Automatic Code Generation for Rocket Chip RoCC Accelerators

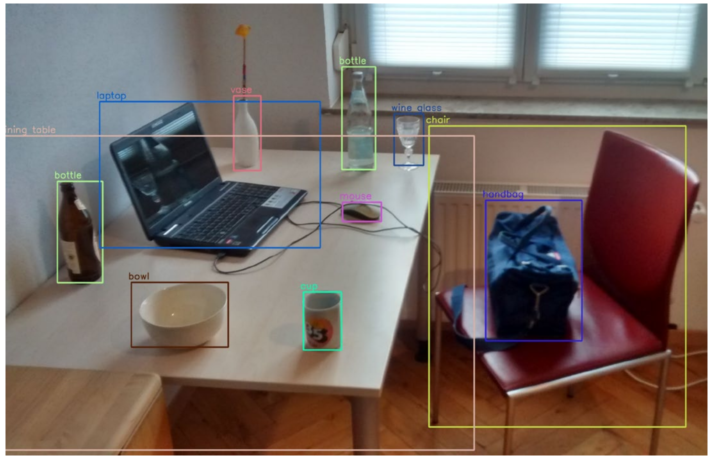
Fourth Workshop on Computer Architecture Research with RISC-V (CARRV 2020)

Pengcheng Xu, Yun Liang
Peking University

北京大學
PEKING UNIVERSITY

北京大学高能效计算与应用中心
Center for Energy-efficient Computing and Applications

Deep Learning is everywhere



Object tracking

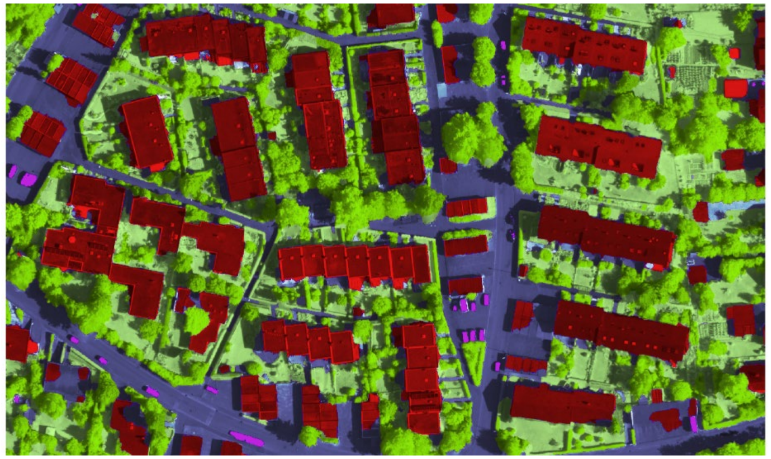
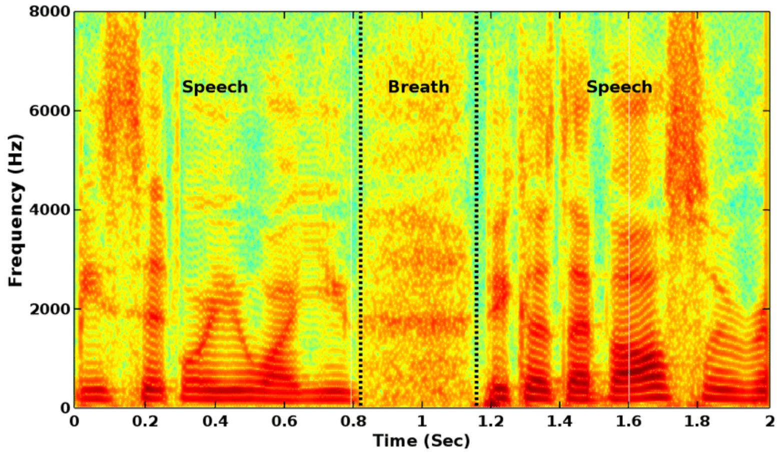
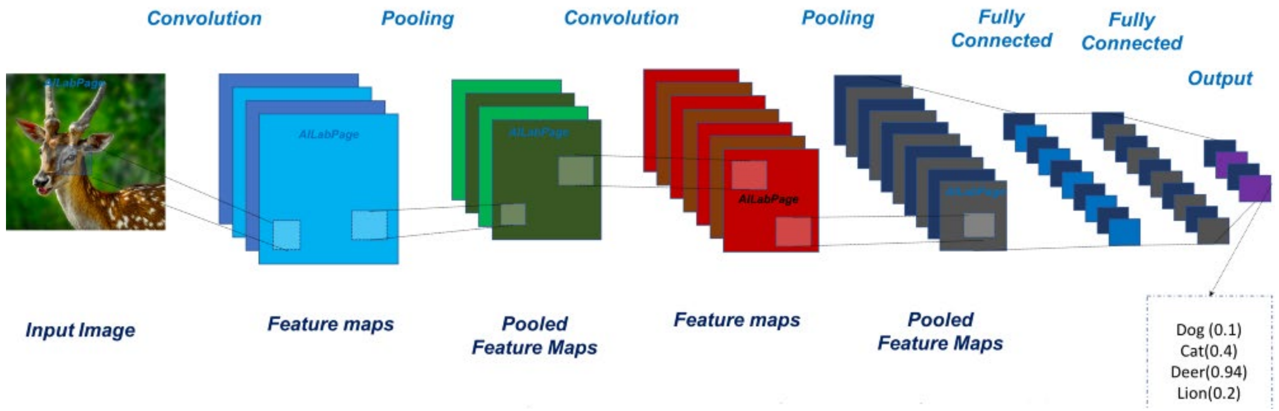


Image Segmentation



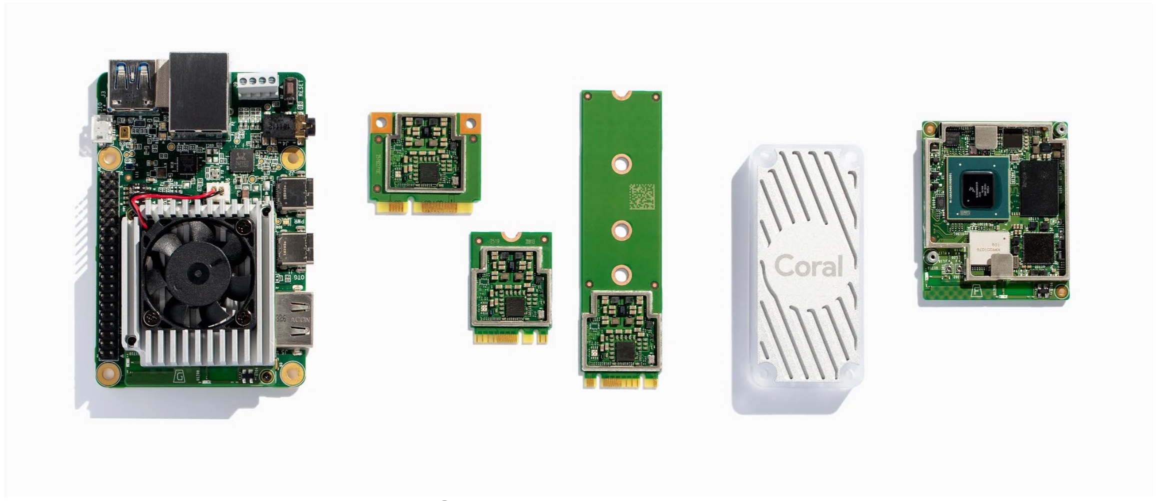
Speech Recognition

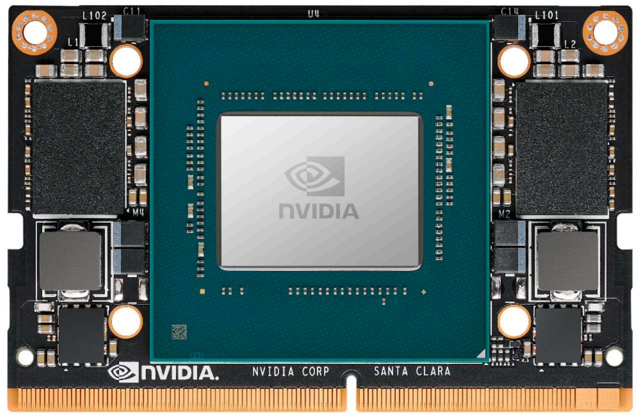


Tensor programs are at the heart of Deep Learning

Images are from web search. Copyright goes to their rightful owners.

Deep Learning at the edge


Coral Edge TPU



NVIDIA Jetson Xavier NX

Huawei Kirin 990

Seeed Studio MAix-1

Images are from web search. Copyright goes to their rightful owners.

These are no easy toys!

- Accelerators require efficient software to achieve potential performance
- However, developing for them is hard
  - Lack of mature SDK: write code in C, handle hardware details directly
  - Cross compiling, lack of OS, etc. makes debugging cumbersome
- Deep Learning code optimizations are repetitive and empirical
  - Loop transformations: split, reorder
  - Need to run program to evaluate performance
  - Insufficient design space exploration leads to suboptimal programs

Outline

- <u>Glossary</u>
- Automatic code generation for RoCC accelerators
- Performance evaluation platform design
- Case study of Gemmini

RoCC

- Rocket Chip Custom Coprocessor Interface
- Specifies an interface between CPU core and custom coprocessors
- Coherent & incoherent memory access

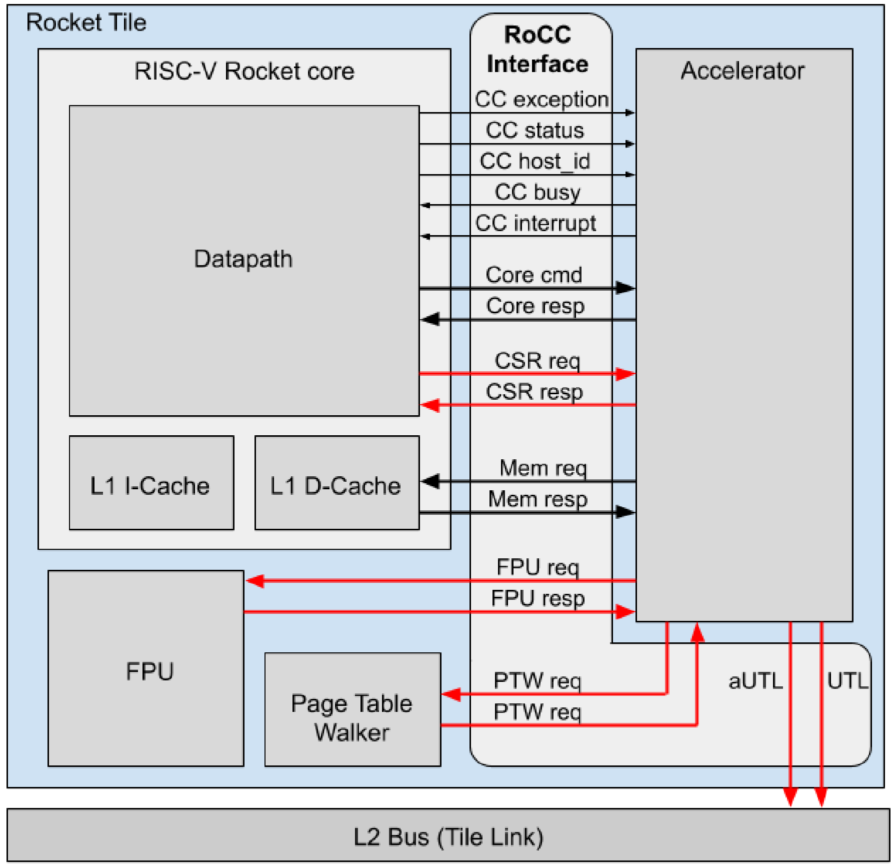
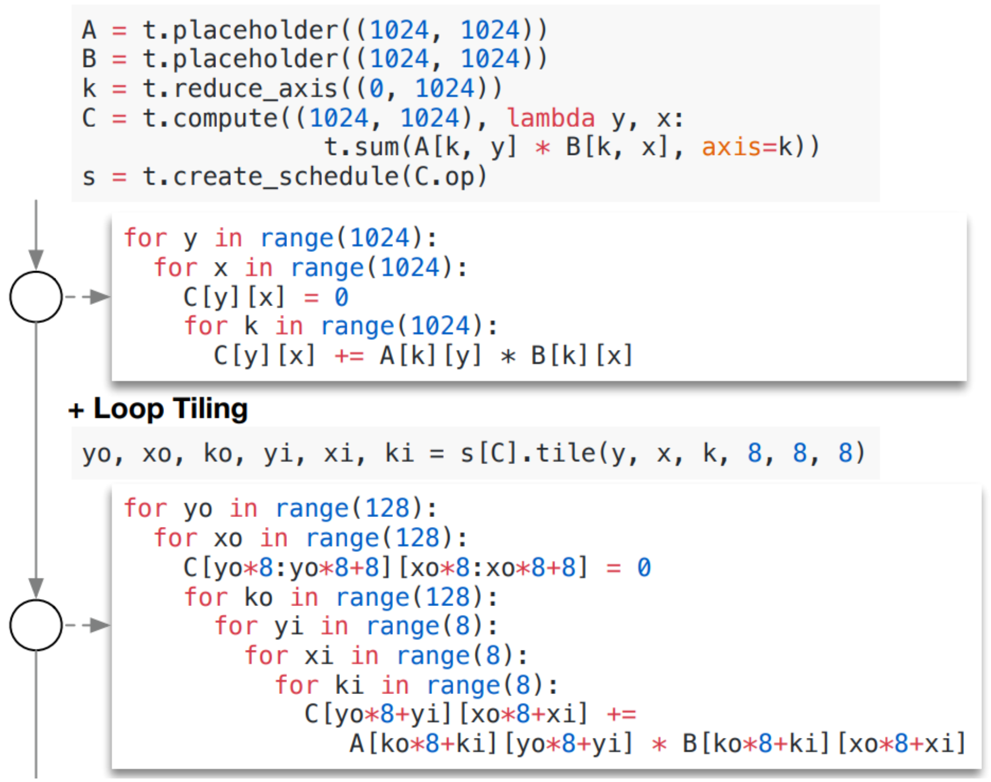


Figure 1: Default (black) & extended (red) signals of the RoCC interface

Automatic code generation

- Separate definition of computation and optimization



```
A = t.placeholder((1024, 1024))
B = t.placeholder((1024, 1024))
k = t.reduce_axis((0, 1024))
C = t.compute((1024, 1024), lambda y, x:
              t.sum(A[k, y] * B[k, x], axis=k))
s = t.create_schedule(C.op)
```

```
for y in range(1024):
  for x in range(1024):
    C[y][x] = 0
    for k in range(1024):
      C[y][x] += A[k][y] * B[k][x]
```

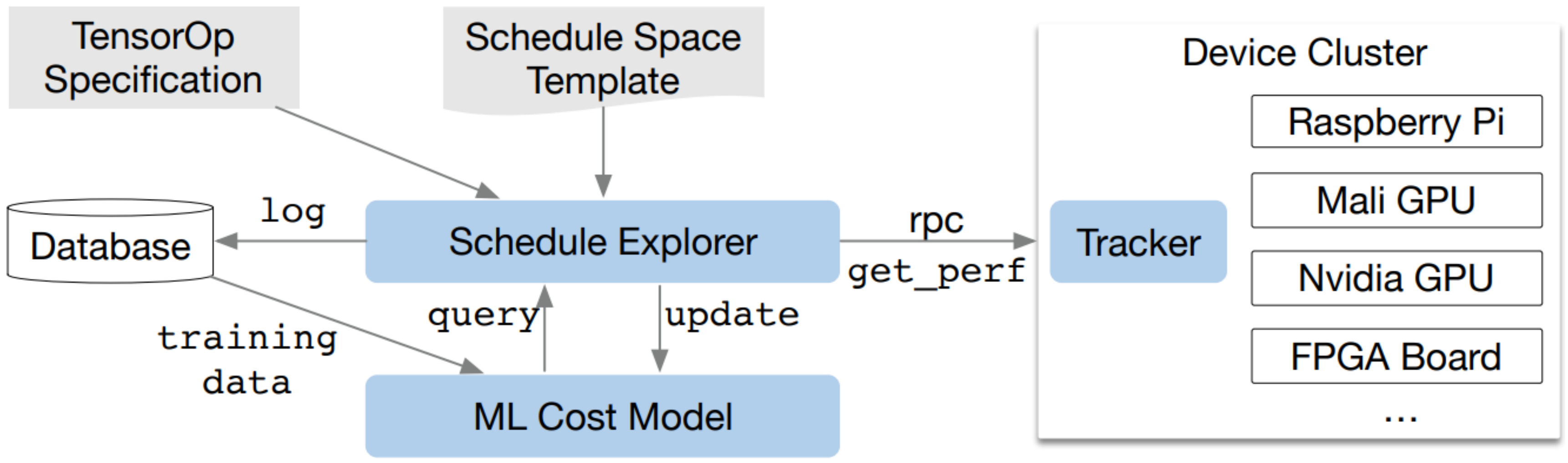
+ Loop Tiling

```
yo, xo, ko, yi, xi, ki = s[C].tile(y, x, k, 8, 8, 8)
```

```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
              A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

Automatic code generation

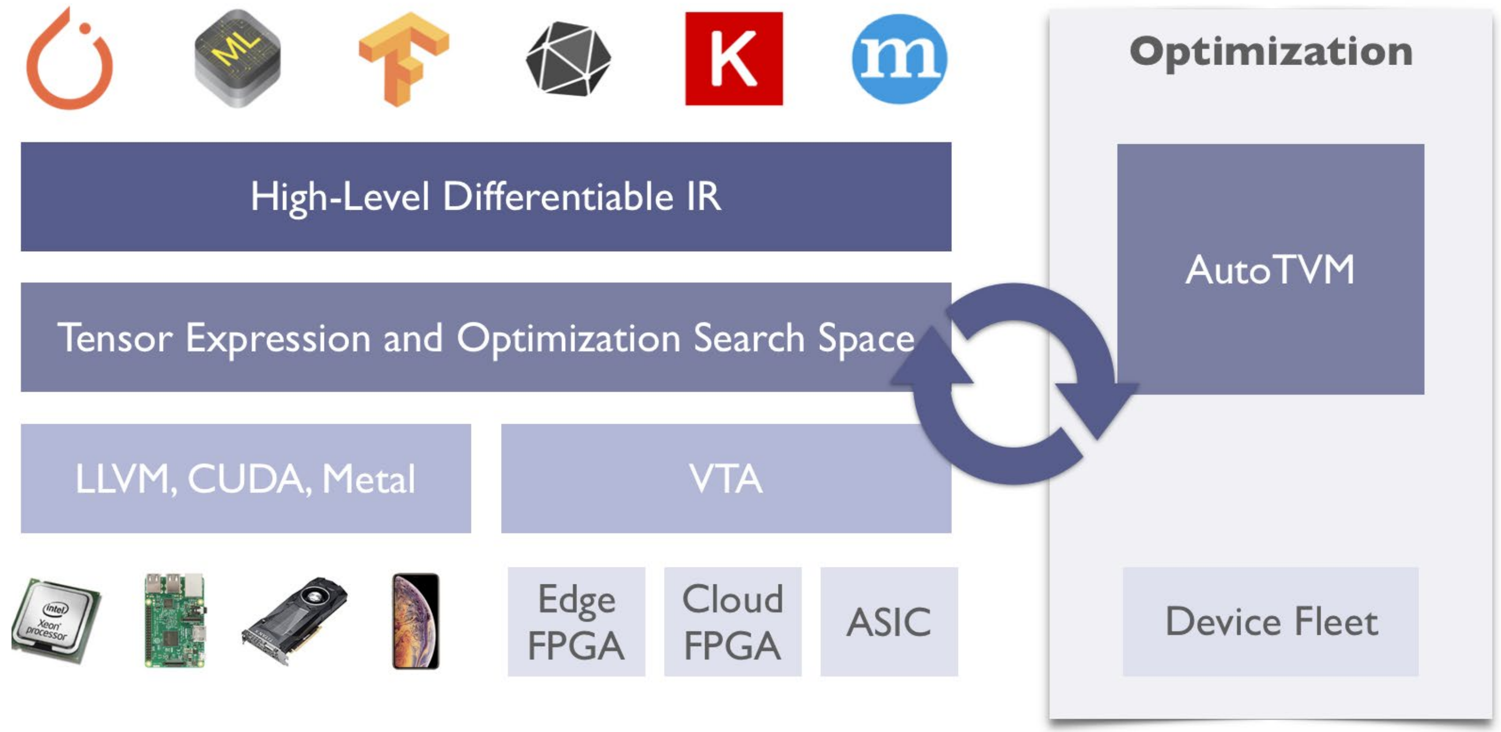
- Automated optimization given schedule space and target device



TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. Tianqi Chen, et, al., University of Washington.

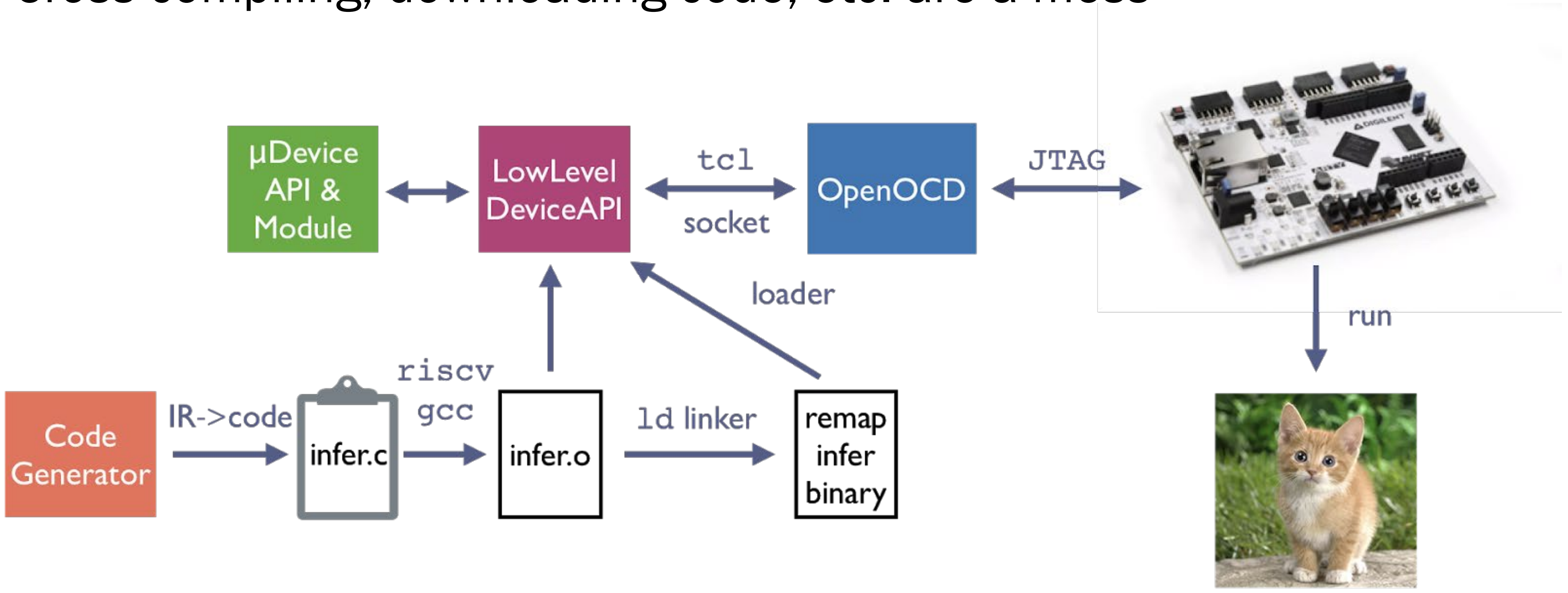
Automatic code generation

- Bridges hardware with high-level deep learning frameworks


Optimization
High-Level Differentiable IR
Tensor Expression and Optimization Search Space
AutoTVM
LLVM, CUDA, Metal
VTA
Edge FPGA
Cloud FPGA
ASIC
Device Fleet

Performance evaluation for SoCs

- Embedded SoCs are resource-constraint
  - RPC-based solutions probably won't work (lack of OS and network)
  - Cross-compiling, downloading code, etc. are a mess



Outline

- Glossary
- **Automatic code generation for RoCC accelerators**
- Performance evaluation platform design
- Case study of Gemmini

The tensorize schedule

- Accelerators provide micro-kernels for specific type of computation
  - Often with limit in input shape (corresponding to memory, computation units, etc.)
  - “Tensor intrinsics”
  - E.g. GEMM, convolution, etc.
- Code generation framework uses such intrinsics to offload computation to accelerator
  - Marks loop layers in nested loop program to be replaced by intrinsic call

Example of generated kernel

```
produce C {
  for (i.o, 0, 8) {
    for (j.o, 0, 8) {
      for (i.i, 0, 8) {
        for (j.i, 0, 8) {
          C[i.o*8+i.i][j.o*8+j.i] = 0}}
      for (k.o, 0, 8) {
        for (i.i, 0, 8) {
          for (j.i, 0, 8) {
            for (k.i, 0, 8) {
              C[i.o*8+i.i][j.o*8+j.i] +=
              ↪  A[i.o*8+i.i][k.o*8+k.i] *
              ↪  B[k.o*8+k.i][j.o*8+j.i]}}}}}}}
```

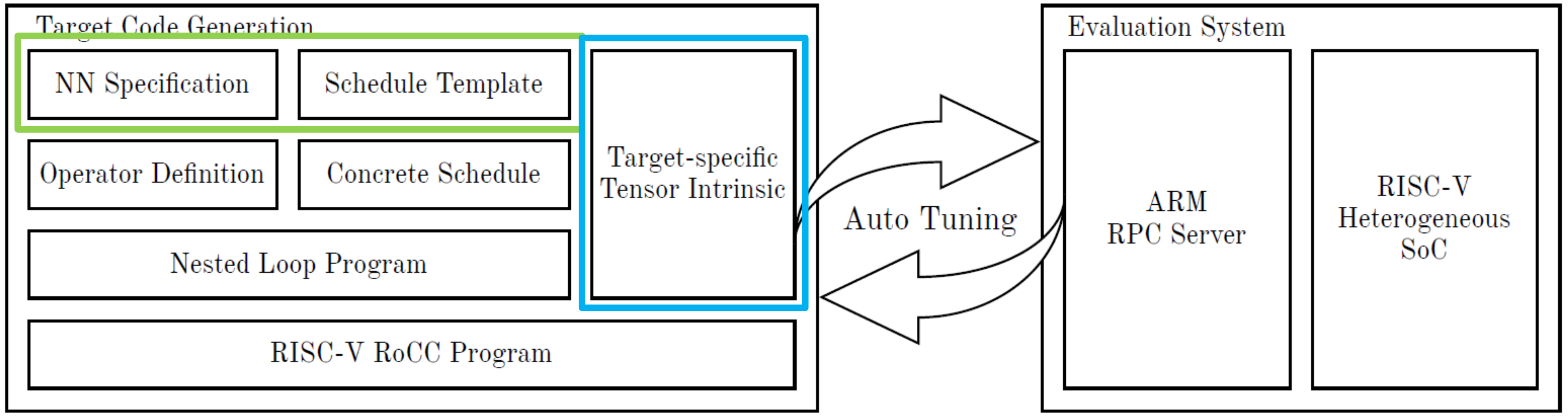
Tensorize

```
produce C {
  // attr pragma_epilogue = "do_fence"
  for (i.o, 0, 8) {
    for (j.o, 0, 8) {
      matmul_reset(access_ptr(C), 512)
      for (k.o, 0, 8) {
        matmul_kernel(access_ptr(A), access_ptr(B),
        ↪  access_ptr(C), 512, 512, 512)}
      matmul_finalize(access_ptr(C), 512)}}}
```

Assuming "matmul" kernel that can handle 8x8x8 GEMM

Overall framework

- Accelerator developer provides tensor intrinsic implementation
- User defines network and schedule template
- Framework generates accelerated target program



Tensor intrinsic design

- An intrinsic should be of the “reset-update-finalize” pattern:
  - Reset is called to initialize output region (in SoC memory)
  - Update is called to combine partial results (in accelerator memory)
  - Finalize is called to move output (back to SoC memory)
- Physical constraints of accelerator (memory, etc.) encoded in the intrinsic declaration
- Focus on data movement
  - Computation is getting fast
  - Data movement takes up about the same time as computation does

Memory consistency

- Memory ordering in heterogeneous SoCs are complicated:
  - Modern SoCs often feature multi-level hierarchical memory
  - Accelerators use asynchronous DMA for high performance
- Enforcing ordering may be necessary
  - Fences
  - TLB flush

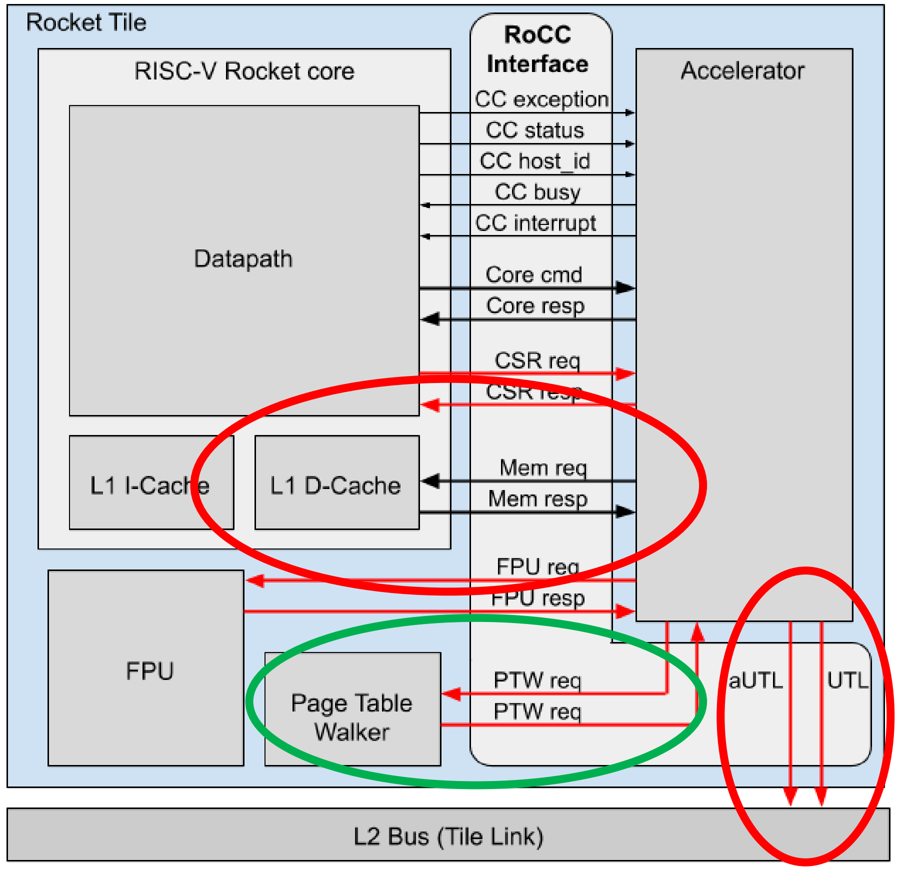


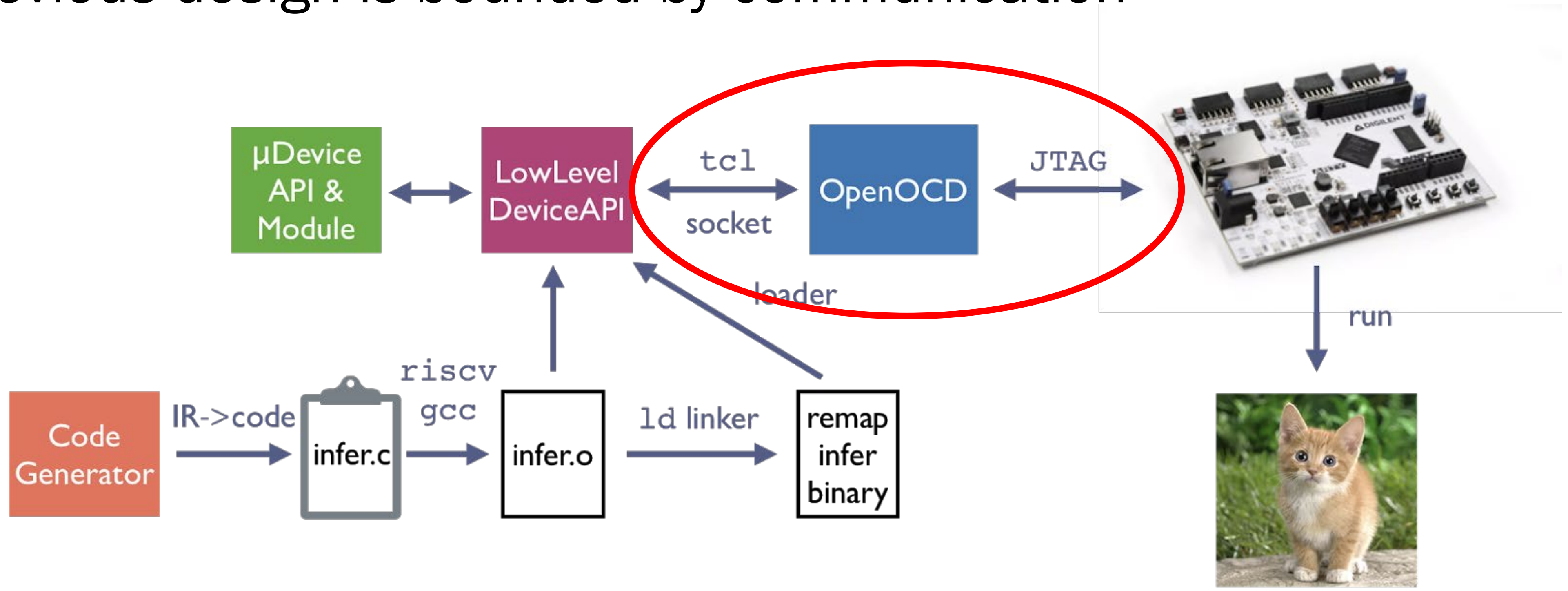
Figure 1: Default (black) & extended (red) signals of the RoCC interface

Outline

- Glossary
- Automatic code generation for RoCC accelerators
- <u>**Performance evaluation platform design**</u>
- Case study of Gemmini

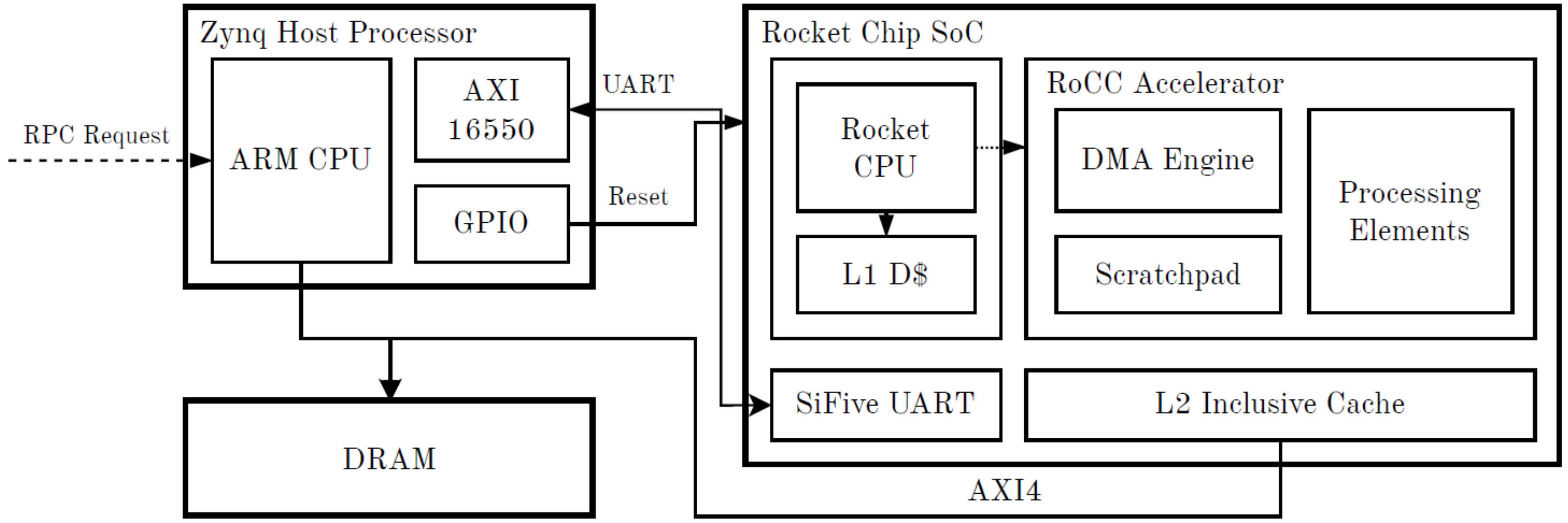
Code quality evaluation for SoCs

- Necessary for automatic code generation
  - Forms the closed ring of automatic tuning
- Previous design is bounded by communication

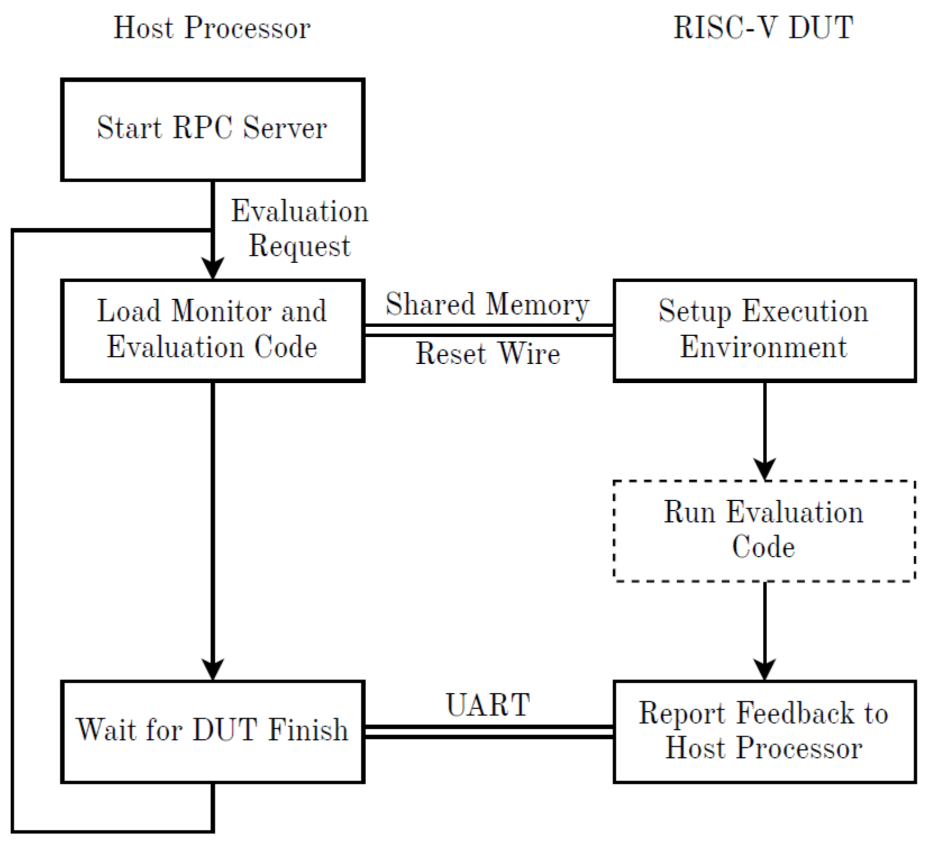


Evaluation system design

- Based on shared-memory FPGA platforms: high bandwidth
  - Zynq, FPGA over PCIe, etc.
- Simplified protocol implementation with UART and reset



Evaluation system workflow

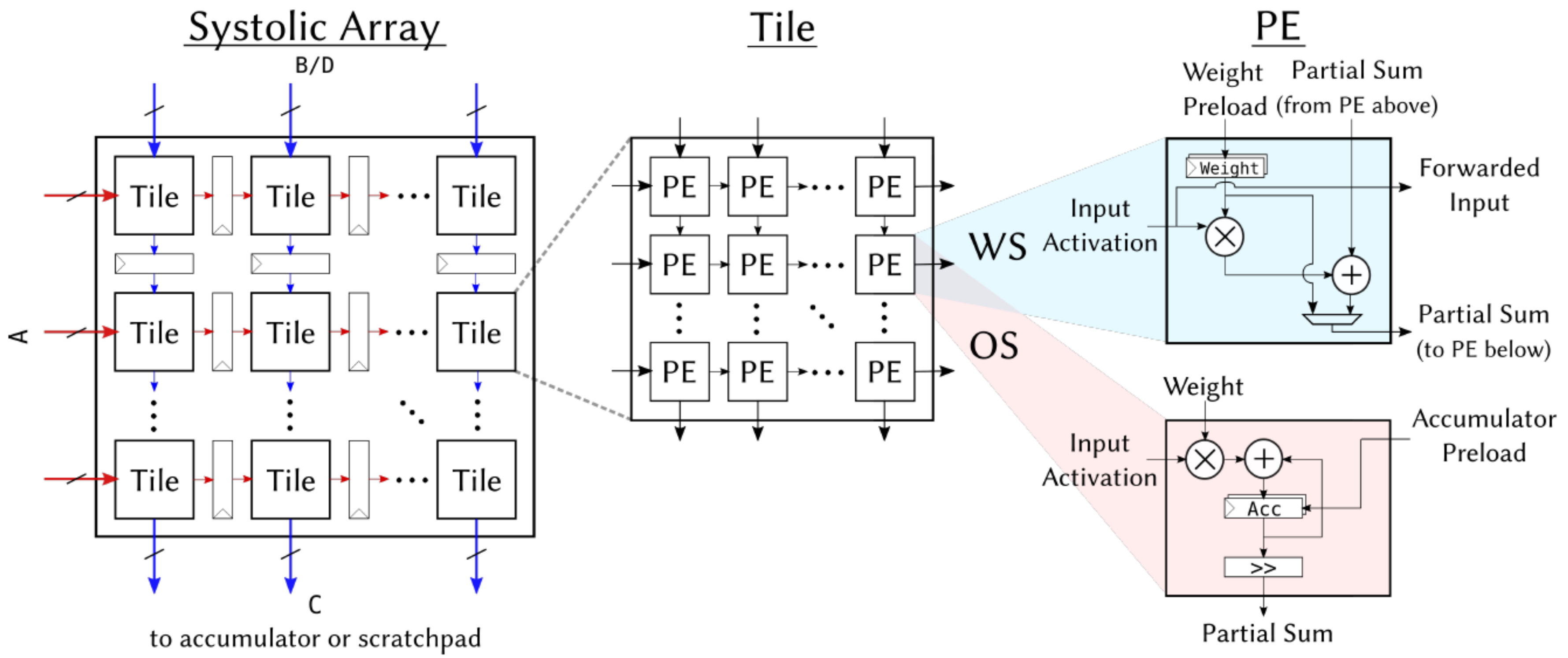


Outline

- Glossary
- Automatic code generation for RoCC accelerators
- Performance evaluation platform design
- **Case study of Gemmini**

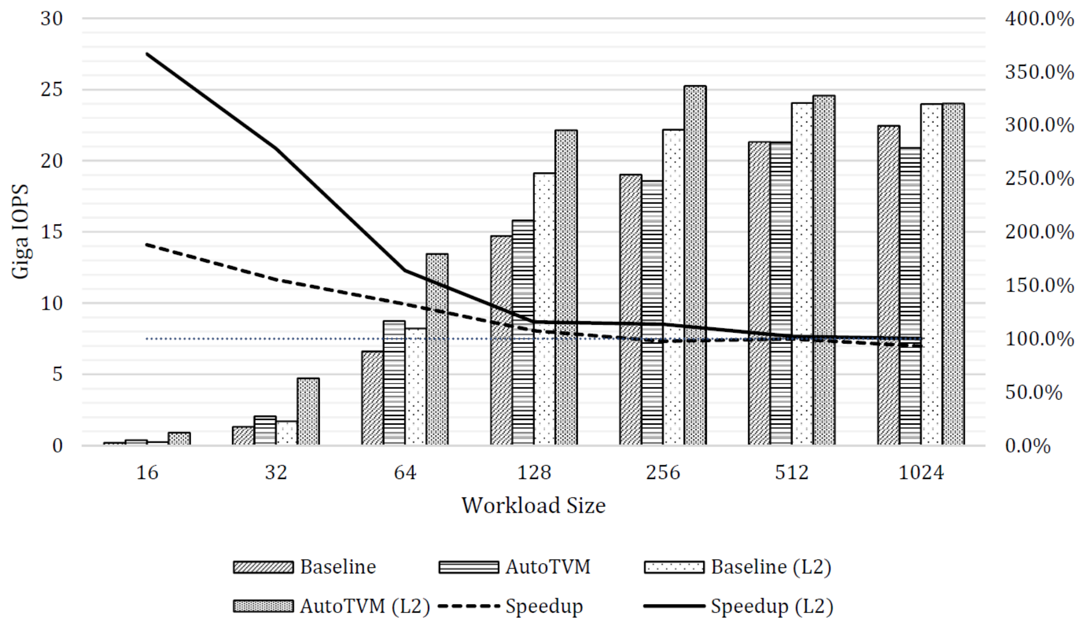
Gemmini the GEMM accelerator

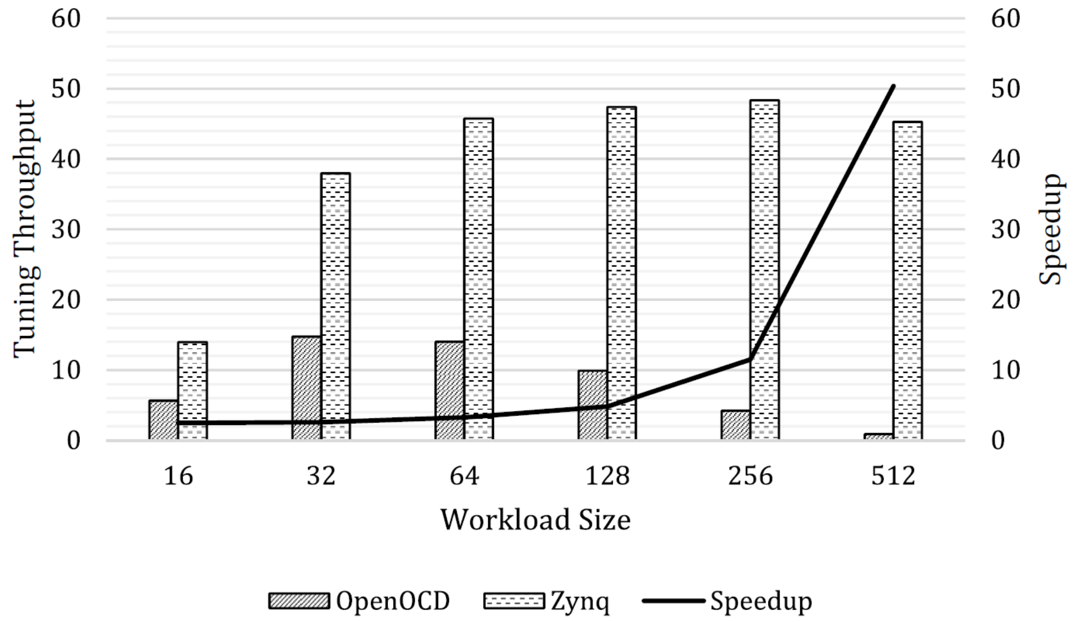
- Systolic array design for GEMM using RoCC interface



Results

- Under 100 MHz clock, compared to hand-tuned results:
  - Best-case 25.24 GIOPS, 3.6x speedup; same performance overall
- Tuning system shows over 50x speedup of tuning throughput
  - Communication bandwidth is no longer the bottleneck





Takeaways

- Automatic code generation flow for RoCC accelerators
  - Improves productivity for system and application developers
- Evaluation platform that make automatic tuning on SoC targets realistic
  - Enables automatic tuning for larger group of accelerators
- Case study of Gemmini under 100 MHz using proposed flow and system
  - Best case speedup in generated code of 3.6x, same performance overall
  - Tuning system show speedup of 50x for tuning throughput

Future work

- The current implementation does not yet support full-network generation due to a limitation in the code generation framework
    - Shall be fixed soon
- Evaluation on a wider range of accelerator designs

Thank you!