Introduction to Polyhedral Compilation

Sanket Tavarageri

Research Scientist at Intel Labs, Bengaluru

Outline

- Polyhedral Compilation and its advantages
- Polyhedral Model Constructs
- Loop Transformations
  - Loop interchange
- Code analysis
  - Data set computation

What is the Polyhedral Model?

Polyhedral model for compilation

Source code to mathematical abstractions

```
#pragma omp parallel for private(ofm_tile, ifm_tile, ij, oj, kj, ki, ii)
for (img = 0; img < nImg; ++img) {
 for (ofm_tile = 0; ofm_tile < nOfm / GEMM_BLOCK; ++ofm_tile) {
  for (ifm_tile = 0; ifm_tile < nIfm / GEMM_BLOCK; ++ifm_tile) {
   for (oj = 0; oj < ofh; ++oj) {
        ij = oj * STRIDE_H;
        for (kj = 0; kj < kh; ++kj) {
         for (ki = 0; ki < kw; ++ki) {

          /* GEMM operation begins */
          for (oi = 0; oi < ofw; ++oi) {
           ii = oi * STRIDE_W;
            for (ofm = 0; ofm < GEMM_BLOCK; ++ofm) {
                for (ifm = 0; ifm < GEMM_BLOCK; ++ifm) {
                 output[img][ofm_tile][oj][oi][ofm] +=
                  filter[ofm_tile][ifm_tile][kj][ki][ifm][ofm]
                   * input[img][ifm_tile][ij+kj][ii+ki][ifm];
                }
           }
          }
         /* GEMM operation ends */
     }}}}}}
```

$$A = \begin{bmatrix} 2 & 1 \\ 0 & -1 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} 1/2 & 1/2 \\ 0 & -1 \end{bmatrix}$$

and

$$\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}.$$

We have $|\det A| = \det S = 2$ *and* $\mathbf{k}_1^t = \begin{bmatrix} 0 & 0 \end{bmatrix}$ *and* $\mathbf{k}_2^t = \begin{bmatrix} 0 & 1 \end{bmatrix}$. *Therefore,*

$$\mathbf{w}_1^t = \left\{\begin{bmatrix} 0 & 0 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1/2 & 1/2 \\ 0 & -1 \end{bmatrix}\right\}\begin{bmatrix} 2 & 1 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

and

$$\begin{aligned} \mathbf{w}_2^t &= \left\{\begin{bmatrix} 0 & 1 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} 1/2 & 1/2 \\ 0 & -1 \end{bmatrix}\right\}\begin{bmatrix} 2 & 1 \\ 0 & -1 \end{bmatrix} \\ &= \begin{bmatrix} 1/2 & 1/2 \end{bmatrix}\begin{bmatrix} 2 & 1 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}. \end{aligned}$$

Mathematical abstractions to source code

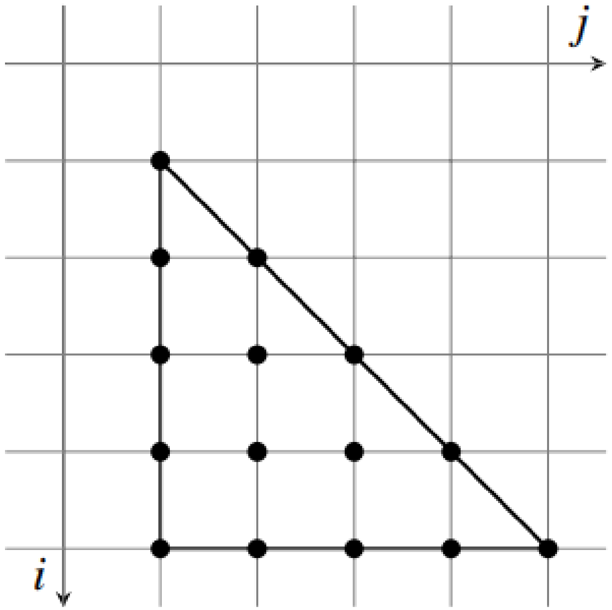
```
// First level of tiling
// Potential parallel loop1: it2
for (it2 = 0; it2 < M; it2 += M2_Tile) {
// Potential parallel loop2: jt2
 for (jt2 = 0; jt2 < N; jt2 += N2_Tile) {
  for (kt2 = 0; kt2 < K; kt2 += K2_Tile) {
  // Second level of tiling
   for(it1=it2;it1<it2+M2_Tile;it1+=M1_Tile){
    for(jt1=jt2;jt1<jt2+N2_Tile;jt1+=N1_Tile){
     for(kt1=kt2;kt1<kt2+K2_Tile;kt1+=K1_Tile){
      //Call to GEMM microkernel of size:
      //M1_Tile,N1_Tile,K1_Tile
      microkernel(..)
      }}}}}}
```

Iteration spaces as polyhedrons

A loop nest

```
for (i = 1; i <= n; ++i)
    for (j = 1; j <= i; ++j)
        /* S */
```

Its iteration space as a 2-D polyhedron



A polyhedron is an n-dimensional geometric object

What can the Polyhedral model be used for?

Applicability

- Polyhedral model provides a powerful mathematical framework to reason about loops in programs
- Polyhedral model can be used to reason about Affine loops:
  - Loops where the loop bounds and array references are affine functions of loop iterators and program parameters
- Affine function: linear + constant
  - Examples: 2*i+10, i+j+k, N*2+3
- Functions that are not affine
  - Examples: i*i, N*i

Affine loop examples

```
for (i = 0; i < M; i++) {
  for (j = 3*i; j < N/2; j++) {
      C[i][2*j] = C[M-1][j] + A[i][j+2];
    }
  }
}
```

Loop bounds: 3*i, N/2
Array access functions:
2*j, M-1, j+2

```
for (i = 0; i < M; i++) {
  for (j = 3*i*i; j < N/2; j++) {
      C[i][2*j] = C[M-1][j] + A[i][j+2];
    }
  }
}
```

j’s lower bound
3*i*i is not affine

Polyhedral model is broadly applicable

- Over 99% loops in a majority of HPC (High Performance Computing) programs are affine [1]
  - [1] C. Bastoul, A. Cohen, S. Girbal, S. Sharma, and O. Temam. Putting polyhedral loop transformations to work. In LCPC, 2003.
- Over 95% of loops in deep learning are affine [2]
  - [2] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA). IEEE, 1–12.

One model, many uses

- Loop transformations
  - Loop tiling
  - Loop peeling
  - Loop permutations
  - Loop reversal
  - Loop skewing
- Memory consumption optimization
  - Calculate memory consumption
  - Array contraction
- Parallelization
  - Determine which loops are parallel

Understanding the polyhedral model with an example

Sets

Sets. A set is a tuple of variables x_is along with a collection of constraints c_ks defined on the tuple variables. $s = \{[x_1, \ldots, x_n] : c_1 \wedge \ldots c_m\}$

Iteration space as a set in matrix multiplication

```
for (i = 0; i < M; i++) {
  for (j = 0; j < N; j++) {
    for (k = 0; k < K; k++) {
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
  }
}
```

Iteration domain as a set:

[M, N, K] ->

{ S[i, j, k] : 0 <= i < M and 0 <= j < N and 0 <= k < K; }

Relations

Relations. A relation is a mapping from input tuple variables x_is to output tuple variables y_js. In addition, a set of constraints c_ks can be defined for a relation that will place constraints on the input/output tuple variables. $r = \{[x_1, \ldots, x_n] \mapsto [y_1, \ldots, y_m] : c_1, \ldots, c_p\}$

Write access relation

```
for (i = 0; i < M; i++) {
  for (j = 0; j < N; j++) {
    for (k = 0; k < K; k++) {
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
  }
}
```

Write access relation:

writes := { S[i, j, k] -> C[i, j] }

Read access relations

```
for (i = 0; i < M; i++) {
  for (j = 0; j < N; j++) {
    for (k = 0; k < K; k++) {
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
  }
}
```

Read access relations:

reads := {S[i, j, k] -> B[k, j], S[i, j, k] -> A[i, k], S[i, j, k] -> C[i, j] }

Execution schedule as a relation

```
for (i = 0; i < M; i++) {
  for (j = 0; j < N; j++) {
    for (k = 0; k < K; k++) {
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
  }
}
```

Schedule

sched := { S[i, j, k] -> [i, j, k] };

Computing data dependences

```
for (i = 0; i < M; i++) {
  for (j = 0; j < N; j++) {
    for (k = 0; k < K; k++) {
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
  }
}
```

<u>Data dependences</u>

RAW := last *writes* before *reads* under sched;

Flow dependence (RAW – Read After Write dependence)

{ S[i, j, k] -> S[i' = i, j' = j, k' = 1 + k] }

Loop Transformations

Loop interchange in the Polyhedral model

```
for (i = 0; i < M; i++) {
  for (k = 0; k < K; k++) {
    for (j = 0; j < N; j++) {
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
  }
}
```

<u>Loop interchange</u>

sched := { S[i, j, k] -> [i, k, j] };

codegen (sched * I);

Loop transformations

- Loop transformations are performed
  - For better data cache locality
  - For better vectorization
- Loop transformations have to respect the data dependences
  - The resulting program should be functionally equivalent to the original program
  - The transformed program should produce the same results
  - Producer – consumer relations should be respected

Exercise

- How to tell if loop permutation is legal?
  - S[i, j, k] -> [i, k, j]
- Hint: examine the data dependences and formulate conditions based on them.

Loop Analysis

Apply operation

Apply operation. When a relation r is applied on a set s, the domain of r will be intersected with s and the resulting range will be a new set s'. The set s' is said to be the result of the apply operation. The operation is mathematically defined as: $(\vec{y} \in s') \iff (\exists \vec{x} \text{ s.t } (\vec{x} \in s \land \vec{x} \mapsto \vec{y}) \in r)$

Data footprint computation

```
for (i = 0; i < M; i++) {
  for (j = 0; j < N; j++) {
    for (k = 0; k < K; k++) {
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
    }
  }
}
```

<u>The number of array A elements accessed in the loop nest</u>

reads_A := [M, N, K] -> { S[i, j, k] -> A[i, k] };

I = { S[i, j, k] : 0 <= i < M and 0 <= j < N and 0 <= k < K; }

reads_A_set := reads_A(I);

Result: { A[i, k] : 0 <= i < M and 0 <= k < K }

Cardinality: M * K

Data footprint analysis

- Can be used to determine the unit of computation to hand over to an accelerator
  - E.g., the data accessed in the task should not exceed the available on-device memory size

Further reading/hands on experience

- The ISL (Integer Set Library) http://barvinok.gforge.inria.fr/
  - "iscc" tool is a command line facility for rapid exploration and prototyping
  - *iscc* operations on Page 15, Table 1: http://barvinok.gforge.inria.fr/barvinok.pdf
- *iscc* tutorial: http://barvinok.gforge.inria.fr/tutorial.pdf
- The *iscc* command lines used while preparing for this lecture are available in the accompanying material