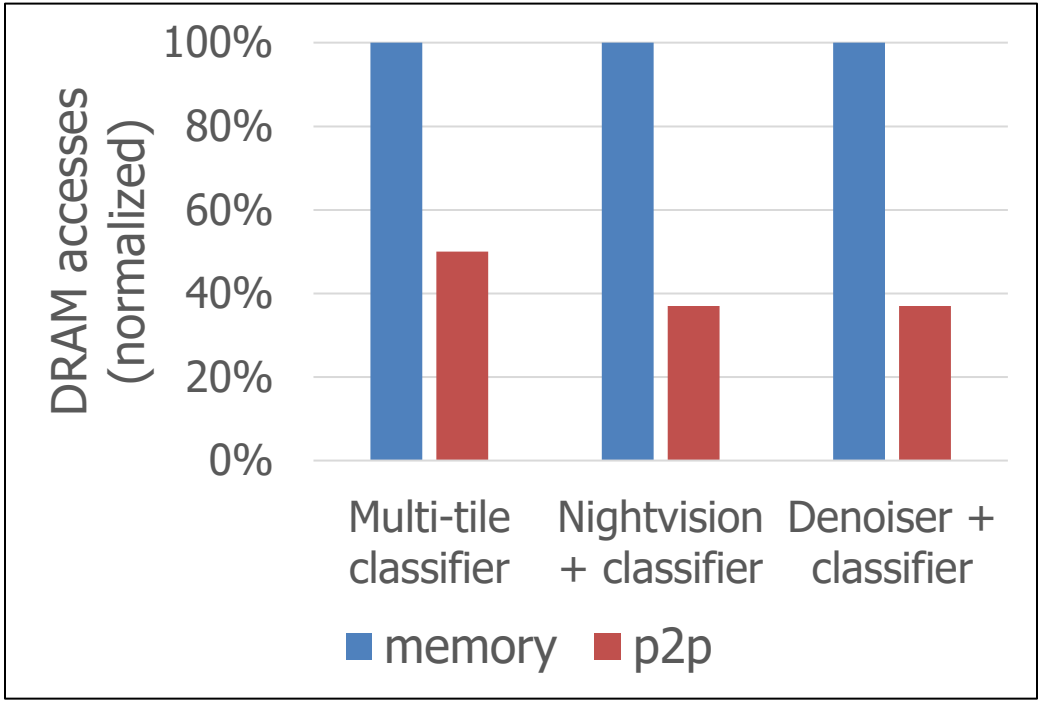
Performance increases to up to 4.5 times thanks to:

- Parallelization
- Chaining (p2p)



Memory accesses

Accelerator chaining (p2p) reduces the memory accesses by 2-3 times



Conclusions

ESP4ML is a complete system-level design flow to implement many-accelerator SoCs and to deploy embedded applications on them.

We enhanced ESP with the following features:

- Fully automatic integration in ESP of accelerators specified in **C/C++** (Vivado HLS) and **Keras/Pytorch/ONNX** (hls4ml)
- Minimal **API** to invoke accelerator for ESP
- Reconfigurable activation of accelerators pipelines through efficient **point-to-point** communication mechanisms

Thank you from the ESP team!

sld.cs.columbia.edu

esp.cs.columbia.edu


sld-columbia/esp

ESP4ML

Platform-Based Design of System-on-Chip
for Embedded Machine Learning

Davide Giri (www.cs.columbia.edu/~davide_giri)
Kuan-Lin Chiu
Giuseppe di Guglielmo
Paolo Mantovani
Luca P. Carloni

DATE 2020