Data Dependences and Parallelization

Agenda

- *Introduction*
- Single Loop
- Nested Loops
- Data Dependence Analysis

Motivation

- DOAL loops: loops whose iterations can execute in parallel

```
for i = 11, 20
  a[i] = a[i] + 3
```

- New abstraction needed
  - Abstraction used in data flow analysis is inadequate
    - Information of all instances of a statement is combined

Examples

```
for i = 11, 20
  a[i] = a[i] + 3
```

Parallel

```
for i = 11, 20
  a[i] = a[i-1] + 3
```

Parallel?

Examples

```
for i = 11, 20
  a[i] = a[i] + 3
```

Parallel

```
for i = 11, 20
  a[i] = a[i-1] + 3
```

Not parallel

```
for i = 11, 20
  a[i] = a[i-10] + 3
```

Parallel?

Agenda

- Introduction
- *Single Loop*
- Nested Loops
- Data Dependence Analysis

Data Dependence of Scalar Variables

- True dependence

  a =

  = a

- Anti-dependence

  = a

  a =

- Output dependence

  a =

  a =

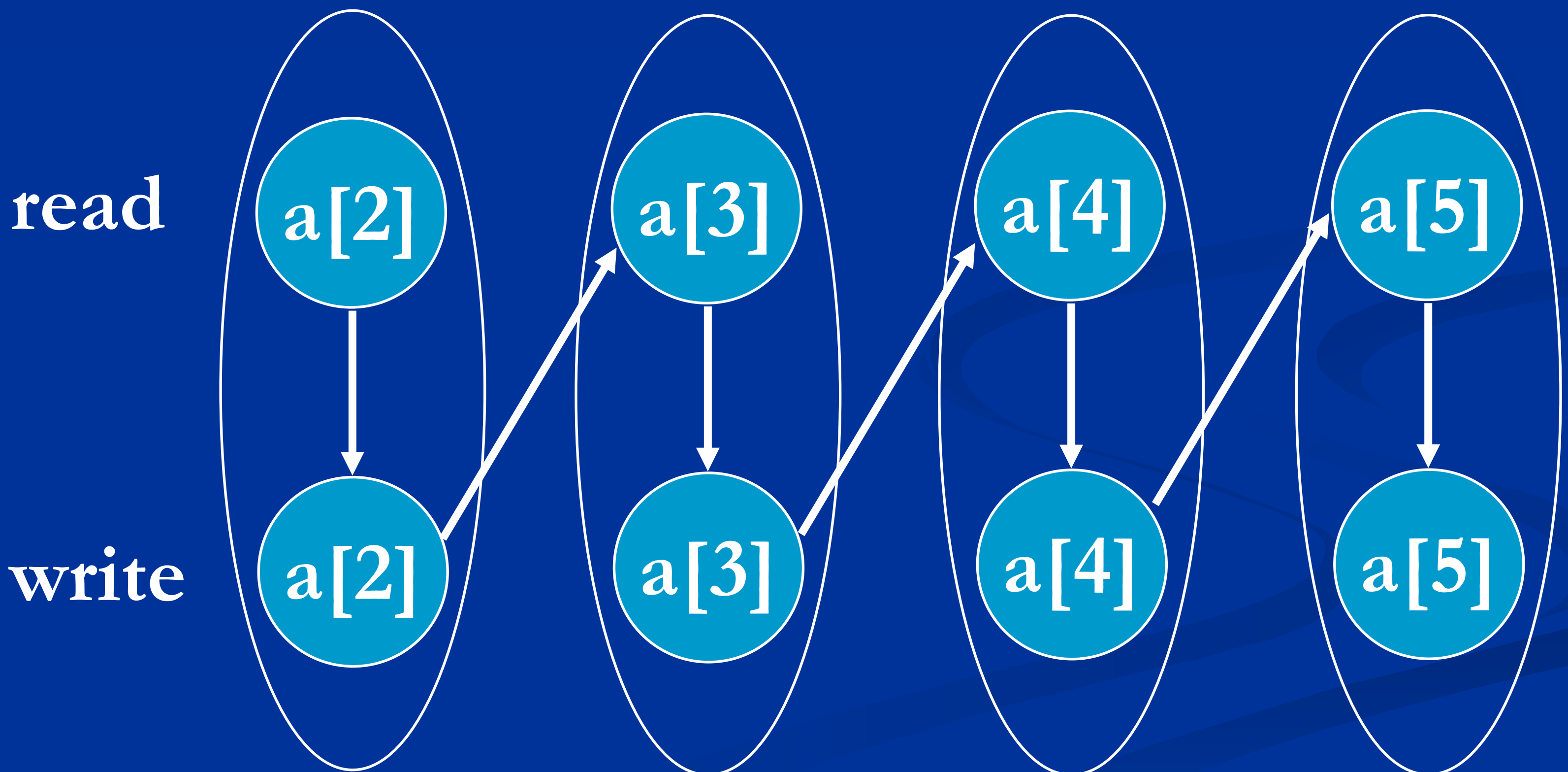
- Input dependence

  = a

  = a

Array Accesses in a Loop

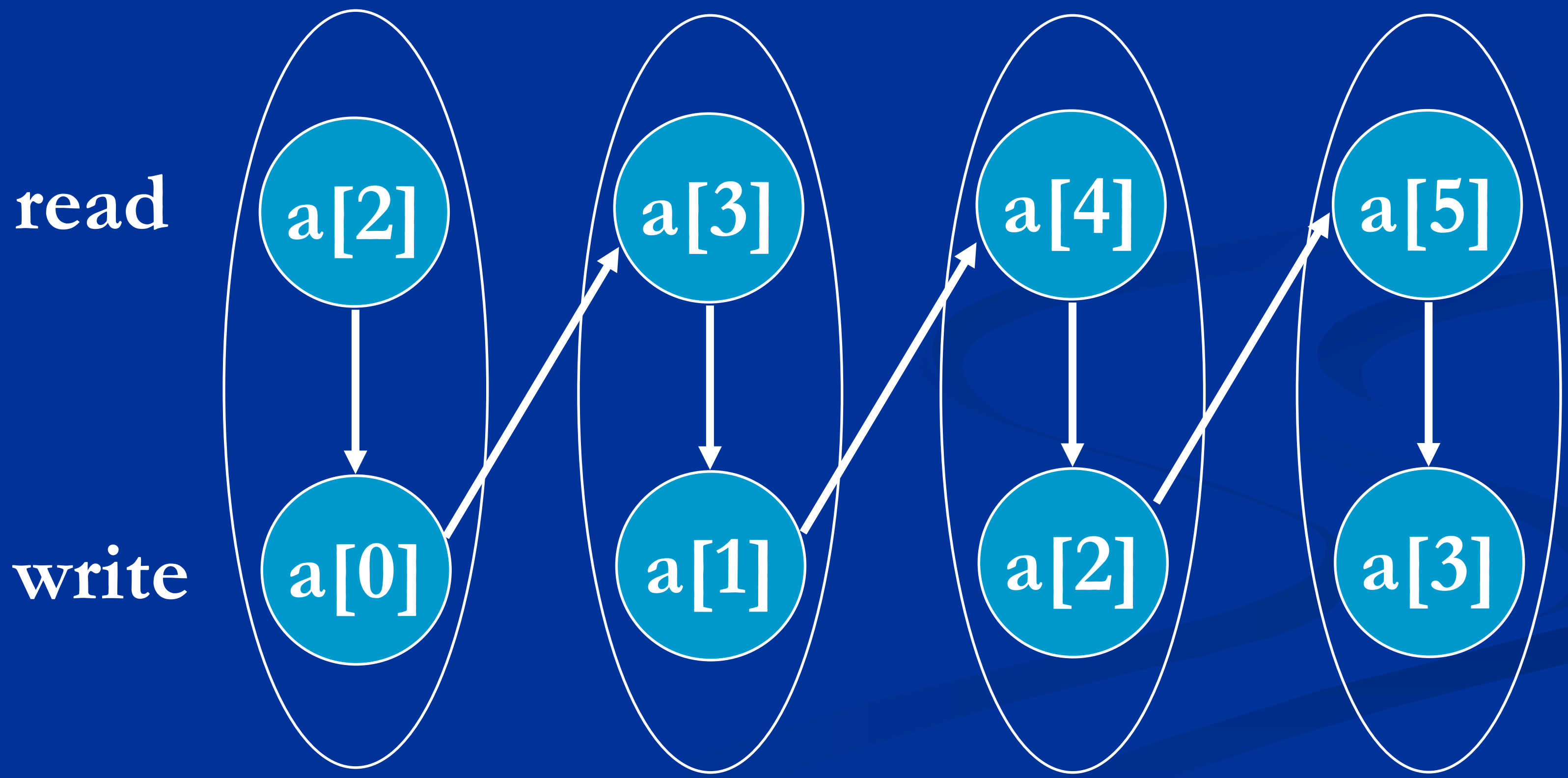
for i= 2, 5
  a[i] = a[i] + 3



Array Anti-dependence

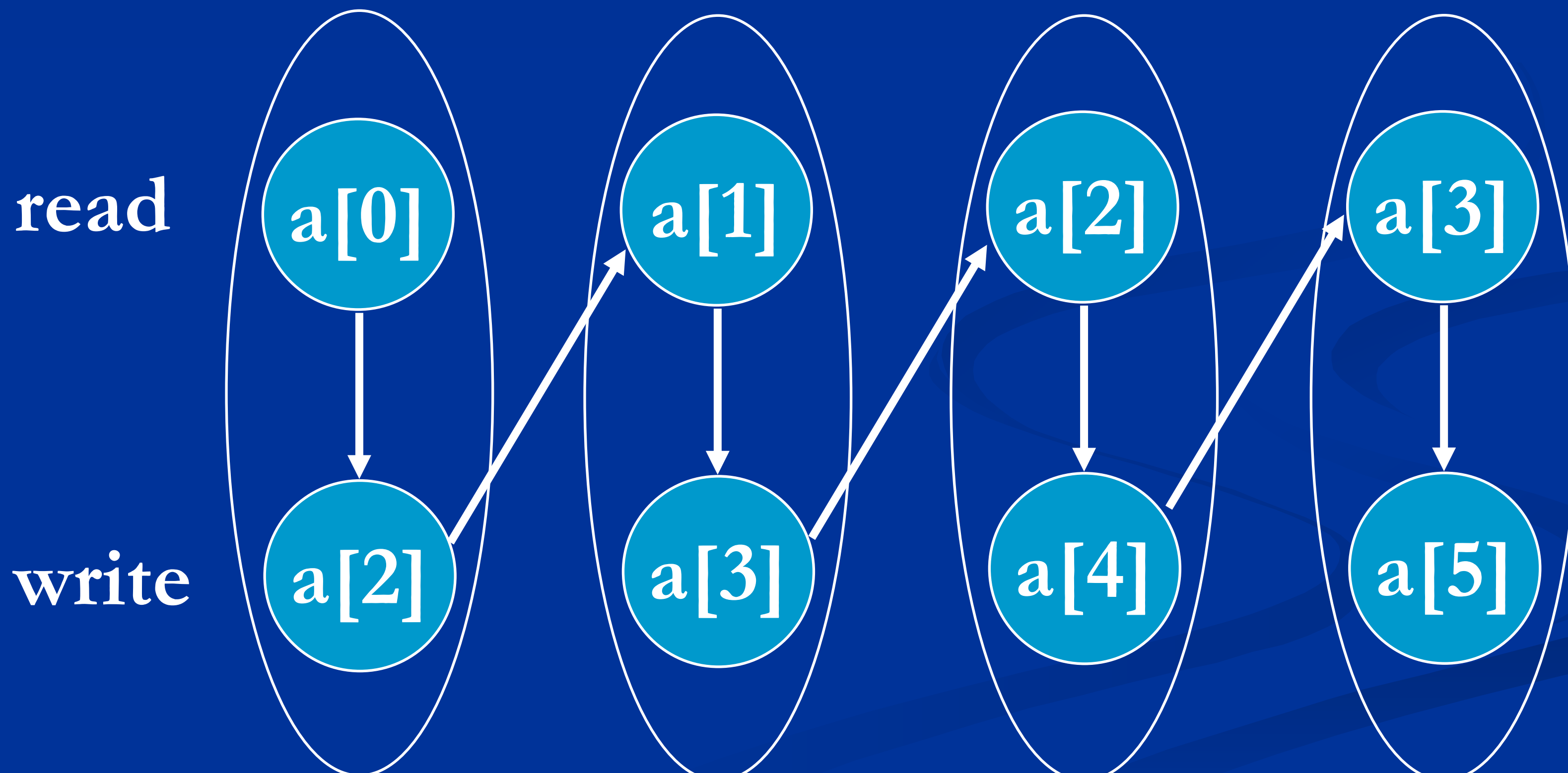
for i= 2, 5

  a[i-2] = a[i] + 3



Array True-dependence

for i= 2, 5
  a[i] = a[i-2] + 3



Dynamic Data Dependence

- Let o and o’ be two (dynamic) operations
- Data dependence exists from o to o’, iff
  - either o or o’ is a write operation
  - o and o’ may refer to the same location
  - o executes before o’

Static Data Dependence

- Let a and a’ be two static array accesses (not necessarily distinct)
- Data dependence exists from a to a’, iff
  - either a or a’ is a write operation
  - There exists a dynamic instance of a (o) and a dynamic instance of a’ (o’) such that
    - o and o’ may refer to the same location
    - o executes before o’

Recognizing DOALL Loops

- Find data dependences in loop
- Definition: a dependence is loop-carried if it crosses an iteration boundary
- If there are no loop-carried dependences then loop is parallelizable

Compute Dependence

for i= 2, 5
  a[i-2] = a[i] + 3

- There is a dependence between a[i] and a[i-2] if
  - There exist two iterations i_r and i_w within the loop bounds such that iterations i_r and i_w read and write the same array element, respectively
  - There exist i_r, i_w , $2 \le i_r, i_w \le 5$, $i_r = i_w - 2$

Compute Dependence

for i= 2, 5
  a[i-2] = a[i] + 3

- There is a dependence between a[i-2] and a[i-2] if
  - There exist two iterations i_v and i_w within the loop bounds such that iterations i_v and i_w write the same array element, respectively
  - There exist i_v, i_w , $2 \le i_v$, $i_w \le 5$, $i_v -2= i_w -2$

Parallelization

```
for i= 2, 5
  a[i-2] = a[i] + 3
```

- Is there a loop-carried dependence between a[i] and a[i-2]?
- Is there a loop-carried dependence between a[i-2] and a[i-2]?

Agenda

- Introduction
- Single Loops
- *Nested Loops*
- Data Dependence Analysis

Nested Loops

- Which loop(s) are parallel?

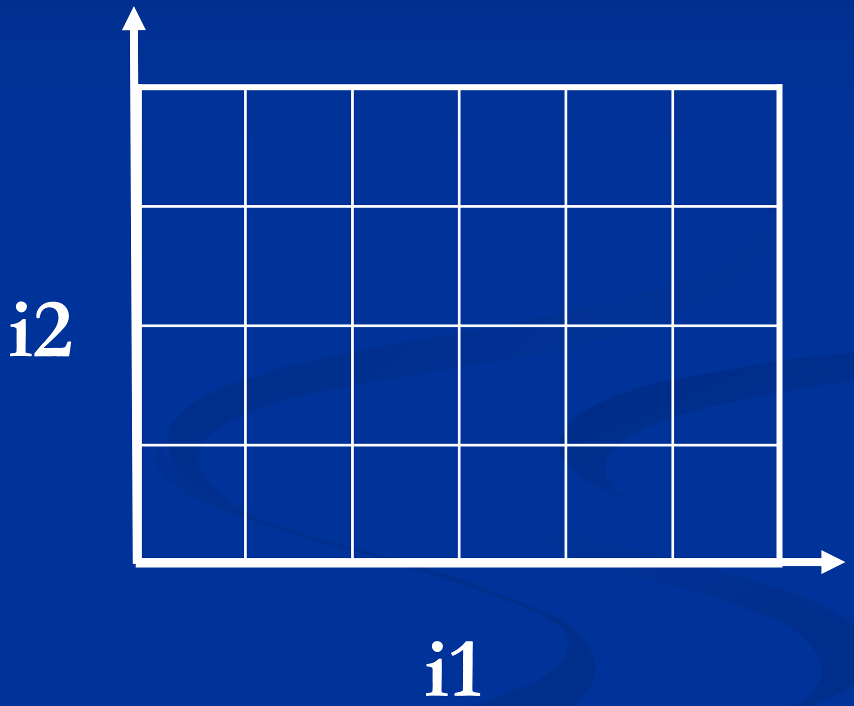
```
for i1 = 0, 5
  for i2 = 0, 3
    a[i1,i2] = a[i1-2,i2-1] + 3
```

Iteration Space

- An abstraction for loops

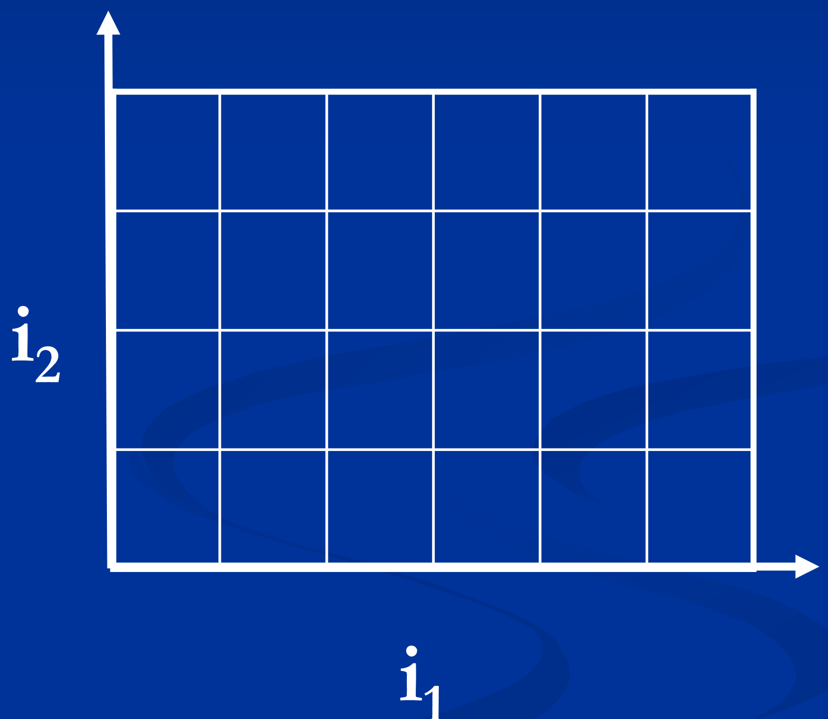
```
for i1 = 0, 5
  for i2 = 0, 3
    a[i1,i2] = 3
```

- Iteration is represented as coordinates in iteration space.



Execution Order

- Sequential execution order of iterations: Lexicographic order [0,0], [0,1], …[0,3], [1,0], [1,1], …[1,3], [2,0]…
- Let $I = (i_1, i_2, \ldots i_n)$. I is lexicographically less than I', $I<I'$, iff there exists k such that $(i_1, \ldots i_{k-1}) = (i'_1, \ldots i'_{k-1})$ and $i_k < i'_k$



Parallelism for Nested Loops

- Is there a data dependence between a[i1,i2] and a[i1-2,i2-1]?
  - There exist $i1_r$, $i2_r$, $i1_w$, $i2_w$, such that
  - $0 \le i1_r, i1_w \le 5$,
  - $0 \le i2_r, i2_w \le 3$,
  - $i1_r - 2 = i1_w$
  - $i2_r - 1 = i2_w$

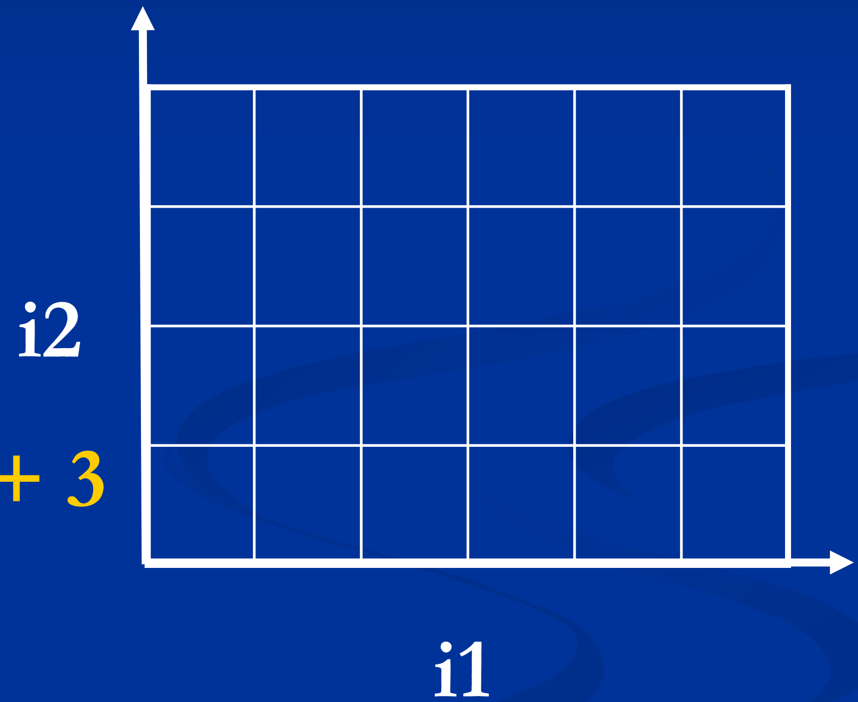
Loop-carried Dependence

- If there are no loop-carried dependences, then loop is parallelizable.
- Dependence carried by outer loop:
  - $i1_r \neq i1_w$
- Dependence carried by inner loop:
  - $i1_r = i1_w$
  - $i2_r \neq i2_w$
- This can naturally be extended to dependence carried by loop level *k*.

Nested Loops

- Which loop carries the dependence?

for i1 = 0, 5
for i2 = 0, 3
a[i1,i2] = a[i1-2,i2-1] + 3



Agenda

- Introduction
- Single Loop
- Nested Loops
- *Data Dependence Analysis*

Solving Data Dependence Problems

- Memory disambiguation is un-decidable at compile-time.

```
read(n)
for i = 0, 3
  a[i] = a[n] + 3
```

Domain of Data Dependence Analysis

- Only use loop bounds and array indices which are integer linear functions of variables.

```
for i1 = 1, n
  for i2 = 2*i1, 100
    a[i1+2*i2+3][4*i1+2*i2][i1*i1] = …
    … = a[1][2*i1+1][i2] + 3
```

Equations

- There is a data dependence, if
  - There exist $i1_r$, $i2_r$, $i1_w$, $i2_w$, such that
  - $0 \le i1_r, i1_w \le n$, $2*i1_r \le i2_r \le 100$, $2*i1_w \le i2_w \le 100$,
  - $i1_w + 2*i2_w + 3 = 1$, $4*i1_w + 2*i2_w = 2*i1_r + 1$
- Note: ignoring non-affine relations

```
for i1 = 1, n
  for i2 = 2*i1, 100
    a[i1+2*i2+3][4*i1+2*i2][i1*i1] = …
    … = a[1][2*i1+1][i2] + 3
```

Solutions

- There is a data dependence, if
  - There exist $i1_r$, $i2_r$, $i1_w$, $i2_w$, such that
  - $0 \le i1_r, i1_w \le n$, $2*i1_w \le i2_w \le 100$, $2*i1_w \le i2_w \le 100$,
  - $i1_w + 2*i2_w + 3 = 1$, $4*i1_w + 2*i2_w - 1 = i1_r + 1$
- No solution → No data dependence
- Solution → there may be a dependence

Form of Data Dependence Analysis

- Data dependence problems originally contains equalities and equalities
- Eliminate inequalities in the problem statement:
  - Replace a ≠ b with two sub-problems: a>b or a<b
  - We get

$$\exists \text{int } \vec{i}, A_1\vec{i} \le \vec{b}_1, A_2\vec{i} = \vec{b}_2$$

Form of Data Dependence Analysis

- Eliminate equalities in the problem statement:
  - Replace a =b with two sub-problems: a≤b and b≤a
  - We get

$$\exists \text{ int } \vec{i}\,,\, A\vec{i} \le \vec{b}$$

- Integer programming is NP-complete, i.e. Expensive

Techniques: Inexact Tests

- Examples: GCD test, Banerjee's test
- 2 outcomes
  - No → no dependence
  - Don't know → assume there is a solution → dependence
- Extra data dependence constraints
- Sacrifice parallelism for compiler efficiency

GCD Test

- Is there any dependence?

```
for i = 1, 100
  a[2*i] = …
  … = a[2*i+1] + 3
```

- Solve a linear Diophantine equation
  - $2*i_w = 2*i_r + 1$

GCD

- The *greatest common divisor* (GCD) of integers $a_1, a_2, \ldots, a_n$, denoted gcd($a_1, a_2, \ldots, a_n$), is the largest integer that evenly divides all these integers.
- Theorem: The linear Diophantine equation

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = c$$

has an integer solution $x_1, x_2, \ldots, x_n$ iff gcd($a_1, a_2, \ldots, a_n$) divides c

Examples

- Example 1: gcd(2,-2) = 2. No solutions

$$2x_1 - 2x_2 = 1$$

- Example 2: gcd(24,36,54) = 6. Many solutions

$$24x + 36y + 54z = 30$$

Multiple Equalities

$$x - 2y + z = 0$$
$$3x + 2y + z = 5$$

- Equation 1: gcd(1,-2,1) = 1. Many solutions
- Equation 2: gcd(3,2,1) = 1. Many solutions
- Is there any solution satisfying both equations?

The Euclidean Algorithm

- Assume a and b are positive integers, and a > b.
- Let c be the remainder of a/b.
  - If c=0, then gcd(a,b) = b.
  - Otherwise, gcd(a,b) = gcd(b,c).
- gcd(a_1, a_2, …, a_n) = gcd(gcd(a_1, a_2), a_3 …, a_n)

Exact Analysis

- Most memory disambiguations are simple integer programs.
- Approach: Solve exactly – yes, or no solution
  - Solve exactly with Fourier-Motzkin + branch and bound
  - Omega package from University of Maryland

Incremental Analysis

- Use a series of simple tests to solve simple programs (based on properties of inequalities rather than array access patterns)
- Solve exactly with Fourier-Motzkin + branch and bound
- Memoization
  - Many identical integer programs solved for each program
  - Save the results so it need not be recomputed

State of the Art

- Multiprocessors need large outer parallel loops
- Many inter-procedural optimizations are needed
  - Interprocedural scalar optimizations
    - Dependence
    - Privatization
    - Reduction recognition
  - Interprocedural array analysis
    - Array section analysis

Summary

- DOALL loops
- Iteration Space
- Data dependence analysis