Data Dependences

Sebastian Hack

http://compilers.cs.uni-saarland.de

Compiler Construction Core Course 2018
Saarland University
Value dependence:
- Determines which variables influence the value of a variable
- Here: $z$ depends on $x$ and $y$
- Foundation of slicing, non-interference, binding time, divergence analyses

Data dependence:
- Relates instructions in the program based on what storage they access
- Here: $C$ depends on $A$ and $B$
- Limits freedom how compiler can arrange code wrt a given storage allocation

```c
if (y < 0)
    x = 0;   // A
else
    x = 1;   // B
z = x + 1;   // C
```
### Data Dependence

$x \leftarrow 1$

$y \leftarrow 2$

$x \leftarrow x + y$

$y \leftarrow 3$

$z \leftarrow 4$

$y \leftarrow y + z$

$x \leftarrow x + y$

---

#### Definition

An instruction $B$ is data dependent on $A$ (write $B \rightarrow A$) if and only if

1. both access the same storage location $x$
2. there is a path from $A$ to $B$ and
   - one of them is a write and
   - the path contains no further write to $x$
An instruction $B$ is data dependent on $A$ (write $B \rightarrow A$) if and only if

1. both access the same storage location $x$
2. there is a path from $A$ to $B$ and
   - one of them is a write and
   - the path contains no further write to $x$
Definition
An instruction $B$ is data dependent on $A$ (write $B \rightarrow A$) if and only if

1. both access the same storage location $x$
2. there is a path from $A$ to $B$ and
   - one of them is a write and
   - the path contains no further write to $x$

Theorem
Any schedule that preserves dependences preserves the semantics of the program
There are three kinds of dependences:

- **RAW:** read after write
- **WAR:** write after read
- **WAW:** write after write

WAR and WAW are called false dependences.

True dependences express data flow.

False dependences are an artifact of storage allocation.
False dependences can be eliminated by providing unique storage for each computation, aka Static Single Assignment (SSA)

- Unifies variables and instructions
- Instruction is the variable
- Enables graph-based program representation
- All modern compilers use it
Data Dependence Graphs

\[
\begin{align*}
x_1 & \leftarrow 1 \\
y_1 & \leftarrow 2 \\
x_2 & \leftarrow x_1 + y_1 \\
y_2 & \leftarrow 3 \\
z_1 & \leftarrow 4 \\
y_3 & \leftarrow y_2 + z_1 \\
x_3 & \leftarrow x_2 + y_3
\end{align*}
\]
Scheduling Computations

- Storage assignment and parallelism influence each other
- More storage
  - less false dependences
  - more freedom
  - more parallelism
- Knowing dependences essential for the compiler to come up with “good” schedules
Scheduling Computations

Storage assignment and parallelism influence each other

More storage
  - less false dependences
  - more freedom
  - more parallelism

Knowing dependences essential for the compiler to come up with “good” schedules

Unfortunately: Finding schedule that maximizes parallelism and not exceeds storage bound is NP-hard
**Definition [recap]**

An instruction $B$ is data dependent on $A$ if and only if

1. both access the same storage location $x$
2. there is a path from $A$ to $B$ and
   - one of them is a write and
   - the path contains no further write to $x$

- What if it is not clear what $x$ is?
  - Here: Dependence if $a = b$

- Undecidable in general

- Compiler has to be conservative:
  Assume dependence exists
for \( i = 1 \) to 4  
for \( j = 1 \) to 4  
\[ X[i, j] = X[i-1, j-1] + X[i-1, j] \]

- Conceptually, unroll loops and construct dependence graph
- Not practical:
  - Loop bounds not constant
  - Graph too big
- We can do better if loops and subscripts are affine
- Relate instances of instructions given by iteration vector
- Represent dependences by polyhedra
for \( i = 1 \) to \( N \)
  for \( j = 1 \) to \( N \)
    \( X[i,j] = X[i, j-1] \quad \text{// } S \)
      + X[i-1, j-1]
      + X[i-1, j]

- Relate instances of instructions
- Instances described by iteration space polyhedron

\[
D_{S,S} \triangleq \begin{bmatrix}
1 & 0 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 \\
0 & -1 & 0 & 0 & 1 & 0 \\
\vdots
\end{bmatrix}
\cdot \begin{bmatrix}
[i] \\
[j] \\
[i'] \\
[j'] \\
[N] \\
[1]
\end{bmatrix}
\geq \vec{0}
\]}
Dependence Polyhedra

```plaintext
for i = 1 to N
    for j = 1 to N
        X[i, j] = X[i, j-1] // S
                    + X[i-1, j-1]
                    + X[i-1, j]
```

- Relate instances of instructions
- Instances described by iteration space polyhedron

Dependence polyhedron for accesses $X[i, j]$ and $X[i, j-1]$:

$$D_{S,S} = \begin{bmatrix}
1 & 0 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 \\
0 & -1 & 0 & 0 & 1 & 0 \\
\vdots
\end{bmatrix} \cdot \begin{bmatrix}
i \\
j \\
i' \\
j' \\
N \\
1
\end{bmatrix} = \vec{0} \geq 0$$
for i = 1 to N
    for j = 1 to N
        \( X[i,j] = X[i-1,j-1] + X[i-1,j] + X[i-1,j-1] \)

**Dependence Polyhedra**

- Relate instances of instructions
- Instances described by iteration space polyhedron

\[
\begin{bmatrix}
    1 & 0 & -1 & 0 & 0 & 0 \\
    0 & 1 & 0 & -1 & 0 & 1 \\
    1 & 0 & 0 & 0 & 0 & -1 \\
    -1 & 0 & 0 & 0 & 1 & 0 \\
    0 & 1 & 0 & 0 & 0 & -1 \\
    0 & -1 & 0 & 0 & 1 & 0 \\
    \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\end{bmatrix}
\begin{bmatrix}
    i \\
    j \\
    i' \\
    j' \\
    N \\
    1 \\
\end{bmatrix}
\geq \begin{bmatrix}
    \tilde{0} \\
\end{bmatrix}
\]
for i = 1 to N
  for j = 1 to N
    X[i,j] = X[i ,j-1] // S
    + X[i-1,j-1]
    + X[i-1,j]

Dependence polyhedron for accesses
X[i,j] and X[i,j-1]:

\[ DS,S \triangleq \begin{bmatrix}
  1 & 0 & -1 & 0 & 0 & 0 \\
  0 & 1 & 0 & -1 & 0 & 1 \\
  1 & 0 & 0 & 0 & 0 & -1 \\
  -1 & 0 & 0 & 1 & 0 \\
  0 & 1 & 0 & 0 & 0 & -1 \\
  0 & -1 & 0 & 0 & 1 & 0 \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots 
\end{bmatrix} \begin{bmatrix}
  i \\
  j \\
  i' \\
  j' \\
  N \\
  1 
\end{bmatrix} \geq \begin{bmatrix}
  0 \\
  0 \\
  \vec{0} \\
  \vec{0} \\
  \vec{0} \\
  \vec{0} \\
\end{bmatrix} \]

- Relate instances of instructions
- Instances described by iteration space polyhedron
for \( i = 1 \) to \( N \)
for \( j = 1 \) to \( N \)
\[
X[i,j] = X[i,j-1] \quad \text{// } S
+ X[i-1,j-1]
+ X[i-1,j]
\]

Dependence polyhedron for accesses \( X[i,j] \) and \( X[i,j-1] \):

\[
D_{S,S} \triangleq \begin{bmatrix}
1 & 0 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
\vdots
\end{bmatrix} \cdot \begin{bmatrix}
i \\
j \\
i' \\
j' \\
N \\
1
\end{bmatrix} \geq \vec{0}
\]

- Relate instances of instructions
- Instances described by iteration space polyhedron
**Dependence Polyhedra**

```
for i = 1 to N
  for j = 1 to N
    X[i,j] = X[i,j-1] // S
    + X[i-1,j-1]
    + X[i-1,j]
```

Dependence polyhedron for accesses $X[i,j]$ and $X[i,j-1]$:

$$D_{S,S} \triangleq \begin{bmatrix}
1 & 0 & -1 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 \\
0 & -1 & 0 & 0 & 1 & 0 \\
\vdots
\end{bmatrix} \cdot \begin{bmatrix}
i \\
j \\
i' \\
j' \\
N \\
1
\end{bmatrix} = \vec{0} \geq \vec{0}
$$

- Loop transformations (schedules) described by affine functions $\Theta_S$ for each statement $S$
- Affine schedules $\Theta_S, \Theta_T$ valid iff for all $\vec{x}, \vec{y} \in D_{S,T}$: $\Theta_T(\vec{x}) > \Theta_S(\vec{y})$
- Via Farkas’ Lemma, we get an affine space of all valid schedules
- Use linear programming to find a “good” one
Affine Schedules

Original Schedule

### Inherently sequential

\[ \Theta \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} i \\ j \end{pmatrix} \]

Optimized Schedule

### Parallelism along the \( j \) dimension

\[ \Theta \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} i + j \\ j \end{pmatrix} \]