TU Wien

Institute of Information Systems Engineering
Research Division of Compilers and Languages

FAKULTÄT FÜR INFORMATIK
Faculty of Informatics

computer languages

Übersetzer für Parallele Systeme
Compilers for Parallel Systems

LVA 185.A64
SS 2018

Hans Moritsch
hans.moritsch@tuwien.ac.at

Data Dependence

* When may we execute statements
  * in a different order than specified in the sequential program,
  * or concurrently,
  * without changing program semantics ?
* Most important source of performance: loops
* Dependence relation specifies semantically relevant constraints on statement execution order

Data Dependence (2)

S_1 `a = 1.0`

***true** dependence*, change order $\Rightarrow$ S_2 reads old value of a

S_2 `b = a+3.14159`

***anti** dependence*, change order $\Rightarrow$ S_2 reads value written by S_3

S_3 `a = 1/3*(c-d)`

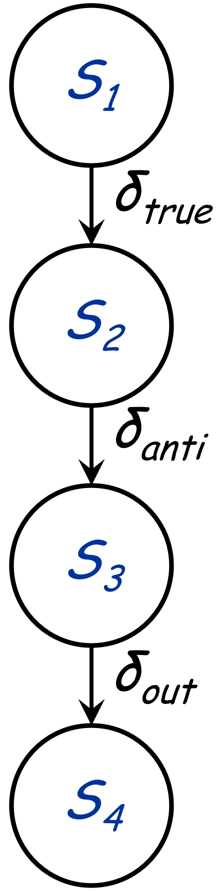
***output** dependence*, change order $\Rightarrow$ wrong values read after S_4

`...`

S_4 `a = (b*3.8)/2.8284`

- Dependences have direction: from statement to statement
- True dependences exist also in functional languages
- Anti- and output dependences due to imperative languages
  - reuse of memory via variables
  - no actual data flow
  - can be eliminated by introducing new variables

Dependence Graph




```
a = 1.0
b = a+3.14159
a = 1/3*(c-d)
a = (b*3.8)/2.8284
```

Finding Dependences

* Example 1

```
for i=1 to 100
    S  a[2*i]=b[i]+1       even elements
    S' d[i]=a[2*i+1]       odd elements
end for
```

  * no dependence
  * have to look at array indices
  * equation: $2x = 2y+1$, $1 \le x,y \le 100$
    * no solution

* Example 2
  * for i=2 to 10

```
    S  a[i]=b[i]+1
    S' d[i]=a[i-1]
end for
```

  * true dependence
  * equation: $x = y-1$, $2 \le x,y \le 10$
  * 8 solutions

```
i=2:
a[2]=b[2]+1
d[2]=a[1]

i=3:
a[3]=b[3]+1
d[3]=a[2]

i=4:
a[4]=b[4]+1
d[4]=a[3]
```

i	i-1
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9

x	y
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

Terminology
Vectors

* Vector $\mathbf{x} \in Z^n$
* $\mathbf{x} = [x_1 : x_n]$ … n elements
* Relation $<_c$
  * $\mathbf{x},\mathbf{y} \in Z^n$, $c \in [1:n]$
* $\mathbf{x} <_c \mathbf{y} \Leftrightarrow (\forall j \in [1:c-1]: x_j = y_j) \wedge (x_c < y_c)$
  * e.g. $[5,3,2,8] <_3 [5,3,4,1]$
* Properties
  * irreflexive partial order
  * for every pair $\mathbf{x},\mathbf{y} \in Z^n$
    * either $\mathbf{x} = \mathbf{y}$, or
    * $\exists\ c \in [1:n]$ such that either $\mathbf{x} <_c \mathbf{y}$ or $\mathbf{y} <_c \mathbf{x}$
  * for every pair $\mathbf{x},\mathbf{y},\mathbf{z} \in Z^n$
    * $\mathbf{x} <_c \mathbf{y} \wedge \mathbf{y} <_{c'} \mathbf{z} \Rightarrow \mathbf{x} <_{c''} \mathbf{z}$ with $c''=\min(c,c')$
* Lexicographical order $<$
  * $\mathbf{x} < \mathbf{y} \Leftrightarrow \exists\ c \in [1:n]$ such that $\mathbf{x} <_c \mathbf{y}$
  * irreflexive linear order
* Used to describe iteration variables in nested loops

Loops

```
for <loop variable> = <lower bound> to <upper bound>
   <body>
end forall
```

- Loop variable
  - integer
  - local to loop
  - must not be written in body
- Bounds are integer expressions
  - lower bound > upper bound ⇒ no execution of loop body
- Loop increment is 1 !
- Normalized loop: lower bound is 1
- Body is sequence of statements
  - assignment
  - loop

Nested Loops

```
L L1 for I1 = lb1 to ub1
       ...
     L2 for I2 = lb2 to ub2
          ...
        Ln for In = lbn to ubn
             ...
              S
             ...
             end for
          ...
          end for
       ...
       end for
```

- Loop nest L
  - n loops L_i at level i
- Perfect loop nest
  - body of L_i is L_{i+1}
- Iteration space of a loop
  - iteration vectors of all iterations

- Vector of loop variables $\boldsymbol{I} = (I_1, \ldots, I_n)$
- Iteration vector $\mathbf{i} = (i_1, \ldots, i_n) \in \mathbb{Z}^n$
  - values of all loop variables for a particular execution of S
  - defines an iteration of L … iteration state
- Execution index set $[S]$ of S: set of all iteration vectors

Statement Instances

* Statement instance $S(\mathbf{i})$
  * execution of statement S in iteration $\mathbf{i}$, $\mathbf{i} \in [S]$
* Execution order
  * irreflexive partial order within statement instances
* Standard execution order «
  * of a sequential imperative programming language
  * e.g. matrix multiplication
    * $[S] = [1{:}100]^3$
    * $S(i,j,k) \ll S(i',j',k')$ iff $(i,j,k) < (i',j',k')$
* Control set $[S,S']$ of two statements
  * which iteration vectors cause S' to be executed after S?
  * set of all iteration vector pairs respecting « … plausible pairs
  * $[S,S'] = \{(\mathbf{i},\mathbf{i}') \in [S]\times[S']: S(\mathbf{i}) \ll S'(\mathbf{i}')\}$
* S bef S'
  * S occurs (in the program code) textually before S'

Nested Loops (2)

```
L L1 L'1 for I1 = lb1 to ub1
            ...
      L3 L'3 for I3 = lb3 to ub3
                ...
         L4 for I4 = lb4 to ub4
                   ...
                   S
                   ...
                end for
                ...
         L'4 for I'4 = lb'4 to ub'4
                   ...
                   S'
                   ...
                end for
                ...
            end for
            ...
        end for
```

- Maximum common loop index
  - level of innermost loop enclosing both S and S'
  - $m := \max\{k : L_k = L'_k\}$
  - example: m=3
  - iteration vector prefix \i = i[1:m]

Nested Loops (3)

```
L L1 L'1 for I1 = lb1 to ub1
           ...
      L3 L'3 for I3 = lb3 to ub3
                ...
           L4 L'4 for I4 = lb4 to ub4
                     ...
                     S1
                     ...
                     S2
                     ...
                     S3
                     ...
                  end for
                ...
              end for
           ...
         end for
```

- Relative order of two statement executions determined by lexicographical order of iteration vectors and textual order of statements
- Iteration vectors are relevant up to maximum common loop index *m* only

$$S(\mathbf{i}) \ll S'(\mathbf{i}') \Leftrightarrow$$
$$(\backslash \mathbf{i} < \backslash \mathbf{i}') \vee (\backslash \mathbf{i} = \backslash \mathbf{i}' \wedge S\ \underline{\text{bef}}\ S')$$

Abstractions of the Control Set

* Control set represents complete information about relative order of statement executions
  * in terms of (a lot of) iteration vector pairs
  * data set possibly too large to be handled by the compiler
  * not always known at compile time
  * need for abstractions ... loss of precision
* Need for abstractions
  * distance
  * direction
  * $<_c$ relation

Distance and Direction Vectors

- Let $(\mathbf{i},\mathbf{i}') \in [S] \times [S]$ ... all possible combinations
- Distance vector $dist(\mathbf{i},\mathbf{i}') := \backslash\mathbf{i}' - \backslash\mathbf{i}$ of length m
- Direction vector $dir(\mathbf{i},\mathbf{i}')$ of length m
  - vector of relation symbols {<,=,>,*}
  - wildcard * for classes of direction vectors
    - e.g. {[=,<,*]} = {[=,<,<], [=,<,=], [=,<,>]}
  - e.g. matrix multiplication
    dir([S,S])={(<,*,*)} ∪ {(=,<,*)} ∪ {(=,=,<)}

$dist_j$	dir_j
> 0	<
= 0	=
< 0	>

- $dir([S,S'])$ ... plausible direction vectors
  - {(<,*,...,*)} ∪ {(=,<,*,...,*)} ∪ {(=,=,<,*,...,*)} ∪ {(=,=,...,=,<)}
  - if S bef S': ∪ {(=,=,...,=,=)}
- $dir(<_3)$:= {(=,=,<,*,...,*)}
  - same iteration on levels < 3
  - earlier iteration on level 3

Extension to sets $M \subseteq [S] \times [S']$
$dist(M) := \{dist(\mathbf{i},\mathbf{i}') : (\mathbf{i},\mathbf{i}') \in M\}$
$dir(M) := \{dir(\mathbf{i},\mathbf{i}') : (\mathbf{i},\mathbf{i}') \in M\}$

Data Dependence

- Is there a pair $(i,i') \in [S,S']$ such that
  - both S and S' access the same variable
  - and at least one write access ?
- Input set of a statement $USE(S)$
  - all variables read in S
  - incl. array elements ... subscripted variables
- Output set of a statement $DEF(S)$
  - all variables written in S
- Subscripted variables $a[exp_1, \ldots, exp_n]$
  - variables occuring in exp_i belong to $USE(S)$
  - no function calls in exp
- Admissible variables
  - loop variables
  - subscripted variables without function calls

Input and Output Sets

- Example

```
for i=1 to 100
    S   a[2*i]=b[i]+1
    S'  d[i]=a[2*i+1]
end for
```

  - *USE*(*S*) = { b[i],i} *DEF*(*S*) = { a[2*i] }
  - *USE*(*S*') = { a[2*i+1],i } *DEF*(*S*') = { d[i] }

- Input and output execution sets

```
for i=1 to 100
    S x[i+1] = x[i] + y[i]
end for
```

  - *USE*(*S*) = { x[i],y[i],i } *DEF*(*S*) = { x[i+1] }
  - for statement instance *S*(10)

    USE(*S*(10)) = { x[10],y[10]),i } *DEF*(*S*(10)) = { x[11] }

Definition of Data Dependence

- S, S' statements in a loop
- **i**, **i'** iteration vectors
- Relation between statement instances $S(\mathbf{i})$ and $S'(\mathbf{i'})$
  - **i**,**i'** ... iteration vectors associated with the dependence

$S'(\mathbf{i'})$ is data dependent on $S(\mathbf{i})$: $S(\mathbf{i})\ \delta\ S'(\mathbf{i'}) \Leftrightarrow$

- (DEP-1) $S(\mathbf{i})$ must be executed before $S'(\mathbf{i'})$: $S(\mathbf{i}) \ll S'(\mathbf{i'})$
- (DEP-2) there is a variable v causing a dependence
  - $v \in DEF(S(\mathbf{i})) \cap USE(S'(\mathbf{i'})) \vee$ (true)
  - $v \in USE(S(\mathbf{i})) \cap DEF(S'(\mathbf{i'})) \vee$ (anti)
  - $v \in DEF(S(\mathbf{i})) \cap DEF(S'(\mathbf{i'}))$ (output)
  - $v \in USE(S(\mathbf{i})) \cap USE(S'(\mathbf{i'}))$ (input)
- (DEP-3) there is no statement instance $\hat{S}$ which covers the dependence
  - $S(\mathbf{i}) \ll \hat{S} \ll S'(\mathbf{i'})$ and $v \in DEF(\hat{S})$

Characterization of Dependences

- Dependence distance
  - $S(\mathbf{i})\ \delta^{d}\ S'(\mathbf{i}')$ ... $S(\mathbf{i}) \ll^{d} S'(\mathbf{i}')$ with $d = dist(\mathbf{i},\mathbf{i}')$
- Dependence direction
  - $S(\mathbf{i})\ \delta^{\theta}\ S'(\mathbf{i}')$ ... $S(\mathbf{i}) \ll^{\theta} S'(\mathbf{i}')$ with $\theta = dir(\mathbf{i},\mathbf{i}')$
- Loop carried dependence at level c
  - $S(\mathbf{i}) <_c S'(\mathbf{i}')$ ... $\backslash\mathbf{i} <_c \backslash\mathbf{i}'$
  - originating from previous iterations
- Loop independent dependence ... level ∞
  - $S(\mathbf{i}) <_\infty S'(\mathbf{i}')$ ... $\backslash\mathbf{i} = \backslash\mathbf{i}'$ and S <u>bef</u> S'
  - within same iteration

Dependent Statements

* Generalization: statement instances $\Rightarrow$ statements
  * $\exists$ **i**,**i**′ with S(**i**) « S(**i**′) $\Leftrightarrow$ S « S′
* S′ is (data) dependent on S
  * $\exists$ **i**,**i**′ with S(**i**) δ S'(**i**′) $\Leftrightarrow$ S δ S′
* Dependence relation is not transitive

S_1 **a** = b + 1
S_2 **b** = **a** * 2
S_3 c = **b** – 1

  * S_3 is not dependent on S_1
  * different variables, iteration vectors, dependence types
  * but restrictions wrt. execution order are transitive

Example

```
for i=1 to 100
    S  x[i+1]=x[i]+y[i]
end for
```

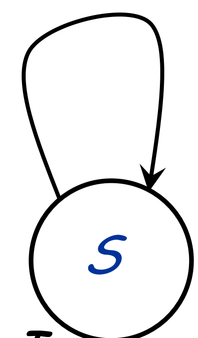
$USE(S(i)) = \{\, x[i], y[i], i \,\}$ $\quad$ $DEF(S(i)) = \{\, x[i+1] \,\}$

- DEP-1 holds, iff $1 \le i < i' \le 100$
- DEP-2

$v \in DEF(S(i)) \cap USE(S'(i')) \vee$
$v \in USE(S(i)) \cap DEF(S'(i')) \vee$
$v \in DEF(S(i)) \cap DEF(S'(i'))$

$\delta^{(<)}_{1,true}$

$v \in \{\, x[i+1] \,\} \cap \{\, x[i'], y[i'], i \,\} \vee$ $\quad$ holds with $i' = i + 1$
$v \in \{\, x[i], y[i], i \,\} \cap \{\, x[i'+1] \,\} \vee$
$v \in \{\, x[i+1] \,\} \cap \{\, x[i'+1] \,\}$



- DEP-3 holds: iteration i' immediately follows iteration I

$\Rightarrow$ Dependence $S(i)\ \delta_{true}\ S(i')$ exists for all $1 \le i < i' \le 100$, $i' = i + 1$