Update on Ara

12/05/2021

Matteo Perotti

Matheus Cavalcante

Nils Wistoff

Professor Luca Benini

Integrated Systems Laboratory

ETH Zürich

Summary

- **Toolchain**
  - From GCC to LLVM
  - CI pipeline
- **Hardware**
  - Transition to RVV 0.10
  - Implementing reductions
- **Software**
  - New benchmarks

Toolchain - From GCC to LLVM

Motivation:
"*Current focus for V compiler work is on LLVM.*" [Jim Wilson]

- Scalar riscv-tests
- Vector riscv-tests
- Vector programs

ETH zürich

Toolchain - From GCC to LLVM

Motivation:
"*Current focus for V compiler work is on LLVM*." [Jim Wilson]

*

- Scalar riscv-tests

- Vector riscv-tests
- Vector programs

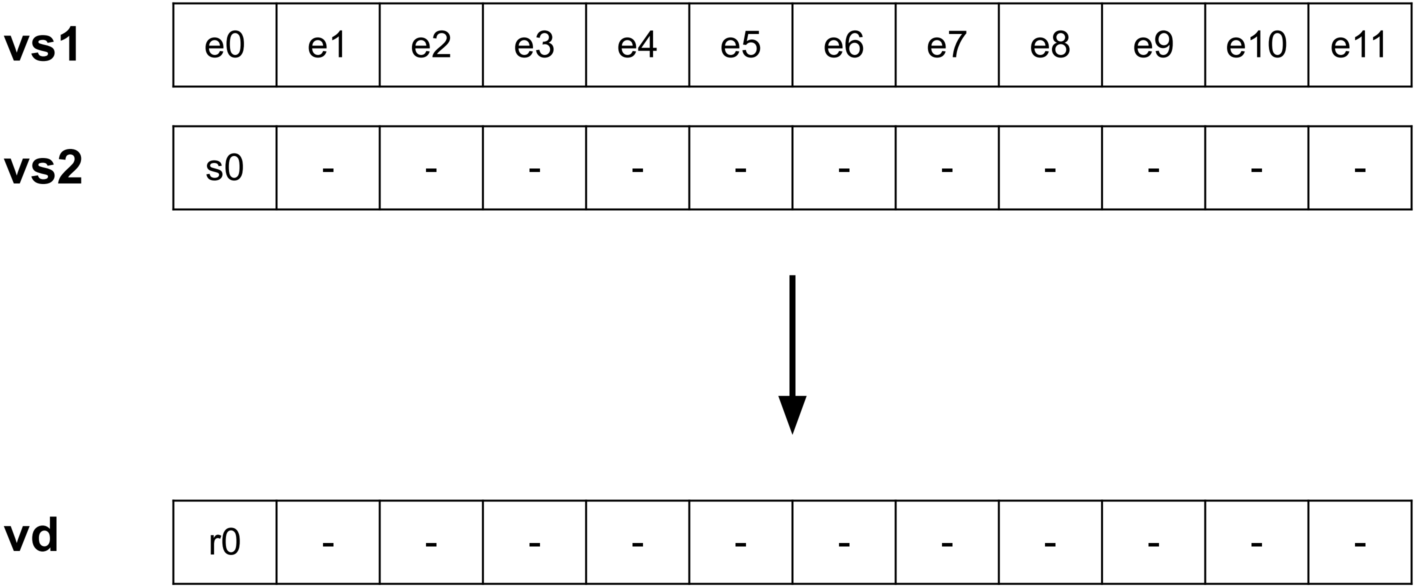
** Kept for compatibility reasons*

Toolchain - From GCC to LLVM

- RVV 0.10
- V-intrinsics support
- Implies:
  - SPIKE RVV 0.10
  - Ara updated to RVV 0.10*

* We adapted the already-present instructions

Hardware - Reductions



vs1	e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11
vs2	s0	-	-	-	-	-	-	-	-	-	-	-
vd	r0	-	-	-	-	-	-	-	-	-	-	-

Hardware - Reductions

Example: vredsum vd, vs1, vs2

vs1	e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11
vs2	s0	-	-	-	-	-	-	-	-	-	-	-

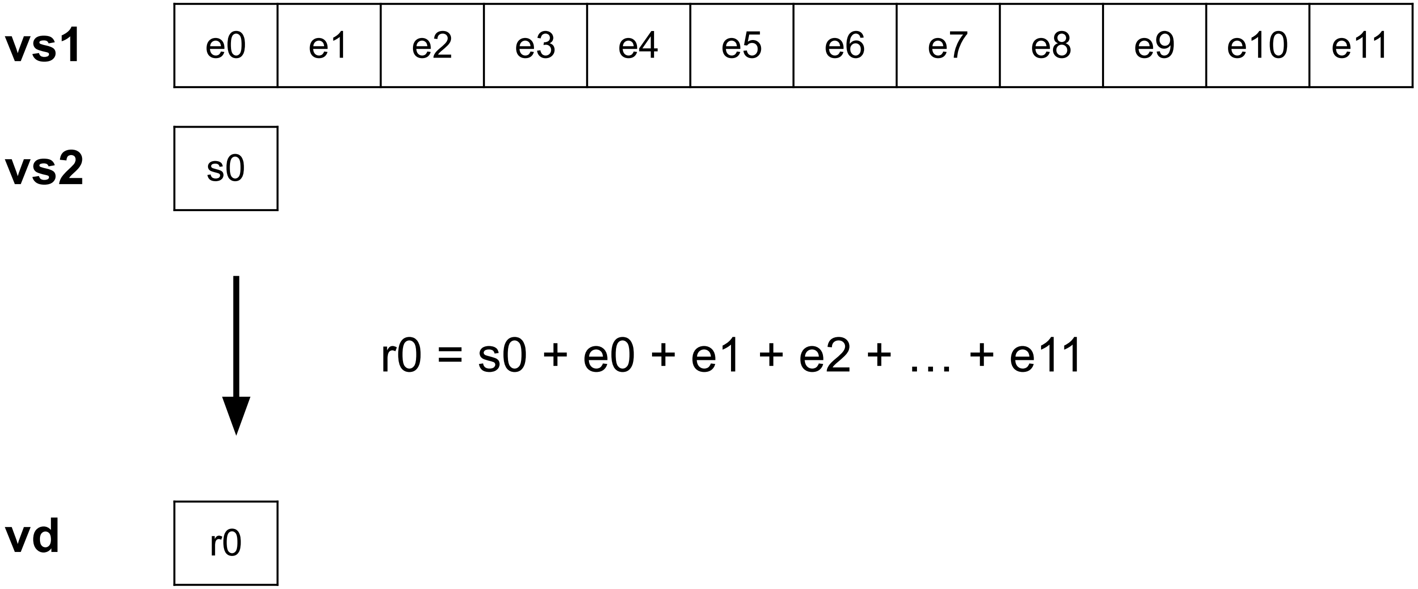
↓

r0 = s0 + e0 + e1 + e2 + … + e11

vd	r0	-	-	-	-	-	-	-	-	-	-	-

Hardware - Reductions

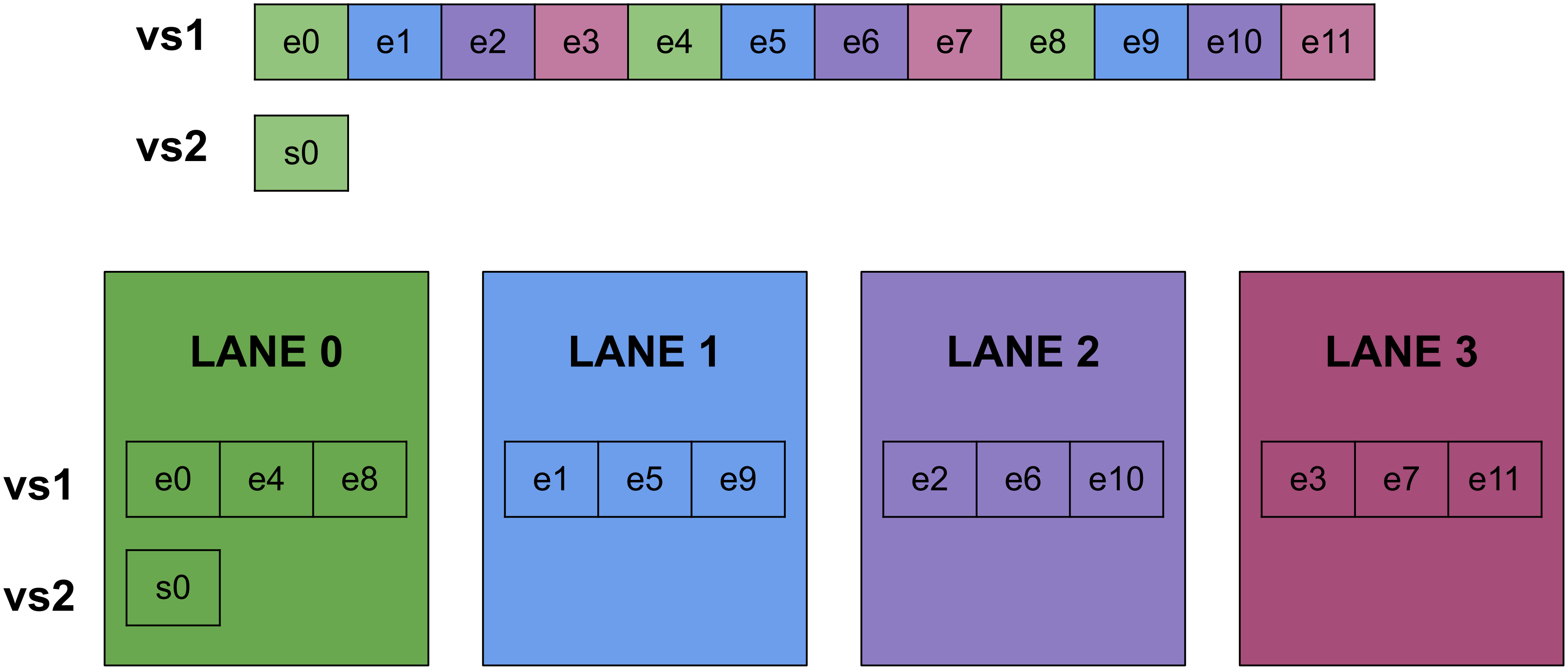
Example: vredsum vd, vs1, vs2



vs1	e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11
vs2	s0											
vd	r0											

$$r0 = s0 + e0 + e1 + e2 + \ldots + e11$$

Hardware - Reductions



vs1 e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11

vs2 s0

LANE 0

vs1 e0 e4 e8

vs2 s0

LANE 1

e1 e5 e9

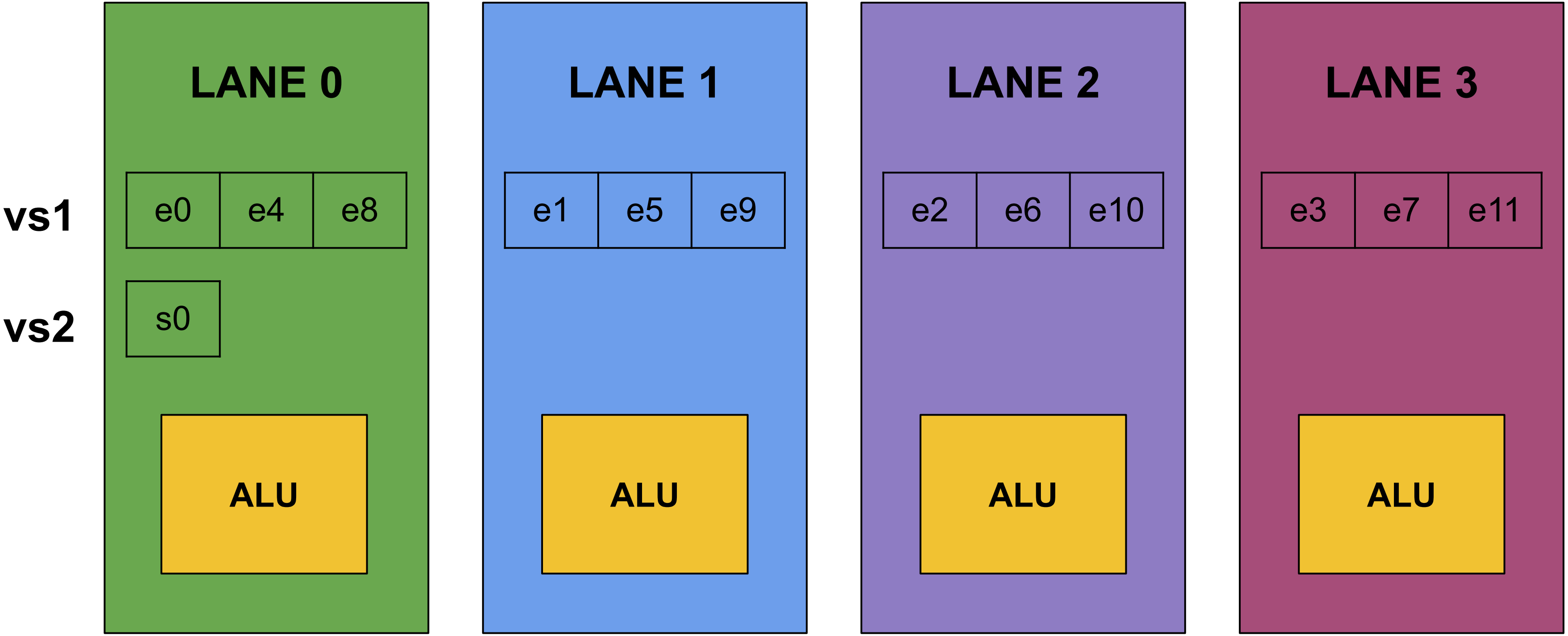
LANE 2

e2 e6 e10

LANE 3

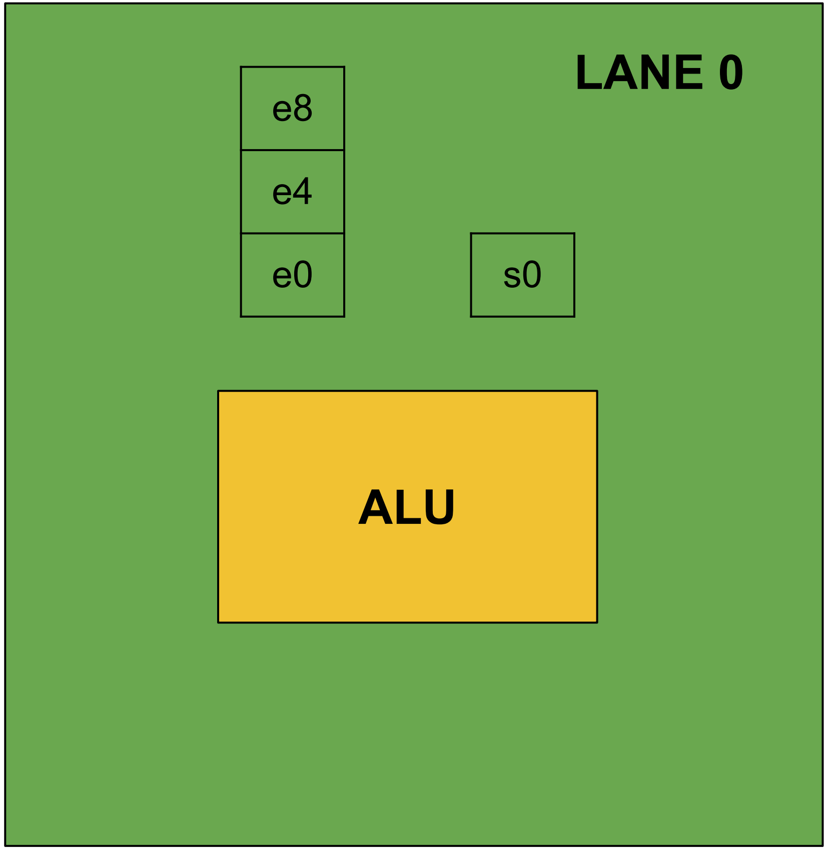
e3 e7 e11

Hardware - Reductions

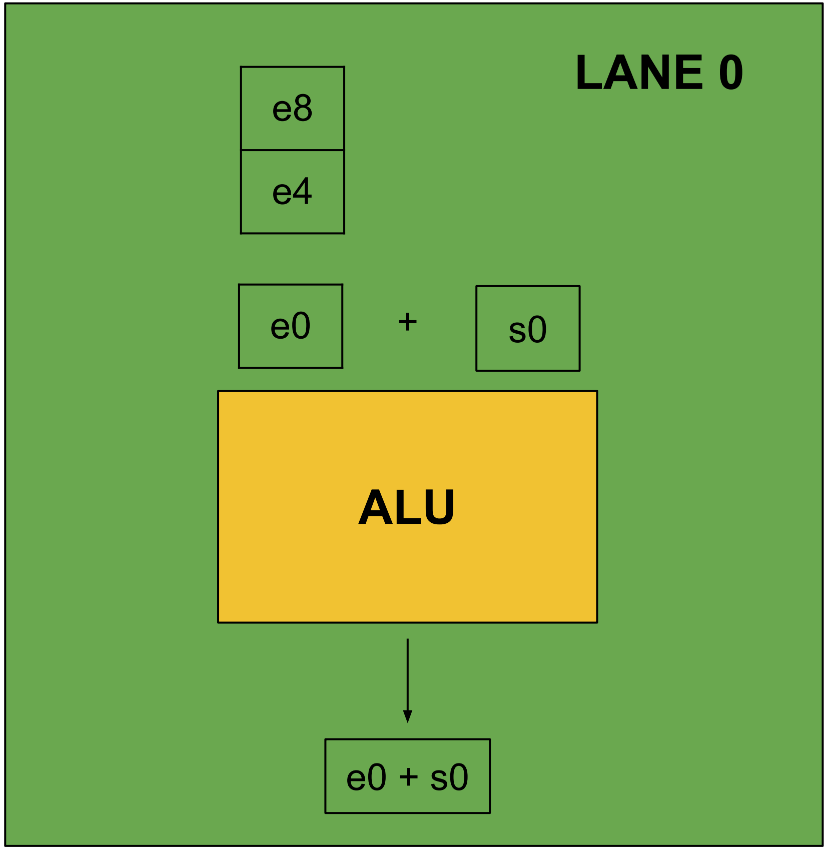


temp0 = s0 + e0 + e4 + e8 temp1 = e1 + e5 + e9 temp2 = e2 + e6 + e10 temp3 = e3 + e7 + e11

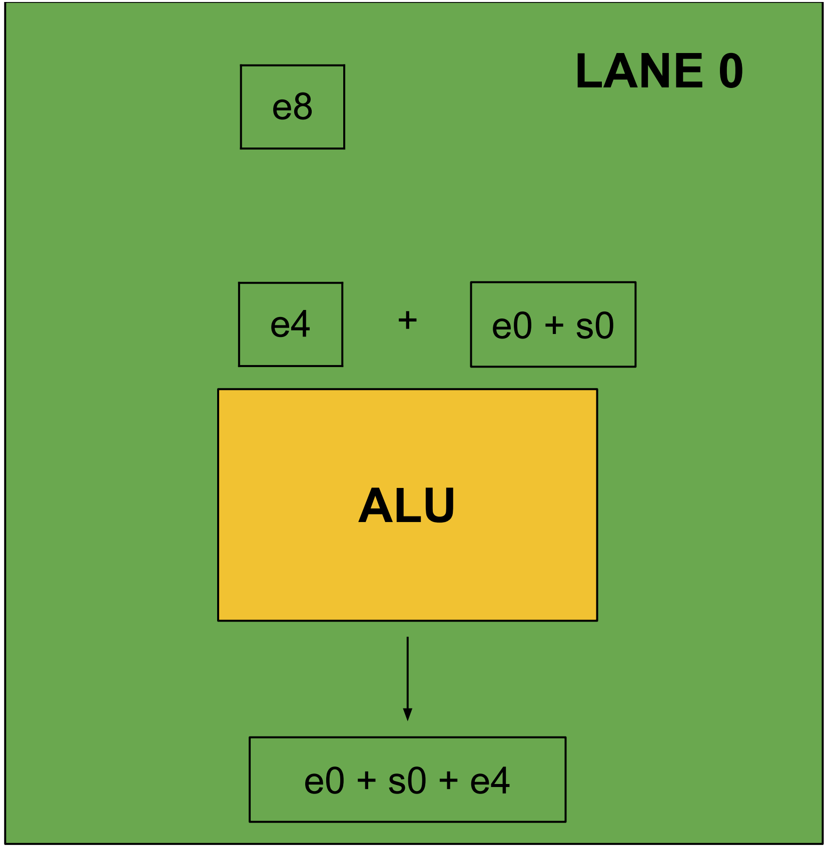
Intra-Lane reduction



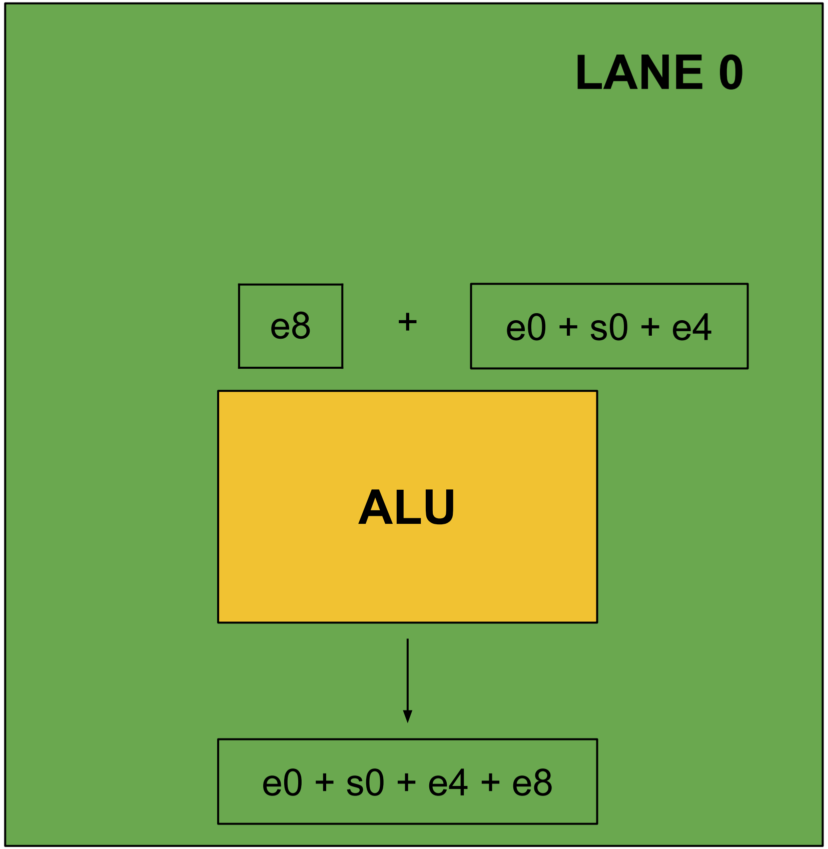
Intra-Lane reduction



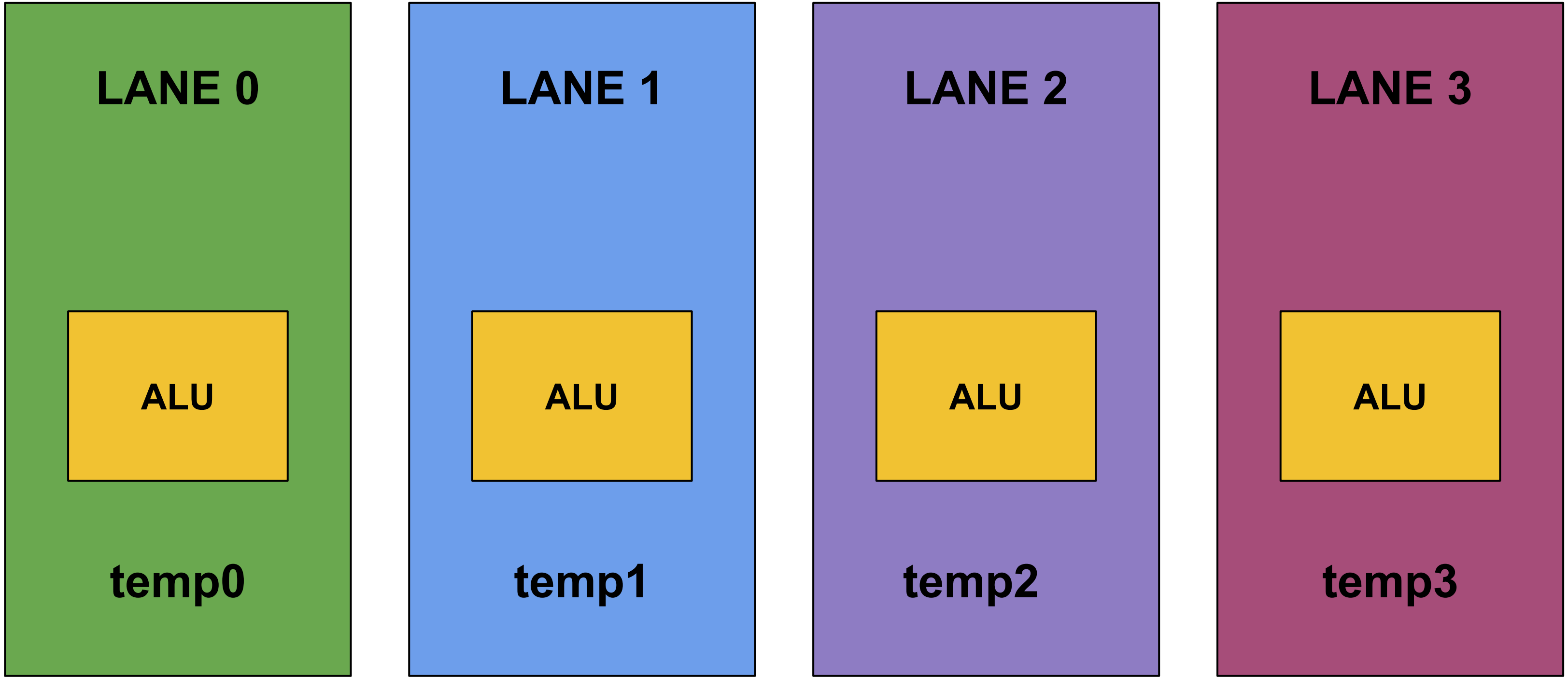
Intra-Lane reduction



Intra-Lane reduction

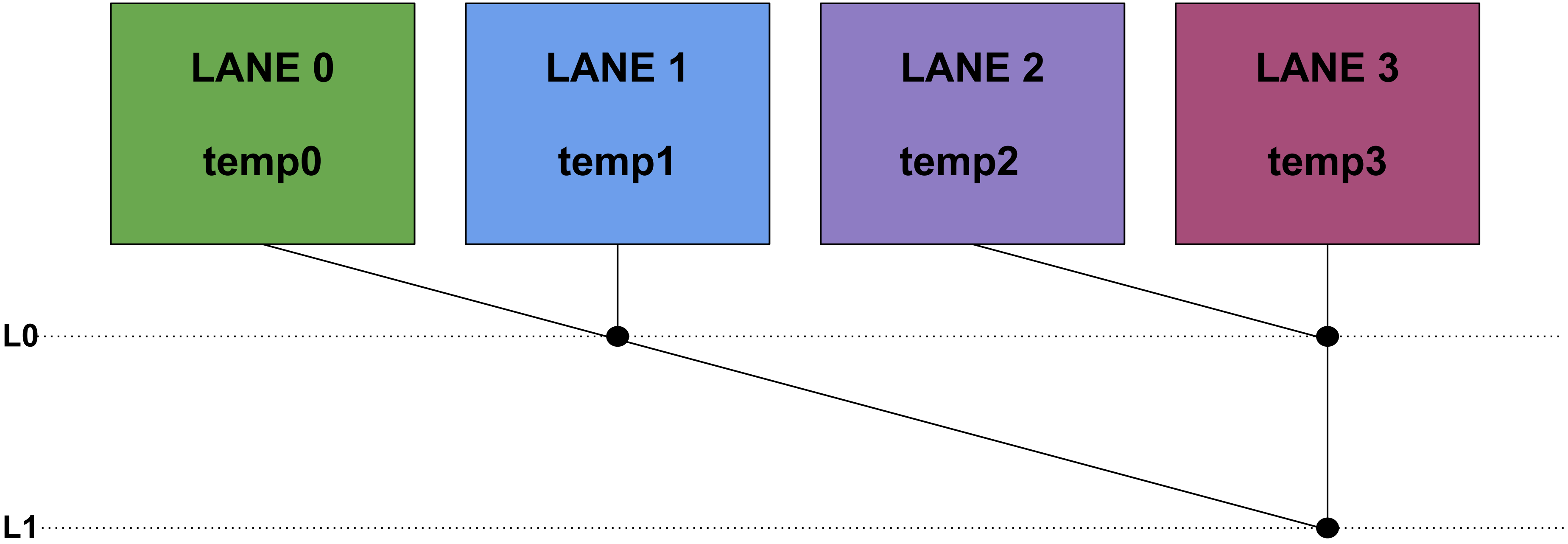


Inter-Lane reduction

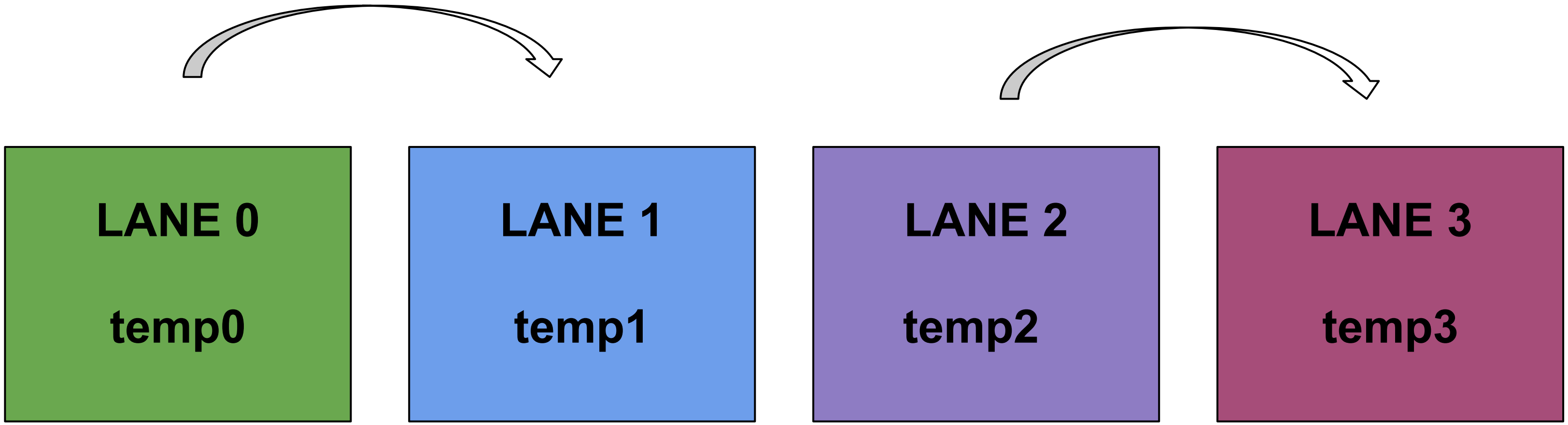


temp0 = s0 + e0 + e4 + e8 **temp1 = e1 + e5 + e9** **temp2 = e2 + e6 + e10** **temp3 = e3 + e7 + e11**

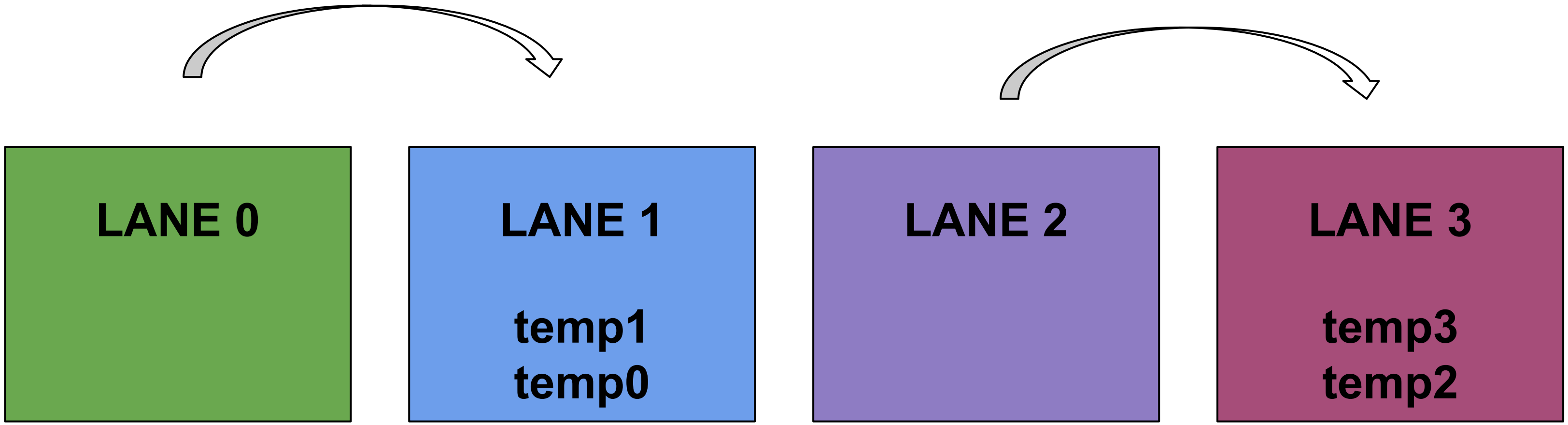
Inter-Lane reduction - Log tree



Inter-Lane reduction



Inter-Lane reduction



LANE 0

LANE 1

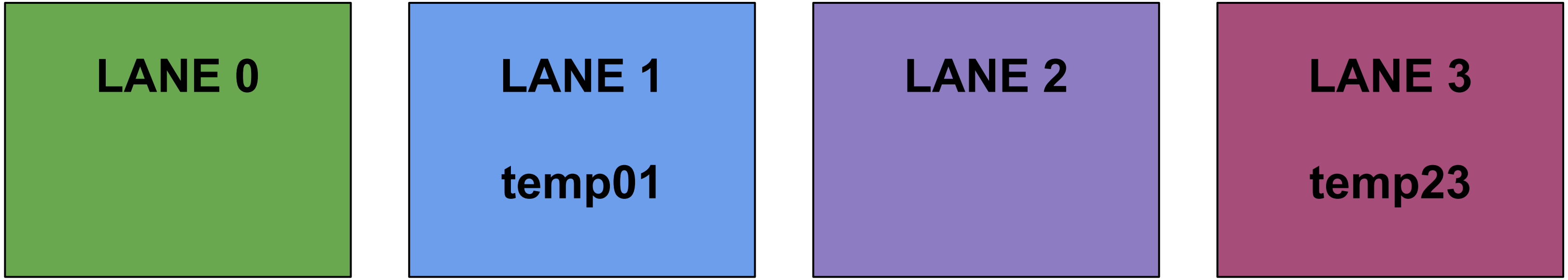
temp1
temp0

LANE 2

LANE 3

temp3
temp2

Inter-Lane reduction



LANE 0

LANE 1

temp01

LANE 2

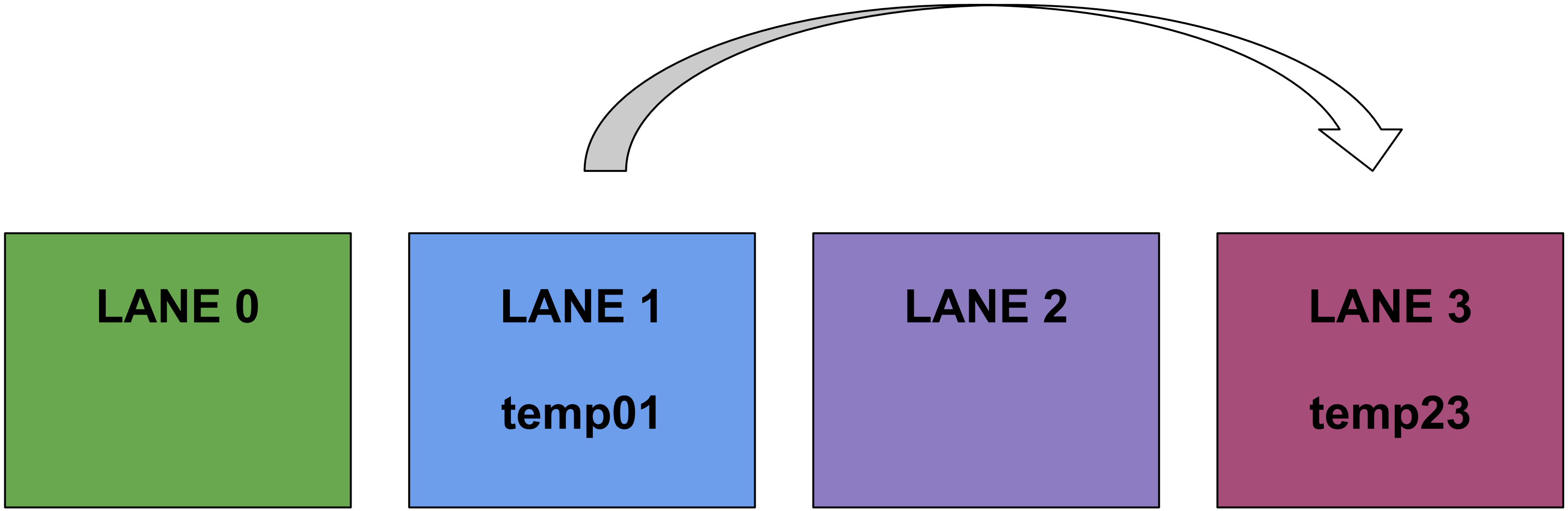
LANE 3

temp23

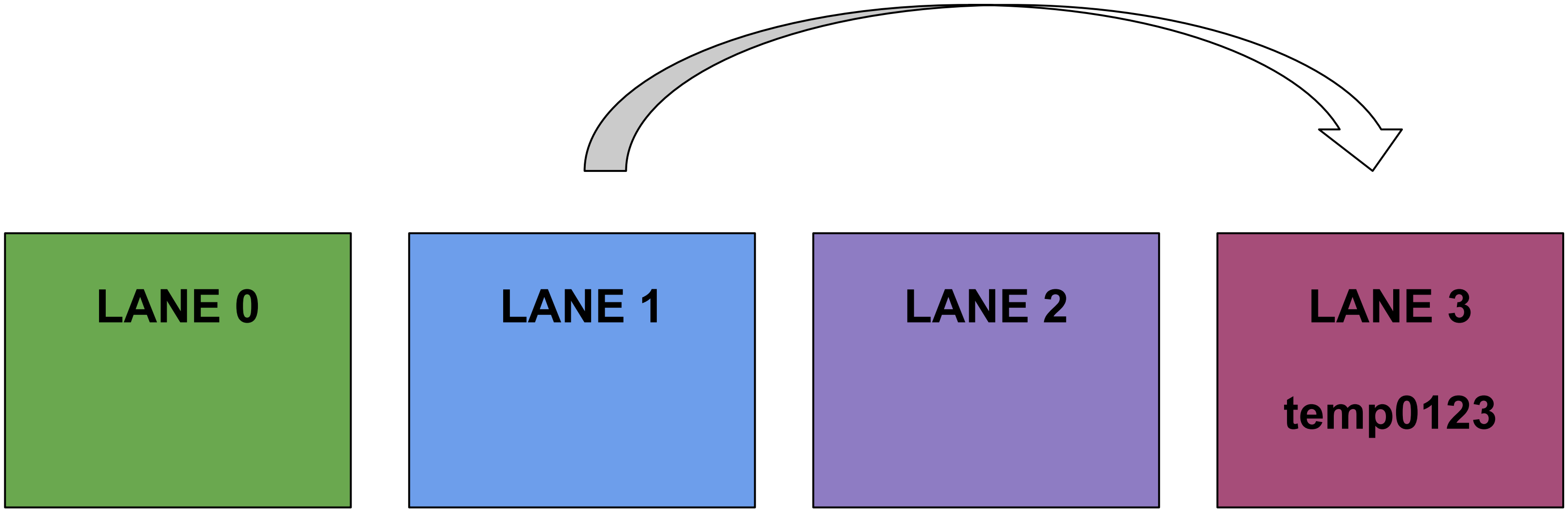
temp01 = temp0 + temp1

temp23 = temp2 + temp3

Inter-Lane reduction

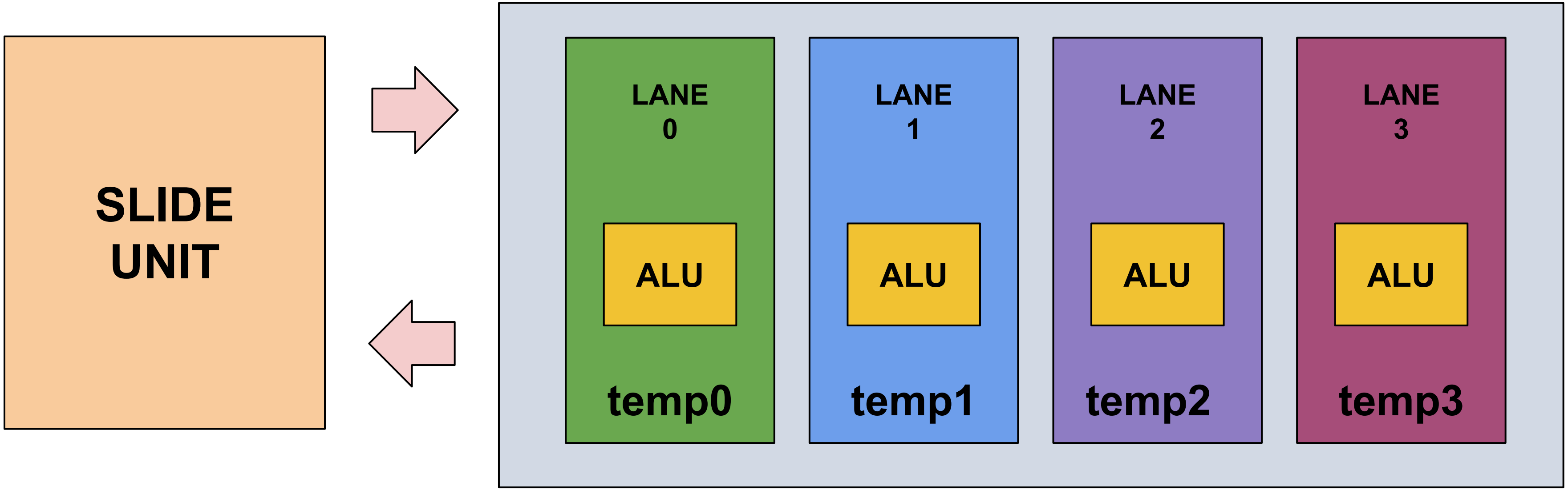


Inter-Lane reduction

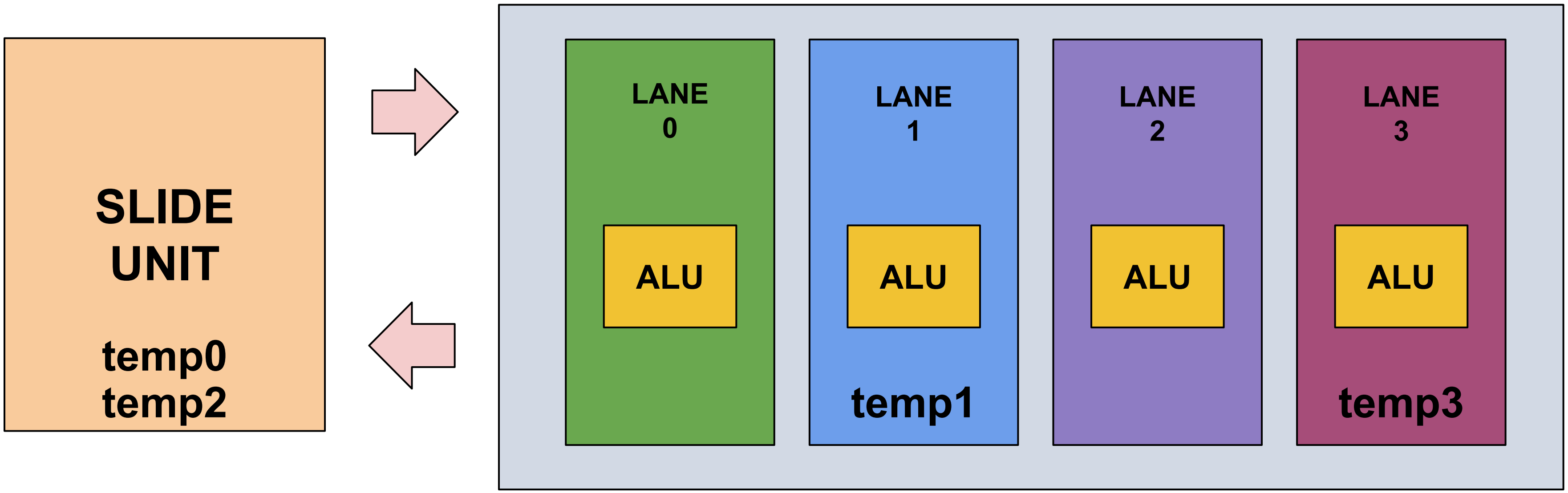


temp0123 = temp01 + temp23

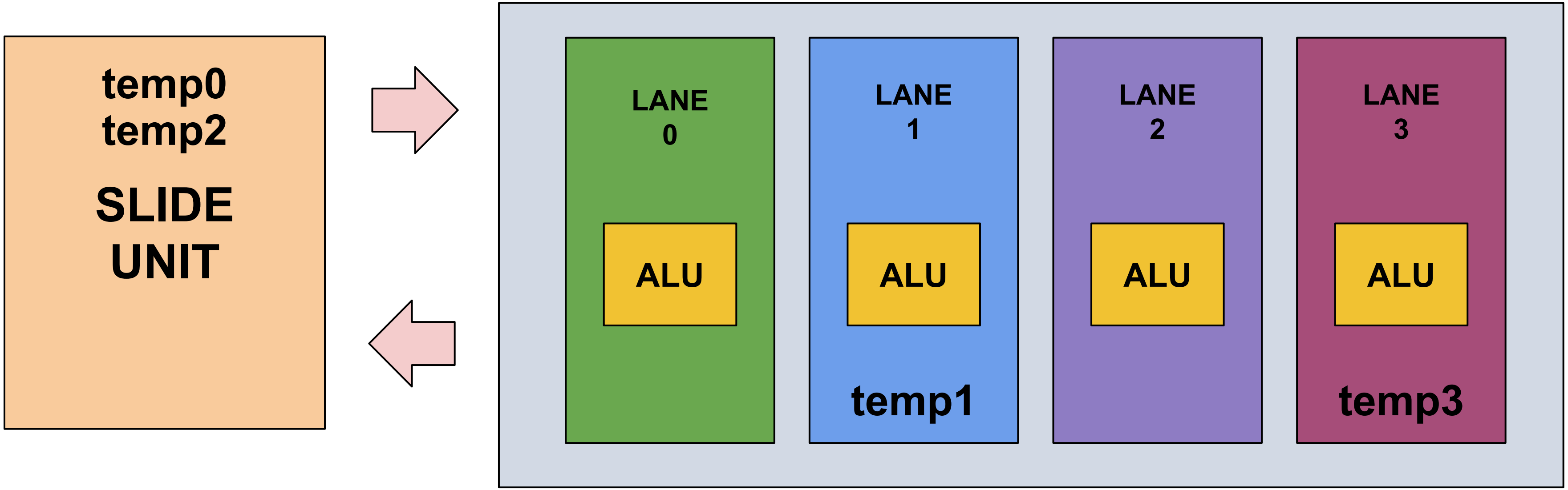
Inter-Lane reduction



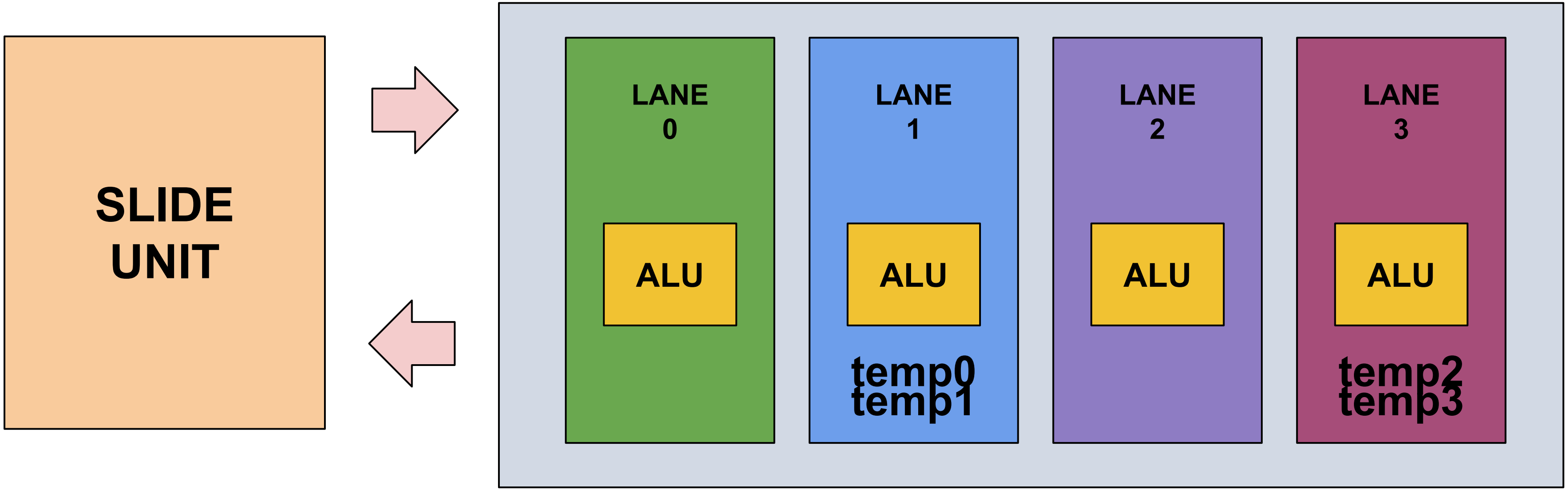
Inter-Lane reduction



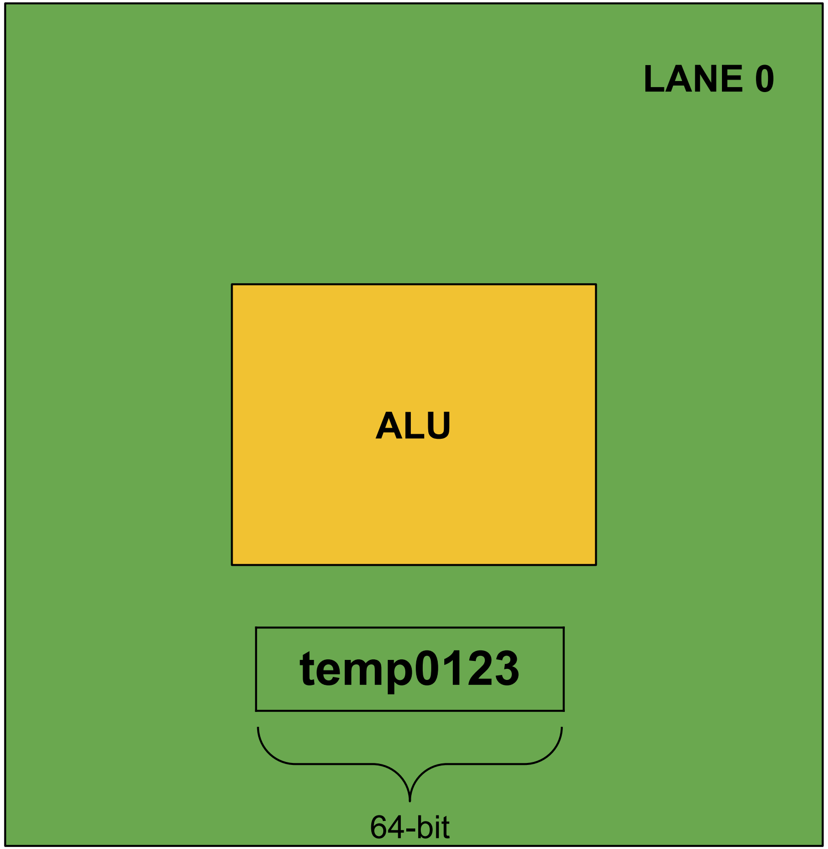
Inter-Lane reduction



Inter-Lane reduction

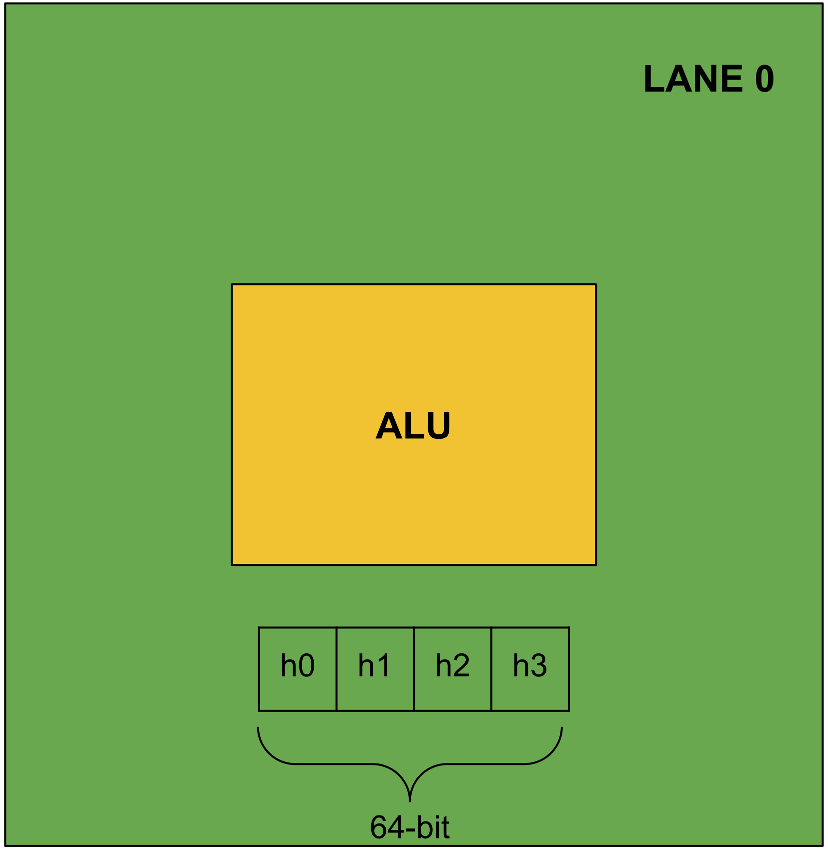


SIMD reduction



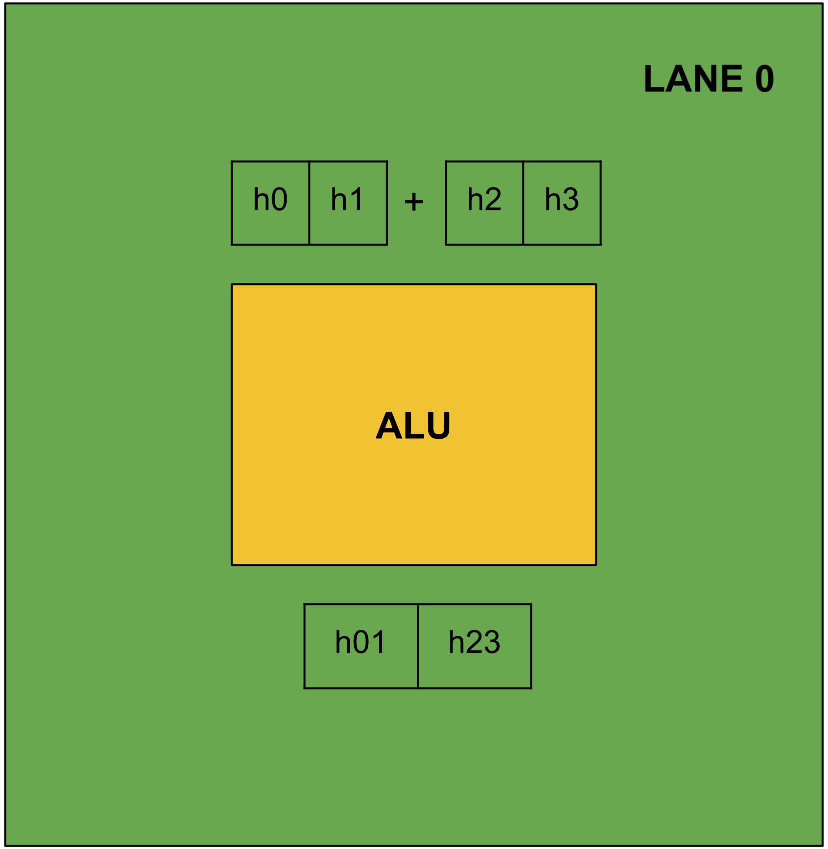
SIMD reduction (if element_width < 64)

- Reduce the 64-bit packet of N elements
- Example: four 16-bit elements
- log2(N) operations for N sub-elements
- Example: log2(4) = 2 operations



SIMD reduction (if element_width < 64)

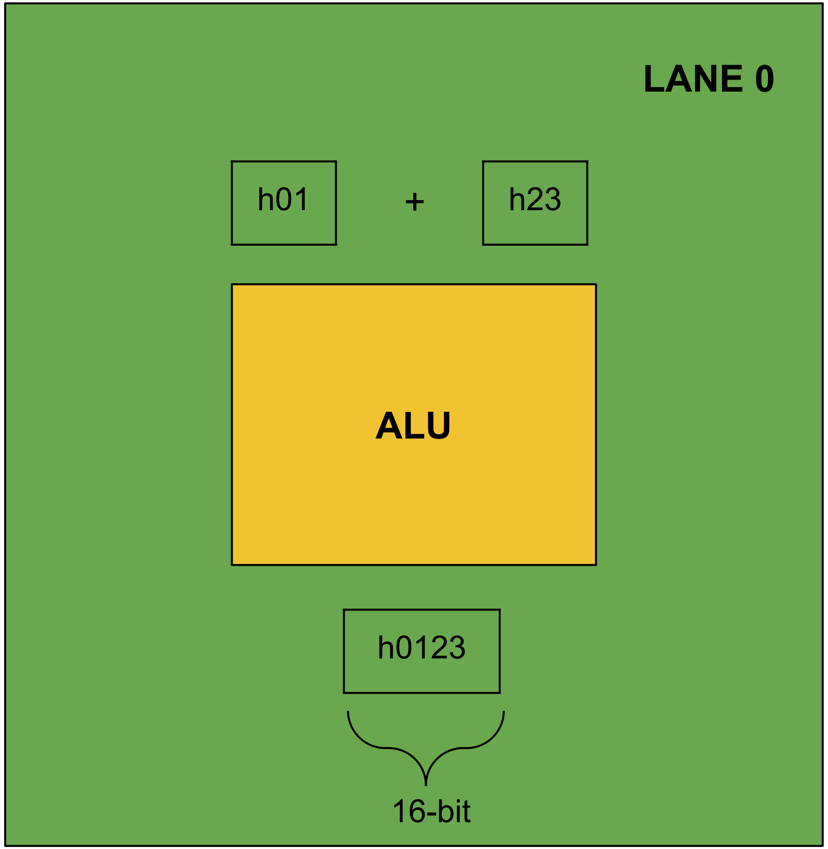
- Reduce the 64-bit packet of N elements
- Example: four 16-bit elements
- log2(N) operations for N sub-elements
- Example: log2(4) = 2 operations



First operation

SIMD reduction (if element_width < 64)

- Reduce the 64-bit packet of N elements
- Example: four 16-bit elements
- log2(N) operations for N sub-elements
- Example: log2(4) = 2 operations

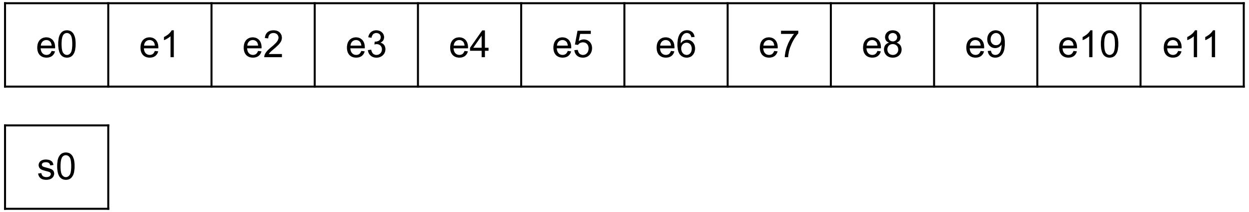


Second operation

Reductions: evaluation

- Compare HW reductions with baseline.

- Baseline: SW sequence of slides + vector operations.

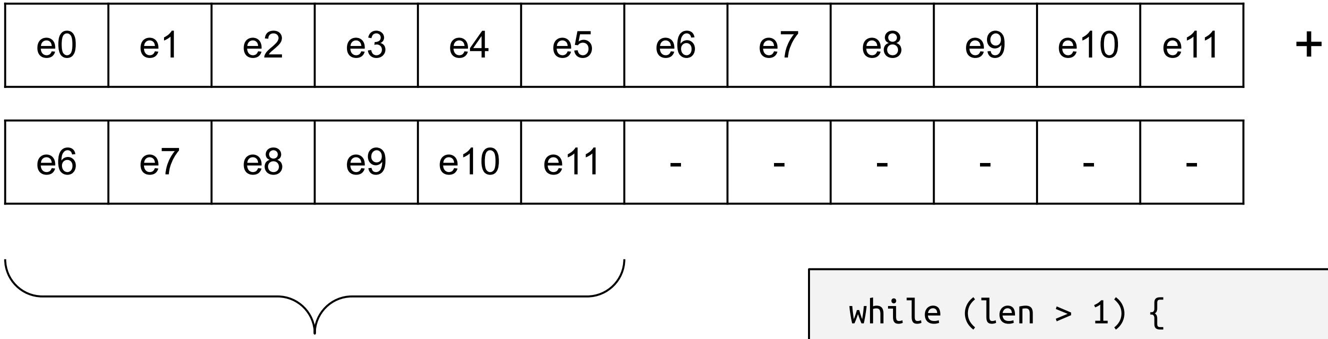


e0 e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11

s0

Reductions: evaluation

- Reduce the vector with a sequence of slides and vector operations



e0	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11	+
e6	e7	e8	e9	e10	e11	-	-	-	-	-	-	

Operate on half of the vector length

```
while (len > 1) {
        len >>= 1;
        vslidedown.vx v0, v8, len;
        vsetvli zero, len, e64, m1;
        vadd.vv v8, v8, v0;
}
```

Reductions: evaluation

- Reduce a vector of 64 elements, 64-bit each
- 120 cycles for the baseline, 23 cycles for the HW implementation
- BUT:
    - This solution worsens the WNS (estimate: -3.6% frequency)
    - The baseline can be optimized:
        - Maximize throughput of the slide unit
        - Eliminate stalls with more fine-grained hazard checks on slides
        - Avoid bank conflicts

- VREDSUM baseline: good benchmark to evaluate the performance of the slide unit

SW benchmarks

- Add benchmarks to the pool and use them for verification purposes
- Optimize the benchmarks and get performance metrics

Porting to Ara already vectorized benchmarks* for RVV 0.10, with LLVM intrinsics:

- **AXPY**
- **ParticleFilter**
- **Pathfinder**
- **Jacobi2d**
- Blackscholes
- Canneal
- LavaMD2
- Others...

https://github.com/RALC88/riscv-vectorized-benchmark-suite/tree/rvv-1.0
from paper https://doi.org/10.1145/3422667

Ongoing

- Complete integer reductions (analysis of the impact on Ara clock period)
- Optimize slide unit and hazard checks
- Finish porting the already-vectorized benchmarks
- Vectorize Softmax, GaussBlur, …
- Verification + Optimization