Berkeley Architecture Research

Gemmini Introduction

Seah Kim
UC Berkeley
seah@berkeley.edu

Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration
DAC Best Paper Award
Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanovic, Borivoje Nikolic, Yakun Sophia Shao
Design Automation Conference (DAC), December 2021.

Getting Started / Logistics

Setup

Why Gemmini?

Gemmini Architecture and Template

Gemmini Programming Model

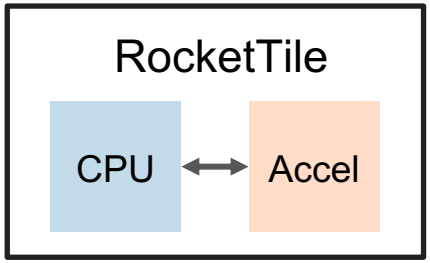
Setup

Pre-Fetch RTL Build

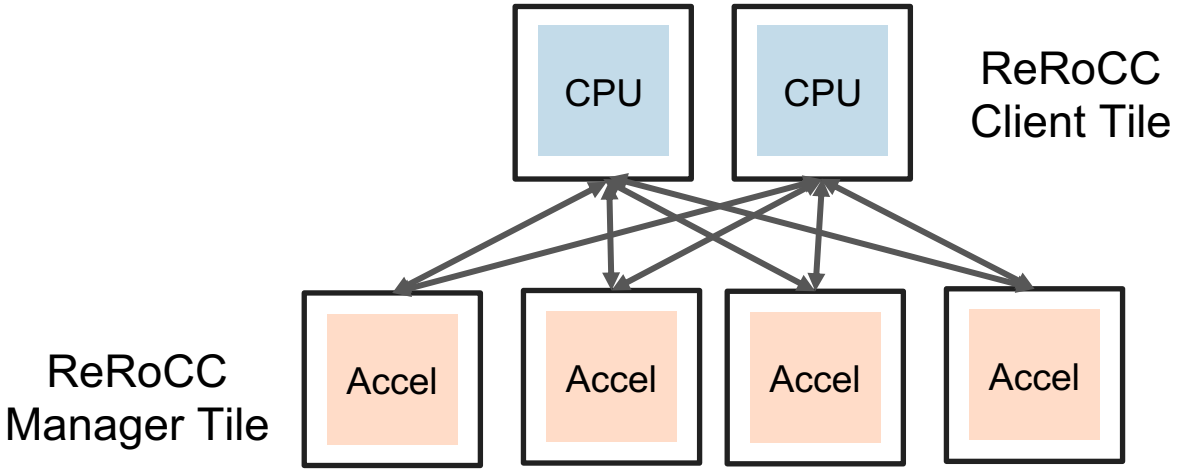
Two configs we will use

- TutorialGemminiRocketConfig: Single-core Rocket + Gemmini (For Gemmini part)
- TutorialGemminiReRoCCConfig: Dual-core Rocket + 4 Gemminis (For AuRORA part)

GemminiRocketConfig



GemminiReRoCCConfig



Pre-Fetch RTL Build

Two configs we will use

- TutorialGemminiRocketConfig: Single-core Rocket + Gemmini (For Gemmini part)
- TutorialGemminiReRoCCConfig: Dual-core Rocket + 4 Gemminis (For AuRORA part)

Build rtl sim for gemmini rerocc config

```
tmux new -s gemminibuild
```

```
cd $MCYDIR/sims/verilator
```

➔ GemminiRocketConfig is prebuilt. Let's run

```
make CONFIG=TutorialGemminiReRoCCConfig
```

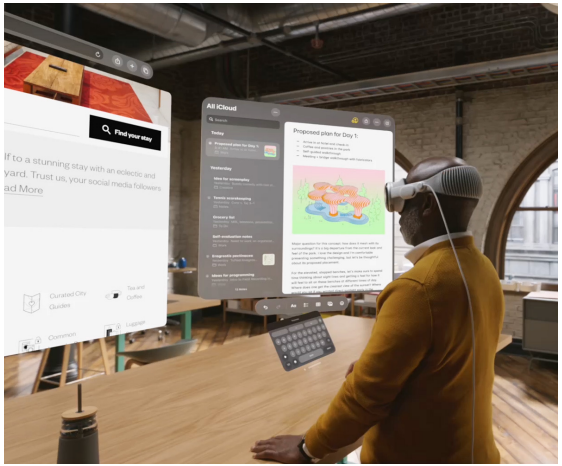
```
[centos@ip-192-168-3-190 chipyard-morning]$ source env.sh
[centos@ip-192-168-3-190 chipyard-morning]$ cd sims/verilator/
[centos@ip-192-168-3-190 verilator]$ make CONFIG=TutorialGemminiReRoCCConfig
```

Why Gemmini?

DNN is Prevalent!

Smartphone, AR/VR, autonomous driving, etc

https://www.samsung.com/us/galaxy-ai/

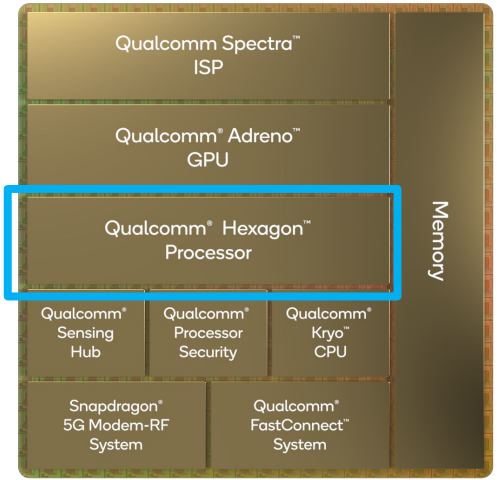


https://www.apple.com/newsroom/2023/06/introducing-apple-vision-pro/

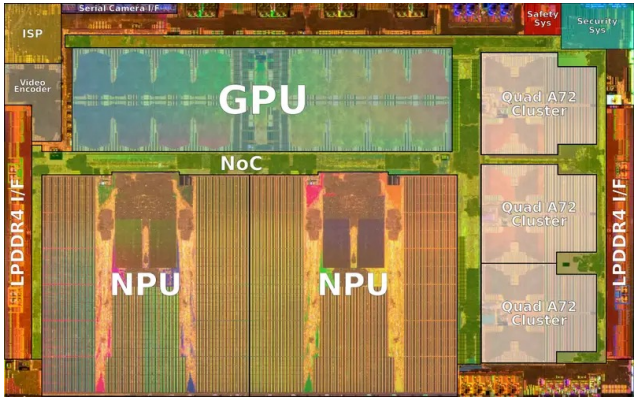
https://www.tesla.com/autopilot

The ERA of DNN Accelerators

DNN accelerators are exploding popularity in different scale…



Qualcomm
Snapdragon



Tesla
FSD

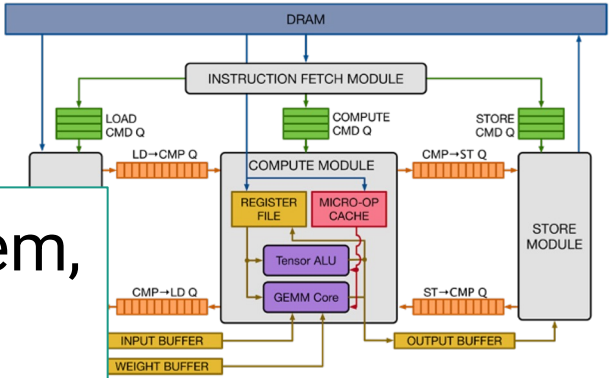
Cloud
TPU

DNN Accelerator Generators

In need of DNN accelerators generators for comp arch research

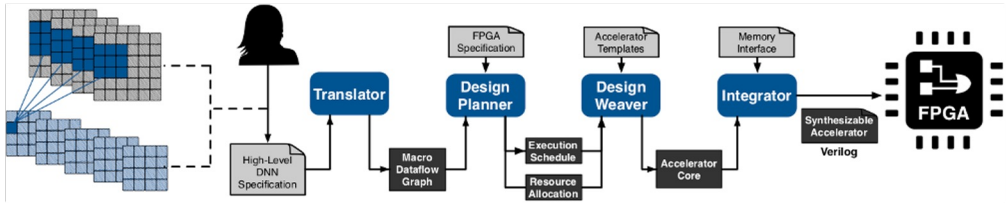
NVDLA.org

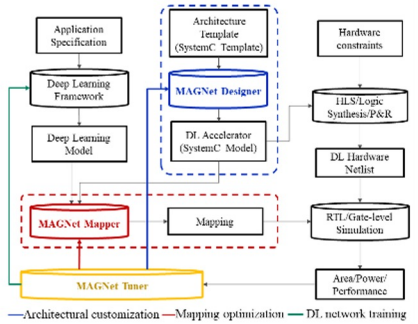
VTA



Yet, we need full-system, full-stack visibility

DNNWeaver

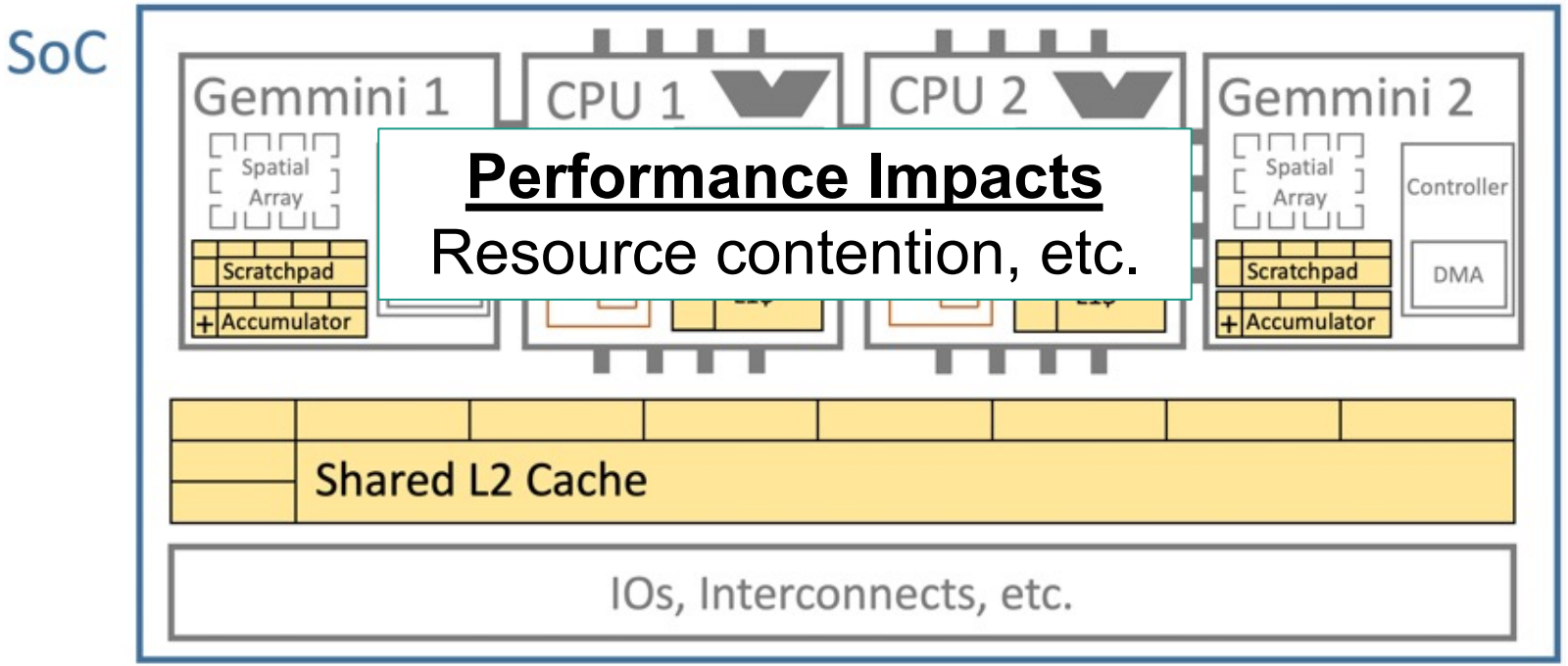




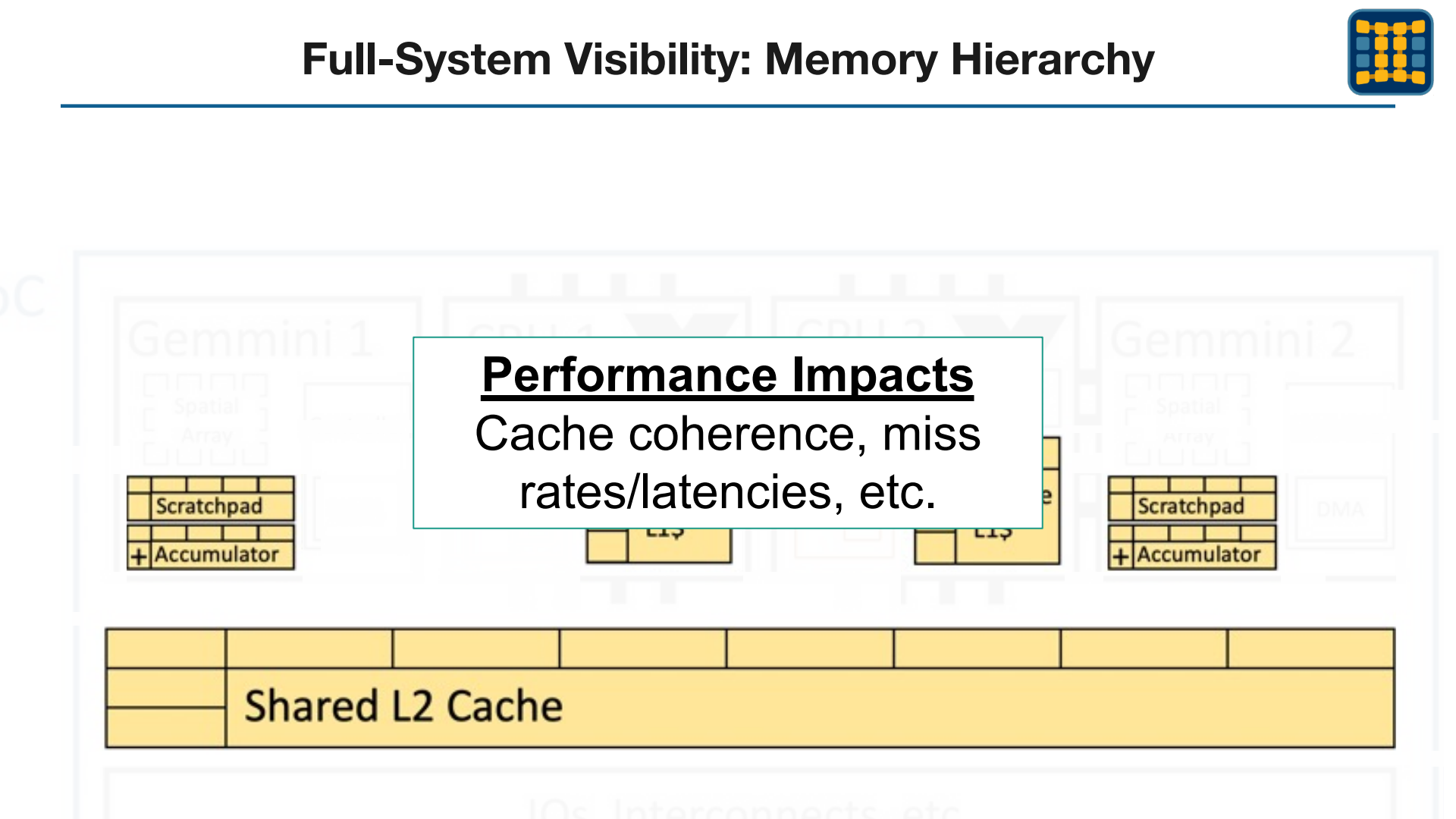
MAGNet

Full-System Visibility

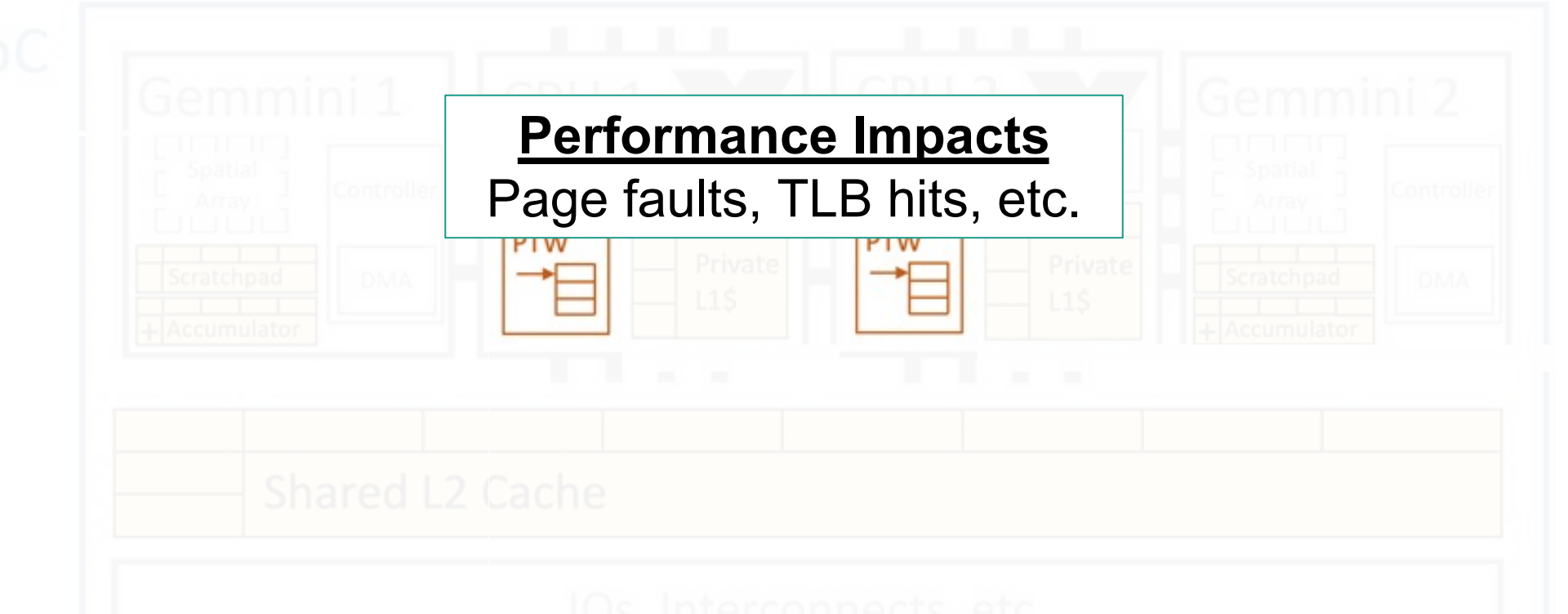
Full Stack Visibility



Full-System Visibility: Memory Hierarchy



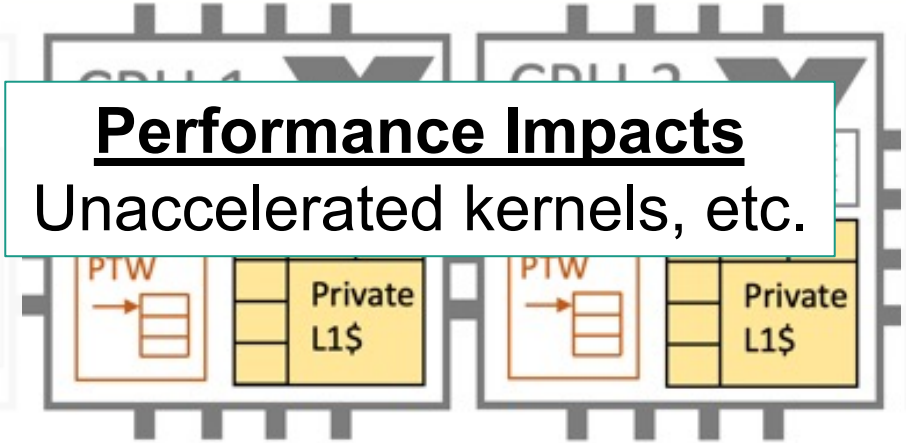
Full-System Visibility: Virtual Addresses



Performance Impacts

Page faults, TLB hits, etc.

Full-System Visibility: Host CPUs



Performance Impacts

Unaccelerated kernels, etc.

Full-Stack Visibility

Direct hardware configuration, low-level ISA

Gemmini Architecture & Template

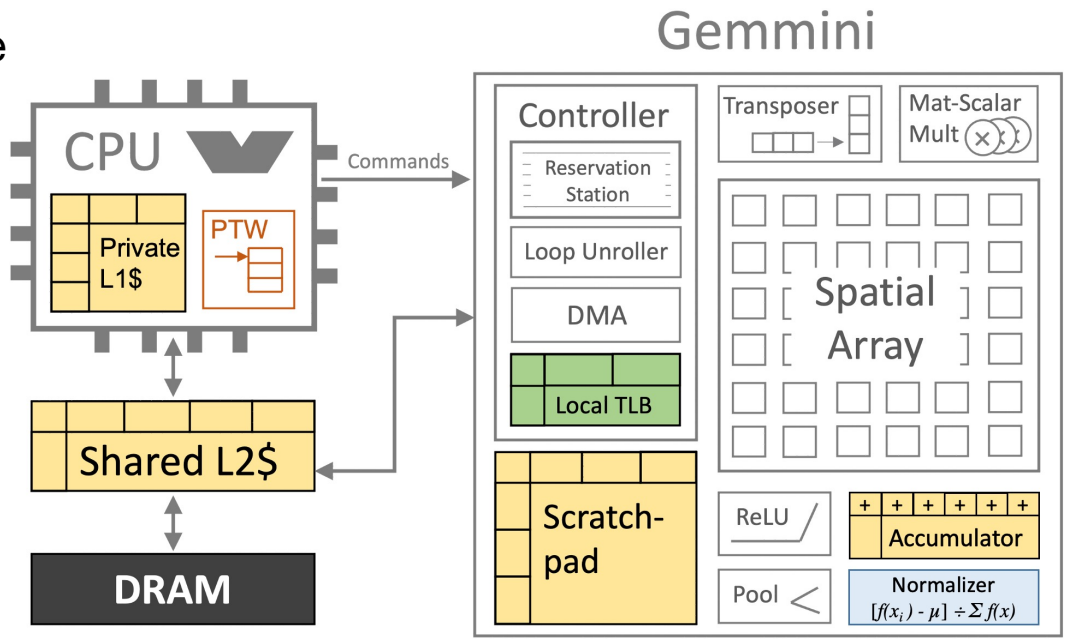
Gemmini

DNN accelerator generator

- High utilization dense matmul
- Direct convolution

Flexible hardware template

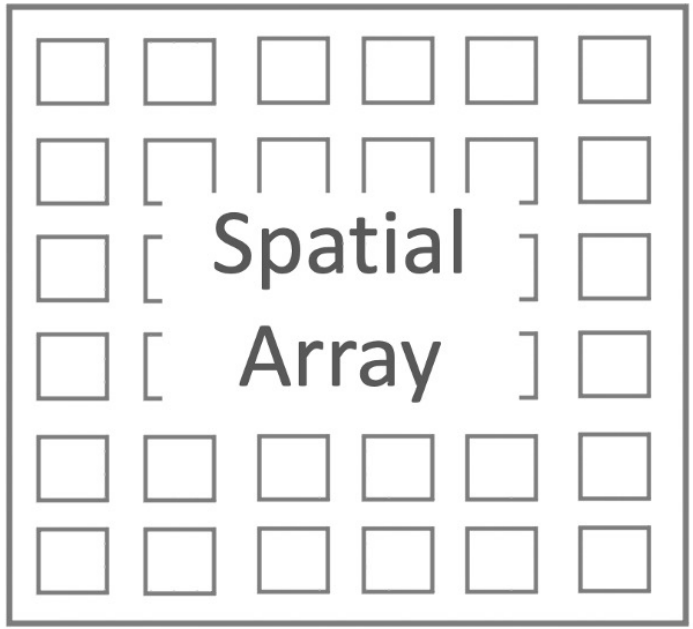
Full-system, full-stack



Gemmini: Spatial Array

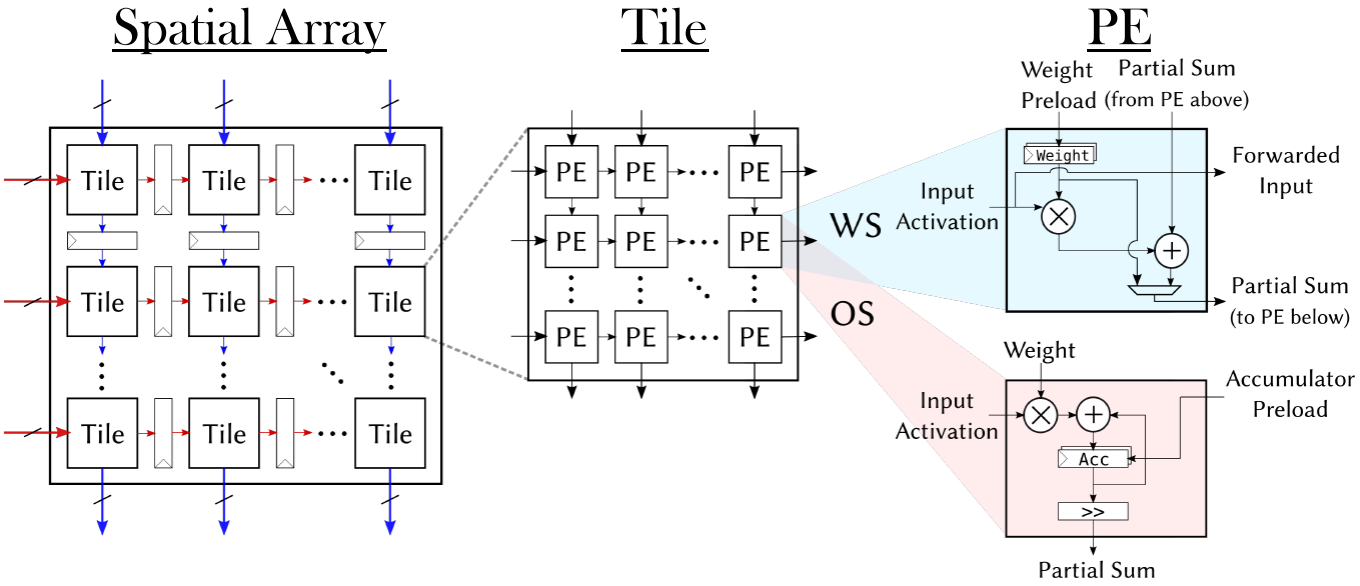
Parameters:

- Dataflow
- Dimensions
- Pipelining



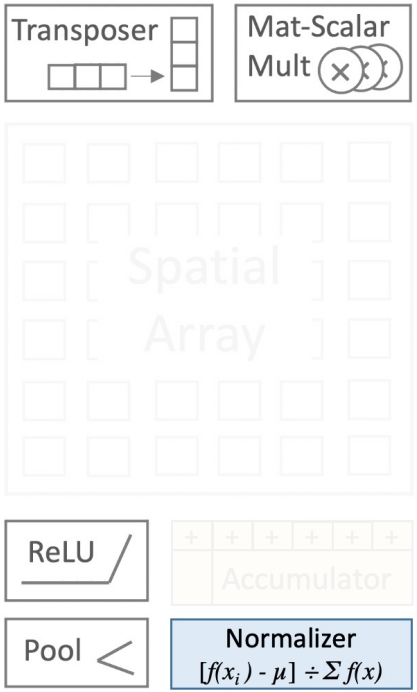
Gemmini: Spatial Array

- Parameters:
  - Dataflow
  - Dimensions
  - Pipelining



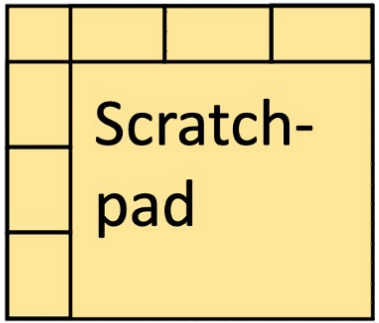
Gemmini: Non-GEMM Functionality

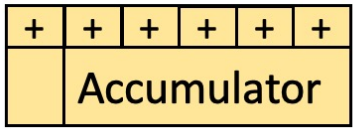
- Can be optimized out at elaboration-time



Gemmini: Local Scratchpad and Accumulator

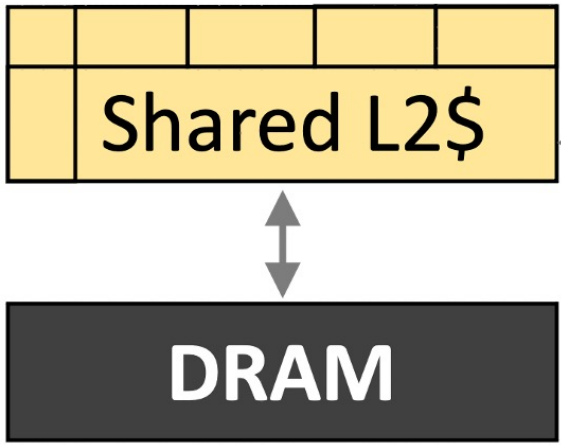
- Parameters:
  - Capacity
  - Banks
  - Single- or dual-port





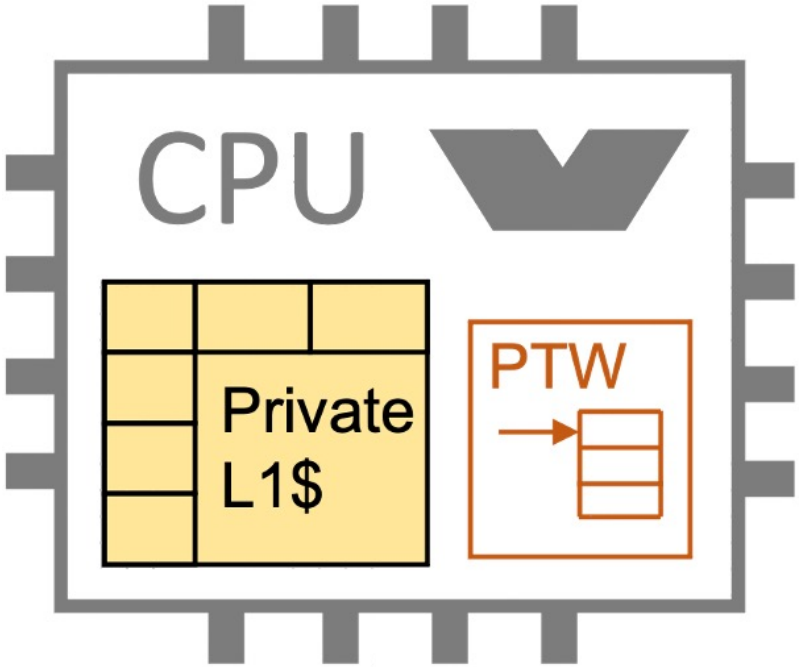
Gemmini: Global Memory

- Parameters:
  - Capacity
  - Banks
  - Optional L3
  - DRAM controller



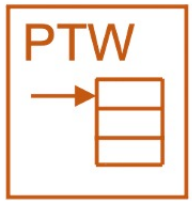
Gemmini: Host CPU

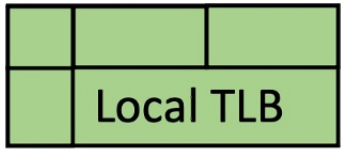
- Parameters:
  - In-order/out-of-order
  - ROB capacity
  - L1 capacity
  - Branch predictor



Gemmini: Virtual Address Translation

- Parameters:
  - TLB capacity
  - TLB hierarchy
    - e.g. L2 TLB





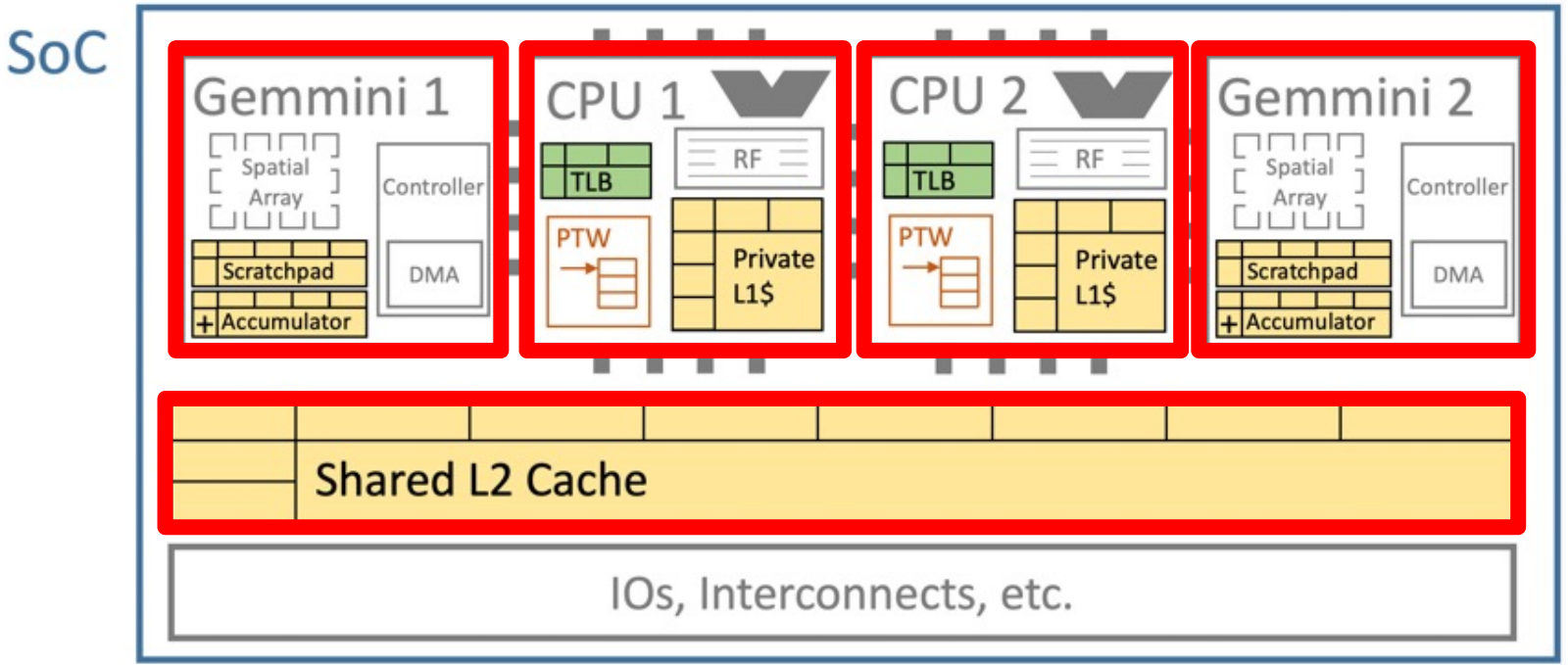
Gemmini: Reservation Station

- Parameters:
  - Size
    - Separate entries for ld-, st-, and execute-type instructions

Gemmini: Loop Unroller

- Dynamically schedules operations
- Parameters:
  - Types of loops to unroll in hardware
    - Only inference kernels?
    - Training *and* inference kernels?

Gemmini: Full SoC



Gemmini Configuration

Look into hardware configuration

How to change parameter

View Gemmini Configs

```
cd $MCYDIR/gemmini-hw
# Symlink to chipyard-morning/generators/gemmini/src/main/scala

vim Configs.scala # ConfigsFP.scala for Floating Point
```

```
// Ld/Ex/St instruction queue lengths
ld_queue_length = 8,
st_queue_length = 2,
ex_queue_length = 8,

// DMA options
max_in_flight_mem_reqs = 16,

dma_maxbytes = 64,
dma_buswidth = 128,

// TLB options
tlb_size = 4,
```

```
meshRows = 16,
meshColumns = 16,

// Spatial array PE options
dataflow = Dataflow.WS,

// Scratchpad and accumulator
sp_capacity = CapacityInKilobytes(256),
acc_capacity = CapacityInKilobytes(64),

sp_banks = 4,
acc_banks = 2,

sp_singleported = true,
acc_singleported = false,
```

View SoC Configs

```
cd $MCYDIR/generators/chipyard/src/main/scala

vim configs/RoCCAcceleratorConfig.scala
```

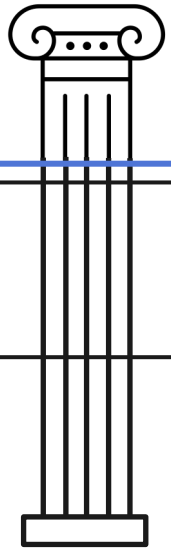
Example SoC config:

```
class TutorialGemminiRocketConfig extends Config(
  new gemmini.LeanGemminiConfig ++
  new freechips.rocketchip.rocket.WithNHugeCores(1) ++
  new chipyard.config.WithSystemBusWidth(128) ++
  new chipyard.config.AbstractConfig)
```

Gemmini Programming Model

Programming Model

ONNX



High

```
matmul(…); conv(…); residual_add(…);
max_pool(…); global_averaging(…)
```

Hand-tuned C library for DNNs

Medium

```
configure_loads(…); configure_stores(…);
preload_spatial_array(…); feed_spatial_array(…)
```

Direct hardware configuration, low-level ISA

Low

Mid-Level: Hand-Tuned Library

C library of kernels

- `#include "gemmini.h"`

```
In $MCYDIR/gemmini-sw/include
-> Symlink to generators/gemmini/software/gemmini-rocc-tests/include
```

```
void tiled_matmul(...

void tiled_resadd(...

// Convs and max-pools are fused
// together
void tiled_conv(...
```

Mid-Level: Hand-Tuned Library

C library of kernels

- `#include "gemmini.h"`

Requires a C description of the accelerator configuration

- `#include "gemmini_params.h"`

```
#define DIM ...

typedef elem_t ...
```

Mid-Level: Hand-Tuned Library

C library of kernels

- `#include "gemmini.h"`

Requires a C description of the accelerator configuration

- `#include "gemmini_params.h"`

Write your own Gemmini applications

```
// My workload

#include “gemmini.h”

int main() {
  elem_t image[...][...][...];

  tiled_conv(image, ...);

  // ...
}
```

Mid-Level: Hand-Tuned Library

C library of kernels

- `#include "gemmini.h"`

Requires a C description of the accelerator configuration

- `#include "gemmini_params.h"`

Write your own Gemmini applications

Performs loop tiling

- Uses heuristics to maximize buffer usage

```
// Global memory level
for (int i0 = 0; i0 < N; i0 += tile_I) {
  // Scratchpad level
  for (int i = 0; i < tile_I; i++) {
    // ...
  }
}
```

Mid-Level: Hand-Tuned Library

C library of kernels

- `#include "gemmini.h"`

Requires a C description of the accelerator configuration

- `#include "gemmini_params.h"`

Write your own Gemmini applications

Performs loop tiling

- Uses heuristics to maximize buffer usage

Assumes NHWC format

Mid-Level: Hand-Tuned Library

Matmuls

- Transposed
- Strided

Convolutions

- Transposed
- Kernel or input dilated
- Strided
- Max-pooling

Matrix additions

- Scaled

Global-averaging

Mid-Level: Gemmini ISA Example Program

```
cd $MCYDIR/gemmini-sw
vim include/gemmini.h
vim bareMetalC/tiled_matmul_ws_perf.c

./build.sh
```

```
tiled_matmul_auto(MAT_DIM_I, MAT_DIM_J, MAT_DIM_K,
        (elem_t*)full_A, (elem_t*)full_B, NO_BIAS ? NULL : &full_D[0][0], (elem_t*)full_C,
        A_STRIDE, B_STRIDE, MAT_DIM_J, MAT_DIM_J,
        MVIN_SCALE_IDENTITY, MVIN_SCALE_IDENTITY, MVIN_SCALE_IDENTITY,
        ACTIVATION, ACC_SCALE_IDENTITY, 0, REPEATING_BIAS,
        A_TRANSPOSE, B_TRANSPOSE,
        false, false,
        0,
        WS);
```

Mid-Level: Gemmini ISA Example Program

```
cd $MCYDIR/sims/verilator

make CONFIG=TutorialGemminiRocketConfig run-binary-hex
BINARY=../../gemmini-sw/build/bareMetalC/tiled_matmul_ws_perf-
baremetal

# WS matmul
```

```
Starting gemmini matmul
I: 128, J: 256, K: 256
NO_BIAS: 0, REPEATING_BIAS: 1
A_TRANSPOSE: 0, B_TRANSPOSE: 0
Cycles taken: 37567
Total macs: 8388608
Ideal cycles: 32768
Utilization: 87%
```

Low-Level: Gemmini ISA

Only necessary if you:

- Want to write your own mid-level kernels
- Performance tuning

Low-level assembly instructions

- Wrapped in C “#define” macros
- `#include "gemmini.h"`

Example Gemmini ISA:

```
#define gemmini_extended_mvout(dram_addr, spad_addr, cols, rows) \
  ROCC_INSTRUCTION_RS1_RS2(XCUSTOM_ACC, dram_addr, ((uint64_t)(rows) << (ADDR_LEN + 16)) | ((uint64_t)(cols) << ADDR_LEN) | (uint64_t)(spad_addr), k_MVOUT)

#define gemmini_extended_mvin(dram_addr, spad_addr, cols, rows) \
  ROCC_INSTRUCTION_RS1_RS2(XCUSTOM_ACC, dram_addr, ((uint64_t)(rows) << (ADDR_LEN + 16)) | ((uint64_t)(cols) << ADDR_LEN) | (spad_addr), k_MVIN)

#define gemmini_extended_compute_preloaded(A, BD, A_cols, A_rows, BD_cols, BD_rows) \
  ROCC_INSTRUCTION_RS1_RS2(XCUSTOM_ACC, ((uint64_t)(A_rows) << (ADDR_LEN + 16)) | ((uint64_t)(A_cols) << ADDR_LEN) | (uint64_t)(A), ((uint64_t)(BD_rows) << (AD
DR_LEN + 16)) | ((uint64_t)(BD_cols) << ADDR_LEN) | (uint64_t)(BD), k_COMPUTE_PRELOADED)
```

Low-Level: Programming Model

Gemmini is programmed using custom RISC-V instructions

- ”RoCC” interface

  “Rocket-Chip Coprocessor”

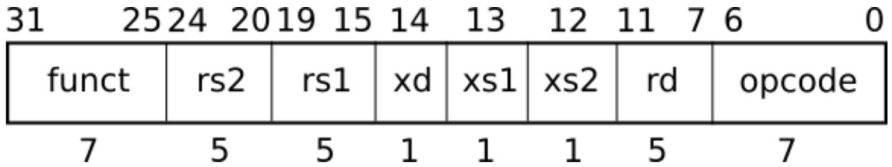
Gemmini instructions are interspersed between ordinary CPU instructions

Gemmini does *not* fetch its own instructions

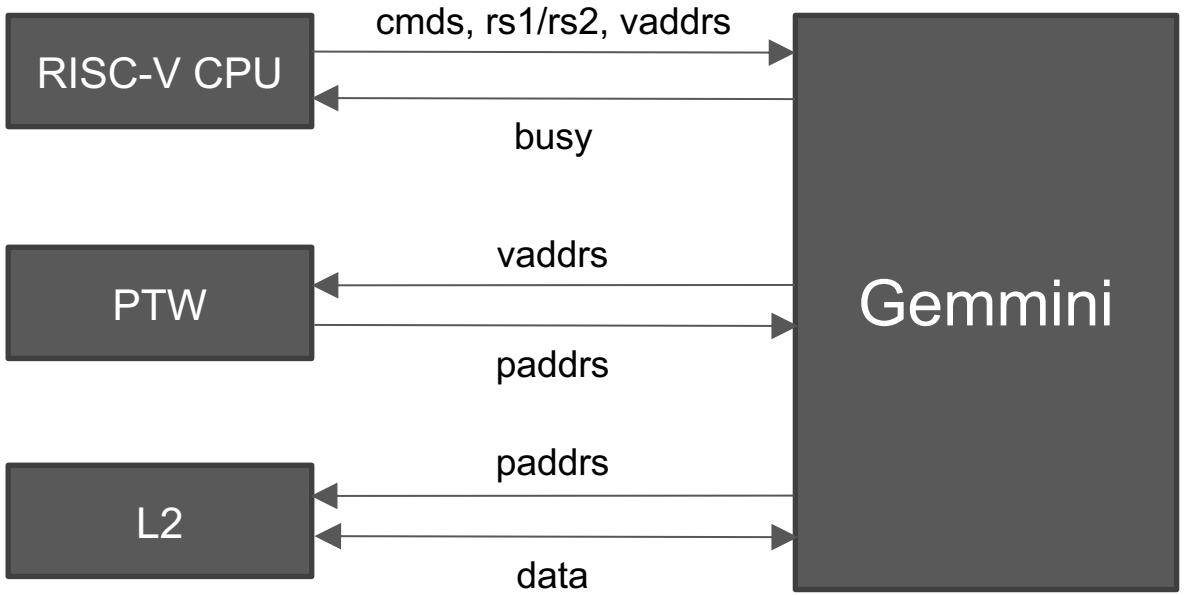
Whenever the CPU encounters a RoCC instruction

- Checks the opcode to find target
- Dispatches to target co-processor
- *Immediately* continues CPU execution

  Synchronize with fences



Low-Level: Gemmini Interface With CPU



Low-Level: Decoupled Access-Execute Pipelines

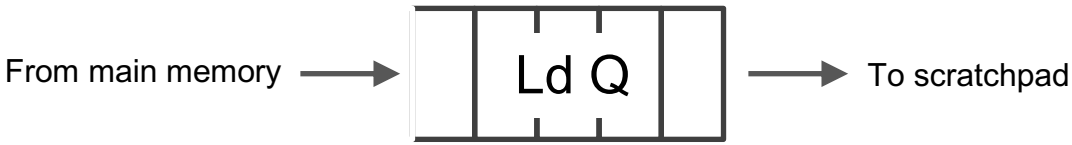
Three main “types” of instructions:

Low-Level: Decoupled Access-Execute Pipelines

Three main “types” of instructions:

- Load instructions

  Main memory -> Scratchpad



Low-Level: Decoupled Access-Execute Pipelines

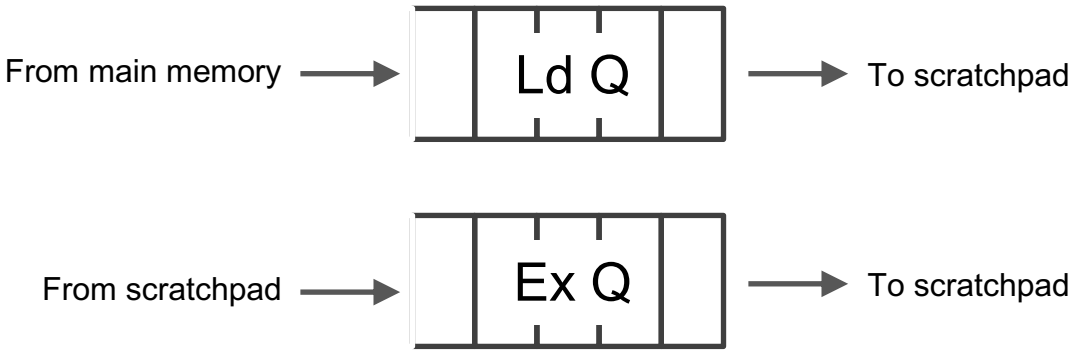
Three main “types” of instructions:

- Load instructions

  Main memory -> Scratchpad

- Execute instructions

  Scratchpad -> Scratchpad



Low-Level: Decoupled Access-Execute Pipelines

Three main “types” of instructions:

- Load instructions

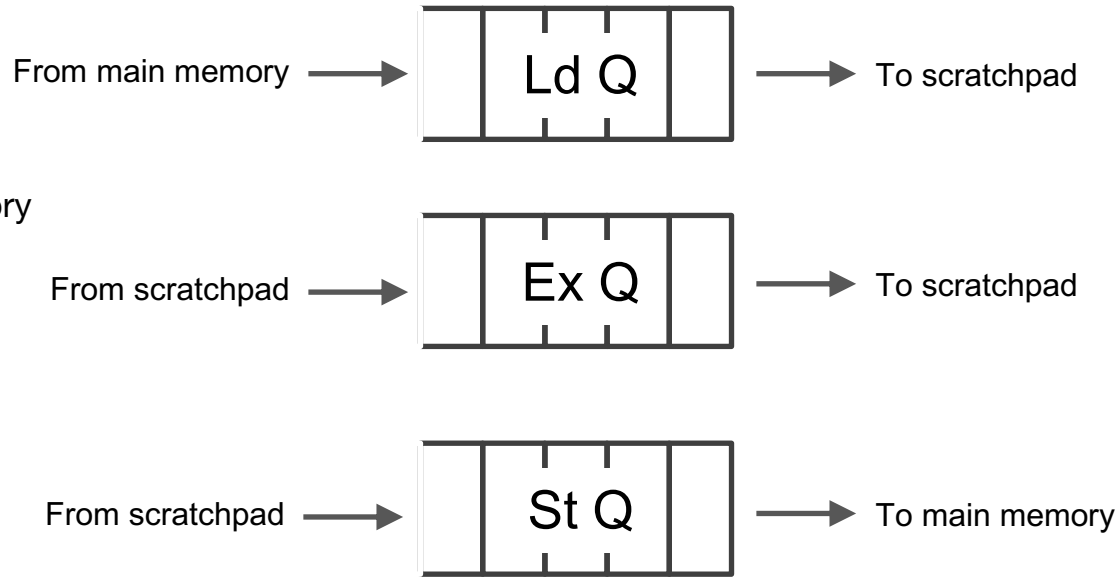
  Main memory -> Scratchpad

- Execute instructions

  Scratchpad -> Scratchpad

- Store instructions

  Scratchpad -> Main memory

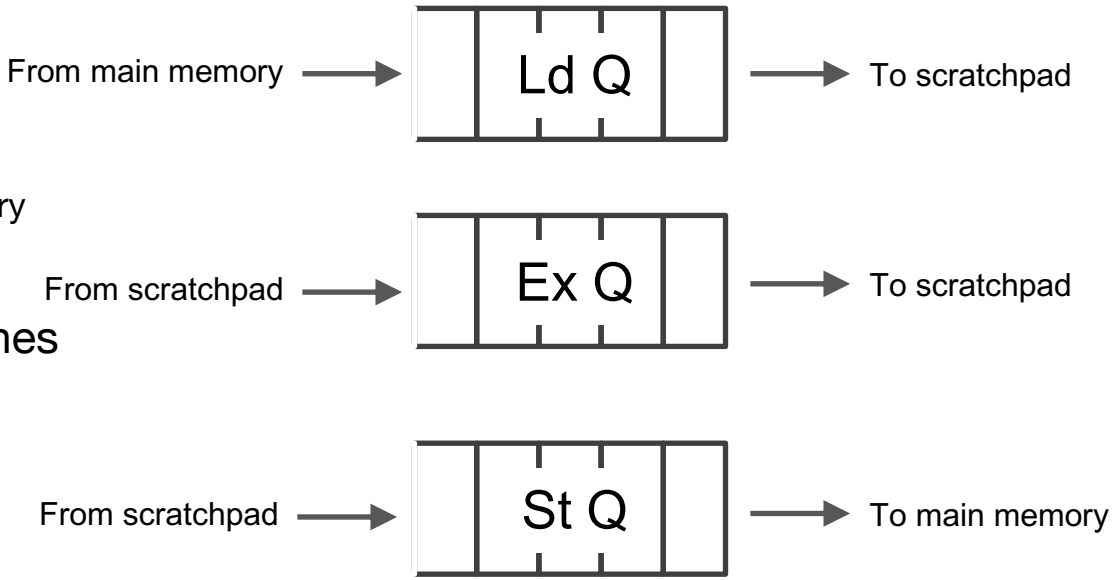


Low-Level: Decoupled Access-Execute Pipelines

Three main “types” of instructions:

- Load instructions
  - Main memory -> Scratchpad
- Execute instructions
  - Scratchpad -> Scratchpad
- Store instructions
  - Scratchpad -> Main memory

Matches Gemmini’s decoupled-access execute pipelines



Low-Level: Load Pipeline

Three “identical” load instructions:

- `gemmini_mvin1`
- `gemmini_mvin2`
- `gemmini_mvin3`

Each mvin has it’s own config registers

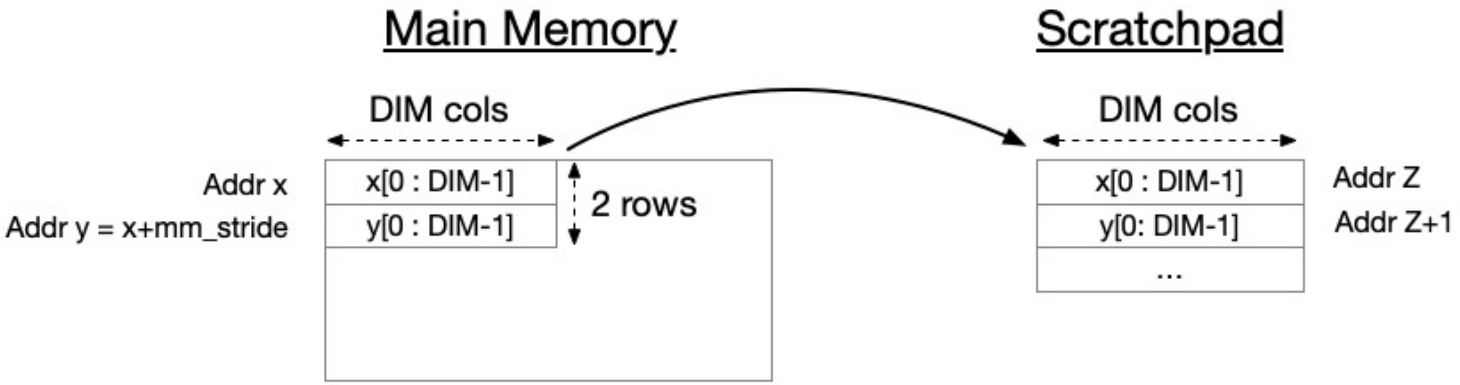
Intended use:

- Different mvin for Inputs, Weights, and Bias

Mvins are meant to move DIMxDIM **submatrices** from main memory into the scratchpad

- DIM is the dimension of the matmul unit

Low-Level: Mvin



Mvin Instruction Parameters

From: x (main memory address)

To: Z (private memory address)

Rows: 2 *Columns:* DIM

Main memory stride: mm_stride

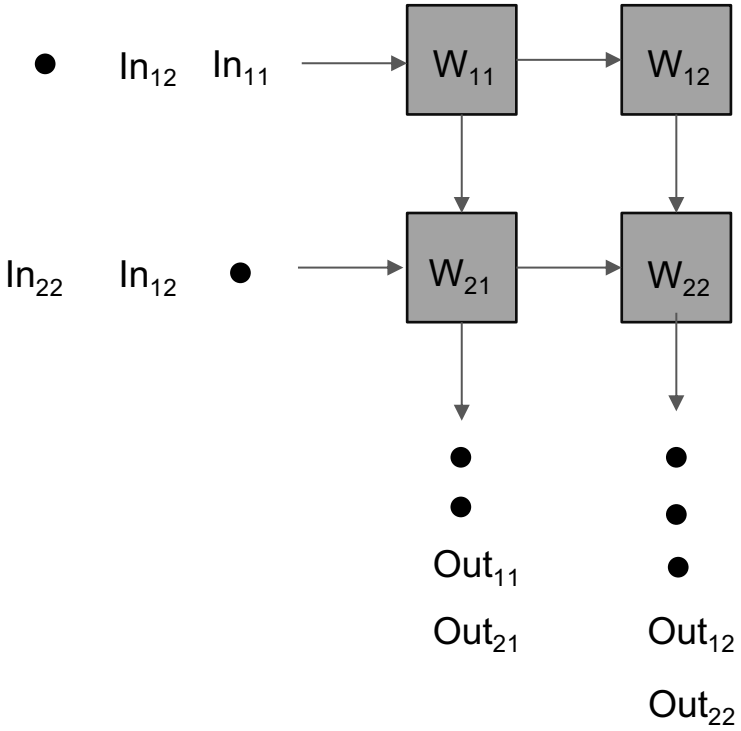
Low-Level: Execute Pipeline

"Preload" instruction loads stationary data into the matmul unit

- `preload`
- Weights for weight-stationary
- Bias/output for output-stationary

"Compute" instruction performs matmul after stationary data has been loaded

- `compute_preloaded`
- `compute_accumulated`



Low-Level: Store Pipeline

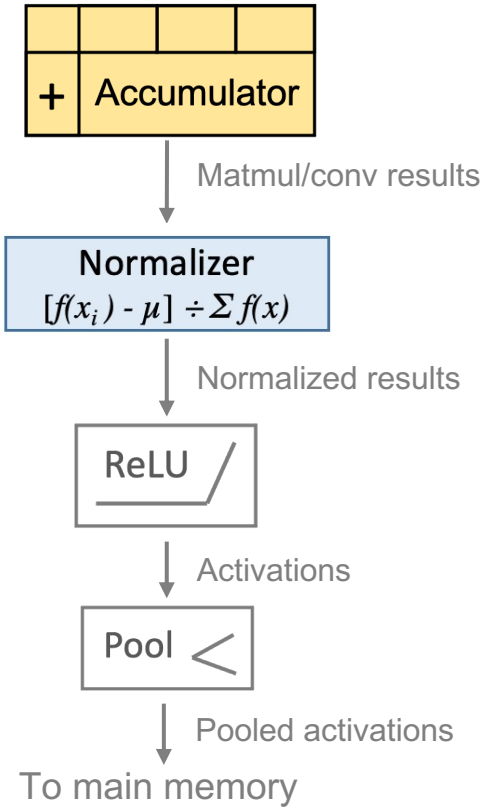
Very similar to load pipeline

- `gemmini_mvout`

Stores DIMxDIM **submatrices** into main memory

Normalization and **activation** during mvouts

Max-pooling happens during mvouts



Low-Level: “Complex” vs “Primitive” Instructions

”Primitive” instructions

- `Mvin`
- `Preload`
- `Compute`
- `Mvout`

Usually operate on DIMxDIM matrices

Explicitly manage scratchpad

“Complex” instructions hardcode these loops

- `loop_matmul`
- `loop_conv`

Unrolled by hardware FSMs

Main memory to main memory

- Scratchpad is not explicitly managed by programmer

Enables performance optimizations

- Dynamic scheduling
  - Good overlap between ld/ex/st instructions
- Double-buffering
- Lower instruction bandwidth requirements

Low-Level: “Primitive”

```
// Main memory tiles
for (int i0 = 0; i0 < N; i0 += TILE_I) {
  for (int j0 = 0; j0 < N; j0 += TILE_J) {
    for (int k0 = 0; k0 < N; k0 += TILE_K) {

      // Scratchpad tiles
      for (int i1 = 0; i1 < N; i1 += DIM) {
        for (int j1 = 0; j1 < N; j1 += DIM) {
          for (int k1 = 0; k1 < N; k1 += DIM) {
            int i = i0 * tile_I + i1;
            int j = j0 * tile_J + j1;
            int k = k0 * tile_K + k1;

            A_addr = ...; B_addr = ...; C_addr = ...;

            if (j1 == 0) gemmini_mvin(&A[i0*tile_I + i1][k*TILE_K + k1], A_addr);
            if (i1 == 0) gemmini_mvin(&B[k0*tile_K + k1][j*TILE_J + j1], B_addr);

            gemmini_preload(B_addr, C_addr);
            gemmini_compute(A_addr);
}}}}}}
```

Low-Level: “Primitive”

```
// Main memory tiles
for (int i0 = 0; i0 < N; i0 += TILE_I) {
  for (int j0 = 0; j0 < N; j0 += TILE_J) {
    for (int k0 = 0; k0 < N; k0 += TILE_K) {

      // Scratchpad tiles
      for (int i1 = 0; i1 < N; i1 += DIM) {
        for (int j1 = 0; j1 < N; j1 += DIM) {
          for (int k1 = 0; k1 < N; k1 += DIM) {
            int i = i0 * tile_I + i1;
            int j = j0 * tile_J + j1;
            int k = k0 * tile_K + k1;

            A_addr = ...; B_addr = ...; C_addr = ...;

            if (j1 == 0) gemmini_mvin(&A[i0*tile_I + i1][k*TILE_K + k1], A_addr);
            if (i1 == 0) gemmini_mvin(&B[k0*tile_K + k1][j*TILE_J + j1], B_addr);

            gemmini_preload(B_addr, C_addr);
            gemmini_compute(A_addr);
}}}}}}
```

Low-Level: “Complex”

Coarse-grained command for a “tile”

- Tile: operand/result fits in internal memory
- Contains load, execute, store

```
// Main memory tiles
for (int i0 = 0; i0 < N; i0 += TILE_I) {
  for (int j0 = 0; j0 < N; j0 += TILE_J) {
    for (int k0 = 0; k0 < N; k0 += TILE_K) {

    // Scratchpad tiles
    gemmini_loop_matmul(...);

}}}
```