Matrix Support in LLVM & Clang

Florian Hahn

Contents

- Motivation
- C/C++ Matrix Types Extension
- Matrix Support in LLVM
- Performance

Motivation

Provide a high-level solution to write high-performance code for (small) matrix operations.

Motivation

- Code that makes heavy use of matrix math with small matrixes
  - Varying sizes 2x2 - 16x16
  - Varying shapes, e.g 2x2 * 2x8
  - Column Major
- Math primitives implemented in C/C++
  - Relies on SLP vectorizer (and others)
  - Relies on loop vectorizer/unroller/SLP
  - Mixing & matching leads to sub-optimal code

```
template <typename T>
void Multiply4x4_4x4(T C[16], const T A[16], const T B[16]) {
    C[0] = A[0] * B[0] + A[3] * B[12] + A[1] * B[4] + A[2] * B[8];
    C[1] = A[0] * B[1] + A[3] * B[13] + A[1] * B[5] + A[2] * B[9];
    C[2] = A[0] * B[2] + A[2] * B[10] + A[3] * B[14] + A[1] * B[6];
    C[3] = A[3] * B[15] + A[0] * B[3] + A[2] * B[11] + A[1] * B[7];
    C[4] = A[4] * B[0] + A[7] * B[12] + A[5] * B[4] + A[6] * B[8];
    C[5] = A[4] * B[1] + A[7] * B[13] + A[5] * B[5] + A[6] * B[9];
    C[6] = A[4] * B[2] + A[6] * B[10] + A[7] * B[14] + A[5] * B[6];
    C[7] = A[7] * B[15] + A[4] * B[3] + A[6] * B[11] + A[5] * B[7];
    C[8] = A[11] * B[12] + A[8] * B[0] + A[10] * B[8] + A[9] * B[4];
    C[9] = A[11] * B[13] + A[8] * B[1] + A[10] * B[9] + A[9] * B[5];
    C[10] = A[10] * B[10] + A[11] * B[14] + A[8] * B[2] + A[9] * B[6];
    C[11] = A[11] * B[15] + A[10] * B[11] + A[8] * B[3] + A[9] * B[7];
    C[12] = A[12] * B[0] + A[15] * B[12] + A[13] * B[4] + A[14] * B[8];
    C[13] = A[12] * B[1] + A[15] * B[13] + A[13] * B[5] + A[14] * B[9];
    C[14] = A[14] * B[10] + A[15] * B[14] + A[12] * B[2] + A[13] * B[6];
    C[15] = A[15] * B[15] + A[14] * B[11] + A[12] * B[3] + A[13] * B[7];
}
```

Motivation

- Code that makes heavy use of matrix math with small matrixes
  - Varying sizes 2x2 - 16x16
  - Varying shapes, e.g 2x2 * 2x8
  - Column Major
- Math primitives implemented in C/C++
  - Relies on SLP vectorizer (and others)
  - Relies on loop vectorizer/unroller/SLP
  - Mixing & matching leads to sub-optimal code

```
template <uint32_t N, typename T>
void MultiplyMatrix_NxN_NxN(T* C, const T* A, const T* B) {
  for (uint32_t i = 0; i < N; ++i) {
      const uint32_t A_offset = i * N;
      const uint32_t B_offset = i * N;
      const T *Aptr = &A[A_offset];
      for (uint32_t j = 0; j < N; ++j) {
          T sum = *Aptr * B[j];
          for (uint32_t k = 1; k < N; ++k) {
              sum += Aptr[k] * B[k * N + j];
          }
          C[B_offset + j] = sum;
      }
  }
}
```

Solution

Support matrix types in LLVM/Clang

 Guarantees vector code generation for operations

♻️ Better removal of unnecessary memory access

🥰 User friendly

Meet Matrix Types (Clang)

- Define a matrix type using `__attribute__((matrix_type()))`
- `+,-,*` math operators
- `[row][column]` element subscript operator
- Builtins
  - `__builtin_matrix_transpose`
  - `__builtin_matrix_column_major_store`
  - `__builtin_matrix_column_major_load`

```
typedef float m4x4_t __attribute__((matrix_type(4, 4)));

void f(float *pa, float *pb) {
  m4x4_t a = __builtin_matrix_column_major_load(pa, 4, 4, 4);
  m4x4_t b = __builtin_matrix_column_major_load(pb, 4, 4, 10);
  m4x4_t r = a * b + 10.0;
  __builtin_matrix_column_major_store(r, pa, 4);
}
```

https://clang.llvm.org/docs/MatrixTypes.html

Matrix support (LLVM)

```
typedef float m4x4_t __attribute__((matrix_type(4, 4)));

void f(float *pa, float *pb) {
  m4x4_t a = __builtin_matrix_column_major_load(pa, 4, 4, 4);
  m4x4_t b =  __builtin_matrix_column_major_load(pb, 4, 4, 10);
  m4x4_t r = a * b + 10.0;
  __builtin_matrix_column_major_store(r, pa, 4);
}
```

```
define void @f(float* %pa, float* %pb) {
  %matrix = call <16 x float> @llvm.matrix.column.major.load.v16f32.p0f32(
            float* %pa, i64 4, i64 4, i64 4)

  %matrix1 = call <16 x float> @llvm.matrix.column.major.load.v16f32.p0f32
       float* %pb, i64 10, i64 4,i64 4)

  %0 = call <16 x float> @llvm.matrix.multiply.v16f32.v16f32.v16f32(
       <16 x float> %matrix, <16 x float> %matrix1, i32 4, i32 4, i32 4)

  %1 = fadd <16 x float> %0, <float 1.000000e+01, float 1.000000e+01,
       float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
       float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
       float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
       float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
       float 1.000000e+01, float 1.000000e+01>

  call void @llvm.matrix.column.major.store.v16f32.p0f32(
       <16 x float> %1, float* %pa, i32 4, i64 4, i64 4)
  ret void
}
```

Matrix support (LLVM)

- Matrixes are embedded in flat vectors.
- Use vector instructions for element wise operations.
- Use intrinsics for shape-dependent operations.

```
define void @f(float* %pa, float* %pb) {
  %matrix = call <16 x float> @llvm.matrix.column.major.load.v16f32.p0f32(
            float* %pa, i64 4, i64 4, i64 4)

  %matrix1 = call <16 x float> @llvm.matrix.column.major.load.v16f32.p0f32
       float* %pb, i64 10, i64 4,i64 4)

  %0 = call <16 x float> @llvm.matrix.multiply.v16f32.v16f32.v16f32(
       <16 x float> %matrix, <16 x float> %matrix1, i32 4, i32 4, i32 4)

  %1 = fadd <16 x float> %0, <float 1.000000e+01, float 1.000000e+01,
       float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
       float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
       float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
       float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
       float 1.000000e+01, float 1.000000e+01, float 1.000000e+01>

  call void @llvm.matrix.column.major.store.v16f32.p0f32(
       <16 x float> %1, float* %pa, i32 4, i64 4, i64 4)
  ret void
}
```

Lowering

1. Collect & propagate shape information (number of rows/ columns).

2. Lower instructions with shape information to operations on column vectors.

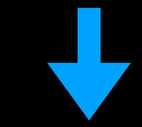
```
define void @f(float* %pa) {
4x4  %matrix = call <16 x float> @llvm.matrix.column.major.load.v16f32.p0f32(
         float* %pa, i64 4, i1 false, i64 4, i64 4)

4x4  %0 = call <16 x float> @llvm.matrix.multiply.v16f32.v16f32.v16f32(
         <16 x float> %matrix, <16 x float> %matrix, i32 4, i32 4, i32 4)

4x4  %1 = fadd <16 x float> %0, <float 1.000000e+01, float 1.000000e+01,
         float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
         float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
         float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
         float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
         float 1.000000e+01, float 1.000000e+01>

  call void @llvm.matrix.column.major.store.v16f32.p0f32(
         <16 x float> %1, float* %pa, i64 4, i1 false, i64 4, i64 4)

  ret void
}
```

Lowering

1. Collect & propagate shape information (number of rows/ columns).

2. Lower instructions with shape information to operations on column vectors.

```
%matrix = call <16 x float> @llvm.matrix.column.major.load.v16f32.p0f32(
     float* %pa, i64 4, i1 false, i64 4, i64 4)
```



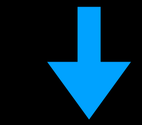
```
%vec.cast = bitcast float* %pa to <4 x float>*
%col.load = load <4 x float>, <4 x float>* %vec.cast, align 4
%vec.gep = getelementptr float, float* %pa, i64 4
%vec.cast1 = bitcast float* %vec.gep to <4 x float>*
%col.load2 = load <4 x float>, <4 x float>* %vec.cast1, align 4
%vec.gep3 = getelementptr float, float* %pa, i64 8
%vec.cast4 = bitcast float* %vec.gep3 to <4 x float>*
%col.load5 = load <4 x float>, <4 x float>* %vec.cast4, align 4
%vec.gep6 = getelementptr float, float* %pa, i64 12
%vec.cast7 = bitcast float* %vec.gep6 to <4 x float>*
%col.load8 = load <4 x float>, <4 x float>* %vec.cast7, align 4
```

Lowering

1. Collect & propagate shape information (number of rows/ columns).

2. Lower instructions with shape information to operations on column vectors.

```
%0 = call <16 x float> @llvm.matrix.multiply.v16f32.v16f32.v16f32(
      <16 x float> %matrix, <16 x float> %matrix, i32 4, i32 4, i32 4)
```



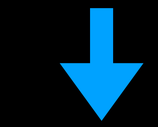
```
%0 = fmul <4 x float> %col.load, %splat.splat
%splat.splat11 = shufflevector <4 x float> %col.load, <4 x float> undef,
  <4 x i32> <i32 1, i32 1, i32 1, i32 1>
%1 = call <4 x float> @llvm.fmuladd.v4f32(
  <4 x float> %col.load2, <4 x float> %splat.splat11, <4 x float> %0)
%splat.splat14 = shufflevector <4 x float> %col.load, <4 x float> undef,
  <4 x i32> <i32 2, i32 2, i32 2, i32 2>
%2 = call <4 x float> @llvm.fmuladd.v4f32(
  <4 x float> %col.load5, <4 x float> %splat.splat14, <4 x float> %1)
%splat.splat17 = shufflevector <4 x float> %col.load, <4 x float> undef,
  <4 x i32> <i32 3, i32 3, i32 3, i32 3>
%3 = call <4 x float> @llvm.fmuladd.v4f32(
  <4 x float> %col.load8, <4 x float> %splat.splat17, <4 x float> %2)
%splat.splat20 = shufflevector <4 x float> %col.load2, <4 x float> undef,
  <4 x i32> zeroinitializer
%4 = fmul <4 x float> %col.load, %splat.splat20
```

Lowering

1. Collect & propagate shape information (number of rows/ columns).

2. Lower instructions with shape information to operations on column vectors.

```
%1 = fadd <16 x float> %0, <float 1.000000e+01, float 1.000000e+01,
     float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
     float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
     float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
     float 1.000000e+01, float 1.000000e+01, float 1.000000e+01,
     float 1.000000e+01, float 1.000000e+01>
```



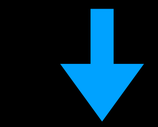
```
%16 = fadd <4 x float> %3, <float 1.000000e+01, float 1.000000e+01,
                                float 1.000000e+01, float 1.000000e+01>
%17 = fadd <4 x float> %7, <float 1.000000e+01, float 1.000000e+01,
                                float 1.000000e+01, float 1.000000e+01>
%18 = fadd <4 x float> %11, <float 1.000000e+01, float 1.000000e+01,
                                 float 1.000000e+01, float 1.000000e+01>
%19 = fadd <4 x float> %15, <float 1.000000e+01, float 1.000000e+01,
                                 float 1.000000e+01, float 1.000000e+01>
```

Lowering

1. Collect & propagate shape information (number of rows/ columns).

2. Lower instructions with shape information to operations on column vectors.

```
call void @llvm.matrix.column.major.store.v16f32.p0f32(
    <16 x float> %1, float* %pa, i64 4, i1 false, i64 4, i64 4)
```



```
%vec.cast54 = bitcast float* %pa to <4 x float>*
store <4 x float> %16, <4 x float>* %vec.cast54, align 4
%vec.gep55 = getelementptr float, float* %pa, i64 4
%vec.cast56 = bitcast float* %vec.gep55 to <4 x float>*
store <4 x float> %17, <4 x float>* %vec.cast56, align 4
%vec.gep57 = getelementptr float, float* %pa, i64 8
%vec.cast58 = bitcast float* %vec.gep57 to <4 x float>*
store <4 x float> %18, <4 x float>* %vec.cast58, align 4
%vec.gep59 = getelementptr float, float* %pa, i64 12
%vec.cast60 = bitcast float* %vec.gep59 to <4 x float>*
store <4 x float> %19, <4 x float>* %vec.cast60, align 4
ret void
```

Lowering

Also supported:

- Tiled code generation for matrix multiplies.
- Codegen for hardware extensions (e.g. AArch64 `udot`).

```
rows.header:                                      ; preds = %rows.latch, %cols.header
  %rows.iv = phi i64 [ 0, %cols.header ], [ %rows.step, %rows.latch ]
  br label %inner.header

inner.header:                                     ; preds = %inner.header, %rows.header
  %inner.iv = phi i64 [ 0, %rows.header ], [ %inner.step, %inner.header ]
  %12 = phi <2 x double> [ zeroinitializer, %rows.header ], [ %21, %inner.header ]
  %13 = phi <2 x double> [ zeroinitializer, %rows.header ], [ %23, %inner.header ]
  %14 = shl i64 %inner.iv, 1
  %15 = add i64 %14, %rows.iv
  %16 = getelementptr <4 x double>, <4 x double>* %5, i64 0, i64 %15
  %vec.cast = bitcast double* %16 to <2 x double>*
  %col.load = load <2 x double>, <2 x double>* %vec.cast, align 8
  %vec.gep = getelementptr double, double* %16, i64 2
  %vec.cast8 = bitcast double* %vec.gep to <2 x double>*
  %col.load9 = load <2 x double>, <2 x double>* %vec.cast8, align 8
  %17 = shl i64 %cols.iv, 1
  %18 = add i64 %17, %inner.iv
  %19 = getelementptr <4 x double>, <4 x double>* %11, i64 0, i64 %18
  %vec.cast11 = bitcast double* %19 to <2 x double>*
  %col.load12 = load <2 x double>, <2 x double>* %vec.cast11, align 8
  %vec.gep13 = getelementptr double, double* %19, i64 2
  %vec.cast14 = bitcast double* %vec.gep13 to <2 x double>*
  %col.load15 = load <2 x double>, <2 x double>* %vec.cast14, align 8
  %splat.splat = shufflevector <2 x double> %col.load12, <2 x double> undef, <2 x i32> zeroinitializer
  %20 = call <2 x double> @llvm.fmuladd.v2f64(<2 x double> %col.load, <2 x double> %splat.splat, <2 x double> %12)
  %splat.splat19 = shufflevector <2 x double> %col.load12, <2 x double> undef, <2 x i32> <i32 1, i32 1>
  %21 = call <2 x double> @llvm.fmuladd.v2f64(<2 x double> %col.load9, <2 x double> %splat.splat19, <2 x double> %20)
  %splat.splat23 = shufflevector <2 x double> %col.load15, <2 x double> undef, <2 x i32> zeroinitializer
  %22 = call <2 x double> @llvm.fmuladd.v2f64(<2 x double> %col.load, <2 x double> %splat.splat23, <2 x double> %13)
  %splat.splat26 = shufflevector <2 x double> %col.load15, <2 x double> undef, <2 x i32> <i32 1, i32 1>
  %23 = call <2 x double> @llvm.fmuladd.v2f64(<2 x double> %col.load9, <2 x double> %splat.splat26, <2 x double> %22)
  %inner.step = add i64 %inner.iv, 2
  %inner.cond = icmp eq i64 %inner.iv, 0
  br i1 %inner.cond, label %rows.latch, label %inner.header

rows.latch:                                       ; preds = %inner.header
  %rows.step = add i64 %rows.iv, 2
  %rows.cond = icmp eq i64 %rows.iv, 0
  %24 = shl i64 %cols.iv, 1
  %25 = add i64 %24, %rows.iv
  %26 = getelementptr <4 x double>, <4 x double>* %C, i64 0, i64 %25
  %vec.cast28 = bitcast double* %26 to <2 x double>*
  store <2 x double> %21, <2 x double>* %vec.cast28, align 8
  %vec.gep29 = getelementptr double, double* %26, i64 2
  %vec.cast30 = bitcast double* %vec.gep29 to <2 x double>*
  store <2 x double> %23, <2 x double>* %vec.cast30, align 8
  br i1 %rows.cond, label %cols.latch, label %rows.header
```

Lowering

Also supported:

- Tiled code generation for matrix multiplies.
- Codegen for hardware extensions (e.g. AArch64 `udot`).

```
%vec.cast = bitcast <16 x i8>* %A to <4 x i8>*
 %col.load = load <4 x i8>, <4 x i8>* %vec.cast, align 16
 %vec.gep = getelementptr <16 x i8>, <16 x i8>* %A, i64 0, i64 4
 %vec.cast1 = bitcast i8* %vec.gep to <4 x i8>*
 %col.load2 = load <4 x i8>, <4 x i8>* %vec.cast1, align 4
 %vec.gep3 = getelementptr <16 x i8>, <16 x i8>* %A, i64 0, i64 8
 %vec.cast4 = bitcast i8* %vec.gep3 to <4 x i8>*
 %col.load5 = load <4 x i8>, <4 x i8>* %vec.cast4, align 8
 %vec.gep6 = getelementptr <16 x i8>, <16 x i8>* %A, i64 0, i64 12
 %vec.cast7 = bitcast i8* %vec.gep6 to <4 x i8>*
 %col.load8 = load <4 x i8>, <4 x i8>* %vec.cast7, align 4
 %vec.cast10 = bitcast <16 x i8>* %B to <4 x i8>*
 %col.load11 = load <4 x i8>, <4 x i8>* %vec.cast10, align 1
 %vec.gep12 = getelementptr <16 x i8>, <16 x i8>* %B, i64 0, i64 4
 %vec.cast13 = bitcast i8* %vec.gep12 to <4 x i8>*
 %col.load14 = load <4 x i8>, <4 x i8>* %vec.cast13, align 1
 %vec.gep15 = getelementptr <16 x i8>, <16 x i8>* %B, i64 0, i64 8
 %vec.cast16 = bitcast i8* %vec.gep15 to <4 x i8>*
 %col.load17 = load <4 x i8>, <4 x i8>* %vec.cast16, align 1
 %vec.gep18 = getelementptr <16 x i8>, <16 x i8>* %B, i64 0, i64 12
 %vec.cast19 = bitcast i8* %vec.gep18 to <4 x i8>*
 %col.load20 = load <4 x i8>, <4 x i8>* %vec.cast19, align 1
 %0 = shufflevector <4 x i8> %col.load, <4 x i8> %col.load2, <8 x i32> <i32 0, i32 1, i32 2, i
 %1 = shufflevector <4 x i8> %col.load5, <4 x i8> %col.load8, <8 x i32> <i32 0, i32 1, i32 2,
 %2 = shufflevector <8 x i8> %0, <8 x i8> %1, <16 x i32> <i32 0, i32 1, i32 2, i32 3, i32 4, i
i32 14, i32 15>
 %3 = shufflevector <4 x i8> %col.load11, <4 x i8> undef, <8 x i32> <i32 0, i32 1, i32 2, i32
 %4 = shufflevector <8 x i8> %3, <8 x i8> undef, <16 x i32> <i32 0, i32 1, i32 2, i32 3, i32 4
i32 6, i32 7>
 %5 = tail call <4 x i32> @llvm.aarch64.neon.udot.v4i32.v16i8(<4 x i32> zeroinitializer, <16 x
 %6 = shufflevector <4 x i8> %col.load14, <4 x i8> undef, <8 x i32> <i32 0, i32 1, i32 2, i32
 %7 = shufflevector <8 x i8> %6, <8 x i8> undef, <16 x i32> <i32 0, i32 1, i32 2, i32 3, i32 4
i32 6, i32 7>
 %8 = tail call <4 x i32> @llvm.aarch64.neon.udot.v4i32.v16i8(<4 x i32> zeroinitializer, <16 x
 %9 = shufflevector <4 x i8> %col.load17, <4 x i8> undef, <8 x i32> <i32 0, i32 1, i32 2, i32
 %10 = shufflevector <8 x i8> %9, <8 x i8> undef, <16 x i32> <i32 0, i32 1, i32 2, i32 3, i32
i32 6, i32 7>
 %11 = tail call <4 x i32> @llvm.aarch64.neon.udot.v4i32.v16i8(<4 x i32> zeroinitializer, <16
 %12 = shufflevector <4 x i8> %col.load20, <4 x i8> undef, <8 x i32> <i32 0, i32 1, i32 2, i32
 %13 = shufflevector <8 x i8> %12, <8 x i8> undef, <16 x i32> <i32 0, i32 1, i32 2, i32 3, i32
i32 6, i32 7>
 %14 = tail call <4 x i32> @llvm.aarch64.neon.udot.v4i32.v16i8(<4 x i32> zeroinitializer, <16
```

Original Proposal

- Add LLVM IR matrix type.
- Use intrinsics for all operations.

✅ Less intrinsic arguments.

🚫 Teach various places about matrix types (e.g. SROA).

🚫 Thread type through many places including LL parser, bitcode reader/writer, instructions.

🚫 Matrix constants.

```
define void @f(float* %pa, float* %pb) {
  %matrix = call <4 x 4 x float> @llvm.matrix.columnwise.load.m4x4f32.p0f32(
            float* %pa, i64 4, i64 4, i64 4)

  %matrix1 = call <4 x 4 x float> @llvm.matrix.columnwise.load.m4x4f32.p0f32(
       float* %pb, i64 10, i64 4,i64 4)

  %0 = call <4 x 4 x float> @llvm.matrix.multiply.m4x4f32.m4x4f32.m4x4f32(
      <4 x 4 x float> %matrix, <4 x 4 x float> %matrix1)

  %1 = call <4 x 4 x float> @llvm.matrix.add.m4x4f32.m4x4f32.m4x4f32(
       <4 x 4 x float> %matrix, <4 x 4 x float> %matrix1)

  call void @llvm.matrix.columnwise.store.m4x4f32.p0f32(
       <4 x 4 x float> %1, float* %pa, i32 4)
  ret void
}
```

Open Sourcing

First version of RFC proposed a matrix IR type (2018).

🍾 Updated RFC using vectors & intrinsics (2019).

🍾 Clang RFC with complete draft spec for matrix types extensions (2020).

Performance

```
// Eigen Version:
//  MatrixTy = Eigen::Matrix<FloatTy, R, C>
// Matrix types version:
//  MatrixTy = FloatTy __attribute__((matrix_type(R, C)));

template <typename MatrixTy>
void bench(MatrixTy &A, MatrixTy &B, MatrixTy &C) {
  C += A * B;
}
```

Size	MT exec_time
3x3 float	20.67%
3x3 double	-16.91%
4x4 float	-0.12%
4x4 double	-23.72%
8x8 float	-33.69%
8x8 double	-14.38%
16x16 float	-74.08%
16x16 double	-67.01%

Runtime change of Matrix Type version compared to Eigen, on ARM64

Performance

```
// Eigen Version:
//  MatrixTy = Eigen::Matrix<FloatTy, R, C>
// Matrix types version:
//  MatrixTy = FloatTy __attribute__((matrix_type(R, C)));

template <typename MatrixTy>
void bench(MatrixTy &A, MatrixTy &B, MatrixTy &C) {
  C += A * B;
}
```

Size	MT exec_time
3x3 float	20.67%
3x3 double	-16.91%
4x4 float	-0.12%
4x4 double	-23.72%
8x8 float	-33.69%
8x8 double	-14.38%
16x16 float	-74.08%
16x16 double	-67.01%

Runtime change of Matrix Type version compared to Eigen, on ARM64

Performance

```
// Eigen Version:
//  MatrixTy = Eigen::Matrix<FloatTy, R, C>
// Matrix types version:
//  MatrixTy = FloatTy __attribute__((matrix_type(R, C)));

template <typename MatrixTy>
void bench(MatrixTy &A, MatrixTy &B, MatrixTy &C) {
  C += A * B;
}
```

Size	MT exec_time
3x3 float	20.67%
3x3 double	-16.91%
4x4 float	-0.12%
4x4 double	-23.72%
8x8 float	-33.69%
8x8 double	-14.38%
16x16 float	-74.08%
16x16 double	-67.01%

Runtime change of Matrix Type version compared to Eigen, on ARM64

Performance

```
// Eigen Version:
//  MatrixTy = Eigen::Matrix<FloatTy, R, C>
// Matrix types version:
//  MatrixTy = FloatTy __attribute__((matrix_type(R, C)));

template <typename MatrixTy>
void bench(MatrixTy &A, MatrixTy &B, MatrixTy &C) {
  C += A * B;
}
```

The goal is not to replace specialized libraries, but to give authors an additional set of tools!

Size	MT exec_time
3x3 float	20.67%
3x3 double	-16.91%
4x4 float	-0.12%
4x4 double	-23.72%
8x8 float	-33.69%
8x8 double	-14.38%
16x16 float	-74.08%
16x16 double	-67.01%

Runtime change of Matrix Type version compared to Eigen, on ARM64

Performance

```
// Eigen Version:
//  MatrixTy = Eigen::Matrix<FloatTy, R, C>
// Matrix types version:
//  MatrixTy = FloatTy __attribute__((matrix_type(R, C)));

template <typename MatrixTy>
void bench(MatrixTy &A, MatrixTy &B, MatrixTy &C) {
  C += A * B;
}
```

The goal is not to replace specialized libraries, but to give authors an additional set of tools!

Size	MT exec_time
3x3 float	20.67%
3x3 double	-16.91%
4x4 float	-0.12%
4x4 double	-23.72%
8x8 float	-33.69%
8x8 double	-14.38%
16x16 float	-74.08%
16x16 double	-67.01%

Runtime change of Matrix Type version compared to Eigen, on ARM64

CodeGen double 4x4

Matrix Types

```
ldp q0, q4, [x20]
ldp q1, q5, [x20, #32]
ldp q2, q6, [x20, #64]
ldp q3, q7, [x20, #96]
ldp q20, q16, [x22]
ldp q21, q17, [x22, #32]
ldp q22, q18, [x22, #64]
ldp q23, q19, [x22, #96]
ldp q25, q24, [x21]
ldp q27, q26, [x21, #32]
ldp q29, q28, [x21, #64]
ldp q31, q30, [x21, #96]
fmul.2d v8, v20, v25[0]
fmla.2d v8, v21, v25[1]
fmla.2d v8, v22, v24[0]
fmla.2d v8, v23, v24[1]
fmul.2d v9, v16, v25[0]
fmla.2d v9, v17, v25[1]
fmla.2d v9, v18, v24[0]
fmla.2d v9, v19, v24[1]
fmul.2d v24, v20, v27[0]
fmla.2d v24, v21, v27[1]
fmla.2d v24, v22, v26[0]
fmla.2d v24, v23, v26[1]
fmul.2d v25, v16, v27[0]
fmla.2d v25, v17, v27[1]
fmla.2d v25, v18, v26[0]
fmla.2d v25, v19, v26[1]
fmul.2d v26, v20, v29[0]
fmla.2d v26, v21, v29[1]
fmla.2d v26, v22, v28[0]
fmla.2d v26, v23, v28[1]
fmul.2d v27, v16, v29[0]
fmla.2d v27, v17, v29[1]
fmla.2d v27, v18, v28[0]
fmla.2d v27, v19, v28[1]
fmul.2d v20, v20, v31[0]
fmla.2d v20, v21, v31[1]
fmla.2d v20, v22, v30[0]
fmla.2d v20, v23, v30[1]
fmul.2d v16, v16, v31[0]
fmla.2d v16, v17, v31[1]
fmla.2d v16, v18, v30[0]
fmla.2d v16, v19, v30[1]
fadd.2d v4, v9, v4
fadd.2d v0, v8, v0
fadd.2d v5, v25, v5
fadd.2d v1, v24, v1
fadd.2d v6, v27, v6
fadd.2d v2, v26, v2
fadd.2d v7, v16, v7
fadd.2d v3, v20, v3
stp q0, q4, [x20]
stp q1, q5, [x20, #32]
stp q2, q6, [x20, #64]
stp q3, q7, [x20, #96]
```

CodeGen double 4x4

Matrix Types

```
ldp q0, q4, [x20]
ldp q1, q5, [x20, #32]
ldp q2, q6, [x20, #64]
ldp q3, q7, [x20, #96]
ldp q20, q16, [x22]
ldp q21, q17, [x22, #32]
ldp q22, q18, [x22, #64]
ldp q23, q19, [x22, #96]
ldp q25, q24, [x21]
ldp q27, q26, [x21, #32]
ldp q29, q28, [x21, #64]
ldp q31, q30, [x21, #96]
fmul.2d v8, v20, v25[0]
fmla.2d v8, v21, v25[1]
fmla.2d v8, v22, v24[0]
fmla.2d v8, v23, v24[1]
fmul.2d v9, v16, v25[0]
fmla.2d v9, v17, v25[1]
fmla.2d v9, v18, v24[0]
fmla.2d v9, v19, v24[1]
fmul.2d v24, v20, v27[0]
fmla.2d v24, v21, v27[1]
fmla.2d v24, v22, v26[0]
fmla.2d v24, v23, v26[1]
fmul.2d v25, v16, v27[0]
fmla.2d v25, v17, v27[1]
fmla.2d v25, v18, v26[0]
fmla.2d v25, v19, v26[1]
```

```
fmul.2d v26, v20, v29[0]
fmla.2d v26, v21, v29[1]
fmla.2d v26, v22, v28[0]
fmla.2d v26, v23, v28[1]
fmul.2d v27, v16, v29[0]
fmla.2d v27, v17, v29[1]
fmla.2d v27, v18, v28[0]
fmla.2d v27, v19, v28[1]
fmul.2d v20, v20, v31[0]
fmla.2d v20, v21, v31[1]
fmla.2d v20, v22, v30[0]
fmla.2d v20, v23, v30[1]
fmul.2d v16, v16, v31[0]
fmla.2d v16, v17, v31[1]
fmla.2d v16, v18, v30[0]
fmla.2d v16, v19, v30[1]
fadd.2d v4, v9, v4
fadd.2d v0, v8, v0
fadd.2d v5, v25, v5
fadd.2d v1, v24, v1
fadd.2d v6, v27, v6
fadd.2d v2, v26, v2
fadd.2d v7, v16, v7
fadd.2d v3, v20, v3
stp q0, q4, [x20]
stp q1, q5, [x20, #32]
stp q2, q6, [x20, #64]
stp q3, q7, [x20, #96]
```

CodeGen double 4x4

Matrix Types

```
ldp q0, q4, [x20]
ldp q1, q5, [x20, #32]
ldp q2, q6, [x20, #64]
ldp q3, q7, [x20, #96]
ldp q20, q16, [x22]
ldp q21, q17, [x22, #32]
ldp q22, q18, [x22, #64]
ldp q23, q19, [x22, #96]
ldp q25, q24, [x21]
ldp q27, q26, [x21, #32]
ldp q29, q28, [x21, #64]
ldp q31, q30, [x21, #96]
fmul.2d v8, v20, v25[0]
fmla.2d v8, v21, v25[1]
fmla.2d v8, v22, v24[0]
fmla.2d v8, v23, v24[1]
fmul.2d v9, v16, v25[0]
fmla.2d v9, v17, v25[1]
fmla.2d v9, v18, v24[0]
fmla.2d v9, v19, v24[1]
fmul.2d v24, v20, v27[0]
fmla.2d v24, v21, v27[1]
fmla.2d v24, v22, v26[0]
fmla.2d v24, v23, v26[1]
fmul.2d v25, v16, v27[0]
fmla.2d v25, v17, v27[1]
fmla.2d v25, v18, v26[0]
fmla.2d v25, v19, v26[1]
fmul.2d v26, v20, v29[0]
fmla.2d v26, v21, v29[1]
fmla.2d v26, v22, v28[0]
fmla.2d v26, v23, v28[1]
fmul.2d v27, v16, v29[0]
fmla.2d v27, v17, v29[1]
fmla.2d v27, v18, v28[0]
fmla.2d v27, v19, v28[1]
fmul.2d v20, v20, v31[0]
fmla.2d v20, v21, v31[1]
fmla.2d v20, v22, v30[0]
fmla.2d v20, v23, v30[1]
fmul.2d v16, v16, v31[0]
fmla.2d v16, v17, v31[1]
fmla.2d v16, v18, v30[0]
fmla.2d v16, v19, v30[1]
fadd.2d v4, v9, v4
fadd.2d v0, v8, v0
fadd.2d v5, v25, v5
fadd.2d v1, v24, v1
fadd.2d v6, v27, v6
fadd.2d v2, v26, v2
fadd.2d v7, v16, v7
fadd.2d v3, v20, v3
stp q0, q4, [x20]
stp q1, q5, [x20, #32]
stp q2, q6, [x20, #64]
stp q3, q7, [x20, #96]
```

CodeGen double 4x4

Matrix Types

```
ldp q0, q4, [x20]
ldp q1, q5, [x20, #32]
ldp q2, q6, [x20, #64]
ldp q3, q7, [x20, #96]
ldp q20, q16, [x22]
ldp q21, q17, [x22, #32]
ldp q22, q18, [x22, #64]
ldp q23, q19, [x22, #96]
ldp q25, q24, [x21]
ldp q27, q26, [x21, #32]
ldp q29, q28, [x21, #64]
ldp q31, q30, [x21, #96]
fmul.2d v8, v20, v25[0]
fmla.2d v8, v21, v25[1]
fmla.2d v8, v22, v24[0]
fmla.2d v8, v23, v24[1]
fmul.2d v9, v16, v25[0]
fmla.2d v9, v17, v25[1]
fmla.2d v9, v18, v24[0]
fmla.2d v9, v19, v24[1]
fmul.2d v24, v20, v27[0]
fmla.2d v24, v21, v27[1]
fmla.2d v24, v22, v26[0]
fmla.2d v24, v23, v26[1]
fmul.2d v25, v16, v27[0]
fmla.2d v25, v17, v27[1]
fmla.2d v25, v18, v26[0]
fmla.2d v25, v19, v26[1]
```

```
fmul.2d v26, v20, v29[0]
fmla.2d v26, v21, v29[1]
fmla.2d v26, v22, v28[0]
fmla.2d v26, v23, v28[1]
fmul.2d v27, v16, v29[0]
fmla.2d v27, v17, v29[1]
fmla.2d v27, v18, v28[0]
fmla.2d v27, v19, v28[1]
fmul.2d v20, v20, v31[0]
fmla.2d v20, v21, v31[1]
fmla.2d v20, v22, v30[0]
fmla.2d v20, v23, v30[1]
fmul.2d v16, v16, v31[0]
fmla.2d v16, v17, v31[1]
fmla.2d v16, v18, v30[0]
fmla.2d v16, v19, v30[1]
fadd.2d v4, v9, v4
fadd.2d v0, v8, v0
fadd.2d v5, v25, v5
fadd.2d v1, v24, v1
fadd.2d v6, v27, v6
fadd.2d v2, v26, v2
fadd.2d v7, v16, v7
fadd.2d v3, v20, v3
stp q0, q4, [x20]
stp q1, q5, [x20, #32]
stp q2, q6, [x20, #64]
stp q3, q7, [x20, #96]
```

Eigen

```
    ldp q0, q1, [sp, #256]
    ldp q2, q3, [sp, #288]
    ldp q4, q5, [sp, #320]
    ldp q6, q7, [sp, #352]

LBB1_4:
    add x13, x8, x12
    mov x14, x13
    ld1r.2d { v16 }, [x14], #8
    fmul.2d v17, v16, v0
    ldr d18, [x14]
    ldp d19, d20, [x13, #16]
    fmla.2d v17, v2, v18[0]
    fmla.2d v17, v4, v19[0]
    add x13, x9, x12
    fmul.2d v16, v16, v1
    fmla.2d v16, v3, v18[0]
    fmla.2d v17, v6, v20[0]
    fmla.2d v16, v5, v19[0]
    fmla.2d v16, v7, v20[0]
    stp q17, q16, [x13]
    add x12, x12, #32           ; =32
    cmp x12, #128               ; =128
    b.ne    LBB1_4
```

```
ldp q0, q1, [x29, #-176]
ldp q2, q3, [sp]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp]
ldp q0, q1, [x29, #-144]
ldp q2, q3, [sp, #32]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp, #32]
ldp q0, q1, [x29, #-112]
ldp q2, q3, [sp, #64]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp, #64]
ldp q0, q1, [x29, #-80]
ldp q2, q3, [sp, #96]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp, #96]
```

CodeGen double 4x4

Matrix Types

```
ldp q0, q4, [x20]
ldp q1, q5, [x20, #32]
ldp q2, q6, [x20, #64]
ldp q3, q7, [x20, #96]
ldp q20, q16, [x22]
ldp q21, q17, [x22, #32]
ldp q22, q18, [x22, #64]
ldp q23, q19, [x22, #96]
ldp q25, q24, [x21]
ldp q27, q26, [x21, #32]
ldp q29, q28, [x21, #64]
ldp q31, q30, [x21, #96]
fmul.2d v8, v20, v25[0]
fmla.2d v8, v21, v25[1]
fmla.2d v8, v22, v24[0]
fmla.2d v8, v23, v24[1]
fmul.2d v9, v16, v25[0]
fmla.2d v9, v17, v25[1]
fmla.2d v9, v18, v24[0]
fmla.2d v9, v19, v24[1]
fmul.2d v24, v20, v27[0]
fmla.2d v24, v21, v27[1]
fmla.2d v24, v22, v26[0]
fmla.2d v24, v23, v26[1]
fmul.2d v25, v16, v27[0]
fmla.2d v25, v17, v27[1]
fmla.2d v25, v18, v26[0]
fmla.2d v25, v19, v26[1]
```

```
fmul.2d v26, v20, v29[0]
fmla.2d v26, v21, v29[1]
fmla.2d v26, v22, v28[0]
fmla.2d v26, v23, v28[1]
fmul.2d v27, v16, v29[0]
fmla.2d v27, v17, v29[1]
fmla.2d v27, v18, v28[0]
fmla.2d v27, v19, v28[1]
fmul.2d v20, v20, v31[0]
fmla.2d v20, v21, v31[1]
fmla.2d v20, v22, v30[0]
fmla.2d v20, v23, v30[1]
fmul.2d v16, v16, v31[0]
fmla.2d v16, v17, v31[1]
fmla.2d v16, v18, v30[0]
fmla.2d v16, v19, v30[1]
fadd.2d v4, v9, v4
fadd.2d v0, v8, v0
fadd.2d v5, v25, v5
fadd.2d v1, v24, v1
fadd.2d v6, v27, v6
fadd.2d v2, v26, v2
fadd.2d v7, v16, v7
fadd.2d v3, v20, v3
stp q0, q4, [x20]
stp q1, q5, [x20, #32]
stp q2, q6, [x20, #64]
stp q3, q7, [x20, #96]
```

Eigen

```
ldp q0, q1, [sp, #256]
ldp q2, q3, [sp, #288]
ldp q4, q5, [sp, #320]
ldp q6, q7, [sp, #352]

LBB1_4:
    add x13, x8, x12
    mov x14, x13
    ld1r.2d { v16 }, [x14], #8
    fmul.2d v17, v16, v0
    ldr d18, [x14]
    ldp d19, d20, [x13, #16]
    fmla.2d v17, v2, v18[0]
    fmla.2d v17, v4, v19[0]
    add x13, x9, x12
    fmul.2d v16, v16, v1
    fmla.2d v16, v3, v18[0]
    fmla.2d v17, v6, v20[0]
    fmla.2d v16, v5, v19[0]
    fmla.2d v16, v7, v20[0]
    stp q17, q16, [x13]
    add x12, x12, #32           ; =32
    cmp x12, #128               ; =128
    b.ne    LBB1_4
```

🛑 Single vector loads

```
ldp q0, q1, [x29, #-176]
ldp q2, q3, [sp]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp]
ldp q0, q1, [x29, #-144]
ldp q2, q3, [sp, #32]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp, #32]
ldp q0, q1, [x29, #-112]
ldp q2, q3, [sp, #64]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp, #64]
ldp q0, q1, [x29, #-80]
ldp q2, q3, [sp, #96]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp, #96]
```

CodeGen double 4x4

Matrix Types

```
ldp q0, q4, [x20]
ldp q1, q5, [x20, #32]
ldp q2, q6, [x20, #64]
ldp q3, q7, [x20, #96]
ldp q20, q16, [x22]
ldp q21, q17, [x22, #32]
ldp q22, q18, [x22, #64]
ldp q23, q19, [x22, #96]
ldp q25, q24, [x21]
ldp q27, q26, [x21, #32]
ldp q29, q28, [x21, #64]
ldp q31, q30, [x21, #96]
fmul.2d v8, v20, v25[0]
fmla.2d v8, v21, v25[1]
fmla.2d v8, v22, v24[0]
fmla.2d v8, v23, v24[1]
fmul.2d v9, v16, v25[0]
fmla.2d v9, v17, v25[1]
fmla.2d v9, v18, v24[0]
fmla.2d v9, v19, v24[1]
fmul.2d v24, v20, v27[0]
fmla.2d v24, v21, v27[1]
fmla.2d v24, v22, v26[0]
fmla.2d v24, v23, v26[1]
fmul.2d v25, v16, v27[0]
fmla.2d v25, v17, v27[1]
fmla.2d v25, v18, v26[0]
fmla.2d v25, v19, v26[1]
```

```
fmul.2d v26, v20, v29[0]
fmla.2d v26, v21, v29[1]
fmla.2d v26, v22, v28[0]
fmla.2d v26, v23, v28[1]
fmul.2d v27, v16, v29[0]
fmla.2d v27, v17, v29[1]
fmla.2d v27, v18, v28[0]
fmla.2d v27, v19, v28[1]
fmul.2d v20, v20, v31[0]
fmla.2d v20, v21, v31[1]
fmla.2d v20, v22, v30[0]
fmla.2d v20, v23, v30[1]
fmul.2d v16, v16, v31[0]
fmla.2d v16, v17, v31[1]
fmla.2d v16, v18, v30[0]
fmla.2d v16, v19, v30[1]
fadd.2d v4, v9, v4
fadd.2d v0, v8, v0
fadd.2d v5, v25, v5
fadd.2d v1, v24, v1
fadd.2d v6, v27, v6
fadd.2d v2, v26, v2
fadd.2d v7, v16, v7
fadd.2d v3, v20, v3
stp q0, q4, [x20]
stp q1, q5, [x20, #32]
stp q2, q6, [x20, #64]
stp q3, q7, [x20, #96]
```

Eigen

```
    ldp q0, q1, [sp, #256]
    ldp q2, q3, [sp, #288]
    ldp q4, q5, [sp, #320]
    ldp q6, q7, [sp, #352]

LBB1_4:
    add x13, x8, x12
    mov x14, x13
    ld1r.2d { v16 }, [x14], #8
    fmul.2d v17, v16, v0
    ldr d18, [x14]
    ldp d19, d20, [x13, #16]
    fmla.2d v17, v2, v18[0]
    fmla.2d v17, v4, v19[0]
    add x13, x9, x12
    fmul.2d v16, v16, v1
    fmla.2d v16, v3, v18[0]
    fmla.2d v17, v6, v20[0]
    fmla.2d v16, v5, v19[0]
    fmla.2d v16, v7, v20[0]
    stp q17, q16, [x13]
    add x12, x12, #32          ; =32
    cmp x12, #128              ; =128
    b.ne    LBB1_4
```

```
ldp q0, q1, [x29, #-176]
ldp q2, q3, [sp]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp]
ldp q0, q1, [x29, #-144]
ldp q2, q3, [sp, #32]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp, #32]
ldp q0, q1, [x29, #-112]
ldp q2, q3, [sp, #64]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp, #64]
ldp q0, q1, [x29, #-80]
ldp q2, q3, [sp, #96]
fadd.2d v0, v2, v0
fadd.2d v1, v3, v1
stp q0, q1, [sp, #96]
```

- Single vector loads
- Additional memory stores & reloads

Performance

```
// Eigen Version:
//  MatrixTy = Eigen::Matrix<FloatTy, R, C>
// Matrix types version:
//  MatrixTy = FloatTy __attribute__((matrix_type(R, C)));

template <typename MatrixTy>
void bench(
  MatrixTy &A, MatrixTy &B, MatrixTy &C, MatrixTy &D,
  MatrixTy &E) {
  E = D.transpose() * ((E + D) + A * B);
}
```

Size	MT exec_time
3x3 float	-49.67%
3x3 double	-57.31%
4x4 float	-50.30%
4x4 double	-60.87%
8x8 float	-35.82%
8x8 double	-73.44%
16x16 float	-67.31%
16x16 double	-49.35%

Runtime change of Matrix Type version compared to Eigen, on ARM64

Performance

```
// Eigen Version:
//  MatrixTy = Eigen::Matrix<FloatTy, R, C>
// Matrix types version:
//  MatrixTy = FloatTy __attribute__((matrix_type(R, C)));

template <typename MatrixTy>
void bench(
  MatrixTy &A, MatrixTy &B, MatrixTy &C, MatrixTy &D,
  MatrixTy &E) {
  E = D.transpose() * ((E + D) + A * B);
}
```

Size	MT exec_time
3x3 float	-49.67%
3x3 double	-57.31%
4x4 float	-50.30%
4x4 double	-60.87%
8x8 float	-35.82%
8x8 double	-73.44%
16x16 float	-67.31%
16x16 double	-49.35%

Runtime change of Matrix Type version compared to Eigen, on ARM64

CodeGen float 4x4

Eigen

Matrix Types

```
ldp q4, q5, [x22]
ldp q6, q7, [x22, #32]
zip1.4s v0, v4, v5
mov.s   v0[2], v6[0]
mov.s   v0[3], v7[0]
trn2.4s v2, v4, v5
ext.16b v1, v2, v4, #8
mov.s   v2[2], v6[1]
mov.s   v2[3], v7[1]
zip2.4s v3, v4, v5
mov.s   v3[2], v6[2]
mov.s   v3[3], v7[2]
mov.s   v1[2], v6[3]
mov x15, x23
mov.s   v1[3], v7[3]
ldp q16, q17, [x15]
ldp q18, q19, [x15, #32]
fadd.4s v4, v16, v4
fadd.4s v5, v17, v5
fadd.4s v6, v18, v6
fadd.4s v7, v19, v7
ldp q16, q17, [x20]
ldp q18, q19, [x20, #32]
ldp q20, q21, [x21]
ldp q22, q23, [x21, #32]
fmul.4s v24, v16, v20[0]
fmla.4s v24, v17, v20[1]
fmla.4s v24, v18, v20[2]
fmla.4s v24, v19, v20[3]
fmul.4s v20, v16, v21[0]
fmla.4s v20, v17, v21[1]
fmla.4s v20, v18, v21[2]
fmla.4s v20, v19, v21[3]
fmul.4s v21, v16, v22[0]
fmla.4s v21, v17, v22[1]
fmla.4s v21, v18, v22[2]
fmla.4s v21, v19, v22[3]
fmul.4s v16, v16, v23[0]
fmla.4s v16, v17, v23[1]
fmla.4s v16, v18, v23[2]
fmla.4s v16, v19, v23[3]
```

```
ldp q4, q5, [x22]
ldp q6, q7, [x22, #32]
zip1.4s v0, v4, v5
mov.s   v0[2], v6[0]
fadd.4s v4, v4, v24
fadd.4s v5, v5, v20
fadd.4s v6, v6, v21
fadd.4s v7, v7, v16
fmul.4s v16, v0, v4[0]
fmla.4s v16, v2, v4[1]
fmla.4s v16, v3, v4[2]
fmla.4s v16, v1, v4[3]
fmul.4s v4, v0, v5[0]
fmla.4s v4, v2, v5[1]
fmla.4s v4, v3, v5[2]
fmla.4s v4, v1, v5[3]
fmul.4s v5, v0, v6[0]
fmla.4s v5, v2, v6[1]
fmla.4s v5, v3, v6[2]
fmla.4s v5, v1, v6[3]
fmul.4s v0, v0, v7[0]
fmla.4s v0, v2, v7[1]
fmla.4s v0, v3, v7[2]
fmla.4s v0, v1, v7[3]
stp q16, q4, [x15]
stp q5, q0, [x15, #32]
```

CodeGen float 4x4

Eigen

Matrix Types

```
ldp q4, q5, [x22]
ldp q6, q7, [x22, #32]
zip1.4s v0, v4, v5
mov.s   v0[2], v6[0]
mov.s   v0[3], v7[0]
trn2.4s v2, v4, v5
ext.16b v1, v2, v4, #8
mov.s   v2[2], v6[1]
mov.s   v2[3], v7[1]
zip2.4s v3, v4, v5
mov.s   v3[2], v6[2]
mov.s   v3[3], v7[2]
mov.s   v1[2], v6[3]
mov x15, x23
mov.s   v1[3], v7[3]
ldp q16, q17, [x15]
ldp q18, q19, [x15, #32]
fadd.4s v4, v16, v4
fadd.4s v5, v17, v5
fadd.4s v6, v18, v6
fadd.4s v7, v19, v7
ldp q16, q17, [x20]
ldp q18, q19, [x20, #32]
ldp q20, q21, [x21]
ldp q22, q23, [x21, #32]
fmul.4s v24, v16, v20[0]
fmla.4s v24, v17, v20[1]
fmla.4s v24, v18, v20[2]
fmla.4s v24, v19, v20[3]
fmul.4s v20, v16, v21[0]
fmla.4s v20, v17, v21[1]
fmla.4s v20, v18, v21[2]
fmla.4s v20, v19, v21[3]
fmul.4s v21, v16, v22[0]
fmla.4s v21, v17, v22[1]
fmla.4s v21, v18, v22[2]
fmla.4s v21, v19, v22[3]
fmul.4s v16, v16, v23[0]
fmla.4s v16, v17, v23[1]
fmla.4s v16, v18, v23[2]
fmla.4s v16, v19, v23[3]
```

```
ldp q4, q5, [x22]
ldp q6, q7, [x22, #32]
zip1.4s v0, v4, v5
mov.s   v0[2], v6[0]
fadd.4s v4, v4, v24
fadd.4s v5, v5, v20
fadd.4s v6, v6, v21
fadd.4s v7, v7, v16
fmul.4s v16, v0, v4[0]
fmla.4s v16, v2, v4[1]
fmla.4s v16, v3, v4[2]
fmla.4s v16, v1, v4[3]
fmul.4s v4, v0, v5[0]
fmla.4s v4, v2, v5[1]
fmla.4s v4, v3, v5[2]
fmla.4s v4, v1, v5[3]
fmul.4s v5, v0, v6[0]
fmla.4s v5, v2, v6[1]
fmla.4s v5, v3, v6[2]
fmla.4s v5, v1, v6[3]
fmul.4s v0, v0, v7[0]
fmla.4s v0, v2, v7[1]
fmla.4s v0, v3, v7[2]
fmla.4s v0, v1, v7[3]
stp q16, q4, [x15]
stp q5, q0, [x15, #32]
```

CodeGen float 4x4

Matrix Types

```
ldp q4, q5, [x22]
ldp q6, q7, [x22, #32]
zip1.4s v0, v4, v5
mov.s   v0[2], v6[0]
mov.s   v0[3], v7[0]
trn2.4s v2, v4, v5
ext.16b v1, v2, v4, #8
mov.s   v2[2], v6[1]
mov.s   v2[3], v7[1]
zip2.4s v3, v4, v5
mov.s   v3[2], v6[2]
mov.s   v3[3], v7[2]
mov.s   v1[2], v6[3]
mov x15, x23
mov.s   v1[3], v7[3]
ldp q16, q17, [x15]
ldp q18, q19, [x15, #32]
fadd.4s v4, v16, v4
fadd.4s v5, v17, v5
fadd.4s v6, v18, v6
fadd.4s v7, v19, v7
ldp q16, q17, [x20]
ldp q18, q19, [x20, #32]
ldp q20, q21, [x21]
ldp q22, q23, [x21, #32]
fmul.4s v24, v16, v20[0]
fmla.4s v24, v17, v20[1]
fmla.4s v24, v18, v20[2]
fmla.4s v24, v19, v20[3]
fmul.4s v20, v16, v21[0]
fmla.4s v20, v17, v21[1]
fmla.4s v20, v18, v21[2]
fmla.4s v20, v19, v21[3]
fmul.4s v21, v16, v22[0]
fmla.4s v21, v17, v22[1]
fmla.4s v21, v18, v22[2]
fmla.4s v21, v19, v22[3]
fmul.4s v16, v16, v23[0]
fmla.4s v16, v17, v23[1]
fmla.4s v16, v18, v23[2]
fmla.4s v16, v19, v23[3]
```

```
ldp q4, q5, [x22]
ldp q6, q7, [x22, #32]
zip1.4s v0, v4, v5
mov.s   v0[2], v6[0]
fadd.4s v4, v4, v24
fadd.4s v5, v5, v20
fadd.4s v6, v6, v21
fadd.4s v7, v7, v16
fmul.4s v16, v0, v4[0]
fmla.4s v16, v2, v4[1]
fmla.4s v16, v3, v4[2]
fmla.4s v16, v1, v4[3]
fmul.4s v4, v0, v5[0]
fmla.4s v4, v2, v5[1]
fmla.4s v4, v3, v5[2]
fmla.4s v4, v1, v5[3]
fmul.4s v5, v0, v6[0]
fmla.4s v5, v2, v6[1]
fmla.4s v5, v3, v6[2]
fmla.4s v5, v1, v6[3]
fmul.4s v0, v0, v7[0]
fmla.4s v0, v2, v7[1]
fmla.4s v0, v3, v7[2]
fmla.4s v0, v1, v7[3]
stp q16, q4, [x15]
stp q5, q0, [x15, #32]
```

🛑 Single vector loads/stores

Eigen

```
Lloh8:
        ldr     x9, [x9]
        stur    x9, [x29, #-8]
        ldr     x9, [x1]
        str     x9, [sp]
        ldp     x9, x10, [x1, #16]
        ldr     q0, [x9]
        ldr     q1, [x10]
        fadd.4s v2, v1, v0
        str     q2, [sp, #16]
        ldr     q0, [x9, #16]
        ldr     q1, [x10, #16]
        fadd.4s v3, v1, v0
        str     q3, [sp, #32]
        ldr     q0, [x9, #32]
        ldr     q1, [x10, #32]
        fadd.4s v1, v1, v0
        str     q1, [sp, #48]
        ldr     q0, [x9, #48]
        ldr     q4, [x10, #48]
        fadd.4s v0, v4, v0
        str     q0, [sp, #64]
        ldp     x10, x11, [x1, #40]
        ldp     q4, q6, [x10]
        mov     x9, x11
        ld1r.4s { v5 }, [x9], #4
        fmul.4s v4, v5, v4
        ldr     s5, [x9]
        ldr     q7, [x10, #32]
        fmla.4s v4, v6, v5[0]
        ldp     s5, s6, [x11, #8]
        fmla.4s v4, v7, v5[0]
        ldr     q5, [x10, #48]
        fmla.4s v4, v5, v6[0]
        fadd.4s v2, v4, v2
        str     q2, [sp, #16]
        ldp     q2, q4, [x10]
        ldp     s6, s16, [x11, #16]
        fmul.4s v2, v2, v6[0]
        mov     x9, sp
        add     x9, x9, #16
        fmla.4s v2, v4, v16[0]
        ldp     s4, s6, [x11, #24]
        fmla.4s v2, v7, v4[0]
        fmla.4s v2, v5, v6[0]
        fadd.4s v2, v2, v3
        str     q2, [sp, #32]
```

```
        ldp     q2, q3, [x10]
        ldp     s4, s6, [x11, #32]
        fmul.4s v2, v2, v4[0]
        fmla.4s v2, v3, v6[0]
        ldr     q3, [x10, #32]
        ldp     s4, s6, [x11, #40]
        fmla.4s v2, v3, v4[0]
        fmla.4s v2, v5, v6[0]
        fadd.4s v1, v2, v1
        str     q1, [sp, #48]
        ldp     q1, q2, [x10]
        ldp     s3, s4, [x11, #48]
        fmul.4s v1, v1, v3[0]
        fmla.4s v1, v2, v4[0]
        ldp     q2, q3, [x10, #32]
        ldp     s4, s5, [x11, #56]
        fmla.4s v1, v2, v4[0]
        fmla.4s v1, v3, v5[0]
        fadd.4s v0, v1, v0
        str     q0, [sp, #64]
        ldr     x10, [sp]
        stp     x10, x9, [sp, #80]
        mov     w11, #4
        str     x11, [sp, #96]
        add     x11, x0, #8
LBB3_1:
        ldr     q0, [x10]
        ldr     q1, [x9, x8]
        fmul.4s v0, v1, v0
        ext.16b v2, v0, v0, #8
        fadd.2s v0, v2, v0
        faddp.2s        v0, v0, v0
        add     x12, x11, x8
        stur    s0, [x12, #-8]
        ldr     q0, [x10, #16]
        fmul.4s v0, v0, v1
        ext.16b v2, v0, v0, #8
        fadd.2s v0, v2, v0
        faddp.2s        v0, v0, v0
        stur    s0, [x12, #-4]
        ldr     q0, [x10, #32]
        fmul.4s v0, v0, v1
        ext.16b v2, v0, v0, #8
        fadd.2s v0, v2, v0
        faddp.2s        v0, v0, v0
        str     s0, [x12]
        ldr     q0, [x10, #48]
        fmul.4s v0, v0, v1
        ext.16b v1, v0, v0, #8
        fadd.2s v0, v1, v0
        faddp.2s        v0, v0, v0
        str     s0, [x12, #4]
        add     x8, x8, #16
        cmp     x8, #64
        b.ne    LBB3_1
```

CodeGen float 4x4

Matrix Types

```
ldp q4, q5, [x22]
ldp q6, q7, [x22, #32]
zip1.4s v0, v4, v5
mov.s   v0[2], v6[0]
mov.s   v0[3], v7[0]
trn2.4s v2, v4, v5
ext.16b v1, v2, v4, #8
mov.s   v2[2], v6[1]
mov.s   v2[3], v7[1]
zip2.4s v3, v4, v5
mov.s   v3[2], v6[2]
mov.s   v3[3], v7[2]
mov.s   v1[2], v6[3]
mov x15, x23
mov.s   v1[3], v7[3]
ldp q16, q17, [x15]
ldp q18, q19, [x15, #32]
fadd.4s v4, v16, v4
fadd.4s v5, v17, v5
fadd.4s v6, v18, v6
fadd.4s v7, v19, v7
ldp q16, q17, [x20]
ldp q18, q19, [x20, #32]
ldp q20, q21, [x21]
ldp q22, q23, [x21, #32]
fmul.4s v24, v16, v20[0]
fmla.4s v24, v17, v20[1]
fmla.4s v24, v18, v20[2]
fmla.4s v24, v19, v20[3]
fmul.4s v20, v16, v21[0]
fmla.4s v20, v17, v21[1]
fmla.4s v20, v18, v21[2]
fmla.4s v20, v19, v21[3]
fmul.4s v21, v16, v22[0]
fmla.4s v21, v17, v22[1]
fmla.4s v21, v18, v22[2]
fmla.4s v21, v19, v22[3]
fmul.4s v16, v16, v23[0]
fmla.4s v16, v17, v23[1]
fmla.4s v16, v18, v23[2]
fmla.4s v16, v19, v23[3]
```

```
ldp q4, q5, [x22]
ldp q6, q7, [x22, #32]
zip1.4s v0, v4, v5
mov.s   v0[2], v6[0]
fadd.4s v4, v4, v24
fadd.4s v5, v5, v20
fadd.4s v6, v6, v21
fadd.4s v7, v7, v16
fmul.4s v16, v0, v4[0]
fmla.4s v16, v2, v4[1]
fmla.4s v16, v3, v4[2]
fmla.4s v16, v1, v4[3]
fmul.4s v4, v0, v5[0]
fmla.4s v4, v2, v5[1]
fmla.4s v4, v3, v5[2]
fmla.4s v4, v1, v5[3]
fmul.4s v5, v0, v6[0]
fmla.4s v5, v2, v6[1]
fmla.4s v5, v3, v6[2]
fmla.4s v5, v1, v6[3]
fmul.4s v0, v0, v7[0]
fmla.4s v0, v2, v7[1]
fmla.4s v0, v3, v7[2]
fmla.4s v0, v1, v7[3]
stp q16, q4, [x15]
stp q5, q0, [x15, #32]
```

🛑 Single vector loads/stores

Eigen

```
Lloh8:
        ldr     x9, [x9]
        stur    x9, [x29, #-8]
        ldr     x9, [x1]
        str     x9, [sp]
        ldp     x9, x10, [x1, #16]
        ldr     q0, [x9]
        ldr     q1, [x10]
        fadd.4s v2, v1, v0
        str     q2, [sp, #16]
        ldr     q0, [x9, #16]
        ldr     q1, [x10, #16]
        fadd.4s v3, v1, v0
        str     q3, [sp, #32]
        ldr     q0, [x9, #32]
        ldr     q1, [x10, #32]
        fadd.4s v1, v1, v0
        str     q1, [sp, #48]
        ldr     q0, [x9, #48]
        ldr     q4, [x10, #48]
        fadd.4s v0, v4, v0
        str     q0, [sp, #64]
        ldp     x10, x11, [x1, #40]
        ldp     q4, q6, [x10]
        mov     x9, x11
        ld1r.4s { v5 }, [x9], #4
        fmul.4s v4, v5, v4
        ldr     s5, [x9]
        ldr     q7, [x10, #32]
        fmla.4s v4, v6, v5[0]
        ldp     s5, s6, [x11, #8]
        fmla.4s v4, v7, v5[0]
        ldr     q5, [x10, #48]
        fmla.4s v4, v5, v6[0]
        fadd.4s v2, v4, v2
        str     q2, [sp, #16]
        ldp     q2, q4, [x10]
        ldp     s6, s16, [x11, #16]
        fmul.4s v2, v2, v6[0]
        mov     x9, sp
        add     x9, x9, #16
        fmla.4s v2, v4, v16[0]
        ldp     s4, s6, [x11, #24]
        fmla.4s v2, v7, v4[0]
        fmla.4s v2, v5, v6[0]
        fadd.4s v2, v2, v3
        str     q2, [sp, #32]
        ldp     q2, q3, [x10]
        ldp     s4, s6, [x11, #32]
        fmul.4s v2, v2, v4[0]
        fmla.4s v2, v3, v6[0]
        ldr     q3, [x10, #32]
        ldp     s4, s6, [x11, #40]
        fmla.4s v2, v3, v4[0]
        fmla.4s v2, v5, v6[0]
        fadd.4s v1, v2, v1
        str     q1, [sp, #48]
        ldp     q1, q2, [x10]
        ldp     s3, s4, [x11, #48]
        fmul.4s v1, v1, v3[0]
        fmla.4s v1, v2, v4[0]
        ldp     q2, q3, [x10, #32]
        ldp     s4, s5, [x11, #56]
        fmla.4s v1, v2, v4[0]
        fmla.4s v1, v3, v5[0]
        fadd.4s v0, v1, v0
        str     q0, [sp, #64]
        ldr     x10, [sp]
        stp     x10, x9, [sp, #80]
        mov     w11, #4
        str     x11, [sp, #96]
        add     x11, x0, #8
LBB3_1:
        ldr     q0, [x10]
        ldr     q1, [x9, x8]
        fmul.4s v0, v1, v0
        ext.16b v2, v0, v0, #8
        fadd.2s v0, v2, v0
        faddp.2s        v0, v0, v0
        add     x12, x11, x8
        stur    s0, [x12, #-8]
        ldr     q0, [x10, #16]
        fmul.4s v0, v0, v1
        ext.16b v2, v0, v0, #8
        fadd.2s v0, v2, v0
        faddp.2s        v0, v0, v0
        stur    s0, [x12, #-4]
        ldr     q0, [x10, #32]
        fmul.4s v0, v0, v1
        ext.16b v2, v0, v0, #8
        fadd.2s v0, v2, v0
        faddp.2s        v0, v0, v0
        str     s0, [x12]
        ldr     q0, [x10, #48]
        fmul.4s v0, v0, v1
        ext.16b v1, v0, v0, #8
        fadd.2s v0, v1, v0
        faddp.2s        v0, v0, v0
        str     s0, [x12, #4]
        add     x8, x8, #16
        cmp     x8, #64
        b.ne    LBB3_1
```

CodeGen float 4x4

Matrix Types

```
ldp q4, q5, [x22]
ldp q6, q7, [x22, #32]
zip1.4s v0, v4, v5
mov.s   v0[2], v6[0]
mov.s   v0[3], v7[0]
trn2.4s v2, v4, v5
ext.16b v1, v2, v4, #8
mov.s   v2[2], v6[1]
mov.s   v2[3], v7[1]
zip2.4s v3, v4, v5
mov.s   v3[2], v6[2]
mov.s   v3[3], v7[2]
mov.s   v1[2], v6[3]
mov x15, x23
mov.s   v1[3], v7[3]
ldp q16, q17, [x15]
ldp q18, q19, [x15, #32]
fadd.4s v4, v16, v4
fadd.4s v5, v17, v5
fadd.4s v6, v18, v6
fadd.4s v7, v19, v7
ldp q16, q17, [x20]
ldp q18, q19, [x20, #32]
ldp q20, q21, [x21]
ldp q22, q23, [x21, #32]
fmul.4s v24, v16, v20[0]
fmla.4s v24, v17, v20[1]
fmla.4s v24, v18, v20[2]
fmla.4s v24, v19, v20[3]
fmul.4s v20, v16, v21[0]
fmla.4s v20, v17, v21[1]
fmla.4s v20, v18, v21[2]
fmla.4s v20, v19, v21[3]
fmul.4s v21, v16, v22[0]
fmla.4s v21, v17, v22[1]
fmla.4s v21, v18, v22[2]
fmla.4s v21, v19, v22[3]
fmul.4s v16, v16, v23[0]
fmla.4s v16, v17, v23[1]
fmla.4s v16, v18, v23[2]
fmla.4s v16, v19, v23[3]
```

```
ldp q4, q5, [x22]
ldp q6, q7, [x22, #32]
zip1.4s v0, v4, v5
mov.s   v0[2], v6[0]
fadd.4s v4, v4, v24
fadd.4s v5, v5, v20
fadd.4s v6, v6, v21
fadd.4s v7, v7, v16
fmul.4s v16, v0, v4[0]
fmla.4s v16, v2, v4[1]
fmla.4s v16, v3, v4[2]
fmla.4s v16, v1, v4[3]
fmul.4s v4, v0, v5[0]
fmla.4s v4, v2, v5[1]
fmla.4s v4, v3, v5[2]
fmla.4s v4, v1, v5[3]
fmul.4s v5, v0, v6[0]
fmla.4s v5, v2, v6[1]
fmla.4s v5, v3, v6[2]
fmla.4s v5, v1, v6[3]
fmul.4s v0, v0, v7[0]
fmla.4s v0, v2, v7[1]
fmla.4s v0, v3, v7[2]
fmla.4s v0, v1, v7[3]
stp q16, q4, [x15]
stp q5, q0, [x15, #32]
```

- Single vector loads/stores
- Uses <2 x float> instead of <4 x float>

Eigen

```
Lloh8:
        ldr     x9, [x9]
        stur    x9, [x29, #-8]
        ldr     x9, [x1]
        str     x9, [sp]
        ldp     x9, x10, [x1, #16]
        ldr     q0, [x9]
        ldr     q1, [x10]
        fadd.4s v2, v1, v0
        str     q2, [sp, #16]
        ldr     q0, [x9, #16]
        ldr     q1, [x10, #16]
        fadd.4s v3, v1, v0
        str     q3, [sp, #32]
        ldr     q0, [x9, #32]
        ldr     q1, [x10, #32]
        fadd.4s v1, v1, v0
        str     q1, [sp, #48]
        ldr     q0, [x9, #48]
        ldr     q4, [x10, #48]
        fadd.4s v0, v4, v0
        str     q0, [sp, #64]
        ldp     x10, x11, [x1, #40]
        ldp     q4, q6, [x10]
        mov     x9, x11
        ld1r.4s { v5 }, [x9], #4
        fmul.4s v4, v5, v4
        ldr     s5, [x9]
        ldr     q7, [x10, #32]
        fmla.4s v4, v6, v5[0]
        ldp     s5, s6, [x11, #8]
        fmla.4s v4, v7, v5[0]
        ldr     q5, [x10, #48]
        fmla.4s v4, v5, v6[0]
        fadd.4s v2, v4, v2
        str     q2, [sp, #16]
        ldp     q2, q4, [x10]
        ldp     s6, s16, [x11, #16]
        fmul.4s v2, v2, v6[0]
        mov     x9, sp
        add     x9, x9, #16
        fmla.4s v2, v4, v16[0]
        ldp     s4, s6, [x11, #24]
        fmla.4s v2, v7, v4[0]
        fmla.4s v2, v5, v6[0]
        fadd.4s v2, v2, v3
        str     q2, [sp, #32]
        ldp     q2, q3, [x10]
        ldp     s4, s6, [x11, #32]
        fmul.4s v2, v2, v4[0]
        fmla.4s v2, v3, v6[0]
        ldr     q3, [x10, #32]
        ldp     s4, s6, [x11, #40]
        fmla.4s v2, v3, v4[0]
        fmla.4s v2, v5, v6[0]
        fadd.4s v1, v2, v1
        str     q1, [sp, #48]
        ldp     q1, q2, [x10]
        ldp     s3, s4, [x11, #48]
        fmul.4s v1, v1, v3[0]
        fmla.4s v1, v2, v4[0]
        ldp     q2, q3, [x10, #32]
        ldp     s4, s5, [x11, #56]
        fmla.4s v1, v2, v4[0]
        fmla.4s v1, v3, v5[0]
        fadd.4s v0, v1, v0
        str     q0, [sp, #64]
        ldr     x10, [sp]
        stp     x10, x9, [sp, #80]
        mov     w11, #4
        str     x11, [sp, #96]
        add     x11, x0, #8
LBB3_1:
        ldr     q0, [x10]
        ldr     q1, [x9, x8]
        fmul.4s v0, v1, v0
        ext.16b v2, v0, v0, #8
        fadd.2s v0, v2, v0
        faddp.2s        v0, v0, v0
        add     x12, x11, x8
        stur    s0, [x12, #-8]
        ldr     q0, [x10, #16]
        fmul.4s v0, v0, v1
        ext.16b v2, v0, v0, #8
        fadd.2s v0, v2, v0
        faddp.2s        v0, v0, v0
        stur    s0, [x12, #-4]
        ldr     q0, [x10, #32]
        fmul.4s v0, v0, v1
        ext.16b v2, v0, v0, #8
        fadd.2s v0, v2, v0
        faddp.2s        v0, v0, v0
        str     s0, [x12]
        ldr     q0, [x10, #48]
        fmul.4s v0, v0, v1
        ext.16b v1, v0, v0, #8
        fadd.2s v0, v1, v0
        faddp.2s        v0, v0, v0
        str     s0, [x12, #4]
        add     x8, x8, #16
        cmp     x8, #64
        b.ne    LBB3_1
```

Remaining Work

- Improve codegen for operations on larger matrixes
  - Split operations on large vectors
  - Generalize tiled loop code generation
  - Row-major support
- Clang polishing
  - Initializer syntax, fast-math flags, wrapping flags

Remaining Work

- Improve codegen for operations on larger matrixes
  - Split operations on large vectors
  - Generalize tiled loop code generation
  - Row-major support 🚀 Contributions Welcome!
- Clang polishing
  - Initializer syntax, fast-math flags, wrapping flags

Questions?