Leverage AArch64 SME/SVE to support Clang matrix_type

Chen Zheng

HUAWEI

matrix_type

- Introduced in ❁ D76612 [Matrix] Add draft specification for matrix support in Clang. by fhahn
- Source code solution to write high-performance code for matrix operation

```
MTy Res;
for (int C = 0; C < col; ++C) {
  for (int R = 0; R < row; ++R) {
    EltTy Elt = 0;
    for (int K = 0; K < inner; ++K) {
      Elt += M1[R][K] * M2[K][C];
    }
    Res[R][C] = Elt;
  }
}
```



```
typedef EltTy m1_t __attribute__((matrix_type(row, inner)));
typedef EltTy m2_t __attribute__((matrix_type(inner, col)));
typedef EltTy mr_t __attribute__((matrix_type(row, col)));

m1_t M1;
m2_t M2;
mr_t Res = M1 * M2;
```

Clang matrix_type limitations

- This is an **experimental feature,** supports only**:**
  > small matrix size
  > four operations matmul, transpose, column major load, column-major store
  > 2-dimensions matrix
  > NEON instructions on ARM

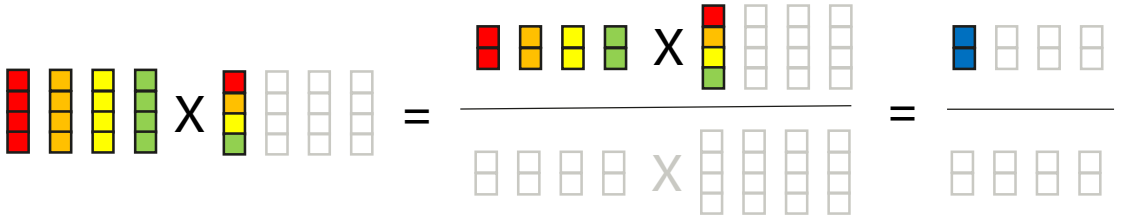
ARM Scalable Matrix Extension (SME)

- To support matrix operations, ARM SME provides:
  - > A matrix register ZA that can be accessed horizontally or vertically and can hold up to 2048 bit
  - > Instructions that accumulate the outer product of two vectors into a ZA tile, like fmopa

SME for matrix matmul

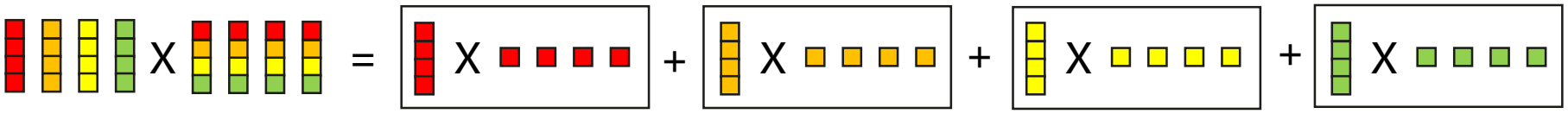
```
typedef double m1_t __attribute__((matrix_type(4, 4)));
typedef double m2_t __attribute__((matrix_type(4, 4)));
typedef double mr_t __attribute__((matrix_type(4, 4)));

mr_t foo(m1_t *a, m2_t *b) {
  return (*a) * (*b);
}
```

- NEON (column major matrix):



> **4 x 4 x 2 = 32 fmul/fma instructions**

- SME (-msve-vector-bits=256, column major matrix):



> **4 fmopa instructions**

HUAWEI

Work in Clang/LLVM

- Base on LowerMatrixIntrinsics
  - > matrix shape propagation
  - > Change the logic for lowering matrix related intrinsics
  - > Change the logic for lowering matrix related operators

4X4

```
define <16 x double> @foo(ptr %0, ptr %1, ptr %2) {
  %4 = load <16 x double>, ptr %0
  %5 = load <16 x double>, ptr %1

  %6 = call <16 x double> @llvm.matrix.multiply.v16f64.v16f64.v16f64(
              <16 x double> %4, <16 x double> %5, i32 4, i32 4, i32 4)

  %7 = load <16 x double>, ptr %2

  %8 = fadd <16 x double> %6, %7

  ret <16 x double> %8
}
```

Work in Clang/LLVM

- Base on LowerMatrixIntrinsics
  - > matrix shape propagation
  - > Change the logic for lowering matrix related intrinsics
  - > Change the logic for lowering matrix related operators

```
%4 = load <16 x double>, ptr %0
%5 = load <16 x double>, ptr %1

%6 = call <16 x double> @llvm.matrix.multiply.v16f64.v16f64.v16f64(
            <16 x double> %4, <16 x double> %5, i32 4, i32 4, i32 4)
```



```
call void @__sme_matmul_float8 (
            ptr %9, ptr %0, ptr %1, i32 4, i32 4, i32 4)
%6 = load <16 x double>, ptr %9
```

Work in Clang/LLVM

- Base on LowerMatrixIntrinsics
  - > matrix shape propagation
  - > Change the logic for lowering matrix related intrinsics
  - > Change the logic for lowering matrix related operators

4X4

```
%6 = load <16 x double>, ptr %1
%7 = load <16 x double>, ptr %2

%8 = fadd <16 x double> %6, %7
```

```
call void @__sve_add_float8 (
                ptr %10, ptr %1, ptr %2, i32 4, i32 4, i32 4)
%8 = load <16 x double>, ptr %10
```

Alternative solution

- Not base on LowerMatrixIntrinsics
  - > Clang directly emits the matrix operations as library calls
  - > **Pros:**
    - easily switch to other matrix library
    - could support large matrix
  - > **Cons:**
    - introduce big function call cost especially for SME za register handling
    - lose middle end optimizations for the flattened vector types and operations

```
typedef double m1_t __attribute__((matrix_type(4, 4)));
typedef double m2_t __attribute__((matrix_type(4, 4)));
typedef double mr_t __attribute__((matrix_type(4, 4)));

mr_t foo(m1_t *a, m2_t *b, mr_t *c) {
  return (*a) * (*b) + (*c);
}
```

```
define void @foo(ptr sret([16 x double]) %agg.result, ptr %a, ptr %b, ptr %c) {
entry:
  %0 = alloca <16 x double>, align 128
  call void @__sme_matmul_float8(ptr %0, ptr %a, ptr %b, i32 4, i32 4, i32 4)
  call void @__sve_add_float8 (ptr %agg.result, ptr %0, ptr %c, i32 4, i32 4)
  ret void
}
```

HUAWEI

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

Bring digital to every person, home and organization for a fully connected, intelligent world.

HUAWEI