Data Dependences

Sebastian Hack

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

Compiler Construction Core Course 2018
Saarland University

Dependence

```
if (y < 0)
    x = 0;   // A
else
    x = 1;   // B

z = x + 1;   // C
```

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

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

Data Dependence

$$\begin{aligned}
x &\leftarrow 1 \\
y &\leftarrow 2 \\
x &\leftarrow x + y \\
y &\leftarrow 3 \\
z &\leftarrow 4 \\
y &\leftarrow y + z \\
x &\leftarrow x + y
\end{aligned}$$

Definition

An instruction *B is data dependent on A* (write $B \to A$) 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

Data Dependence

$$
\begin{aligned}
x &\leftarrow 1 \\
y &\leftarrow 2 \\
x &\leftarrow x + y \\
y &\leftarrow 3 \\
z &\leftarrow 4 \\
y &\leftarrow y + z \\
x &\leftarrow x + y
\end{aligned}
$$

Definition

An instruction B is data dependent on A (write $B \to A$) 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

True and False Dependences

$$
\begin{aligned}
x &\leftarrow 1\\
y &\leftarrow 2\\
x &\leftarrow x + y\\
y &\leftarrow 3\\
z &\leftarrow 4\\
y &\leftarrow y + z\\
x &\leftarrow x + y
\end{aligned}
$$

- 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

Removing False Dependences, SSA

$$
\begin{aligned}
x_1 &\leftarrow 1 \\
y_1 &\leftarrow 2 \\
x_2 &\leftarrow x_1 + y_1 \\
y_2 &\leftarrow 3 \\
z_1 &\leftarrow 4 \\
y_3 &\leftarrow y_2 + z_1 \\
x_3 &\leftarrow x_2 + y_3
\end{aligned}
$$

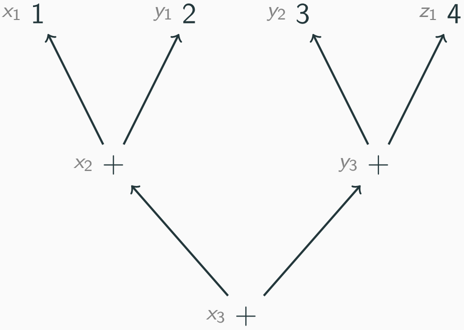
False dependences can be eliminated by providing unique storage for each computation, aka
Static Single Assignment (SSA)

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

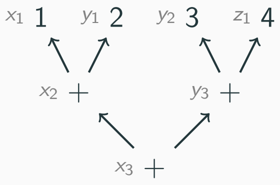
Data Dependence Graphs

$$
\begin{aligned}
x_1 &\leftarrow 1\\
y_1 &\leftarrow 2\\
x_2 &\leftarrow x_1 + y_1\\
y_2 &\leftarrow 3\\
z_1 &\leftarrow 4\\
y_3 &\leftarrow y_2 + z_1\\
x_3 &\leftarrow x_2 + y_3
\end{aligned}
$$

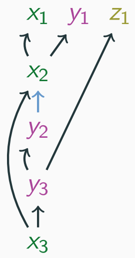
Data Dependence Graph



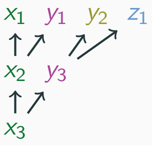
Scheduling Computations



Schedule 1:
Storage = 3
Latency = 5

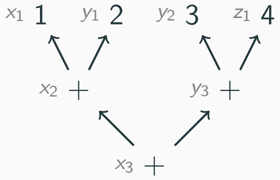


Schedule 2:
Storage = 4
Latency = 3

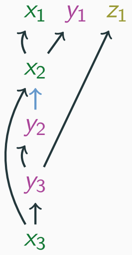


- Storage assignment and parallelism influence each other
- More storage
  - less false dependences
  - more freedom
  - more parallelism
- Knowing dependences essential for the compiler to come up with "good" schedules

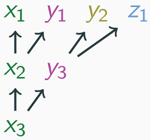
Scheduling Computations



Schedule 1:
Storage = 3
Latency = 5



Schedule 2:
Storage = 4
Latency = 3



- 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

Memory

$$
\begin{aligned}
x_1 &\leftarrow ? \\
M[a] &\leftarrow x_1 + 1 \\
x_2 &\leftarrow x_1 * 2 \\
& \quad a = b? \\
z_1 &\leftarrow M[b] \\
& \quad \vdots
\end{aligned}
$$

Definition [recap]

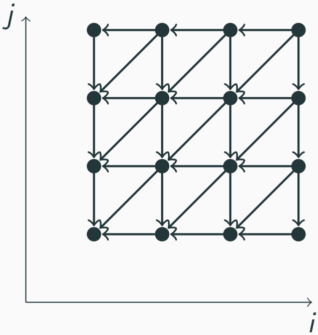
An instruction B is data dependent on A if and only if

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

Dependence Analysis in Loops

```
for i = 1 to 4
  for j = 1 to 4
    X[i,j] = X[i  ,j-1]
           + 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

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]
```

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

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

$$D_{S,S} \triangleq \left[\begin{array}{rrrrrr} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ \vdots & & & & & \end{array}\right] \cdot \begin{bmatrix} i \\ j \\ i' \\ j' \\ N \\ 1 \end{bmatrix} \begin{array}{c} = \vec{0} \\ \hline \geq \vec{0} \end{array}$$

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]`:

$$D_{S,S} \triangleq \left[\begin{array}{cccccc} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ \vdots & & & & & \end{array}\right] \cdot \begin{bmatrix} i \\ j \\ i' \\ j' \\ N \\ 1 \end{bmatrix} \frac{= \vec{0}}{\geq \vec{0}}$$

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

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]
```

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

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

$$D_{S,S} \triangleq \left[\begin{array}{cccccc} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ \vdots & & & & & \end{array}\right] \cdot \begin{bmatrix} i \\ j \\ i' \\ j' \\ N \\ 1 \end{bmatrix} \frac{= \vec{0}}{\geq \vec{0}}$$

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]`:

$$
D_{S,S} \triangleq \left[\begin{array}{cccccc} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ \vdots & & & & & \end{array} \right] \cdot \begin{bmatrix} i \\ j \\ i' \\ j' \\ N \\ 1 \end{bmatrix} \begin{array}{c} = \vec{0} \\ \hline \geq \vec{0} \end{array}
$$

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

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]`:

$$D_{S,S} \triangleq \left[\begin{array}{cccccc} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ \vdots & & & & & \end{array}\right] \cdot \begin{bmatrix} i \\ j \\ i' \\ j' \\ N \\ 1 \end{bmatrix} \begin{array}{c} = \vec{0} \\ \hline \geq \vec{0} \end{array}$$

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

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]
```

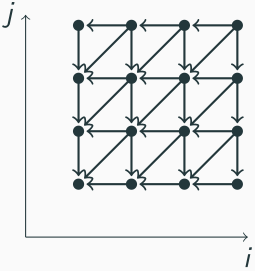
Dependence polyhedron for accesses `X[i,j]` and `X[i,j-1]`:

$$D_{S,S} \triangleq \left[\begin{array}{cccccc} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ \vdots & & & & & \end{array}\right] \cdot \begin{bmatrix} i \\ j \\ i' \\ j' \\ N \\ 1 \end{bmatrix} \begin{array}{c} = \vec{0} \\ \hline \geq \vec{0} \end{array}$$

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

Affine Schedules

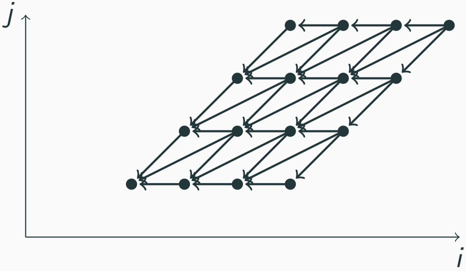
Original Schedule



$$\Theta \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} i \\ j \end{pmatrix}$$

Inherently sequential

Optimized Schedule



$$\Theta \begin{pmatrix} i \\ j \end{pmatrix} = \begin{pmatrix} i + j \\ j \end{pmatrix}$$

Parallelism along the j dimension