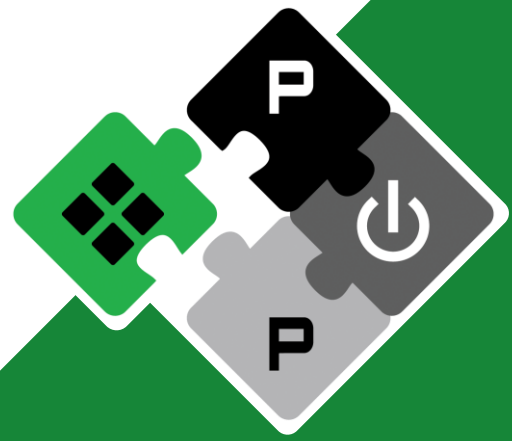
ETH zürich ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

CachePool: Many-core cluster of customizable, lightweight scalar-vector PEs for irregular L2 data-plane workloads

Integrated Systems Laboratory (ETH Zürich)

Zexin Fu, Diyou Shen zexifu, dishen@iis.ee.ethz.ch

Alessandro Vanelli-Coralli avanelli@iis.ee.ethz.ch
Luca Benini lbenini@iis.ee.ethz.ch



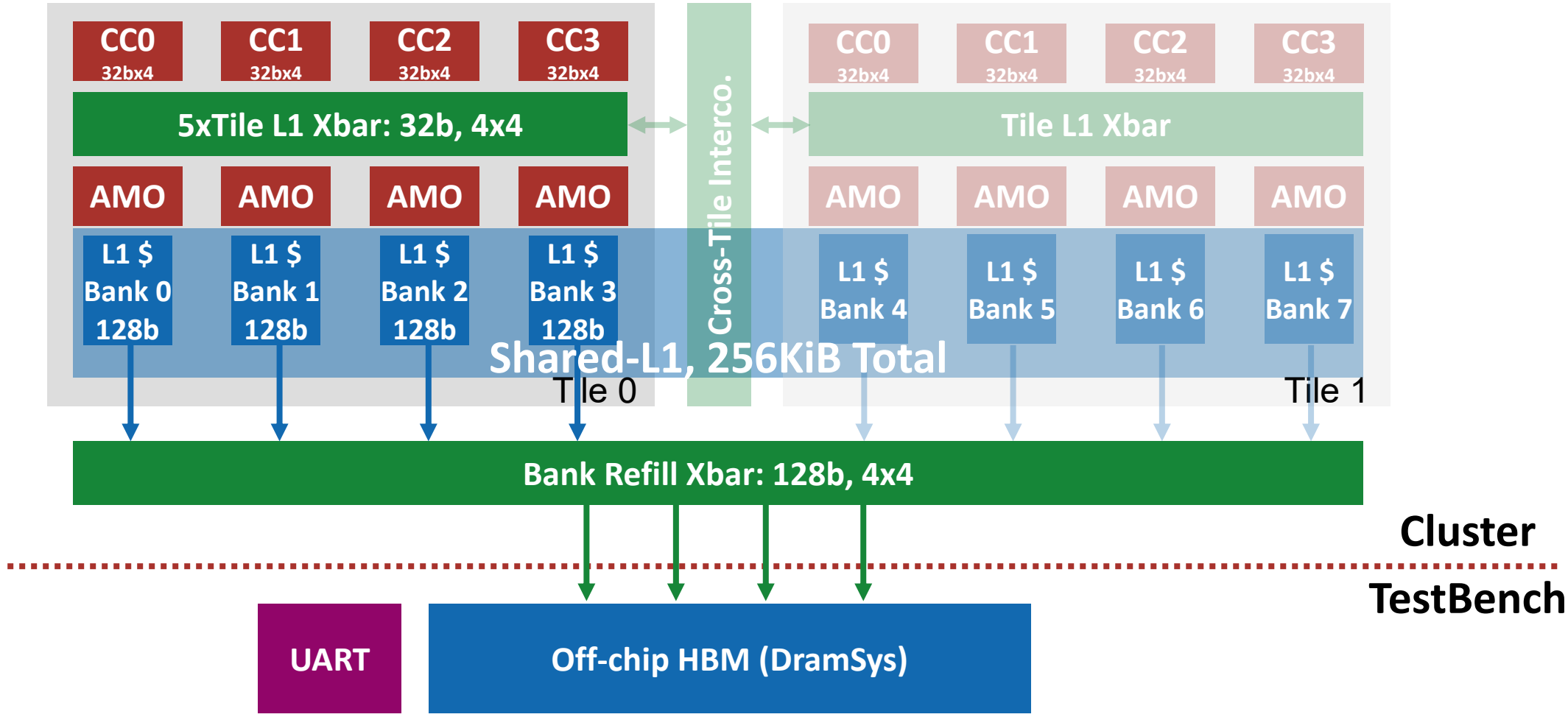
@pulp_platform

pulp-platform.org

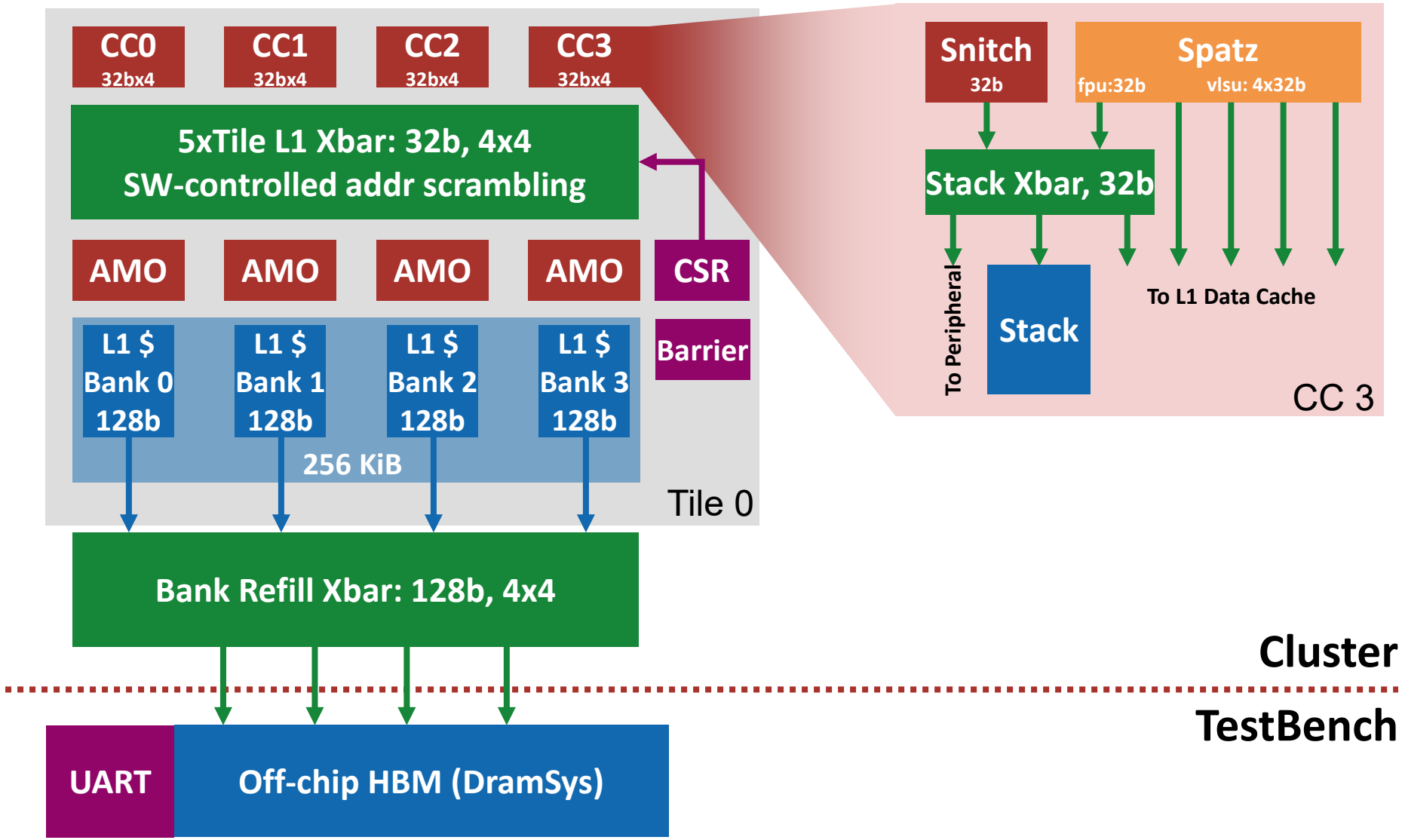
youtube.com/pulp_platform

PULP Platform
Open Source Hardware, the way it should be!

Hardware Development



Hardware Development



Hardware Development -- Stack

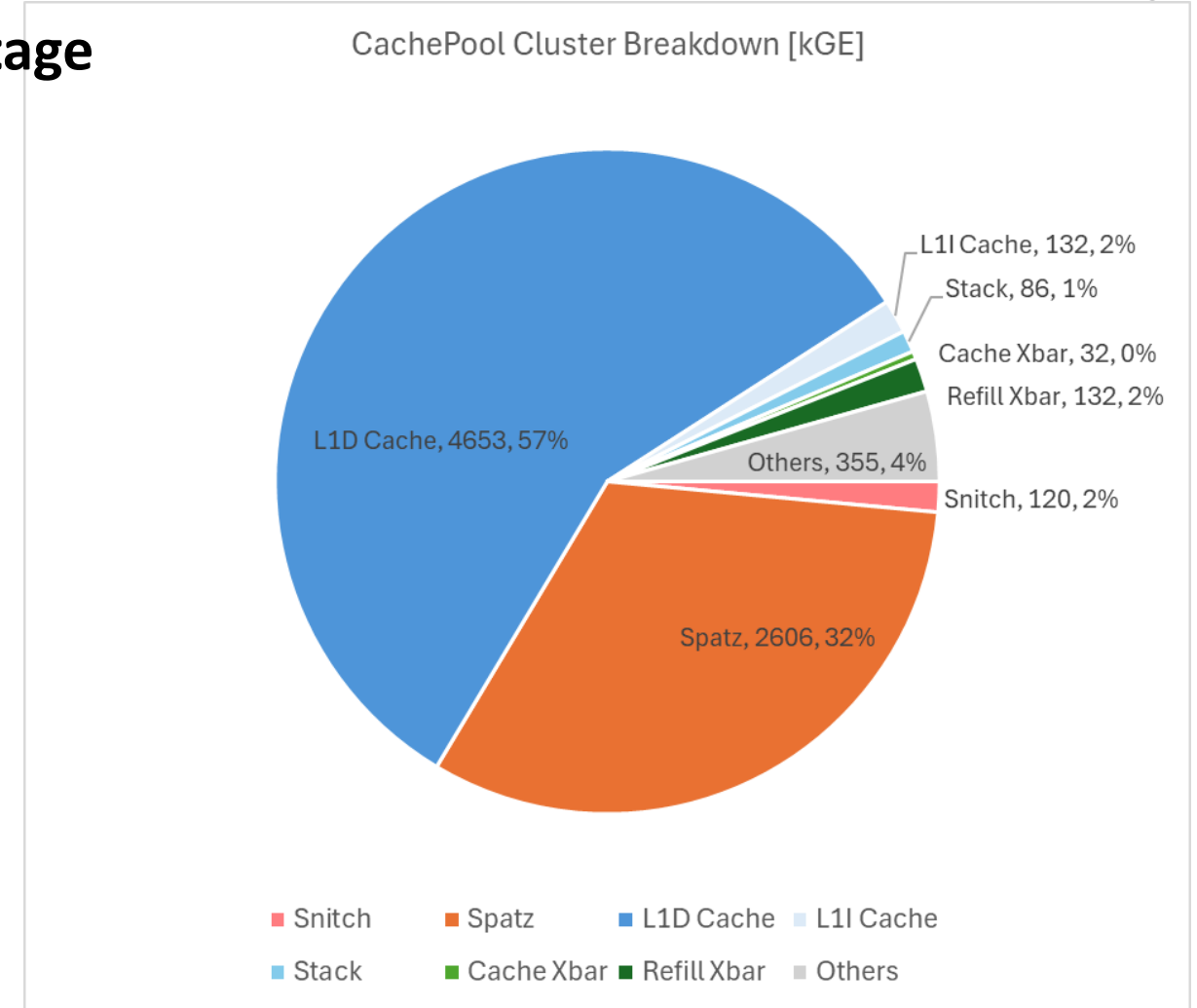
- **Previously Stack region is allocated in the shared SPM of the cluster**
  - In shared memory, but are private to each core
- **Currently for quick exploration, we use a physical SRAM for stack**
  - Private to each core
  - Benefits: simplify interconnects, faster access
  - Stack overflow?
    - Route to Cache when overflow (not implemented yet)
  - Current size: 512x32 for each core
    - Will reduce its size later

Hardware Development

- **Performance Bottleneck – AXI Interconnection**
  - Cache Refill Xbar uses AXI protocol for compatibility
  - AXI requires in-order behavior under same ID
    - In our modules: xbar, datawidth converter, it handles request in same ID one-by-one
    - This causes the system not able to hide any DRAM latency
  - AXI module size linearly scales with ID width (maxtrans)
    - Basically one xbar per transaction channel
    - Not scalable
  - We have fixed the upsizer module for multiple outstandings, but Xbar is complex to fix
    - Demux inside xbar needs ROB if require outstanding support
  - We are now working on changing the internal interconnection to TCDM protocol for better performance and scalability

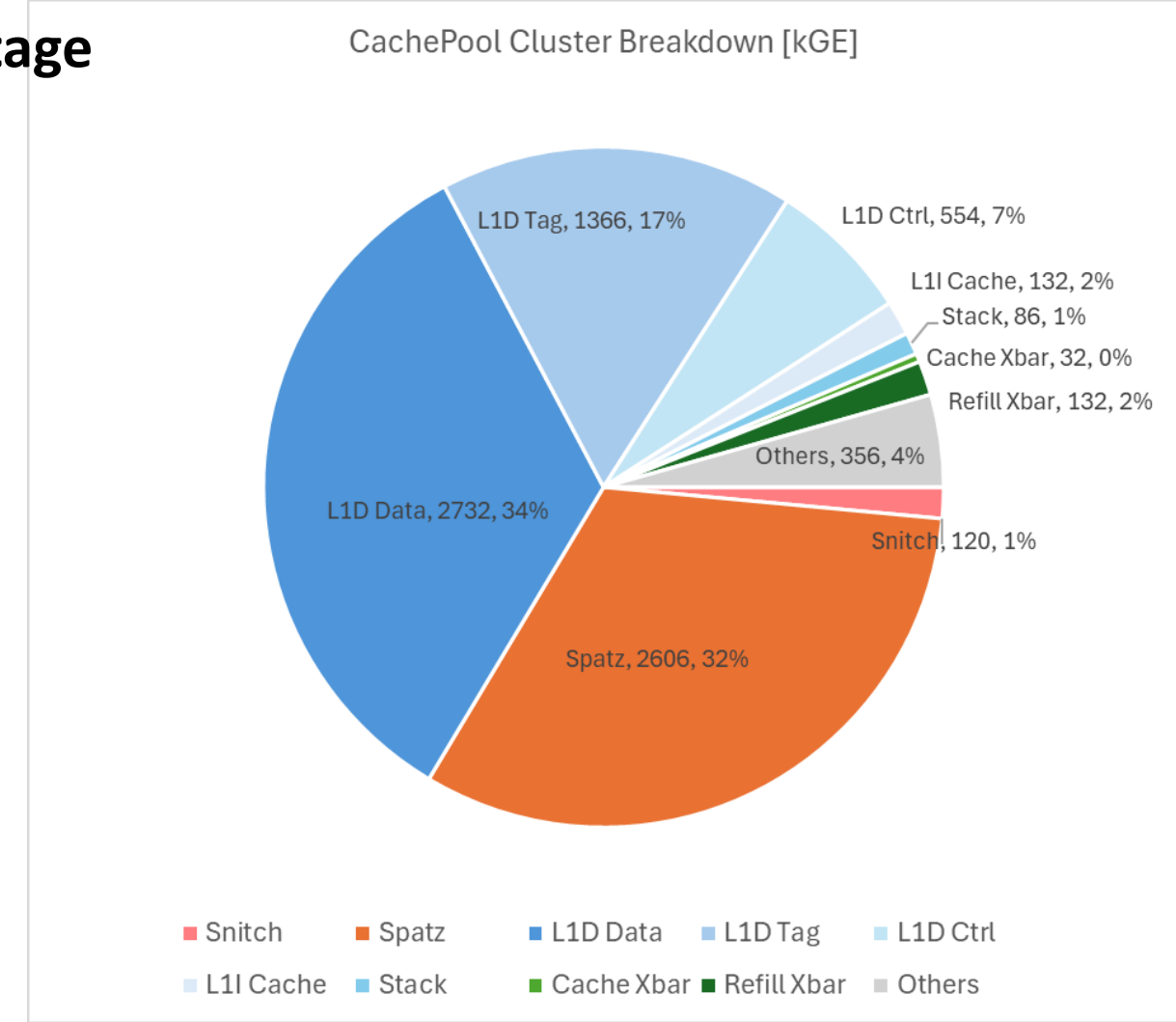
Initial PPA Results

- **Finished initial run up to placement stage**
  - 256 KiB L1
  - 12nm FinFET Technology
  - 931 MHz @ TT Corner
    - **Same as SPM version of Spatz cluster**
  - Timing closed
  - Area:
    - Cluster: 8 MGE
    - L1 Ctrl: 4x138K
    - L1 SRAM: 4 MGE
    - Spatz: 4x650K
    - Snitch: 4x30K



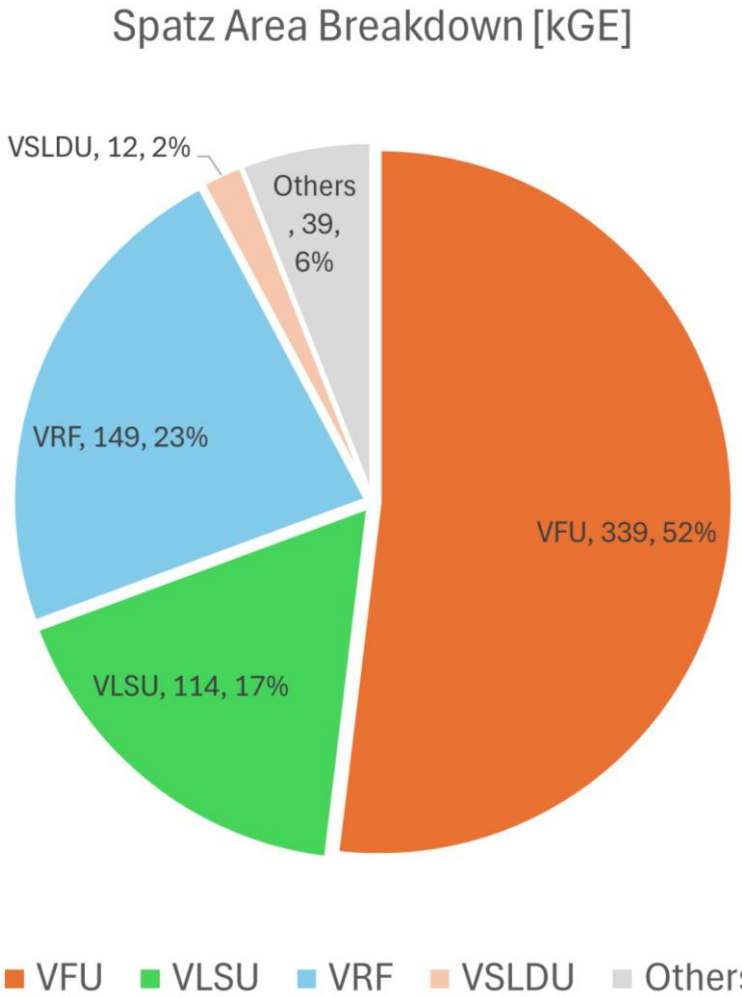
Initial PPA Results

- **Finished initial run up to placement stage**
  - 256 KiB L1
  - 12nm FinFET Technology
  - 931 MHz @ TT Corner
    - **Same as SPM version of Spatz cluster**
  - Timing closed
  - Area:
    - Cluster: 8 MGE
    - L1 Ctrl: 4x138K
    - L1 SRAM: 4 MGE
    - Spatz: 4x650K
    - Snitch: 4x30K



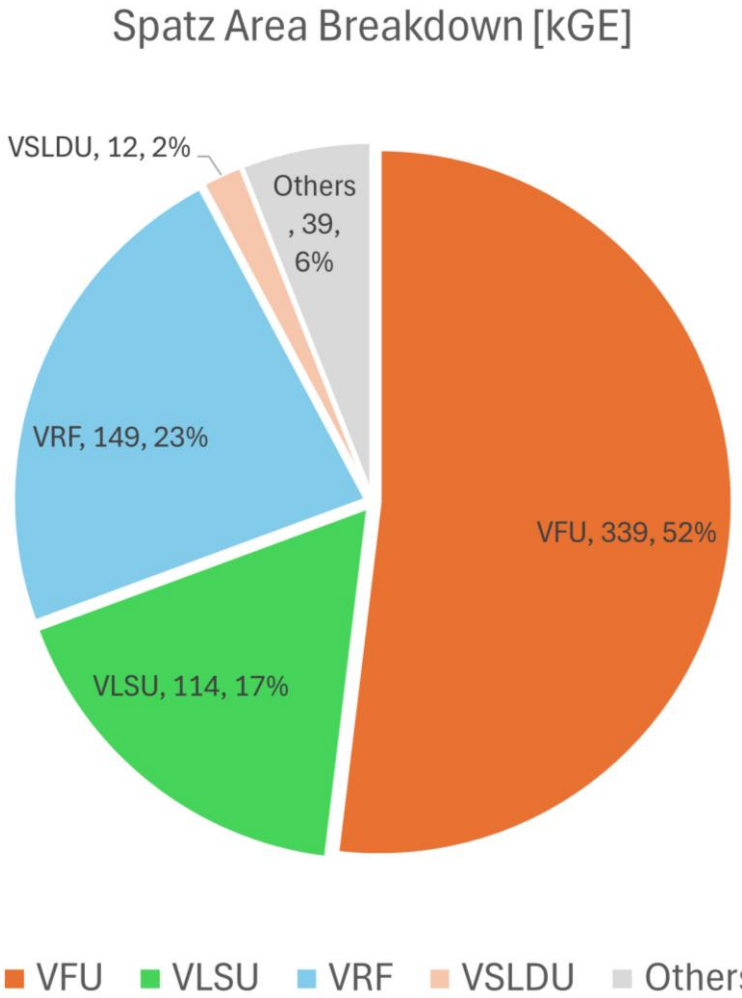
Initial PPA Results

- **Finished initial run up to placement stage (256 KiB L1)**
  - 12nm FinFET Technology
  - 931 MHz @ TT Corner
  - Timing closed
  - Area:
    - Spatz: 650 kGE
    - VFU: 339 kGE
    - VRF: 149 kGE
    - VLSU: 114 kGE



Initial PPA Results

- **Finished initial run up to placement stage (256 KiB L1)**
  - 12nm FinFET Technology
  - 931 MHz @ TT Corner
  - Timing closed
  - Spatz Breakdown
    - VLSU is large here due to ROB depth (64)
      - Temporary measure for latency tolerance
      - Not final solution
      - HW-SW hybrid prefetcher—DMA like
    - We will reduce it after fixing the interco. issues
    - We currently also support FP8, FP16, FP32



Hardware Development

- **Complete**
  - SPM interconnection, stack modification
  - First PPA analysis
  - Add simulation time performance monitors in Spatz
  - Fix a performance bug in AXI DW Upsizer
- **In Progress**
  - Change Cache-Refill interconnection from AXI to TCDM for better performance
- **TODO**
  - Partition support

```
**********************************************************************
***            Spatz Controller Utilization Report                 ***
**********************************************************************
   VLSU:
   VLSU Active (not accurate):                                  4224
   VLSU Num Instructions:                                       4224
   VLSU Stall Cycles:                                              0
   VLSU WR Stalls:                                             34069

   VFU:
   VFU Active (not accurate):                                  20634
   VFU Num Instructions:                                       20634
   VFU Stall Cycles:                                               0
   VFU WR Stalls:                                               7690

   VSLDU:
   VSLDU Active (not accurate):                                    0
   VSLDU Num Instructions:                                         0
   VSLDU Stall Cycles:                                             0
   VSLDU WR Stalls:                                                0

**********************************************************************
***            Spatz VLSU Utilization Report                       ***
**********************************************************************
   Number of VRF Valid Cycles:                                 99605
   Number of VRF Transaction Counts:                           65536
   VRF W Utilization:                                          65.00
   VRF W AVG Cycles:                                            1.00

   Number of Mem Valid Cycles:                                 91719
   Number of Mem Transaction Counts:                           67584
   Mem Utilization:                                            73.69
   Mem AVG Req Accept Cycles:                                   1.36

   ROB AVG Utilization:                                        14.00
   ROB Peak Utilization:                                       50.00

   Total Insn Count:                                         4224.00
   AVG Insn Cycles:                                            41.00
```

Performance Analysis of Vector Kernel

- **GEMM Kernel**
  - Inner loop

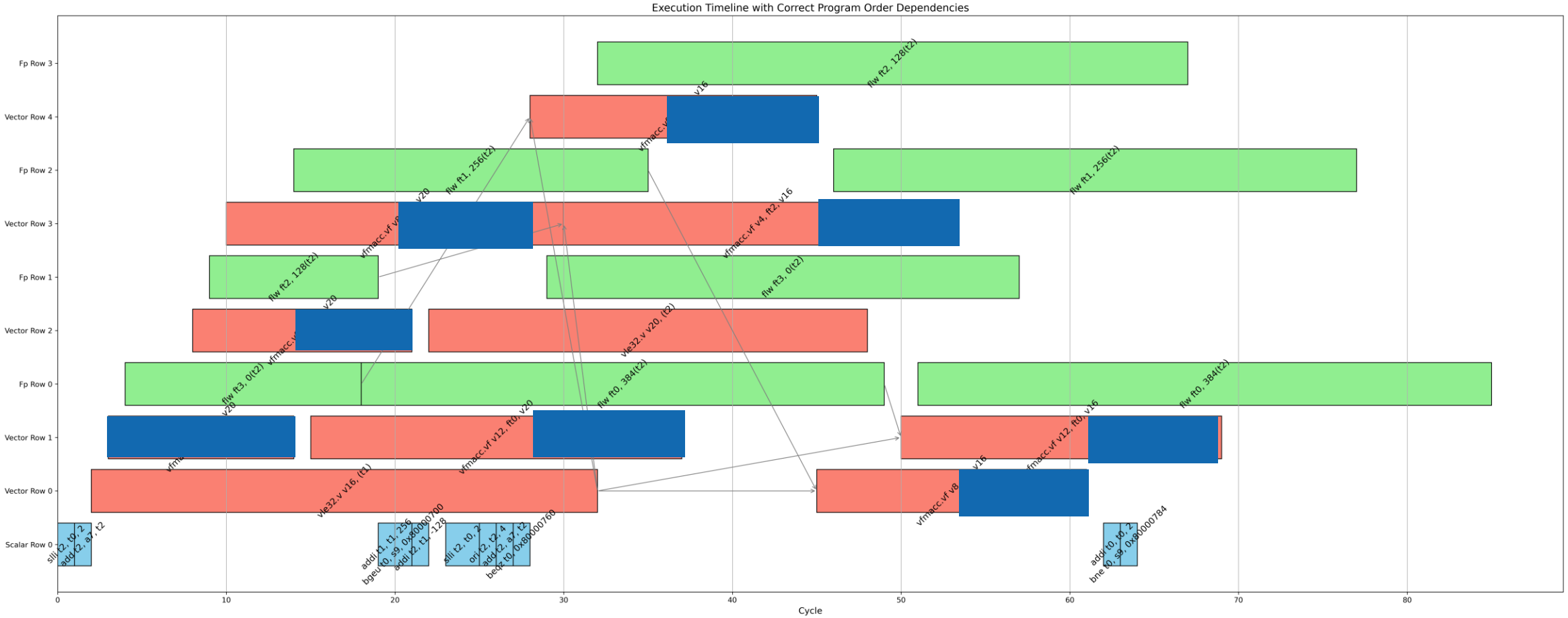
```
while (n < N) {
  asm volatile("vle32.v v20, (%0);" ::"r"(b__));
  b__ += P;

  a__ = a_ + ++n;

  if (n == 1) {
    asm volatile("vfmul.vf v0, v16, %0" ::"f"(t0));
    t0 = *a__;
    a__ += N;
    asm volatile("vfmul.vf v4, v16, %0" ::"f"(t1));
    t1 = *a__;
    a__ += N;
    asm volatile("vfmul.vf v8, v16, %0" ::"f"(t2));
    t2 = *a__;
    a__ += N;
    asm volatile("vfmul.vf v12, v16, %0" ::"f"(t3));
    t3 = *a__;
  } else {
    asm volatile("vfmacc.vf v0, %0, v16" ::"f"(t0));
    t0 = *a__;
    a__ += N;
    asm volatile("vfmacc.vf v4, %0, v16" ::"f"(t1));
    t1 = *a__;
    a__ += N;
    asm volatile("vfmacc.vf v8, %0, v16" ::"f"(t2));
    t2 = *a__;
    a__ += N;
    asm volatile("vfmacc.vf v12, %0, v16" ::"f"(t3));
    t3 = *a__;
  }
```

```
  a__ = a_ + ++n;

  if (n == N)
    break;

  asm volatile("vle32.v v16, (%0);" ::"r"(b__));
  b__ += P;

  asm volatile("vfmacc.vf v0, %0, v20" ::"f"(t0));
  t0 = *a__;
  a__ += N;
  asm volatile("vfmacc.vf v4, %0, v20" ::"f"(t1));
  t1 = *a__;
  a__ += N;
  asm volatile("vfmacc.vf v8, %0, v20" ::"f"(t2));
  t2 = *a__;
  a__ += N;
  asm volatile("vfmacc.vf v12, %0, v20" ::"f"(t3));
  t3 = *a__;
}
```

inst	type
slli t2, t0, 2	scalar
add t2, a7, t2	scalar
vle32.v v16, (t1)	vector
vfmacc.vf v0, ft3, v20	vector
flw ft3, 0(t2)	fp
vfmacc.vf v4, ft2, v20	vector
flw ft2, 128(t2)	fp
vfmacc.vf v8, ft1, v20	vector
flw ft1, 256(t2)	fp
vfmacc.vf v12, ft0, v20	vector
flw ft0, 384(t2)	fp
addi t1, t1, 256	scalar
bgeu t0, s9, 0x80000700	scalar
addi t2, t1, -128	scalar
vle32.v v20, (t2)	vector
slli t2, t0, 2	scalar
ori t2, t2, 4	scalar
add t2, a7, t2	scalar
beqz t0, 0x80000760	scalar
vfmacc.vf v0, ft3, v16	vector
flw ft3, 0(t2)	fp
vfmacc.vf v4, ft2, v16	vector
flw ft2, 128(t2)	fp
vfmacc.vf v8, ft1, v16	vector
flw ft1, 256(t2)	fp
vfmacc.vf v12, ft0, v16	vector
flw ft0, 384(t2)	fp
addi t0, t0, 2	scalar
bne t0, s9, 0x80000784	scalar

Performance Analysis of Vector Kernels

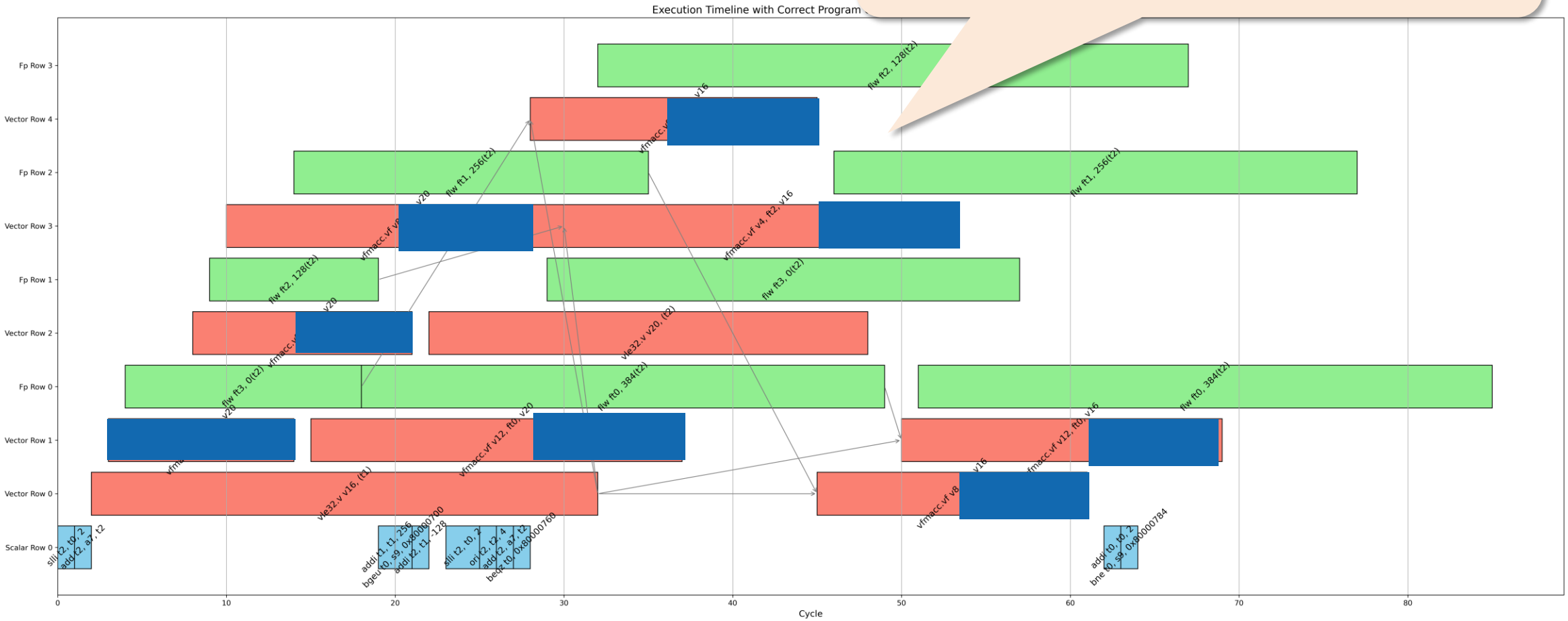
- **GEMM kernel inner loop execution timeline**
  - One Core Complex (Snitch + Spatz)



Performance Analysis of Vector Kernels

- **GEMM kernel inner loop execution timeline**
  - One Core Complex (Snitch + Spatz)

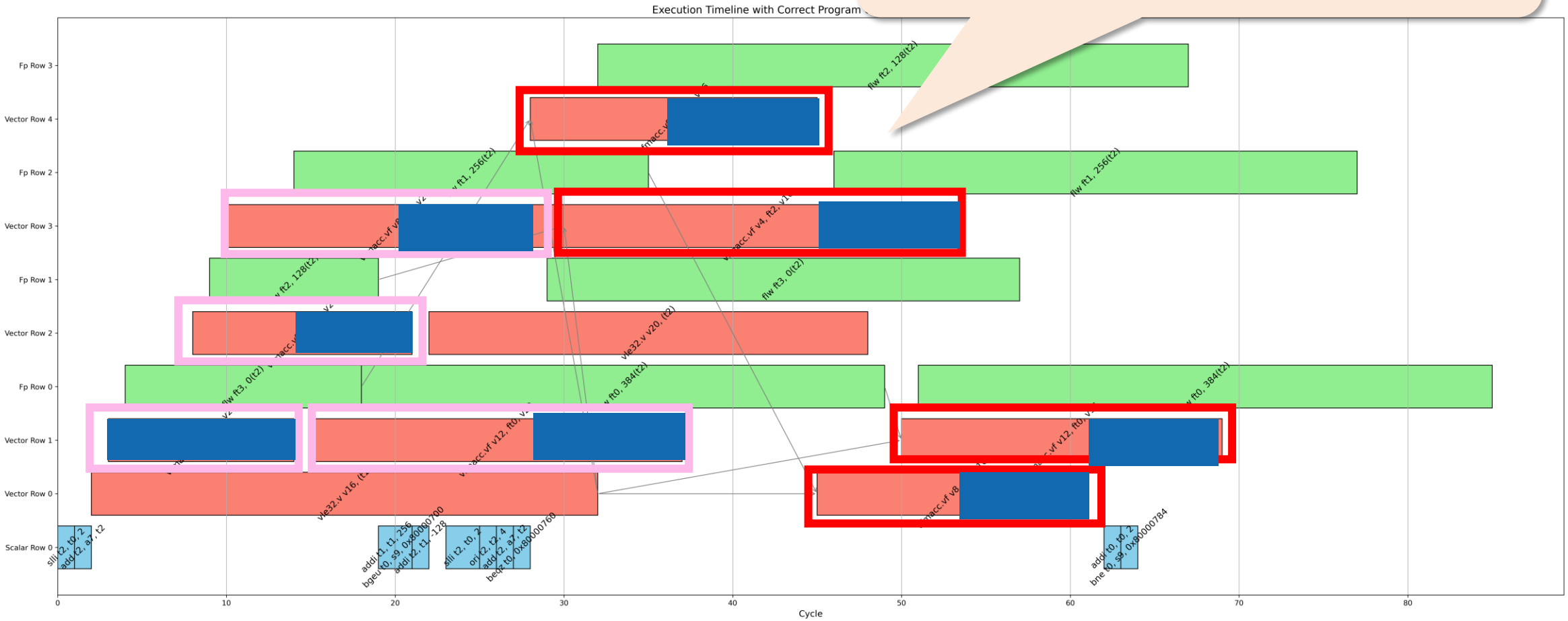
Low switch overhead between scalar and vector



Performance Analysis of Vector Kernels

- **GEMM kernel inner loop execution timeline**
  - One Core Complex (Snitch + Spatz)

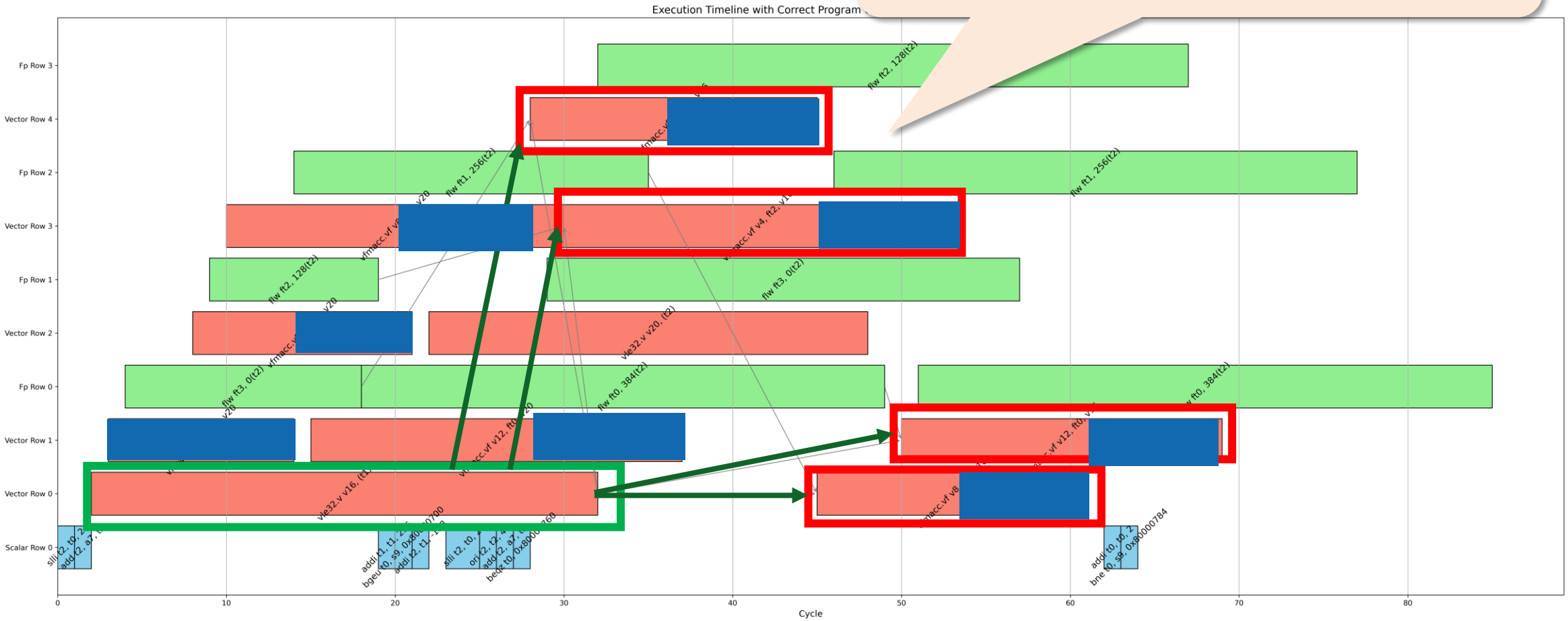
The vector computing units are highly utilized

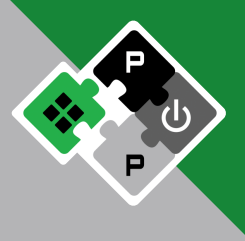


Performance Analysis of Vector Kernels

- **GEMM kernel inner loop execution timeline**
  - One Core Complex (Snitch + Spatz)

The vector load can provide data in time





Thank you!