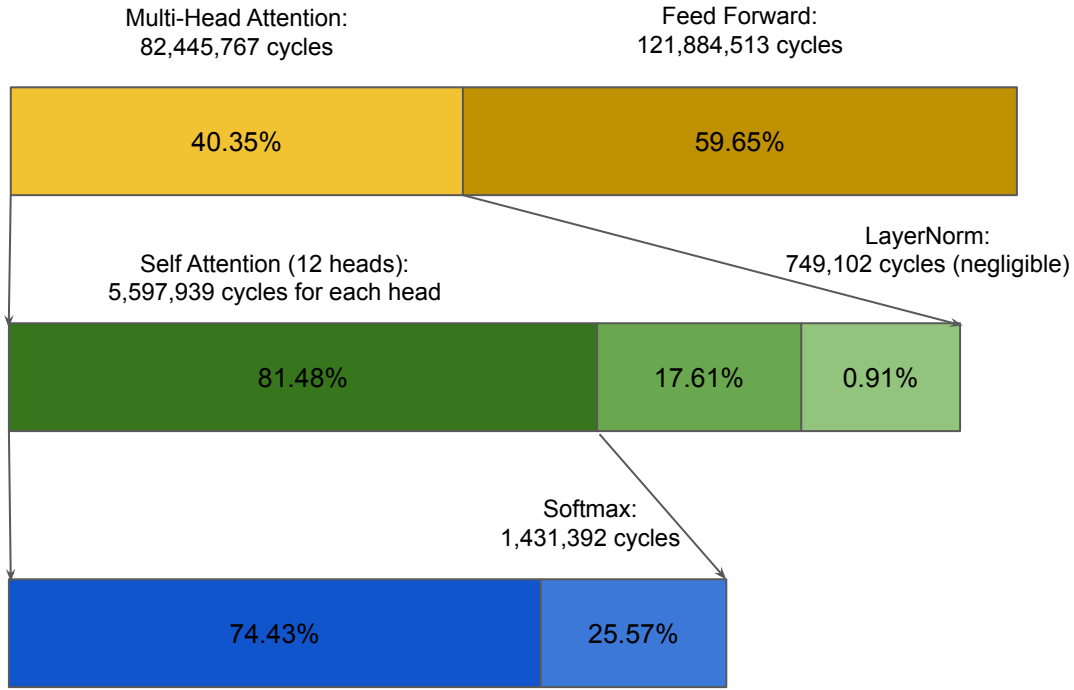
Week 8

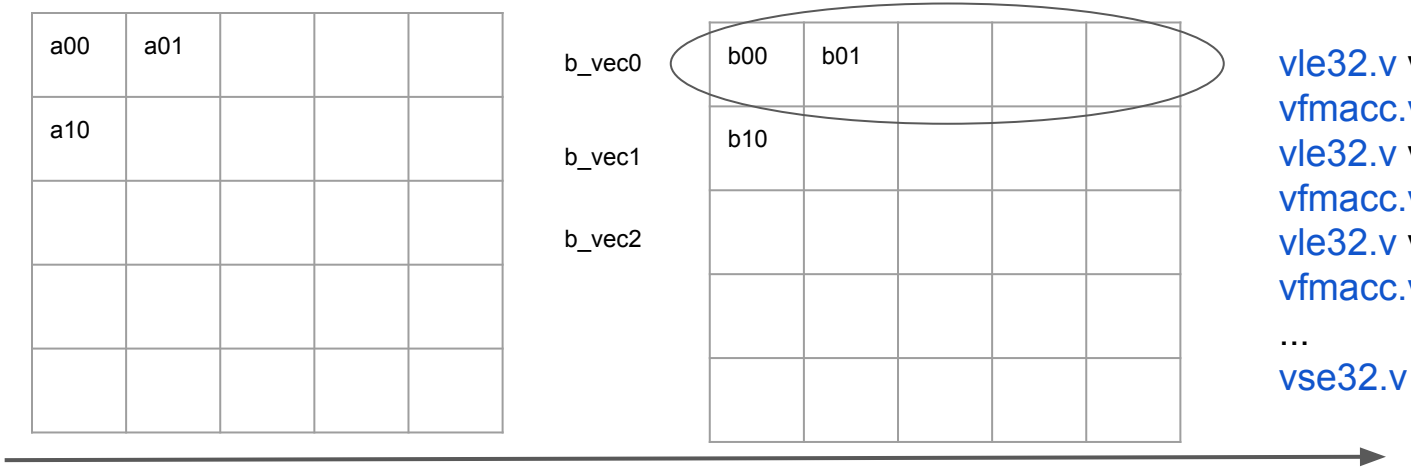
Transformer Performance



MatMul Benchmark

Input dimension	rvv-intrinsic (32-float)	assembly (64-double)
(64x64) * (64x64)	3.6 FLOP/cycle, 44.98% utilization	5.65 FLOP/cycle, 70.7% utilization
(128x128) * (128x128)	4.96 FLOP/cycle, 62.0% utilization	6.66 FLOP/cycle, 83.21% utilization

RVV-Intrinsic vs Assembly




```
vle32.v v9, (a3)
vfmacc.vf v8, ft0, v9
vle32.v v9, (a3)
vfmacc.vf v8, ft0, v9
vle32.v v9, (a3)
vfmacc.vf v8, ft0, v9
...
vse32.v v8, (a3)
```

(a00 * b_vec0) + (a01 * b_vec1) + (a02 * b_vec2) + … + (a0n * b_vec_n) = C_row_0
(a10 * b_vec0) + (a11 * b_vec1) + (a12 * b_vec2) + … + (a1n * b_vec_n) = C_row_1
…
(an0 * b_vec0) + (an1 * b_vec1) + (an2 * b_vec2) + … + (ann * b_vec_n) = C_row_n

Problem: data dependent all the time

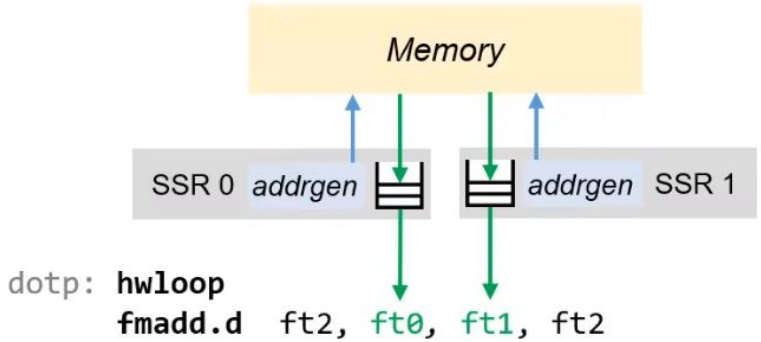
Optimized MatMul

- Use four destination vector registers (partial_sum_i)
- Performance: (fp32, 64x64)
  - 5.97 FLOP/Cycle (5.65)
  - 74.58% utilization (70.7%)

```
void matmul(float *mat_a, float *mat_b, float *o, int dim1, int dim2, int dim3, int transposed) {
  size_t vlmax = vsetvlmax_e32m1();
  vfloat32m1_t partial_sum_1, partial_sum_2, partial_sum_3, partial_sum_4;

  for (int i = 0; i < dim1; i = i + 4) {
    for (int j = 0; j < dim3;) {
      int vl = vlmax;
      if (j + vlmax > dim3) vl = vsetvl_e32m1(dim3 - j);

      partial_sum_1 = vfmv_v_f_f32m1(0, vl);
      partial_sum_2 = vfmv_v_f_f32m1(0, vl);
      partial_sum_3 = vfmv_v_f_f32m1(0, vl);
      partial_sum_4 = vfmv_v_f_f32m1(0, vl);

      for (int k = 0; k < dim2; k++) {
        vfloat32m1_t b_vec = vle32_v_f32m1(&mat_b[k * dim3 + j], vl);
        partial_sum_1 = vfmacc_vf_f32m1(partial_sum_1, mat_a[i * dim2 + k], b_vec, vl);
        partial_sum_2 = vfmacc_vf_f32m1(partial_sum_2, mat_a[(i + 1) * dim2 + k], b_vec, vl);
        partial_sum_3 = vfmacc_vf_f32m1(partial_sum_3, mat_a[(i + 2) * dim2 + k], b_vec, vl);
        partial_sum_4 = vfmacc_vf_f32m1(partial_sum_4, mat_a[(i + 3) * dim2 + k], b_vec, vl);
      }

      if (transposed == 0) {
        vse32_v_f32m1(&o[i * dim3 + j], partial_sum_1, vl);
        vse32_v_f32m1(&o[(i + 1) * dim3 + j], partial_sum_2, vl);
        vse32_v_f32m1(&o[(i + 2) * dim3 + j], partial_sum_3, vl);
        vse32_v_f32m1(&o[(i + 3) * dim3 + j], partial_sum_4, vl);
      } else {
        vsse32_v_f32m1(&o[j * dim1 + i], dim1 * 4, partial_sum_1, vl);
        vsse32_v_f32m1(&o[j * dim1 + i + 1], dim1 * 4, partial_sum_2, vl);
        vsse32_v_f32m1(&o[j * dim1 + i + 2], dim1 * 4, partial_sum_3, vl);
        vsse32_v_f32m1(&o[j * dim1 + i + 3], dim1 * 4, partial_sum_4, vl);
      }
      j += vl;
    }
  }
```

Stream Semantic Register (SSR)

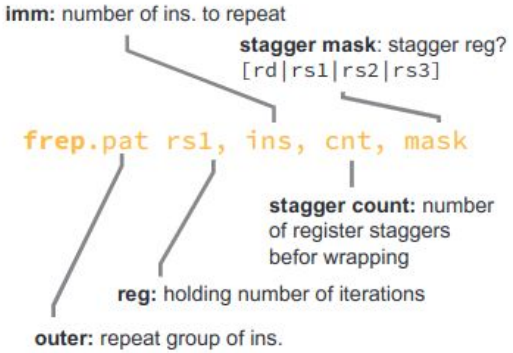
- For non-superscalar cpu, at least one cycle for ld/st instruction
  - Implicite load/store instructions:
    - memory access→ registers access
- Modification to the core
  - Intercept access to register file
    - If ssr enabled, and vd is ssr
  - Additional CSR
- Performance
  - single-core : 2x ~ 3.7x speed-up

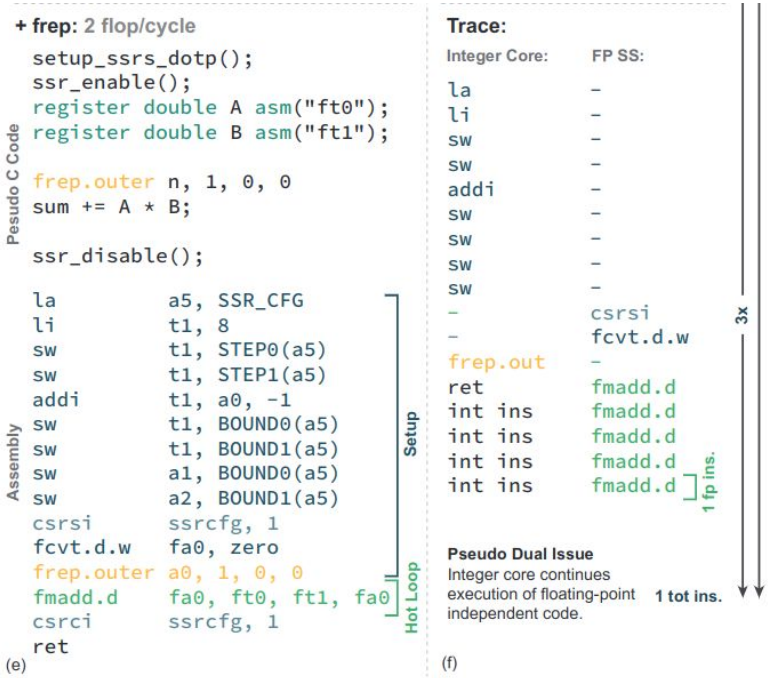


Stream Semantic Registers: A Lightweight RISC-V ISA Extension Achieving Full Compute Utilization in Single-Issue Cores, Fabian Schuiki, Florian Zaruba, Torsten Hoefler, and Luca Benini
Snitch: A tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads, Florian Zaruba, Fabian Schuiki, Torsten Hoefler, and Luca Benin
Indirection Stream Semantic Register Architecture for Efficient Sparse-Dense Linear Algebra

FPU Sequence Buffer (frep)

- Repeat the FPU instructions





Snitch: A tiny Pseudo Dual-Issue Processor for Area and Energy Efficient Execution of Floating-Point Intensive Workloads, Florian Zaruba, Fabian Schuiki, Torsten Hoefler, and Luca Benini

Possible Hardware Improvements to Ara

- Hierarchical 2D Architecture (enhance scalability)
- SSR & FREP
- Apply sparse data format
- …