Week 13

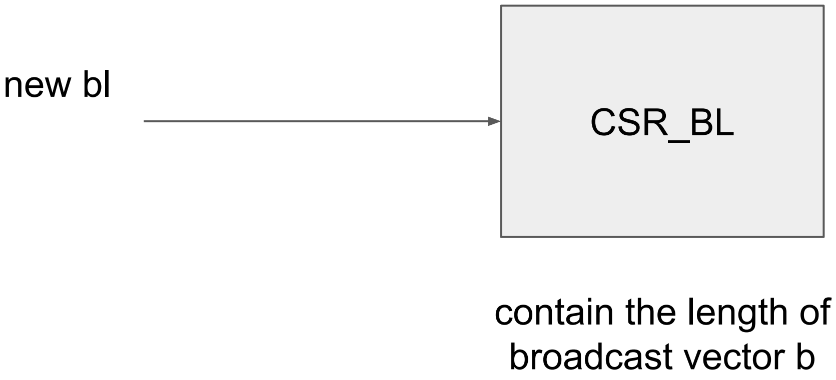
New Instructions

- **vsetbl rd rs1**
- **vle<eew>bc rs1**
- **vfbmacc.vf vd, rs1, vs2**
- **vfbmacc.vv vd, vs2**

LLVM & Spike

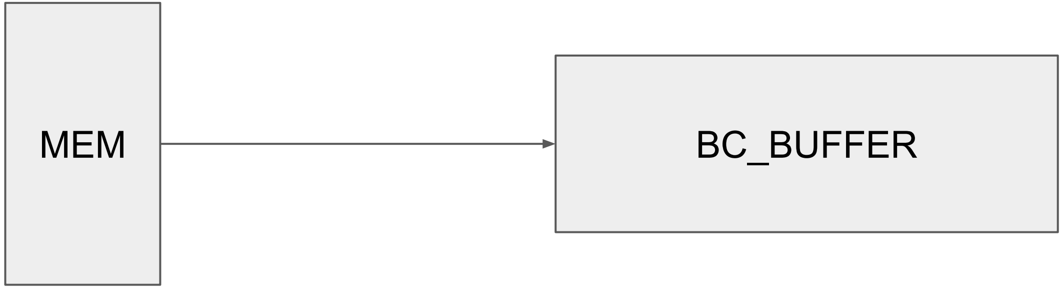
vsetbl

- set the broadcast length CSR
- CSR_BL = rs1, rd = rs1
- 31-26: 1010000 (vsetvl: 1000000)



vle<eew>bc

- load vector to the broadcast buffer
- support all element widths
- use bl instead vl
- lumop: 11000 (unit-stride load: 00000)



vfbmacc

- vfbmacc.vv vd, (vs1), vs2
    - vs1 unused
    - vd[i, :] = bc_vec[:] * vs2[i] + vd[i, :]
- vfbmacc.vf vd, rs1, vs2
    - vd[i, :] = bc_vec[:] * vs2[i] + rs1
    - to avoid initialization (slow)
    - rs1 = 0
- func6: 111001 (vfmacc: 101100)

New MatMul Program

```c
void fmatmul(float *c, const float *a, const float *b, const unsigned long int M,
             const unsigned long int N, const unsigned long int P) {
  const int REUSE_SIZE = 1;
  const int stride_a = 4 * N;
  const int stride_c = 4 * P;
  // We work on 64 elements of the matrix B at once
  // TODO: use 64 with LMUL=m2
  const unsigned long int block_size_p = 32;
  // block_size_m <= M, REUSE_SIZE * length of b_vec * 32 < VRF capacity
  const unsigned long int block_size_m = NR_LANES * REUSE_SIZE;

  for (unsigned long int p = 0; p < P; p += block_size_p) {
    // Find pointers to the submatrices
    const float *b_ = b + p;
    float *c_ = c + p;

    // Set the broadcast length
    const unsigned long int p_ = MIN(P - p, block_size_p);
    int tmp1 = 0;
    int tmp2 = 0;
    asm volatile("vsetbl %0, %1, %2" ::"r"(tmp2), "r"(p_), "r"(tmp1));

    // Iterate over the rows
    for (unsigned long int m = 0; m < M; m += block_size_m) {
      // Set the vector length
      const unsigned long int m_ = MIN(M - m, block_size_m);
      asm volatile("vsetvli zero, %0, e32, m1, ta, ma" ::"r"(m_));

      // Find pointer to the submatrices
      const float *a_ = a + m * N;
      float *c__ = c_ + m * P;

      // First op with scalar zero
      // load vec_a
      asm volatile("vlse32.v v0, (%0), %1" ::"r"(a_), "r"(stride_a));
      // load vec_b
      asm volatile("vle32bc.v v31, (%0)" ::"r"(b_));
      float t0 = 0; // First Operation, accumulated result is 0
      asm volatile("vfbmacc.vf v8, %0, v0" ::"f"(t0));

      for (unsigned long int n = 1; n < N; n++) {
        // load vec_a
        asm volatile("vlse32.v v0, (%0), %1" ::"r"(a_ + n), "r"(stride_a));
        // load vec_b
        asm volatile("vle32bc.v v31, (%0)" ::"r"(b_ + n * P));
        asm volatile("vfbmacc.vv v8, v31, v0");
      }

      asm volatile("vsetvli zero, %0, e32, m1, ta, ma" ::"r"(block_size_p * NR_LANES));
      asm volatile("vsse32.v v8, (%0), %1" ::"r"(c__), "r"(stride_c));
    }
  }
}
```

New MatMul Program

- broadcast length = 32
  - vlen = 4096 (length of one vreg)
  - vlen / #lanes / sizeof(float) = 32
  - bl = 64 (LMUL = 2)

```c
void fmatmul(float *c, const float *a, const float *b, const unsigned long int M,
             const unsigned long int N, const unsigned long int P) {
  const int REUSE_SIZE = 1;
  const int stride_a = 4 * N;
  const int stride_c = 4 * P;
  // We work on 64 elements of the matrix B at once
  // TODO: use 64 with LMUL=m2
  const unsigned long int block_size_p = 32;
  // block_size_m <= M, REUSE_SIZE * length of b_vec * 32 < VRF capacity
  const unsigned long int block_size_m = NR_LANES * REUSE_SIZE;

  for (unsigned long int p = 0; p < P; p += block_size_p) {
    // Find pointers to the submatrices
    const float *b_ = b + p;
    float *c_ = c + p;

    // Set the broadcast length
    const unsigned long int p_ = MIN(P - p, block_size_p);
    int tmp1 = 0;
    int tmp2 = 0;
    asm volatile("vsetbl %0, %1, %2" ::"r"(tmp2), "r"(p_), "r"(tmp1));

    // Iterate over the rows
    for (unsigned long int m = 0; m < M; m += block_size_m) {
      // Set the vector length
      const unsigned long int m_ = MIN(M - m, block_size_m);
      asm volatile("vsetvli zero, %0, e32, m1, ta, ma" ::"r"(m_));

      // Find pointer to the submatrices
      const float *a_ = a + m * N;
      float *c__ = c_ + m * P;

      // First op with scalar zero
      // load vec_a
      asm volatile("vlse32.v v0, (%0), %1" ::"r"(a_), "r"(stride_a));
      // load vec_b
      asm volatile("vle32bc.v v31, (%0)" ::"r"(b_));
      float t0 = 0; // First Operation, accumulated result is 0
      asm volatile("vfbmacc.vf v8, %0, v0" ::"f"(t0));

      for (unsigned long int n = 1; n < N; n++) {
        // load vec_a
        asm volatile("vlse32.v v0, (%0), %1" ::"r"(a_ + n), "r"(stride_a));
        // load vec_b
        asm volatile("vle32bc.v v31, (%0)" ::"r"(b_ + n * P));
        asm volatile("vfbmacc.vv v8, v31, v0");
      }

      asm volatile("vsetvli zero, %0, e32, m1, ta, ma" ::"r"(block_size_p * NR_LANES));
      asm volatile("vsse32.v v8, (%0), %1" ::"r"(c__), "r"(stride_c));
    }
  }
}
```

New MatMul Program

- broadcast length = 32
  - vlen = 4096 (length of one vreg)
  - vlen / #lanes / sizeof(float) = 32
  - bl = 64 (LMUL = 2)
- reuse_size
  - = number of accumulated registers
  - reuse_size = 2, vd = v8 & v9
  - decrease memory access

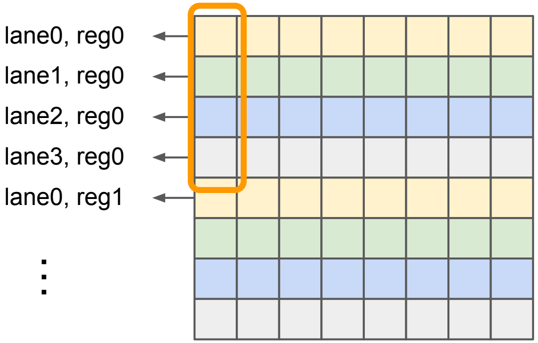
```
void fmatmul(float *c, const float *a, const float *b, const unsigned long int M,
             const unsigned long int N, const unsigned long int P) {
  const int REUSE_SIZE = 1;
  const int stride_a = 4 * N;
  const int stride_c = 4 * P;
  // We work on 64 elements of the matrix B at once
  // TODO: use 64 with LMUL=m2
  const unsigned long int block_size_p = 32;
  // block_size_m <= M, REUSE_SIZE * length of b_vec * 32 < VRF capacity
  const unsigned long int block_size_m = NR_LANES * REUSE_SIZE;

  for (unsigned long int p = 0; p < P; p += block_size_p) {
    // Find pointers to the submatrices
    const float *b_ = b + p;
    float *c_ = c + p;

    // Set the broadcast length
    const unsigned long int p_ = MIN(P - p, block_size_p);
    int tmp1 = 0;
    int tmp2 = 0;
    asm volatile("vsetbl %0, %1, %2" ::"r"(tmp2), "r"(p_), "r"(tmp1));

    // Iterate over the rows
    for (unsigned long int m = 0; m < M; m += block_size_m) {
      // Set the vector length
      const unsigned long int m_ = MIN(M - m, block_size_m);
      asm volatile("vsetvli zero, %0, e32, m1, ta, ma" ::"r"(m_));

      // Find pointer to the submatrices
      const float *a_ = a + m * N;
      float *c__ = c_ + m * P;

      // First op with scalar zero
      // load vec_a
      asm volatile("vlse32.v v0, (%0), %1" ::"r"(a_), "r"(stride_a));
      // load vec_b
      asm volatile("vle32bc.v v31, (%0)" ::"r"(b_));
      float t0 = 0; // First Operation, accumulated result is 0
      asm volatile("vfbmacc.vf v8, %0, v0" ::"f"(t0));

      for (unsigned long int n = 1; n < N; n++) {
        // load vec_a
        asm volatile("vlse32.v v0, (%0), %1" ::"r"(a_ + n), "r"(stride_a));
        // load vec_b
        asm volatile("vle32bc.v v31, (%0)" ::"r"(b_ + n * P));
        asm volatile("vfbmacc.vv v8, v31, v0");
      }

      asm volatile("vsetvli zero, %0, e32, m1, ta, ma" ::"r"(block_size_p * NR_LANES));
      asm volatile("vsse32.v v8, (%0), %1" ::"r"(c__), "r"(stride_c));
    }
  }
}
```

New MatMul Program

- broadcast length = 32
  - vlen = 4096 (length of one vreg)
  - vlen / #lanes / sizeof(float) = 32
  - bl = 64 (LMUL = 2)
- reuse_size
  - = number of accumulated registers
  - reuse_size = 2, vd = v8 & v9
  - decrease memory access
- remove initialization

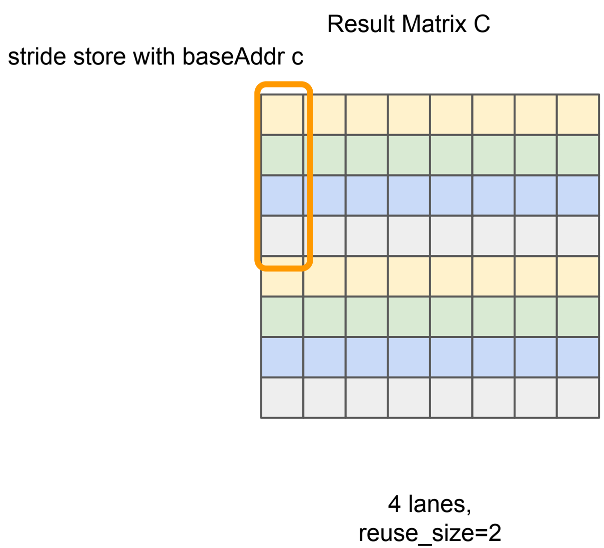
```c
void fmatmul(float *c, const float *a, const float *b, const unsigned long int M,
             const unsigned long int N, const unsigned long int P) {
  const int REUSE_SIZE = 1;
  const int stride_a = 4 * N;
  const int stride_c = 4 * P;
  // We work on 64 elements of the matrix B at once
  // TODO: use 64 with LMUL=m2
  const unsigned long int block_size_p = 32;
  // block_size_m <= M, REUSE_SIZE * length of b_vec * 32 < VRF capacity
  const unsigned long int block_size_m = NR_LANES * REUSE_SIZE;

  for (unsigned long int p = 0; p < P; p += block_size_p) {
    // Find pointers to the submatrices
    const float *b_ = b + p;
    float *c_ = c + p;

    // Set the broadcast length
    const unsigned long int p_ = MIN(P - p, block_size_p);
    int tmp1 = 0;
    int tmp2 = 0;
    asm volatile("vsetbl %0, %1, %2" ::"r"(tmp2), "r"(p_), "r"(tmp1));

    // Iterate over the rows
    for (unsigned long int m = 0; m < M; m += block_size_m) {
      // Set the vector length
      const unsigned long int m_ = MIN(M - m, block_size_m);
      asm volatile("vsetvli zero, %0, e32, m1, ta, ma" ::"r"(m_));

      // Find pointer to the submatrices
      const float *a_ = a + m * N;
      float *c__ = c_ + m * P;

      // First op with scalar zero
      // load vec_a
      asm volatile("vlse32.v v0, (%0), %1" ::"r"(a_), "r"(stride_a));
      // load vec_b
      asm volatile("vle32bc.v v31, (%0)" ::"r"(b_));
      float t0 = 0; // First Operation, accumulated result is 0
      asm volatile("vfbmacc.vf v8, %0, v0" ::"f"(t0));

      for (unsigned long int n = 1; n < N; n++) {
        // load vec_a
        asm volatile("vlse32.v v0, (%0), %1" ::"r"(a_ + n), "r"(stride_a));
        // load vec_b
        asm volatile("vle32bc.v v31, (%0)" ::"r"(b_ + n * P));
        asm volatile("vfbmacc.vv v8, v31, v0");
      }

      asm volatile("vsetvli zero, %0, e32, m1, ta, ma" ::"r"(block_size_p * NR_LANES));
      asm volatile("vsse32.v v8, (%0), %1" ::"r"(c__), "r"(stride_c));
    }
  }
}
```

Result Matrix C

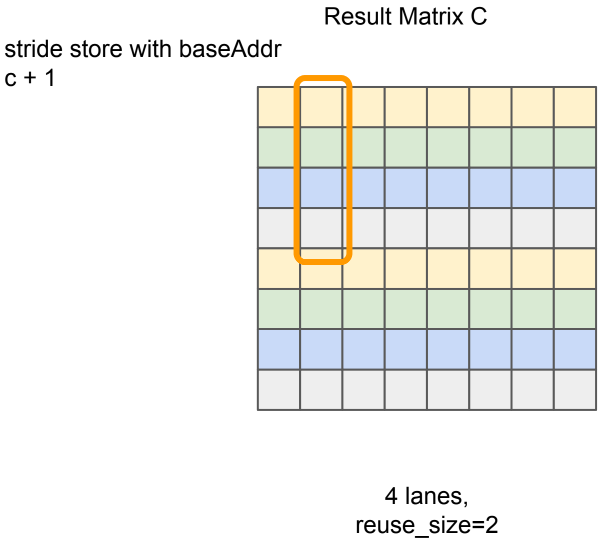


4 lanes,
reuse_size=2

New store instruction



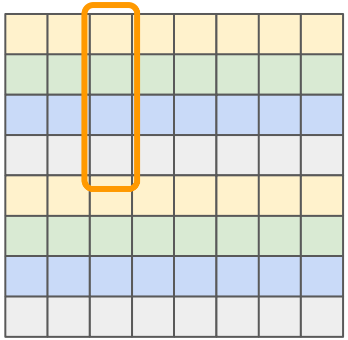
New store instruction



New store instruction

Result Matrix C

stride store with baseAddr
c + 2



4 lanes,
reuse_size=2