Update on Ara

01/06/2022

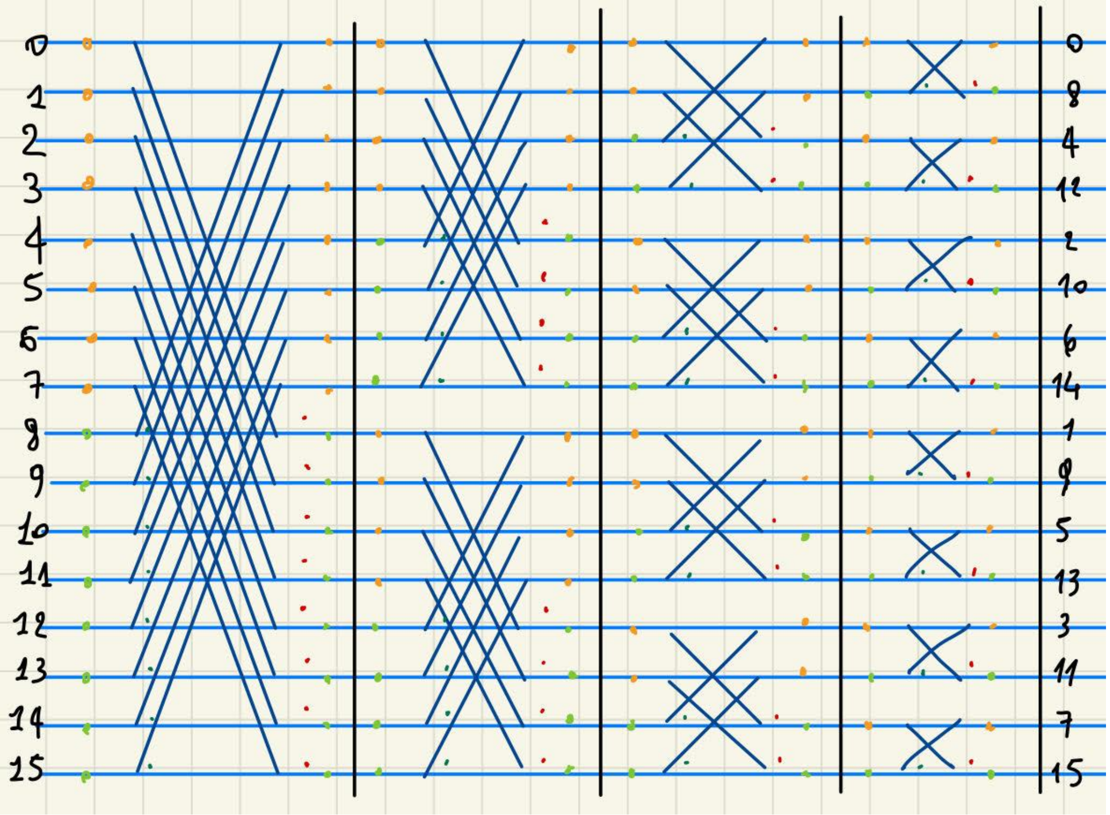
Matteo Perotti
Matheus Cavalcante
Nils Wistoff
Gianmarco Ottavi

Professor Luca Benini
Integrated Systems Laboratory
ETH Zürich

FFT timeline

- ✓ Python golden model
- ✓ Scalar DIT + DIF (CVA6 only)
- ➢ Vectorized DIF algorithm
  - ✓ First complete implementation
  - ➢ Debugging
- ✗ Performance analysis
- ✗ Optimization
- ✗ ISA extension?

Vectorized FFT - DIF



Vectorized FFT - Issues

- Problem with intrinsics
  - Typed
  - Missing *mask-vector slides*

Vectorized FFT - Issues

- Problem with intrinsics
  - Typed
  - Missing *mask-vector slides*

↓

- Load mask vectors from memory
- Check if bottleneck

Vectorized FFT - Issues

- Ara hangs
  - First complex algorithm
  - All the units are stressed together

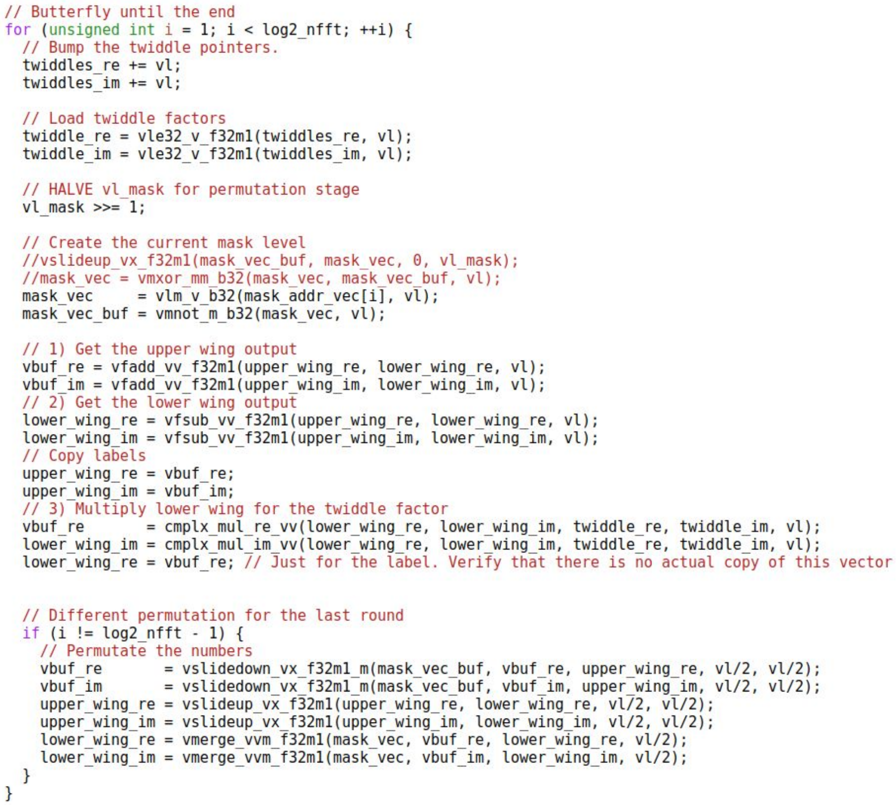
Vectorized FFT - Issues

- Ara hangs
  - First complex algorithm
  - All the units are stressed together

↓

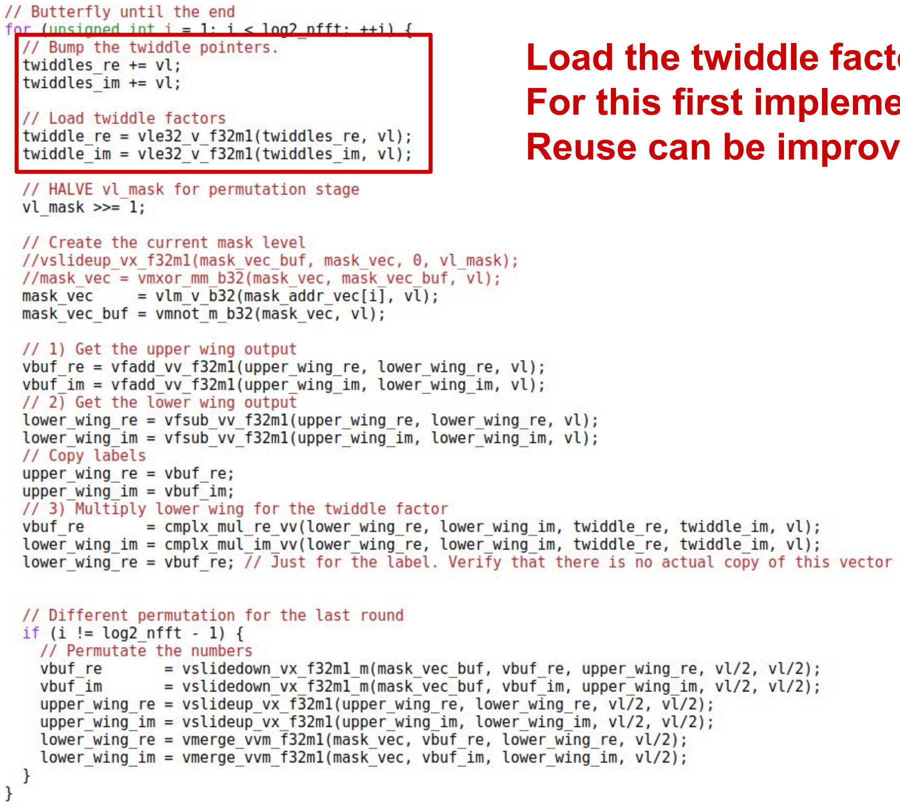
- Debugging

Vector FFT - DIF



```
// Butterfly until the end
for (unsigned int i = 1; i < log2_nfft; ++i) {
  // Bump the twiddle pointers.
  twiddles_re += vl;
  twiddles_im += vl;

  // Load twiddle factors
  twiddle_re = vle32_v_f32m1(twiddles_re, vl);
  twiddle_im = vle32_v_f32m1(twiddles_im, vl);

  // HALVE vl_mask for permutation stage
  vl_mask >>= 1;

  // Create the current mask level
  //vslideup_vx_f32m1(mask_vec_buf, mask_vec, 0, vl_mask);
  //mask_vec = vmxor_mm_b32(mask_vec, mask_vec_buf, vl);
  mask_vec     = vlm_v_b32(mask_addr_vec[i], vl);
  mask_vec_buf = vmnot_m_b32(mask_vec, vl);

  // 1) Get the upper wing output
  vbuf_re = vfadd_vv_f32m1(upper_wing_re, lower_wing_re, vl);
  vbuf_im = vfadd_vv_f32m1(upper_wing_im, lower_wing_im, vl);
  // 2) Get the lower wing output
  lower_wing_re = vfsub_vv_f32m1(upper_wing_re, lower_wing_re, vl);
  lower_wing_im = vfsub_vv_f32m1(upper_wing_im, lower_wing_im, vl);
  // Copy labels
  upper_wing_re = vbuf_re;
  upper_wing_im = vbuf_im;
  // 3) Multiply lower wing for the twiddle factor
  vbuf_re       = cmplx_mul_re_vv(lower_wing_re, lower_wing_im, twiddle_re, twiddle_im, vl);
  lower_wing_im = cmplx_mul_im_vv(lower_wing_re, lower_wing_im, twiddle_re, twiddle_im, vl);
  lower_wing_re = vbuf_re; // Just for the label. Verify that there is no actual copy of this vector


  // Different permutation for the last round
  if (i != log2_nfft - 1) {
    // Permutate the numbers
    vbuf_re       = vslidedown_vx_f32m1_m(mask_vec_buf, vbuf_re, upper_wing_re, vl/2, vl/2);
    vbuf_im       = vslidedown_vx_f32m1_m(mask_vec_buf, vbuf_im, upper_wing_im, vl/2, vl/2);
    upper_wing_re = vslideup_vx_f32m1(upper_wing_re, lower_wing_re, vl/2, vl/2);
    upper_wing_im = vslideup_vx_f32m1(upper_wing_im, lower_wing_im, vl/2, vl/2);
    lower_wing_re = vmerge_vvm_f32m1(mask_vec, vbuf_re, lower_wing_re, vl/2);
    lower_wing_im = vmerge_vvm_f32m1(mask_vec, vbuf_im, lower_wing_im, vl/2);
  }
}
```

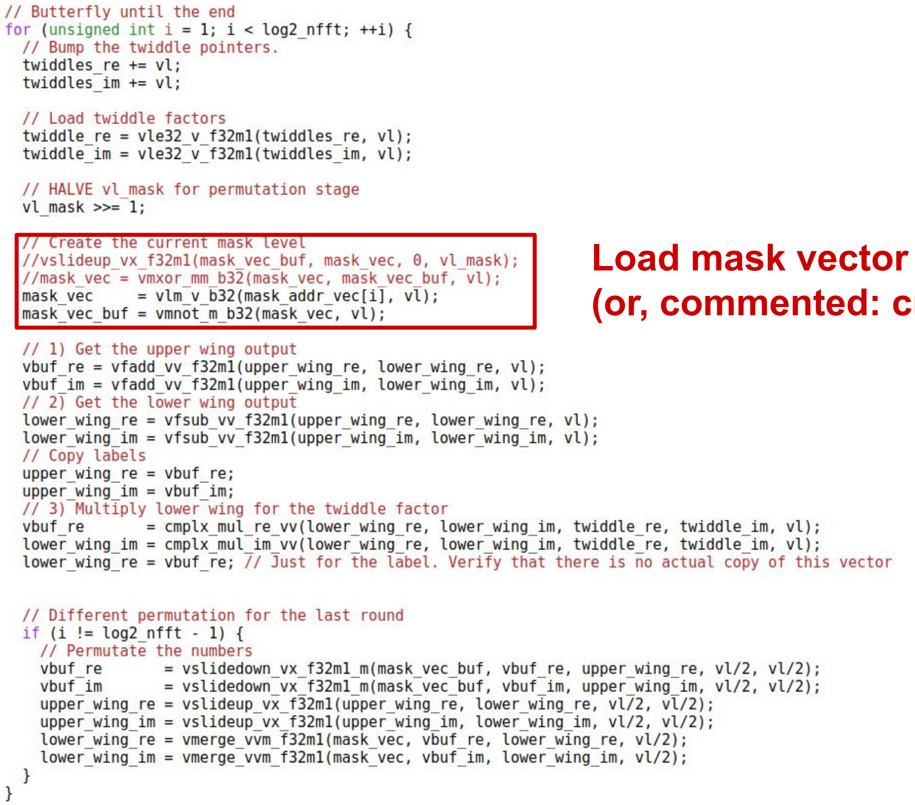
Vectorized FFT - DIF



```
// Butterfly until the end
for (unsigned int i = 1; i < log2_nfft; ++i) {
  // Bump the twiddle pointers.
  twiddles_re += vl;
  twiddles_im += vl;

  // Load twiddle factors
  twiddle_re = vle32_v_f32m1(twiddles_re, vl);
  twiddle_im = vle32_v_f32m1(twiddles_im, vl);

  // HALVE vl_mask for permutation stage
  vl_mask >>= 1;

  // Create the current mask level
  //vslideup_vx_f32m1(mask_vec_buf, mask_vec, 0, vl_mask);
  //mask_vec = vmxor_mm_b32(mask_vec, mask_vec_buf, vl);
  mask_vec     = vlm_v_b32(mask_addr_vec[i], vl);
  mask_vec_buf = vmnot_m_b32(mask_vec, vl);

  // 1) Get the upper wing output
  vbuf_re = vfadd_vv_f32m1(upper_wing_re, lower_wing_re, vl);
  vbuf_im = vfadd_vv_f32m1(upper_wing_im, lower_wing_im, vl);
  // 2) Get the lower wing output
  lower_wing_re = vfsub_vv_f32m1(upper_wing_re, lower_wing_re, vl);
  lower_wing_im = vfsub_vv_f32m1(upper_wing_im, lower_wing_im, vl);
  // Copy labels
  upper_wing_re = vbuf_re;
  upper_wing_im = vbuf_im;
  // 3) Multiply lower wing for the twiddle factor
  vbuf_re       = cmplx_mul_re_vv(lower_wing_re, lower_wing_im, twiddle_re, twiddle_im, vl);
  lower_wing_im = cmplx_mul_im_vv(lower_wing_re, lower_wing_im, twiddle_re, twiddle_im, vl);
  lower_wing_re = vbuf_re; // Just for the label. Verify that there is no actual copy of this vector


  // Different permutation for the last round
  if (i != log2_nfft - 1) {
    // Permutate the numbers
    vbuf_re       = vslidedown_vx_f32m1_m(mask_vec_buf, vbuf_re, upper_wing_re, vl/2, vl/2);
    vbuf_im       = vslidedown_vx_f32m1_m(mask_vec_buf, vbuf_im, upper_wing_im, vl/2, vl/2);
    upper_wing_re = vslideup_vx_f32m1(upper_wing_re, lower_wing_re, vl/2, vl/2);
    upper_wing_im = vslideup_vx_f32m1(upper_wing_im, lower_wing_im, vl/2, vl/2);
    lower_wing_re = vmerge_vvm_f32m1(mask_vec, vbuf_re, lower_wing_re, vl/2);
    lower_wing_im = vmerge_vvm_f32m1(mask_vec, vbuf_im, lower_wing_im, vl/2);
  }
}
```

Load the twiddle factors from memory
For this first implementation, they are already in memory
Reuse can be improved in next implementations

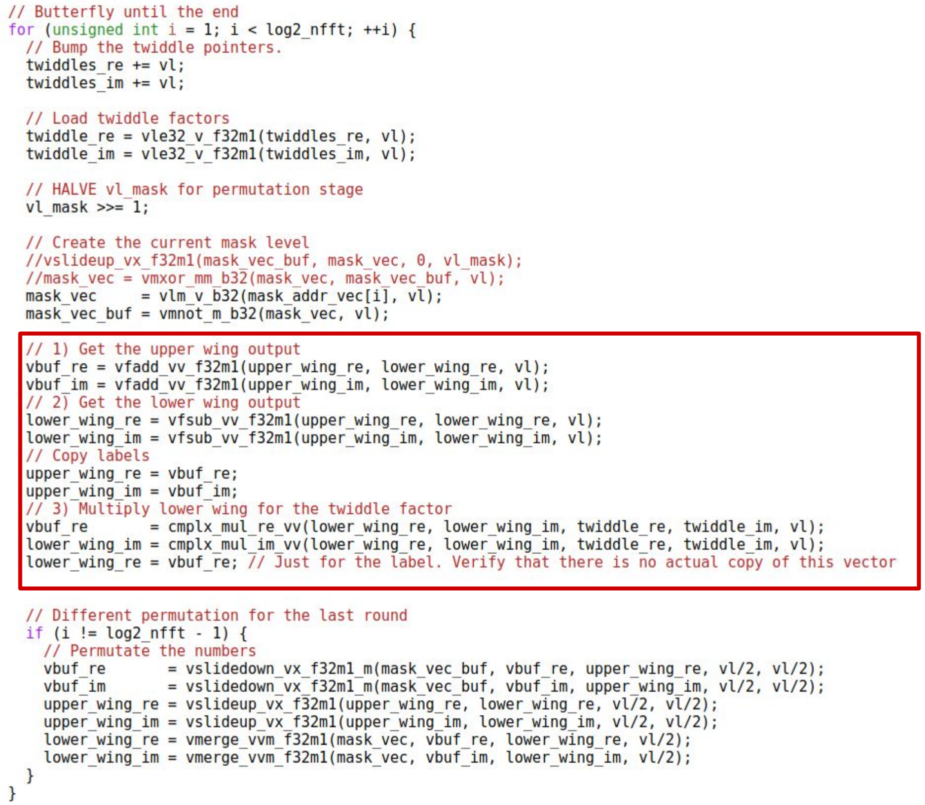
Vectorized FFT - DIF



```
// Butterfly until the end
for (unsigned int i = 1; i < log2_nfft; ++i) {
  // Bump the twiddle pointers.
  twiddles_re += vl;
  twiddles_im += vl;

  // Load twiddle factors
  twiddle_re = vle32_v_f32m1(twiddles_re, vl);
  twiddle_im = vle32_v_f32m1(twiddles_im, vl);

  // HALVE vl_mask for permutation stage
  vl_mask >>= 1;

  // Create the current mask level
  //vslideup_vx_f32m1(mask_vec_buf, mask_vec, 0, vl_mask);
  //mask_vec = vmxor_mm_b32(mask_vec, mask_vec_buf, vl);
  mask_vec     = vlm_v_b32(mask_addr_vec[i], vl);
  mask_vec_buf = vmnot_m_b32(mask_vec, vl);

  // 1) Get the upper wing output
  vbuf_re = vfadd_vv_f32m1(upper_wing_re, lower_wing_re, vl);
  vbuf_im = vfadd_vv_f32m1(upper_wing_im, lower_wing_im, vl);
  // 2) Get the lower wing output
  lower_wing_re = vfsub_vv_f32m1(upper_wing_re, lower_wing_re, vl);
  lower_wing_im = vfsub_vv_f32m1(upper_wing_im, lower_wing_im, vl);
  // Copy labels
  upper_wing_re = vbuf_re;
  upper_wing_im = vbuf_im;
  // 3) Multiply lower wing for the twiddle factor
  vbuf_re       = cmplx_mul_re_vv(lower_wing_re, lower_wing_im, twiddle_re, twiddle_im, vl);
  lower_wing_im = cmplx_mul_im_vv(lower_wing_re, lower_wing_im, twiddle_re, twiddle_im, vl);
  lower_wing_re = vbuf_re; // Just for the label. Verify that there is no actual copy of this vector


  // Different permutation for the last round
  if (i != log2_nfft - 1) {
    // Permutate the numbers
    vbuf_re       = vslidedown_vx_f32m1_m(mask_vec_buf, vbuf_re, upper_wing_re, vl/2, vl/2);
    vbuf_im       = vslidedown_vx_f32m1_m(mask_vec_buf, vbuf_im, upper_wing_im, vl/2, vl/2);
    upper_wing_re = vslideup_vx_f32m1(upper_wing_re, lower_wing_re, vl/2, vl/2);
    upper_wing_im = vslideup_vx_f32m1(upper_wing_im, lower_wing_im, vl/2, vl/2);
    lower_wing_re = vmerge_vvm_f32m1(mask_vec, vbuf_re, lower_wing_re, vl/2);
    lower_wing_im = vmerge_vvm_f32m1(mask_vec, vbuf_im, lower_wing_im, vl/2);
  }
}
```

Load mask vector for permutation from memory (or, commented: create the mask vector within Ara)

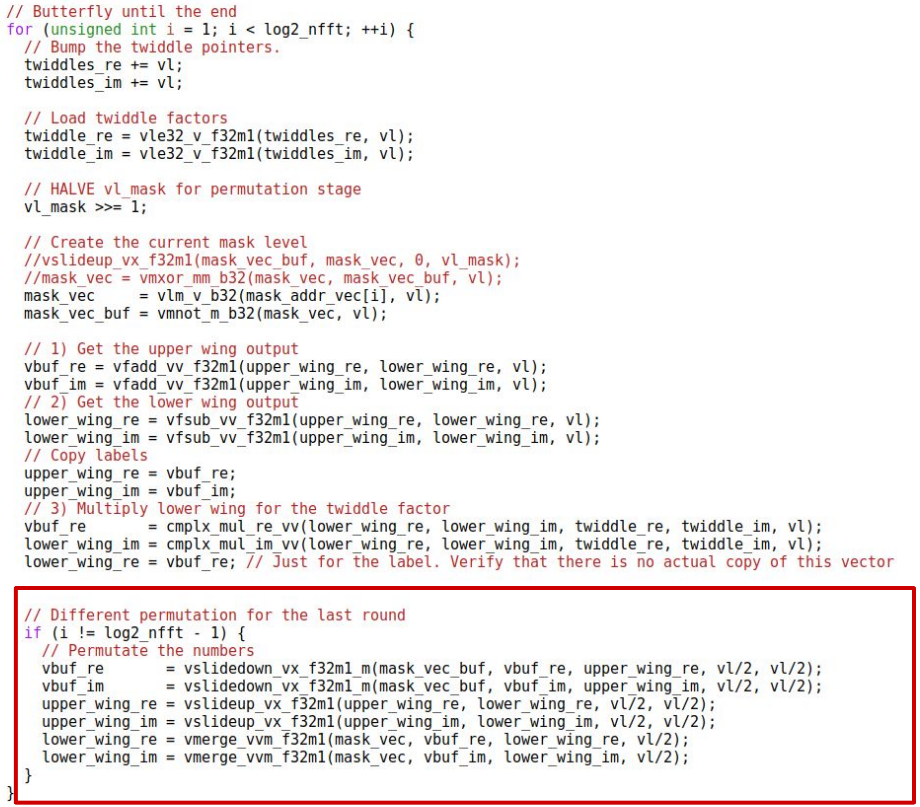
Vectorized FFT - DIF



```
// Butterfly until the end
for (unsigned int i = 1; i < log2_nfft; ++i) {
  // Bump the twiddle pointers.
  twiddles_re += vl;
  twiddles_im += vl;

  // Load twiddle factors
  twiddle_re = vle32_v_f32m1(twiddles_re, vl);
  twiddle_im = vle32_v_f32m1(twiddles_im, vl);

  // HALVE vl_mask for permutation stage
  vl_mask >>= 1;

  // Create the current mask level
  //vslideup_vx_f32m1(mask_vec_buf, mask_vec, 0, vl_mask);
  //mask_vec = vmxor_mm_b32(mask_vec, mask_vec_buf, vl);
  mask_vec     = vlm_v_b32(mask_addr_vec[i], vl);
  mask_vec_buf = vmnot_m_b32(mask_vec, vl);

  // 1) Get the upper wing output
  vbuf_re = vfadd_vv_f32m1(upper_wing_re, lower_wing_re, vl);
  vbuf_im = vfadd_vv_f32m1(upper_wing_im, lower_wing_im, vl);
  // 2) Get the lower wing output
  lower_wing_re = vfsub_vv_f32m1(upper_wing_re, lower_wing_re, vl);
  lower_wing_im = vfsub_vv_f32m1(upper_wing_im, lower_wing_im, vl);
  // Copy labels
  upper_wing_re = vbuf_re;
  upper_wing_im = vbuf_im;
  // 3) Multiply lower wing for the twiddle factor
  vbuf_re       = cmplx_mul_re_vv(lower_wing_re, lower_wing_im, twiddle_re, twiddle_im, vl);
  lower_wing_im = cmplx_mul_im_vv(lower_wing_re, lower_wing_im, twiddle_re, twiddle_im, vl);
  lower_wing_re = vbuf_re; // Just for the label. Verify that there is no actual copy of this vector

  // Different permutation for the last round
  if (i != log2_nfft - 1) {
    // Permutate the numbers
    vbuf_re       = vslidedown_vx_f32m1_m(mask_vec_buf, vbuf_re, upper_wing_re, vl/2, vl/2);
    vbuf_im       = vslidedown_vx_f32m1_m(mask_vec_buf, vbuf_im, upper_wing_im, vl/2, vl/2);
    upper_wing_re = vslideup_vx_f32m1(upper_wing_re, lower_wing_re, vl/2, vl/2);
    upper_wing_im = vslideup_vx_f32m1(upper_wing_im, lower_wing_im, vl/2, vl/2);
    lower_wing_re = vmerge_vvm_f32m1(mask_vec, vbuf_re, lower_wing_re, vl/2);
    lower_wing_im = vmerge_vvm_f32m1(mask_vec, vbuf_im, lower_wing_im, vl/2);
  }
}
```

Butterfly the vectors

Vectorized FFT - DIF



```
// Butterfly until the end
for (unsigned int i = 1; i < log2_nfft; ++i) {
  // Bump the twiddle pointers.
  twiddles_re += vl;
  twiddles_im += vl;

  // Load twiddle factors
  twiddle_re = vle32_v_f32m1(twiddles_re, vl);
  twiddle_im = vle32_v_f32m1(twiddles_im, vl);

  // HALVE vl_mask for permutation stage
  vl_mask >>= 1;

  // Create the current mask level
  //vslideup_vx_f32m1(mask_vec_buf, mask_vec, 0, vl_mask);
  //mask_vec = vmxor_mm_b32(mask_vec, mask_vec_buf, vl);
  mask_vec     = vlm_v_b32(mask_addr_vec[i], vl);
  mask_vec_buf = vmnot_m_b32(mask_vec, vl);

  // 1) Get the upper wing output
  vbuf_re = vfadd_vv_f32m1(upper_wing_re, lower_wing_re, vl);
  vbuf_im = vfadd_vv_f32m1(upper_wing_im, lower_wing_im, vl);
  // 2) Get the lower wing output
  lower_wing_re = vfsub_vv_f32m1(upper_wing_re, lower_wing_re, vl);
  lower_wing_im = vfsub_vv_f32m1(upper_wing_im, lower_wing_im, vl);
  // Copy labels
  upper_wing_re = vbuf_re;
  upper_wing_im = vbuf_im;
  // 3) Multiply lower wing for the twiddle factor
  vbuf_re       = cmplx_mul_re_vv(lower_wing_re, lower_wing_im, twiddle_re, twiddle_im, vl);
  lower_wing_im = cmplx_mul_im_vv(lower_wing_re, lower_wing_im, twiddle_re, twiddle_im, vl);
  lower_wing_re = vbuf_re; // Just for the label. Verify that there is no actual copy of this vector

  // Different permutation for the last round
  if (i != log2_nfft - 1) {
    // Permutate the numbers
    vbuf_re       = vslidedown_vx_f32m1_m(mask_vec_buf, vbuf_re, upper_wing_re, vl/2, vl/2);
    vbuf_im       = vslidedown_vx_f32m1_m(mask_vec_buf, vbuf_im, upper_wing_im, vl/2, vl/2);
    upper_wing_re = vslideup_vx_f32m1(upper_wing_re, lower_wing_re, vl/2, vl/2);
    upper_wing_im = vslideup_vx_f32m1(upper_wing_im, lower_wing_im, vl/2, vl/2);
    lower_wing_re = vmerge_vvm_f32m1(mask_vec, vbuf_re, lower_wing_re, vl/2);
    lower_wing_im = vmerge_vvm_f32m1(mask_vec, vbuf_im, lower_wing_im, vl/2);
  }
}
```

Permutate the vectors for the “next step”
4 * (vl/2)-long slides -> costly

ETH zürich

FFT timeline

- ✓ Python golden model
- ✓ Scalar DIT + DIF (CVA6 only)
- ➢ Vectorized DIF algorithm
  - ✓ First complete implementation
  - ➢ Debugging
- ✗ Performance analysis
- ✗ Optimization
- ✗ ISA extension?