Wild idea - Polyhedral model and linear algebra (LAPACK)

Guillaume Iooss

January 28th, 2026

Let's start with a wild opinion ...

Let's start with a wild opinion ...

Polyhedral programs are too general.

(... No tomato/torch/pitchfork (yet) ? Great !)

Let's start with a wild opinion ...

Polyhedral programs are too general.

(... No tomato/torch/pitchfork (yet) ? Great !)

Class of polyhedral programs is too general to reach best perf :

- Focus on applicability of mathematical framework
  - How far can a polyhedral analysis/transformation be applied ?

Let's start with a wild opinion ...

Polyhedral programs are too general.

(... No tomato/torch/pitchfork (yet) ? Great !)

Class of polyhedral programs is too general to reach best perf :

- Focus on applicability of mathematical framework
  - How far can a polyhedral analysis/transformation be applied ?
- But, covers multiple application domains (cf Polybench)
  - Different properties / bottlenecks / difficulties
  - They are not optimized the same way.
    Ex : Stencils (`jacobi2d`) VS `gemm`

Polyhedral application subdomains

More info on a prog = better optimization work

- Specialization is advantageous
  - Constraining the input program (ex : DSL)
  - Detecting patterns/raising abstraction

Polyhedral application subdomains

More info on a prog = better optimization work

- Specialization is advantageous
  - Constraining the input program (ex : DSL)
  - Detecting patterns/raising abstraction

$\Rightarrow$ Deploy a more pertinent compilation pipeline

(more general view : how to estimate the most important bottleneck in a program, and adapt the compilation pipeline to it ?)

Polyhedral application subdomains

More info on a prog = better optimization work

- Specialization is advantageous
  - Constraining the input program (ex : DSL)
  - Detecting patterns/raising abstraction

$\Rightarrow$ Deploy a more pertinent compilation pipeline

(more general view : how to estimate the most important bottleneck in a program, and adapt the compilation pipeline to it ?)

Some concrete examples :

- Matrix multiplication (abstraction rising in MLIR)
- Tensor operators ("the rectangular model")
- Stencils (DSL)
- Image processing / network of convolutions (Polymage)
- ...

Polyhedral application subdomains

More info on a prog = better optimization work

- Specialization is advantageous
  - Constraining the input program (ex : DSL)
  - Detecting patterns/raising abstraction

$\Rightarrow$ Deploy a more pertinent compilation pipeline

(more general view : how to estimate the most important bottleneck in a program, and adapt the compilation pipeline to it ?)

Some concrete examples :

- Matrix multiplication (abstraction rising in MLIR)
- Tensor operators ("the rectangular model")
- Stencils (DSL)
- Image processing / network of convolutions (Polymage)
- ...

- What about linear algebra ?

Polyhedral application subdomains

More info on a prog = better optimization work

- Specialization is advantageous
  - Constraining the input program (ex : DSL)
  - Detecting patterns/raising abstraction

$\Rightarrow$ Deploy a more pertinent compilation pipeline

(more general view : how to estimate the most important bottleneck in a program, and adapt the compilation pipeline to it ?)

Some concrete examples :

- Matrix multiplication (abstraction rising in MLIR)
- Tensor operators ("the rectangular model")
- Stencils (DSL)
- Image processing / network of convolutions (Polymage)
- ...

- What about linear algebra ?

(as in LAPACK, not `mlir.linalg`)

What is a linear algebra program ?

- What properties could be useful for compilation ?

What is a linear algebra program ?

- What properties could be useful for compilation ?

(Unchecked wild) intuition on Cholesky ($\approx$ Polybench version) :

```
for (i = 0; i < N; i++) {
  for (j = 0; j < i; j++) {
    for (k = 0; k < j; k++) {
      A[i][j] -= A[i][k] * A[j][k];
    }
  }
  for (j = 0; j < i; j++) {
    A[i][j] /= A[j][j];
  }
  for (k = 0; k < i; k++) {
    A[i][i] -= A[i][k] * A[i][k];
  }
  A[i][i] = sqrt(A[i][i]);
}
```

- **Structure of loops :** temporal dimensions surrounding operations with reduction/parallel dimensions.
  - Work progression over matrix : row-by-row/column-by-column

Further example - blocked Cholesky

Blocked version : `spotrf.f` from LAPACK (excuse my Fren- Fortran)

```
*           Compute the Cholesky factorization A = L*L**T.
*
            DO 20 J = 1, N, NB
*
*              Update and factorize the current diagonal block and test
*              for non-positive-definiteness.
*
               JB = MIN( NB, N-J+1 )
               CALL SSYRK( 'Lower', 'No transpose', JB, J-1, -ONE,
     $                     A( J, 1 ), LDA, ONE, A( J, J ), LDA )
               CALL SPOTRF2( 'Lower', JB, A( J, J ), LDA, INFO )
               IF( INFO.NE.0 )
     $            GO TO 30
               IF( J+JB.LE.N ) THEN
*
*                 Compute the current block column.
*
                  CALL SGEMM( 'No transpose', 'Transpose', N-J-JB+1, JB,
     $                        J-1, -ONE, A( J+JB, 1 ), LDA, A( J, 1 ),
     $                        LDA, ONE, A( J+JB, J ), LDA )
                  CALL STRSM( 'Right', 'Lower', 'Transpose', 'Non-unit',
     $                        N-J-JB+1, JB, ONE, A( J, J ), LDA,
     $                        A( J+JB, J ), LDA )
               END IF
   20       CONTINUE
```

- 3 BLAS operators (+ a recursion) / same structure of loops

Conclusion

Wild intuition (no systematic study yet) :

- Linear algebra programs have properties on its structure
- Other constraints (shape of access function, iteration domains, . . .)

Conclusion

Wild intuition (no systematic study yet) :

- Linear algebra programs have properties on its structure
- Other constraints (shape of access function, iteration domains, . . .)

Properties probably exploitable for performance :

- Similarities with network of tensor operations ?
- Intuitively, register tiling is a good strategy for both
  - Of course, major differences : triangular domains, . . .
  - Gap between the two domains might not be “that far” ? (expertise transferable ?)

Conclusion

Wild intuition (no systematic study yet) :

- Linear algebra programs have properties on its structure
- Other constraints (shape of access function, iteration domains, ...)

Properties probably exploitable for performance :

- Similarities with network of tensor operations ?
- Intuitively, register tiling is a good strategy for both
  - Of course, major differences : triangular domains, ...
  - Gap between the two domains might not be "that far" ? (expertise transferable ?)

Thanks for listening !