TECHNISCHE
UNIVERSITÄT
WIEN

Multidimensional Match, Transform and Replace

Benedikt Huber

Intro

- Single dimensional vectorizers are well established
  - SIMD instructions
- How can we make use of multidimensional optimized target implementations?
  - Optimized library functions like `gemm` in OpenBLAS
  - Matrix or tensor instructions like `TDPBUUD` in Intel AMX
- Accessed via compiler intrinsics or library calls
  - Not portable and error prone
- Goal of MMTR
  - Identify and transform nested loops and replace them
  - Only based on a description of the optimized target implementation

Overview

- An optimized target implementation often is semantically equivalent to
  - A perfectly nested loop
  - With a dense and rectangular iteration domain
  - A constant step size
  - A loop invariant iteration count
  - A small loop body
  - Without complex control flow
- Suitable for
  - Polyhedral model (see next slide)
  - Pattern matching
- MMTR implemented as
  - Extension pass in Polly
  - LLVM 19
- Two modes for dimensions
  - parametrized
  - fixed
- Convenient Tablegen description

Polyhedral model

- Compact representation of control flow and memory accesses
- Array indices and Branch conditions have to be
  - affine functions of loop invariant values
- SCoP: Static Control Part
- Polyhedral statement: Sequence of LLVM IR without branches
  `Stmt_for_body4`
- Polyhedral statement instance: One execution of the polyhedral statement
  `Stmt_for_body4[m=2, n=3, l=5]`
- Instance set
  `Stmt_for_body4[m, n, l]: 0<m<9 and 0<n<4 and 0<l<7`
- Access to an array
  `Stmt_for_body4[m, n, l] -> Arr_B[l, n]`
- Polyhedral schedule: The order of the polyhedral statement instances
  `Stmt_for_body4[m, n, l] -> [n, 0, l, m, 1]`
- Capable of specifying loop transformations
  - loop fission, loop fusion, strip mining, tiling, ...

MMTR

- Each target specific MPattern consists of
  - InstrPat: Instruction Tree Pattern
  - Dimensions
  - MemAccPats: List of array accesses
  - ReplacementBlock: LLVM IR to be inserted

Example: Input

```
void matmul(unsigned l, unsigned m, unsigned n,
            TY z[restrict l][m],
            TY x[restrict l][n],
            TY y[restrict n][m]) {
  for (unsigned i = 0; i < l; i++)
    for (unsigned j = 0; j < m; j++) {
      // Polly reports this as Stmt_for_body3
      z[i][j] = 0;
      for (unsigned k = 0; k < n; k++)
        // Polly reports this as Stmt_for_body8
        z[i][j] += x[i][k] * y[k][j];
    }
}
```

Listing 1: Input Program: Matrix Multiplication

Example: MPattern of Vector Matrix Multiplication

```
def MACPat :
  Store<
    FAdd<
      FMul<Load<A>, Load<B>>,
      Load<C>>,
    C>;

// indices are j, k, l, ...
def LA : ReadAccess<A,  [k]>;
def LB : ReadAccess<B,  [k, j]>;
def LC : ReadAccess<C,  [j]>;
def SC : WriteAccess<C, [j]>;

def VectMatMulPattern :
  MPattern<
    MACPat
  , Fixed<[6, 4]>
  , [LA, LB, LC, SC]
  , Emitter<"VectMatMul", [C, A, B, empty, strideB]>>;
```

Listing 2: MPattern

Example: Algorithm

- Match LLVM IR Instructions with the InstrPat
  and capture the needed values
  `def MACPat : Store< FAdd< FMul<Load<A>, Load<B>>, Load<C>>, C>`
- Check size and shape of instance set as reported by Polly
- Match all memory accesses
  by projecting on the innermost dimensions
  - Polly reports access to `x` as
    `Stmt_for_body8[i0, i1, i2] -> MemRef_x [i0, i2]`
  - to match with
    `Fixed<[6, 4]> -> ReadAccess<A, [k]>`
  - After projection
    `Stmt_for_body8[    i1, i2] -> MemRef_x [    i2]`
- Determine strides as reported by ArrayInfo

Example: Algorithm

- Move other statements out of the loop
  Use Polly's ScheduleTree for loop fission

```
for (int c0 = 0; c0 < l; c0 += 1) {
  for (int c1 = 0; c1 < m; c1 += 1)
    Stmt_for_body3(c0, c1); // Zero init moved
  for (int c1 = 0; c1 < m; c1 += 1)
    for (int c2 = 0; c2 < n; c2 += 1)
      Stmt_for_body8(c0, c1, c2);
}
```

Listing 3: Loop Fission

Example: Algorithm

- Loop Tiling for Fixed Dimensions
  Again with Polly's functionality

```
for (int c0 = 0; c0 < l; c0 += 1) {
  for (int c1 = 0; c1 < m; c1 += 1)
    Stmt_for_body3(c0, c1);
  for (int c1 = 0; c1 < 6 * floord(m, 6); c1 += 6)
    for (int c2 = 0; c2 < 4 * floord(n, 4); c2 += 4)

      // REPLACE_LOOP
      for (int c3 = 0; c3 <= 5; c3 += 1)
        for (int c4 = 0; c4 <= 3; c4 += 1)
          Stmt_for_body8(c0, c1 + c3, c2 + c4);

  for (int c1 = 0; c1 < 6 * floord(m, 6); c1 += 1)
    for (int c2 = -(n % 4) + n; c2 < n; c2 += 1)
      Stmt_for_body8(c0, c1, c2);
  for (int c1 = -(m % 6) + m; c1 < m; c1 += 1)
    for (int c2 = 0; c2 < n; c2 += 1)
      Stmt_for_body8(c0, c1, c2);
}
```

Listing 4: Loop Tiling

Example: Algorithm

- Check if all data dependencies are still observed
  `isValidSchedule`
- Set the replacement mark
- During emission replace the loop with the ReplacementBlock
  `Emitter<"VectMatMul", [C, A, B, empty, strideB]>`

Evaluation

- Optimized target implementation was OpenBLAS on x86-64
- MPatterns for
  - Matrix Mult `cblas_sgemm` (3D)
  - Vector Matrix Mult `cblas_sgemv` (2D)
  - Dot Product `cblas_sdot` (1D)
  - Vector Addition `cblas_saxpy` (1D)

- PolyBench
  - 7 out of 30 with matches

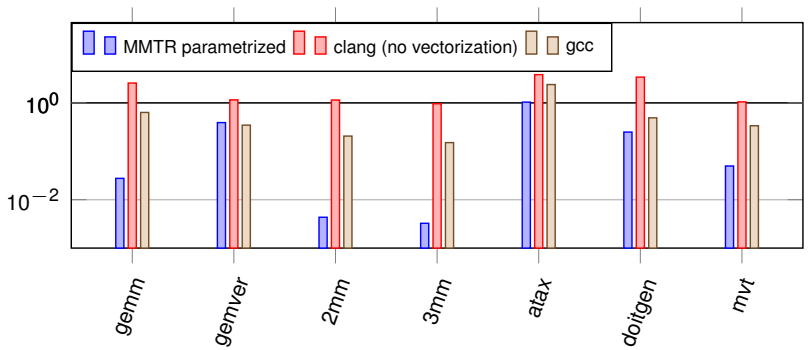

Figure: PolyBench Runtime relative to Clang with vectorization
logarithmic scale, smaller is better

- Visual Wake Words (VWW)
  - MLPerf Tiny
  - Classify images
- C code generated from TFLite model by TVM
- 10 MMTR matches
  4 in hot functions
- All matches are Vector Matrix Mult
- Reduces runtime by 66%

Outro

- Future work
  - Extend handling of access patterns
  - Generalize iteration domains
- MMTR source code available at
  - `github.com/OpenVADL/llvm-project/tree/mmtr`
- Contact
  - `benedikt.huber@tuwien.ac.at`
- My stay at TU Wien ends 2025-09