# Case study of sparse matrix intrinsics in LLVM

Eric Feng & Chris Yoon (2024)

## Sparse Matrices

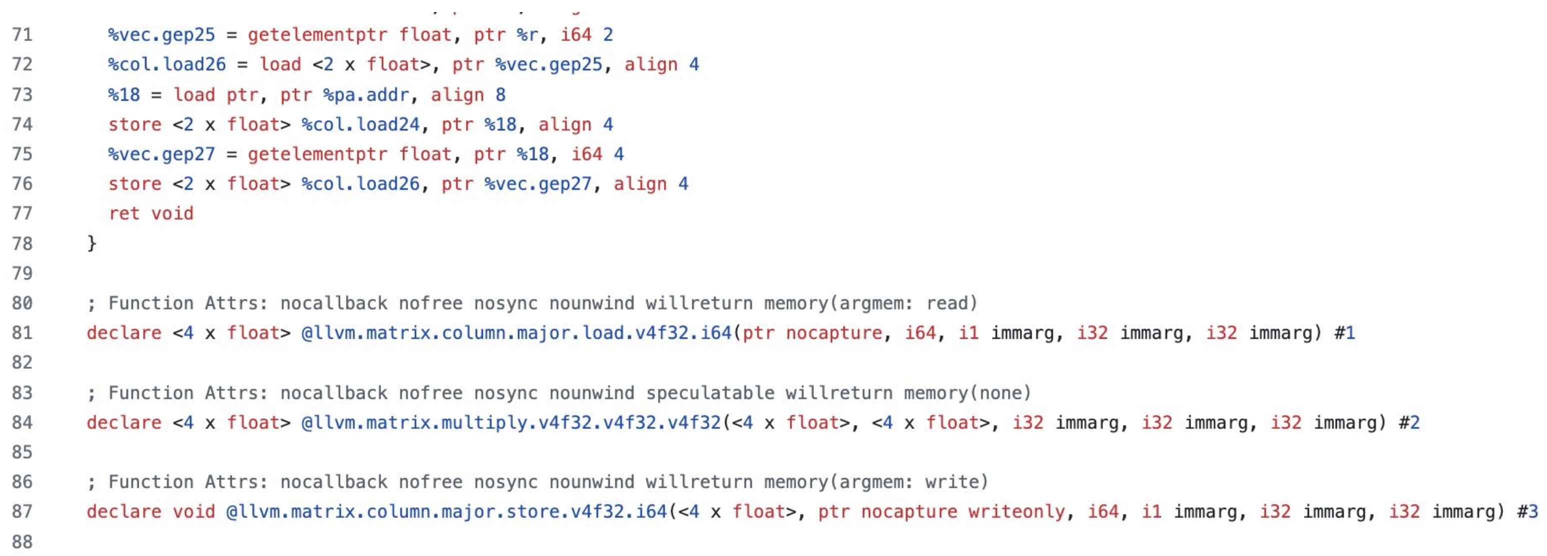
- Compact representation of matrices with lots of zero elements
- Generally stores only the nonzero elements
- Sparse matrix operations only operate on nonzero elements
- Given the right circumstances, huge memory and performance gains!

$$A = \begin{pmatrix} a_{0,0} & 0 & 0 & a_{0,3} \\ 0 & 0 & 0 & 0 \\ a_{2,0} & 0 & 0 & 0 \end{pmatrix}$$

| | | | | |
|---|---|---|---|---|
| `pointers[1]:` | 0 | 2 | 2 | 3 |
| `indices[1]:` | 0 | 3 | 0 | |
| `values:` | $a_{0,0}$ | $a_{0,3}$ | $a_{2,0}$ | |

# LLVM Intrinsics

- Operations that can be matched in the compiler backend to produce the corresponding machine code

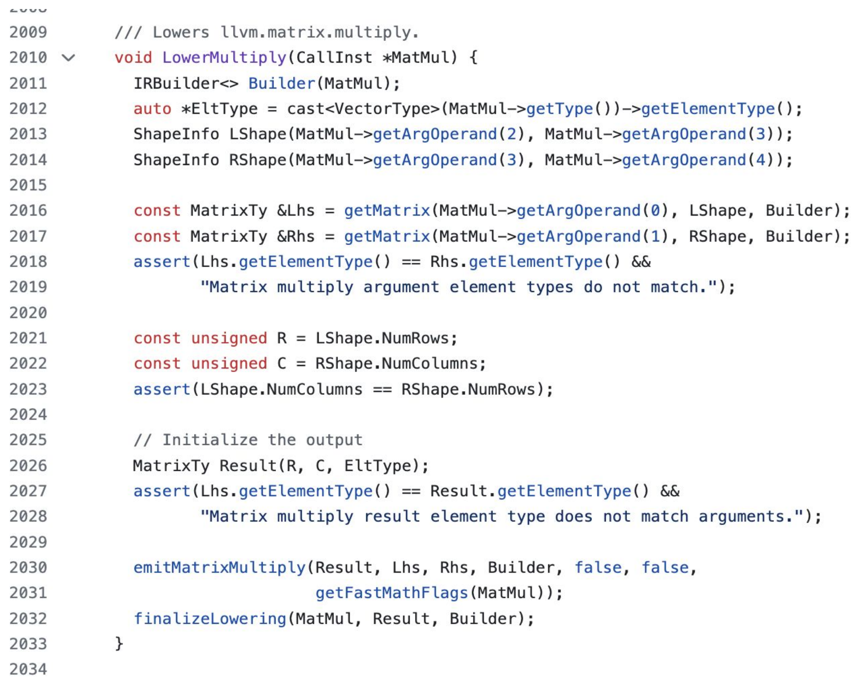


```
  %vec.gep25 = getelementptr float, ptr %r, i64 2
  %col.load26 = load <2 x float>, ptr %vec.gep25, align 4
  %18 = load ptr, ptr %pa.addr, align 8
  store <2 x float> %col.load24, ptr %18, align 4
  %vec.gep27 = getelementptr float, ptr %18, i64 4
  store <2 x float> %col.load26, ptr %vec.gep27, align 4
  ret void
}

; Function Attrs: nocallback nofree nosync nounwind willreturn memory(argmem: read)
declare <4 x float> @llvm.matrix.column.major.load.v4f32.i64(ptr nocapture, i64, i1 immarg, i32 immarg, i32 immarg) #1

; Function Attrs: nocallback nofree nosync nounwind speculatable willreturn memory(none)
declare <4 x float> @llvm.matrix.multiply.v4f32.v4f32.v4f32(<4 x float>, <4 x float>, i32 immarg, i32 immarg, i32 immarg) #2

; Function Attrs: nocallback nofree nosync nounwind willreturn memory(argmem: write)
declare void @llvm.matrix.column.major.store.v4f32.i64(<4 x float>, ptr nocapture writeonly, i64, i1 immarg, i32 immarg, i32 immarg) #3

```

In LLVM IR emitted from C programs

## LLVM Intrinsics

- Operations that can be matched in the compiler backend to produce the corresponding machine code



```
/// Lowers llvm.matrix.multiply.
void LowerMultiply(CallInst *MatMul) {
  IRBuilder<> Builder(MatMul);
  auto *EltType = cast<VectorType>(MatMul->getType())->getElementType();
  ShapeInfo LShape(MatMul->getArgOperand(2), MatMul->getArgOperand(3));
  ShapeInfo RShape(MatMul->getArgOperand(3), MatMul->getArgOperand(4));

  const MatrixTy &Lhs = getMatrix(MatMul->getArgOperand(0), LShape, Builder);
  const MatrixTy &Rhs = getMatrix(MatMul->getArgOperand(1), RShape, Builder);
  assert(Lhs.getElementType() == Rhs.getElementType() &&
         "Matrix multiply argument element types do not match.");

  const unsigned R = LShape.NumRows;
  const unsigned C = RShape.NumColumns;
  assert(LShape.NumColumns == RShape.NumRows);

  // Initialize the output
  MatrixTy Result(R, C, EltType);
  assert(Lhs.getElementType() == Result.getElementType() &&
         "Matrix multiply result element type does not match arguments.");

  emitMatrixMultiply(Result, Lhs, Rhs, Builder, false, false,
                     getFastMathFlags(MatMul));
  finalizeLowering(MatMul, Result, Builder);
}
```

**Generating machine code for the matrix multiplication intrinsics**

# Sparse matrix intrinsics in LLVM

**Lowering pipeline:**

1. Specify tablegen for desired intrinsic (e.g load, store, multiply)
2. Extend pass manager with option to lower sparse matrix intrinsics
3. Traverse IR and replace all calls to desired intrinsics with custom lowering scheme

**Internal representation:**

- Store CSC representation as a contiguous flat vector
  - [col_ptrs + row_indices + values]
- Recover CSC for sparse matrix operations

## Limitations of LLVM

- Intrinsics are limited to zero to one return values (LLVM first class types); so we must store CSC representation as a flat vector
- But recovering the flat vector requires us to know values (such as number of nonzeros) that are only available at runtime
- Unable to read the values of col_ptrs and row_indices at compile time; which are necessary for operations such as SpGEMM, SpMM etc

Thank you