The polyhedral model

Roel Jordans
Introduction

• Remember dependency checking and the iteration domain?

\[
\begin{align*}
& \text{DO } I = 1, 4 \\
& \quad \text{DO } J = 1, 4 \\
& \quad S_1 \quad A(I, J+1) = A(I-1, J) \\
& \quad \text{ENDDO} \\
& \text{ENDDO}
\end{align*}
\]
What can we do with it?

• Used for optimization passes
  – Each pass checks the **legality**
  – Analyzes **profitability**
  – Applies the **transformation**

• Things like
  – Auto vectorization
  – Loop unrolling

• *Fast results for specialized optimizations*
But in which order?

- Optimizations can enable other optimizations
- Iterative optimization
  - Finding the right sequence of optimizations
  - **Huge** exploration space
  - Extremely **slow**
  - Repeats the analysis for each step
- Can we combine steps?
for(i=0; i<3; ++i)
S: B[i] = f(A[i]);
for(i=0; i<3; ++i)
T: C[i] = g(B[2-i]);

Only few are legal!

6 statements
720 (6!) orderings

for(c=0; c<3; ++c){
  B[c] = f(A[c]);
  C[2-c] = g(B[c]);
}

{S[i]-> [i]}; {T[i]-> [2-i]}

{S[i]}, {T[i]}
The polyhedral model

- Focus on Static Control Parts (ScoP)
- The iteration space is usually quite regular
  - We can define an access function!
  - But arbitrary access functions may be a bit much
  - Use a polyhedral abstraction to represent program information
- Use iterative optimization techniques on the model
Advantages

• In the polyhedral model (*Feautrier, '92*)
  – Compositions of transformations are easily expressed
  – Transformation legality is easily checked
  – Natural expression of parallelism
ScoP restrictions

• Static control: *Control flow within the loop only depends on the loop iterators*
  – if(A[i] == X) …; ← *not allowed*
  – for(i=max(1,t-3); i < max(t-1, 3); i++) ← *allowed*

• Loops with (quasi-)affine access functions allow dense storage
  – Only a*i+b*j*c*k (with a, b, c integers)
  – No i*i, abs(i), ...
A three stage process

1 Analysis: from code to model

```
do i=1,3
do j=1,3
A(i+j) = ...
```

2 Transformation in the model
Here: $\Theta(i,j) = t = i+j$

3 Code generation: from model to code

```
do t=2,6
do i=max(1,t-3), min(t-1,3)
A(t) = ...
```
Extract the *Instance Set*

\[
\text{do } \text{i=}0, n
\]

\[
R \quad s(i) = 0
\]

\[
\text{do } \text{j=}0, n
\]

\[
S \quad s(i) = s(i) + a(i,j)x(j)
\]

\[
\text{end do}
\]

\[
\text{end do}
\]
Extract the *Instance Set*

```
do  i=0,n
    R  s(i) = 0
    do  j=0,n
        S  s(i) = s(i) + a(i,j)*x(j)
    end do
end do
```

**Iteration domain of \( R \)**

- **Iteration vector:** \( x_R = (i) \)
- **Exact set of instances of \( R \):** \( D_R : \{ i \mid 0 \leq i \leq n \} \)
Extract the *Instance Set*

\[\begin{align*}
\text{do } i &= 0, n \\
R & \quad s(i) = 0 \\
\text{do } j &= 0, n \\
S & \quad s(i) = s(i) + a(i,j) \times x(j) \\
\text{end do} \\
\text{end do}
\end{align*}\]

**Iteration domain of \( S \)**

- Iteration vector: \( \mathbf{x}_S = (i, j) \)
- Exact set of **instances** of \( S \):
  \[D_s : \{i,j \mid 0 \leq i \leq n, 0 \leq j \leq n\}\]
Scheduling a program

A schedule of a program is a function which associates a logical date (a timestamp) to each instance of each statement.

\[ \Theta_S(\vec{x}_S) = T \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix} \]

• Two instances having the same date can run in parallel
• Schedule dimensions corresponds to the number of nested sequential loops
Program transformations

• Every composition of loop transformations can be expressed as affine schedules (Wolf, '92)

• A schedule is the result of an arbitrary complex composition of transformations
A scheduling example

\[
\begin{align*}
\text{do } i &= 1,2 \\
\text{do } j = 1,3 \\
a(i,j) &= a(i,j) * 0.2
\end{align*}
\]

Original loop

\[
\Theta_R \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}
\]

\[
\begin{align*}
\text{do } i &= 1,2 \\
\text{do } j = 1,3 \\
a(i,j) &= a(i,j) * 0.2
\end{align*}
\]
A scheduling example

\[
\Theta_R \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} j \\ i \end{pmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix}
\]

Loop interchange

\[
\begin{align*}
do i &= 1, 2 \\
do j &= 1, 3 \\
a(i, j) &= a(i, j) \times 0.2
\end{align*}
\]

\[
\begin{align*}
do j &= 1, 3 \\
do i &= 1, 2 \\
a(i, j) &= a(i, j) \times 0.2
\end{align*}
\]
Finding legal schedules

Build the set of all *legal* program versions (i.e. which respects all the data dependence in the program)

• Perform an exact dependence analysis
• Build the set of all possible values of T

• The resulting space represents all the distinct possible ways to *legally reschedule* the program, using arbitrarily complex sequences of transformations
Dependence expression

• Need to represent the exact set of instances in dependence
• Exact computation made possible thanks to the ScoP and Static reference assumptions
• Use a subset of the Cartesian product of iteration domains

\[
\begin{align*}
\text{do } &i=0,n \\
s(i) &= 0 \\
\text{do } &j=0,n \\
 s(i) &= s(i) + a(i,j) \times x(j) \\
\text{end do} \\
\text{end do}
\end{align*}
\]
Dependence expression

\[
\begin{align*}
\text{do} & \quad i = 1, n \\
R & \quad s(i) = 0 \\
\text{do} & \quad j = 1, n \\
S & \quad s(i) = s(i) + a(i,j) * x(j) \\
\text{end do} & \\
\text{end do} & \\
\end{align*}
\]

\[
D_{R\delta S}:
\begin{bmatrix}
1 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 1 & 0
\end{bmatrix}
\begin{pmatrix}
\begin{bmatrix} i_R \\ i_S \\ j_S \\ n \end{bmatrix} \\
1
\end{pmatrix}
\]

Iterations of R
Dependence expression

\[
\begin{align*}
d & \text{ do } i = 0, n \\
R & \text{ s}(i) = 0 \\
& \text{ do } j = 0, n \\
S & \text{ s}(i) = \text{s}(i) + a(i,j) \times x(j) \\
& \text{ end do} \\
& \text{ end do}
\end{align*}
\]

\[
D_{R \delta S} = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & -1 \\
0 & -1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & -1 \\
0 & 0 & -1 & 1 & 0 \\
0 & 0 & 0 & -1 & 1 & 0
\end{bmatrix} \cdot \begin{bmatrix}
i_R \\
i_S \\
j_S \\
n \\
1
\end{bmatrix}
\]

Iterations of S
Dependence expression

\[
\begin{align*}
\text{do } & \ i = 0, n \\
R & \quad s(i) = 0 \\
\text{do } & \ j = 0, n \\
S & \quad s(i) = s(i) + a(i,j) \cdot x(j) \\
\text{end do} \\
\text{end do}
\end{align*}
\]

\[
D_{R\delta S} = \begin{bmatrix}
1 & -1 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
i_R \\
i_S \\
j_S \\
n \\
1
\end{bmatrix}
\]

S depends on R
Dependence expression

\[
\begin{align*}
d & \text{ do } i=0,n \\
R & \quad s(i) = 0 \\
& \text{ do } j=0,n \\
S & \quad s(i) = s(i) + a(i,j) \times x(j) \\
& \text{ end do} \\
& \text{ end do}
\end{align*}
\]

Complete result

\[
D_{R\delta S}:
\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & -1 \\
0 & -1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & -1 \\
0 & 0 & -1 & 1 & 0 \\
1 & -1 & 0 & 0 & 0 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
\dot{i}_R \\
\dot{i}_S \\
\dot{j}_S \\
\dot{n} \\
1 \\
\end{pmatrix}
\]
Definition: Legal schedule

Assuming $R \delta S$, schedules $\Theta_R(x_R)$ and $\Theta_S(x_S)$ are legal iff:

$$\Delta_{R,S} = \Theta_S(\vec{x}_S) - \Theta_R(\vec{x}_R) - 1$$

Is non-negative for each point in $D_{R \delta S}$.
Polyhedral tools

- **PET (Polyhedral Extraction Tool)**
  - Analysis only
  - Based on clang AST
- **Polly**
  - Analyzes and transforms LLVM IR
- **GRAPHITE**
  - Like Polly but works on GCC's GIMPLE
- **ISL (Integer Set Library)**
  - Set calculus for polyhedral analysis
  - Used by PET and Polly
Effects 1 (2011): Sequential

Small data size

Large data size
Effects 2: Parallel (24 thread)

Small data size

Large data size
Useful material

Tobias Grosser & Johannes Doerfert (2015) – *Polly: Optimistic Loop Nest Optimizations with Schedule Trees*
- Tutorial session from the LLVM developers' meeting
- https://www.youtube.com/watch?v=mIBUY20d8c8

Sven Verdoolage (2016) – *Presburger Formulas and Polyhedral Compilation*
- Tutorial style book with all the details if you really want to dive into this
- *Check chapter 5 for more info on today's lecture*

- More of the proofs presented today