# CS6013 - Modern Compilers: Theory and Practise

## Dependence testing

**V. Krishna Nandivada**

IIT Madras

## Example dependence testing

```
for i ← 1 to 4 do
   b[i] ← a[4*i] + 2.0
   a[2*i+1] ← 1.0/i
endfor
for i ← 1 to 4 do
   b[i] ← a[3*i-5] + 2.0
   a[2*i+1] ← 1.0/i
endfor
```



## linear Diophantine equation

$$a1 * x1 + a2 * x2 + \cdots an * xn = c$$

has an integer solution for $x1, x2, ..,$ iff

GCD $(a1, a2, \cdots an)$ divides $c$.



## GCD test - intuition

- A simple and sufficient test
- if a loop carried dependency exists between $X[a*i+b]$ and $X[c*i+d]$, then GCD $(c, a)$ must divide $(d-b)$.

## GCD Test Generalization

```
for i1 ← 1 to hi1 do
  for i2 ← 1 to hi2 do
    ...
      for in ← 1 to hin do
        ...
        ··· x[..., a0 + a1 * i1 + ··· + an * in, ...] ···
        ...
        ··· x[..., b0 + b1 * i1 + ··· + bn * in, ...] ···
        ...
      endfor
    ...
  endfor
endfor
```

- may be accessed inside loop nest using indices of multiple loops.
- Array may be multi-dimensional.
- Dependence present iff, for each subscript position in the equation

$$a_0 + \sum_{j=1}^{n} a_j * i_{j_1} = b_0 + \sum_{j=1}^{n} b_j * i_{j_2}$$

and the following inequalities are satisfied:

$$\forall j = 1 \cdots n$$
$$1 \le i_{j_1} \le hi_j$$
$$1 \le i_{j_2} \le hi_j$$



## GCD Test formula

- Developed by Utpal Bannerjee and Robert Towle (1976).
- Comparatively weak test (Marks too many accesses as dependent).
- If for any one subscript position

$$GCD\left(\bigcup_{j=1}^{n} Sep(a_j, b_j, j)\right) \neg/ \sum_{j=0}^{n}(a_j - b_j)$$

where

  - GCD - computes the Greatest common divisor for the set of numbers.
  - "$a \neg/ b$" means that $a$ does not divide $b$.
  - $$Sep(a,b,j) = \begin{cases} \{a-b\} & \text{looking for intra iteration dependence} \\ \{a,b\} & \text{otherwise} \end{cases}$$

then the two references to the array x are independent.
- Other words: dependence ⇒ GCD divides the sum.



## GCD test for loops with arbitrary bounds

Say the loops are not canonical, but are of the form:

for $i_j \leftarrow lo_j$ by $inc_j$ to $hi_j$

$$GCD\left(\bigcup_{j=1}^{n} Sep(a_j * inc_j, b_j * inc_j, j)\right) \neg/ a_0 - b0 + \sum_{j=0}^{n}(a_j - b_j) * lo_j$$



## Dependence testing based on separability

- A pair of array references is separable if in each pair of subscript positions, the expressions found are of the form: $a * x + b1$ and $a * x + b2$.
- A pair of array references is weakly separable if in each pair of subscript positions, the expressions found are of the form: $a1 * x + b1$ and $a2 * x + b2$.

## Dependence testing for separable array references

If the two array references are separable, then dependence exists if

- $a = 0$ and $b1 = b2$ or
- $(b1 - b2)/a \leq hi_j$



## Dependence testing for weakly separable array references

- For each subscript position, we have equations of the form: $a1 * y + b1 = a2 * x + b2$, or $a1 * y = a2 * x + (b2 - b1)$
- Dependence exists if for a particular value of $j$ has a solution that satisfies inequalities given by the loop bounds of loop $j$.
- List all such constraints for each reference.
- For any given reference if there is only one equation:
  - Say it is given by: $a1 * y = a2 * x + (b2 - b1)$
  - One linear equation, two unknowns:
    Solution exists iff $GCD(a1, a2)\%(b2 - b1) = 0$



## Dependence testing for weakly separable array references (contd)

- If the set of equations has two members of the form

  $$a_{11} * y = a21 * x + (b21 - b11)$$
  $$a_{12} * y = a22 * x + (b22 - b12)$$

  Two equations and two unknowns.
  If $a21/a11 = a22/a12$ then rational solution exists: iff $b21 - b11)/a11 = (b22 - b12)/a12$.
  If $a21/a11 \neq a22/a12$ then there is one rational solution.
  Once we obtain the solutions, check that they are integers and inequalities are satisfied.
- If set of equations have $n$ $(> 2)$ members, either $n - 2$ are redundant $\rightarrow$ use previous methods.
  Else we have more equations compared to the unknowns $\rightarrow$ overdetermined.



## Example: analyzing weak separable references

```
for i ← 1 to n do
   for j ← 1 to n do
      f[i] ← g[2*i,j] + 1.0
      g[i+1,3*j] ← h[i,i] - 1.5
      h[i+2,2*i-2] ← 1.0/i
   endfor
endfor
```

# Closing remarks

What did we do today?

- Dependence testing.