This lecture:

- Parallelism
- Types of dependence flow, anti and output
- Distance and direction vectors
- Classification of loop based data dependences
- Dependence tests: gcd, Banerjee and Omega
References

- Today : The Omega Test: a fast and practical integer programming algorithm for dependence analysis Supercomputing 1992
Parallelism
Programming parallel computers

- Schools of thought:
  1. User specifies low-level parallelism and mapping
  2. User specifies parallelism (e.g. skeletons) - system tunes mapping
  3. Compiler finds parallelism in sequential code

- Popular approach is to break the transformation process into stages
- Transform to maximise parallelism i.e. minimise critical path of program execution graph
- Map parallelism to minimise “significant” machine costs i.e. communication/ non-local access etc.
Parallelism
Different forms of parallelism

Statement parallelism
\[ a = b + c \]
\[ d = e + f \]

Operation parallelism
\[ a = (b + c) \times (e + f) \]

Function parallelism
\[ f(a) = \begin{cases} 0 & \text{if } a \leq 1 \\ f(a-1) + f(a-2) & \text{else} \end{cases} \]

Loop parallelism
\[ \text{for}(i = 1 \text{ to } n) \]
\[ A[i] = b[i] + c \]
Parallelism
Loop parallelism / Array parallelism

Original loop
\[
\text{for}(i = 1 \text{ to } n) \\
A[i] = b[i] + c
\]

Parallel loop
\[
\text{parfor}(i = 1 \text{ to } n) \\
A[i] = b[i] + c
\]

- All iterations of the iterator \( i \) can be performed independently
- Independence implies parallelism
- Loop parallelism \( O(n) \) potential parallelism
- Compare statement and operation parallelism - \( O(1) \).
- Recursive parallelism rich but dynamic. Exploited in functional computational models
Parallelism

Parallelism and data dependence

for (i = 1 to n)
    $A[i] = b[i] + c$

Each iteration independent — completely parallel

for (i = 1 to n)

Each iteration dependent on previous — completely serial

Note: iterations NOT array elements
Parallelism
Parallelism and data dependence

Need to apply transformations and know when it is safe to do so

**Reordering transformation**

A reordering transformation is any program transformation that only changes the execution order of statements without adding or deleting statements.

A reordering transformation that preserves every dependence, preserves the meaning of the program.

Parallelising loop iterations allows random interleaving (reordering) of statements in loop body.
Data dependence
Types of data dependence

Relationship between reads and writes to memory has critical impact on parallelism
3 types of data dependence

<table>
<thead>
<tr>
<th>Flow (True) RAW hazard</th>
<th>Anti WAR hazard</th>
<th>Output WAW hazard</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1: a = S_2: a = a$</td>
<td>$S_1: = a S_2: a =$</td>
<td>$S_1: a = S_2: a =$</td>
</tr>
<tr>
<td>Denoted $S_2 \delta S_1$</td>
<td>Denoted $S_2 \delta^{-1} S_1$</td>
<td>Denoted $S_2 \delta^0 S_1$</td>
</tr>
</tbody>
</table>

Only data flow dependences are true dependences. Anti and output can be removed by renaming
Data-flow analysis can be used to define data dependences on a per block level for scalars but fails in presence of arrays.

Need finer grained analysis – determine if statements’ array usage access same memory location and type of dependence.
Consider two loops:

\[
\begin{array}{|c|c|}
\hline
S & \text{for}(i = 1 \text{ to } n) \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
S & \text{for}(i = 1 \text{ to } n) \\
\hline
\end{array}
\]

- In both cases, statement $S$ depends on itself.
- However, there is a significant difference.
- Need formalism to describe and distinguish such dependences.
Dependence

Iteration vectors

### Iteration number

Each iteration in a loop has an **iteration number** which is the value of the loop index at that iteration.

### Normalised iteration number

For iteration number \( i \) in loop with bounds \( L, U \), and stride \( S \), the **normalised iteration number** is\(^a\)

\[
\frac{(i - L + S)}{S}
\]

Convenient to normalise

\(^a\)This definition is one-based
Iteration vectors extend this notion to loop nests

Iteration vector $\mathcal{I}$ of iteration is the vector of integers containing iteration numbers for loops in order of nesting level

for($i = 1$ to 4)
  for($j = 1$ to 6)
    some-statement

Iteration vector, (2, 1) of $S$ is when $i = 2$ and $j = 1$
Dependence
Iteration vectors

Iteration vectors for simple loop

\[
\begin{array}{cccccc}
(4,1) & (4,2) & (4,3) & (4,4) & (4,5) & (4,6) \\
(3,1) & (3,2) & (3,3) & (3,4) & (3,5) & (3,6) \\
(2,1) & (2,2) & (2,3) & (2,4) & (2,5) & (2,6) \\
(1,1) & (1,2) & (1,3) & (1,4) & (1,5) & (1,6) \\
\end{array}
\]

\[
\begin{array}{c}
\text{for}(i = 1 \text{ to } 4) \\
\text{for}(j = 1 \text{ to } 6) \\
S \text{ some-statement}
\end{array}
\]
Iteration vectors ordered by execution order
For normalised vectors this is lexicographical ordering

**Lexicographical ordering**

For two iteration vectors, $\mathcal{I}$ and $\mathcal{J}$, $\mathcal{I} < \mathcal{J}$ iff

1. $\mathcal{I}[1 : n - 1] < \mathcal{J}[1 : n - 1]$, or
2. $\mathcal{I}[1 : n - 1] = \mathcal{J}[1 : n - 1]$ and $\mathcal{I}_n < \mathcal{J}_n$

I.e. compare $<$ by first element, if $=$ compare $<$ next element, etc.

Why normalised?
Dependence
Iteration vector ordering

Iteration vectors ordered by execution order
For normalised vectors this is lexicographical ordering

 Lexicographical ordering
For two iteration vectors, $\mathcal{I}$ and $\mathcal{J}$,
$\mathcal{I} < \mathcal{J}$ iff

1. $\mathcal{I}[1 : n - 1] < \mathcal{J}[1 : n - 1]$, or
2. $\mathcal{I}[1 : n - 1] = \mathcal{J}[1 : n - 1]$ and $\mathcal{I}_n < \mathcal{J}_n$

l.e. compare $<$ by first element, if $=$ compare $<$ next element, etc.

Why normalised?
Consider induction variable going backwards
Dependence
Iteration vector ordering

Lexicographical ordering

\[
\begin{align*}
S & \quad \text{for}(i = 1 \text{ to } 4) \\
& \quad \text{for}(j = 1 \text{ to } 6) \\
& \quad \text{some-statement}
\end{align*}
\]
Dependence

Dependencies between iterations

Loop-independent dependency
If statement $S_2$ depends on $S_1$ and $S_1, S_2$ execute in same iteration

Loop-carried dependency
If statement $S_2$ depends on $S_1$ and $S_1, S_2$ execute in different iterations
Dependence
Dependencies between iterations

**Dependence distance**
If dependence is between iterations $I_{\text{write}}$ and $I_{\text{read}}$, then distance is $I_{\text{read}} - I_{\text{write}}$.

**Distance example**
Write $A[10,11]$ at iteration $(5, 5)$. Read $A[10,11]$ at $(5, 6)$. Distance is?
Dependence
Dependencies between iterations

**Dependence distance**
If dependence is between iterations $I_{\text{write}}$ and $I_{\text{read}}$, then distance is $I_{\text{read}} - I_{\text{write}}$

**Distance example**
Write $A[10,11]$ at iteration $(5,5)$. Read $A[10,11]$ at $(5,6)$. Distance is $(5,6) - (5,5) = (0,1)$
Dependence
Dependencies between iterations

- If dependence distances all same, then say loop has that dependence distance
- But, loop may have many different dependence distances
- Direction vector summarises directions
- If first non ‘=’ element is ‘<’ then indicates flow dependence

### Dependence direction

Direction vector summary of distance dimensions
i.e. per dimension

- < All +ve
- > All -ve
- = All 0
- * Mixed

### Direction example

Given distances:
(0, 1, -1, -1)
(0, 2, -2, 0)
(0, 3, -3, 1)
Direction is: ?

\(^1\) (Why?)
Dependence

Dependencies between iterations

- If dependence distances all same, then say loop has that dependence distance
- But, loop may have many different dependence distances
- Direction vector summarises directions
- If first non ‘=’ element is ‘<’ then indicates flow dependence

Dependence direction

Direction vector summary of distance dimensions
i.e. per dimension

- < All +ve
- > All -ve
- = All 0
- * Mixed

Direction example

Given distances:
(0, 1,-1,-1)
(0, 2,-2, 0)
(0, 3,-3, 1)
Direction is:
(=, <, >, *)

¹(Why?)
Dependence

Dependencies between iterations

Where are dependences, distances, directions here?

\[
\begin{align*}
&\text{for}(i = 1 \text{ to } 4) \\
&\quad \text{for}(j = 1 \text{ to } 6) \\
&S_2 \quad A[i, j] = \\
&S_2 \quad = A[i, j] + 1
\end{align*}
\]
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

\[
\begin{align*}
&\text{for}(i = 1 \text{ to } 4) \\
&\quad \text{for}(j = 1 \text{ to } 6) \\
&S_2 \quad A[i, j] = \\
&S_2 \quad = A[i, j] + 1
\end{align*}
\]
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

for(i = 1 to 4)
  for(j = 1 to 6)
    S[ i, j ] = A[ i, j ] + 1

Statement can depend on itself
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

\[
\begin{array}{cccccccc}
(4,1) & (4,2) & (4,3) & (4,4) & (4,5) & (4,6) \\
(3,1) & (3,2) & (3,3) & (3,4) & (3,5) & (3,6) \\
(2,1) & (2,2) & (2,3) & (2,4) & (2,5) & (2,6) \\
(1,1) & (1,2) & (1,3) & (1,4) & (1,5) & (1,6) \\
\end{array}
\]

\[\text{for}(i = 1 \text{ to } 4)\]
\[\text{for}(j = 1 \text{ to } 6)\]
\[S \quad A[j+10, i] = A[j+10, i] + 1\]

Iterations, not array elements!
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

\[
\begin{align*}
S & \quad \text{for}(i = 1 \text{ to } 4) \\
& \quad \text{for}(j = 1 \text{ to } 6) \\
& \quad A[i, j] = A[i, j - 1] + 1
\end{align*}
\]
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

<table>
<thead>
<tr>
<th>for(i = 1) to (4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>for(j = 1) to (6)</td>
</tr>
<tr>
<td>(S)</td>
</tr>
</tbody>
</table>
Dependence

Dependencies between iterations

Where are dependences, distances, directions here?

```
for (i = 1 to 4)
  for (j = 1 to 6)
    S[i][j+1] = S[i][j] + 1
```

Clearly the same thing
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

for (i = 1 to 4)
  for (j = 1 to 6)
    $S_{i,j} = S_{i,j-2} + 1$
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

```
for(i = 1 to 4)
    for(j = 1 to 6)
        S
```
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

\[
\begin{align*}
\text{for}(i = 1 \text{ to } 4) & \\
& \quad \text{for}(j = 1 \text{ to } 6) \\
S & \quad A[i, j] = A[i-1, j-1] + 1
\end{align*}
\]
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

\[
\text{for}(i = 1 \text{ to } 4) \\
\quad \text{for}(j = 1 \text{ to } 6) \\
\quad S[A[i, j]] = A[i - 1, j - 1] + 1
\]
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

\[
\text{for}(i = 1 \text{ to } 4) \\
\quad \text{for}(j = 1 \text{ to } 6) \\
S \quad A[i, j] = A[i, 1] + j
\]
Dependence

Dependencies between iterations

Where are dependences, distances, directions here?

for(i = 1 to 4)
  for(j = 1 to 6)
    \( S_{i,j} = A_{i,1} + j \)
Dependence

Dependencies between iterations

Where are dependences, distances, directions here?

\[
\begin{align*}
  \text{for} (i = 1 \text{ to } 4) \quad & \quad \text{for} (j = 1 \text{ to } 6) \\
\end{align*}
\]
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

\[
\text{for}(i = 1 \text{ to } 4) \quad \text{for}(j = 1 \text{ to } 6) \quad
\]
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

\[
\begin{align*}
\text{for}(i = 1 \text{ to } 4) \\
\quad \text{for}(j = 1 \text{ to } 6) \\
S_{i,j} &= A[0,0]
\end{align*}
\]
Dependence
Dependencies between iterations

Where are dependences, distances, directions here?

for(i = 1 to 4)
  for(j = 1 to 6)
    \[ S_{i,j} = A_{0,0} \]

There are none!
Dependence
Solving the dependence problem

Question: is there dependence between array write in $S_1$ and read in $S_2$?

\[ f_{w}(I_w) - f_{r}(I_r) = 0 \]

This is a diophantine equation, which is generally undecidable. However, it can be limited to linear diophantine equations with constraints.

\[ \sum_{i=1}^{n} a_i x_i = 0 \]

After Diophantus of Alexandria c. 210AD

This is Hilbert's tenth problem – set in 1900, proven in 1970

Consider $n \geq 2, \forall a, b, c > 0; a^n + b^n - c^n \neq 0$ (Fermat's last theorem)

Integer linear programming is NP-complete
Dependence
Solving the dependence problem

• Question: is there dependence between array write in $S_1$ and read in $S_2$?
• Assume write in iteration $\mathcal{I}_w$, read in $\mathcal{I}_r$

\[ f_w(I_w) - f_r(I_r) = 0 \] for integer solutions (inside iteration space)

This is diophantine equation

\underline{2} After *Diophantus of Alexandria* c. 210AD
\underline{3} This is Hilberts tenth problem – set in 1900, proven in 1970
\underline{4} Consider $n \geq 2, \forall a, b, c > 0; a^n + b^n - c^n \neq 0$ (*Fermat’s last theorem*)
\underline{5} Integer linear programming is NP-complete
Dependence
Solving the dependence problem

- Question: is there dependence between array write in $S_1$ and read in $S_2$?
- Assume write in iteration $I_w$, read in $I_r$
- Assume write of $A[f_w(I_w)]$, read of $A[f_r(I_r)]$, with $f_w$ and $f_r$ as polynomials

---

2 After *Diophantus of Alexandria* c. 210AD
3 This is Hilberts tenth problem – set in 1900, proven in 1970
4 Consider $n \geq 2, \forall a, b, c > 0; a^n + b^n - c^n \neq 0$ (*Fermat’s last theorem*)
5 Integer linear programming is NP-complete
Dependence
Solving the dependence problem

- Question: is there dependence between array write in $S_1$ and read in $S_2$?
- Assume write in iteration $I_w$, read in $I_r$
- Assume write of $A[f_w(I_w)]$, read of $A[f_r(I_r)]$, with $f_w$ and $f_r$ as polynomials
- Solve $f_w(I_w) - f_r(I_r) = 0$ for integer solutions (inside iteration space)

---

2 After Diophantus of Alexandria c. 210AD
3 This is Hilbert's tenth problem – set in 1900, proven in 1970
4 Consider $n \geq 2, \forall a, b, c > 0; a^n + b^n - c^n \neq 0$ (Fermat's last theorem)
5 Integer linear programming is NP-complete
Dependence
Solving the dependence problem

- Question: is there dependence between array write in $S_1$ and read in $S_2$?
- Assume write in iteration $I_w$, read in $I_r$
- Assume write of $A[f_w(I_w)]$, read of $A[f_r(I_r)]$, with $f_w$ and $f_r$ as polynomials
- Solve $f_w(I_w) - f_r(I_r) = 0$ for integer solutions (inside iteration space)
- This is diophantine equation\(^2\)

---

\(^2\)After *Diophantus of Alexandria* c. 210AD
\(^3\)This is Hilberts tenth problem – set in 1900, proven in 1970
\(^4\)Consider $n \geq 2, \forall a, b, c > 0; a^n + b^n - c^n \neq 0$ (*Fermat’s last theorem*)
\(^5\)Integer linear programming is NP-complete
Dependence
Solving the dependence problem

- Question: is there dependence between array write in $S_1$ and read in $S_2$?
- Assume write in iteration $I_w$, read in $I_r$
- Assume write of $A[f_w(I_w)]$, read of $A[f_r(I_r)]$, with $f_w$ and $f_r$ as polynomials
- Solve $f_w(I_w) - f_r(I_r) = 0$ for integer solutions (inside iteration space)
- This is diophantine equation\(^2\)
- Undecidable in general\(^3,4\)

---

\(^2\)After *Diophantus of Alexandria* c. 210AD
\(^3\)This is Hilbert's tenth problem – set in 1900, proven in 1970
\(^4\)Consider $n \geq 2, \forall a, b, c > 0; a^n + b^n - c^n \neq 0$ (*Fermat's last theorem*)
\(^5\)Integer linear programming is NP-complete
Dependence
Solving the dependence problem

- Question: is there dependence between array write in $S_1$ and read in $S_2$?
- Assume write in iteration $I_w$, read in $I_r$
- Assume write of $A[f_w(I_w)]$, read of $A[f_r(I_r)]$, with $f_w$ and $f_r$ as polynomials
- Solve $f_w(I_w) - f_r(I_r) = 0$ for integer solutions (inside iteration space)
- This is diophantine equation
- Undecidable in general
- Limit to linear diophantine equations with constraints
  \[ a_n x_1 + a_{n-1} x_{n-1} + \ldots + a_1 x_1 + a_0 = 0 \]

---

2 After *Diophantus of Alexandria* c. 210AD
3 This is Hilbert's tenth problem – set in 1900, proven in 1970
4 Consider $n \geq 2, \forall a, b, c > 0; a^n + b^n - c^n \neq 0$ (*Fermat's last theorem*)
5 Integer linear programming is NP-complete
Dependence
Solving the dependence problem

Example

for(i = 1 to 100)
    for(j = i to 100)

Let $I_w = (i_w, j_w)$ and $I_r = (i_r, j_r)$

Let $f_w(i_w, j_w) =$
Dependence
Solving the dependence problem

Example

\[
\text{for}(i = 1 \text{ to } 100)
\]
\[
\quad \text{for}(j = i \text{ to } 100)
\]
\[
\quad A[i, j+1] = A[i, j]
\]

Let \( I_w = (i_w, j_w) \) and \( I_r = (i_r, j_r) \)

Let \( f_w(i_w, j_w) = (i_w, j_w + 1) \) and \( f_r(i_r, j_r) = \)
## Dependence

### Solving the dependence problem

#### Example

```plaintext
for(i = 1 to 100)
    for(j = i to 100)
```

Let $I_w = (i_w, j_w)$ and $I_r = (i_r, j_r)$

Let $f_w(i_w, j_w) = (i_w, j_w + 1)$ and $f_r(i_r, j_r) = (i_r, j_r)$

First constrain induction variables
Dependence
Solving the dependence problem

Example

for(i = 1 to 100)
    for(j = i to 100)

Let $\mathcal{I}_w = (i_w, j_w)$ and $\mathcal{I}_r = (i_r, j_r)$

Let $f_w(i_w, j_w) = (i_w, j_w + 1)$ and $f_r(i_r, j_r) = (i_r, j_r)$

First constrain induction variables

$$1 \leq i_w \leq 100,$$
Dependence
Solving the dependence problem

Example

for(i = 1 to 100)
    for(j = i to 100)

Let $I_w = (i_w, j_w)$ and $I_r = (i_r, j_r)$

Let $f_w(i_w, j_w) = (i_w, j_w + 1)$ and $f_r(i_r, j_r) = (i_r, j_r)$

First constrain induction variables

\[1 \leq i_w \leq 100, \quad i_w \leq j_w \leq 100,\]
Dependence
Solving the dependence problem

Example

for(i = 1 to 100)
    for(j = i to 100)

Let $I_w = (i_w, j_w)$ and $I_r = (i_r, j_r)$
Let $f_w(i_w, j_w) = (i_w, j_w + 1)$ and $f_r(i_r, j_r) = (i_r, j_r)$
First constrain induction variables

\[
1 \leq i_w \leq 100, \quad i_w \leq j_w \leq 100, \\
1 \leq i_r \leq 100, \quad i_r \leq j_r \leq 100
\]
Dependence
Solving the dependence problem

Example

for(i = 1 to 100)
    for(j = i to 100)

Let $I_w = (i_w, j_w)$ and $I_r = (i_r, j_r)$

Let $f_w(i_w, j_w) = (i_w, j_w + 1)$ and $f_r(i_r, j_r) = (i_r, j_r)$

First constrain induction variables

\[ 1 \leq i_w \leq 100,\ i_w \leq j_w \leq 100,\]
\[ 1 \leq i_r \leq 100,\ i_r \leq j_r \leq 100 \]

Constraints so $f_w(i_w, j_w) = f_r(i_r, i_r)$
Dependence
Solving the dependence problem

Example

for(i = 1 to 100)
    for(j = i to 100)

Let \(I_w = (i_w, j_w)\) and \(I_r = (i_r, j_r)\)

Let \(f_w(i_w, j_w) = (i_w, j_w + 1)\) and \(f_r(i_r, j_r) = (i_r, j_r)\)

First constrain induction variables

\[1 \leq i_w \leq 100, \quad i_w \leq j_w \leq 100,\]
\[1 \leq i_r \leq 100, \quad i_r \leq j_r \leq 100\]

Constraints so \(f_w(i_w, j_w) = f_r(i_r, i_r)\)

\[i_w = i_r, \quad j_w + 1 = j_r\]
**Dependence**

Solving the dependence problem

**Example**

```plaintext
for(i = 1 to 100)
    for(j = i to 100)
```

Let $\mathcal{I}_w = (i_w, j_w)$ and $\mathcal{I}_r = (i_r, j_r)$

Let $f_w(i_w, j_w) = (i_w, j_w + 1)$ and $f_r(i_r, j_r) = (i_r, j_r)$

First constrain induction variables

\[
1 \leq i_w \leq 100, \ i_w \leq j_w \leq 100, \\
1 \leq i_r \leq 100, \ i_r \leq j_r \leq 100
\]

Constraints so $f_w(i_w, j_w) = f_r(i_r, i_r)$

\[
i_w = i_r, j_w + 1 = j_r
\]

Are there integer solutions for $i_w, j_w, i_r, j_r$?
Dependence
Solving the dependence problem

Example

for(i = 1 to 100)
    for(j = i to 100)
Let $I_w = (i_w, j_w)$ and $I_r = (i_r, j_r)$
Let $f_w(i_w, j_w) = (i_w, j_w + 1)$ and $f_r(i_r, j_r) = (i_r, j_r)$
First constrain induction variables

\[
1 \leq i_w \leq 100, \quad i_w \leq j_w \leq 100, \\
1 \leq i_r \leq 100, \quad i_r \leq j_r \leq 100
\]

Constraints so $f_w(i_w, j_w) = f_r(i_r, i_r)$

\[
i_w = i_r, j_w + 1 = j_r
\]
Are there integer solutions for $i_w, j_w, i_r, j_r$? Yes, lots:

\[
1 \leq i_w = i_r \leq j_w = (j_r - 1) \leq 99
\]
Dependence
Hierarchical computation of dependence directions in loops

for(i = 1 to n)
    A[f(i)] =
    = A[g(i)]

- Test for any dependence from iteration $I_w$ to $I_r$:
  $1 \leq I_w \leq n, 1 \leq I_r \leq n \land f(I_w) = g(I_r)$
- Use this test to test any direction [*]
- If solutions add additional constraints:
  - $<$ direction : add $I_w < I_r$,
  - $=$ direction : add $I_w = I_r$
- Extend for multi loops, [*] then [<, *], [=, *] etc - hierarchical testing
- If $L$ is loop depth, requires $O(3^L)$ tests per array access pair!
Dependence
Conservative testing

- Full problem is NP-complete; use some quick and dirty tests
- Apply test
- If fail to accurately solve dependence, try more tests
- Still fail? Assume dependency
- Never dangerous, may be sub-optimal
- Works correctly for vast majority of code
Dependence
Classification for simplification: Kennedy approach

- Test for each subscript in turn, if any subscript has no dependence - then no solution

ZIV, SIV, MIV

- Subscript is pair of expressions at same dimension
  - ZIV if it contains no index - e.g. \( \langle 2, 10 \rangle \)
  - SIV if it contains only one index - e.g. \( \langle i, i + 2 \rangle \)
  - MIV if it contains more than one index - e.g. \( \langle i + j, j \rangle \)

Example classification

\[ A[5, i+1, j] = A[10, i, k] + c \]

- Subscript in 1st dim
Dependence
Classification for simplification : Kennedy approach

- Test for each subscript in turn, if any subscript has no dependence - then no solution

ZIV, SIV, MIV

- Subscript is pair of expressions at same dimension
  - ZIV if it contains no index - e.g. \( \langle 2, 10 \rangle \)
  - SIV if it contains only one index - e.g. \( \langle i, i + 2 \rangle \)
  - MIV if it contains more than one index - e.g. \( \langle i + j, j \rangle \)

Example classification

\[
A[5, i+1, j] = A[10, i, k] + c
\]

- Subscript in 1st dim contains zero index variables (ZIV)
- Subscript in 2nd dim
Dependence
Classification for simplification: Kennedy approach

- Test for each subscript in turn, if any subscript has no dependence - then no solution

ZIV, SIV, MIV

- Subscript is pair of expressions at same dimension
  - ZIV if it contains no index - e.g. \( \langle 2, 10 \rangle \)
  - SIV if it contains only one index - e.g. \( \langle i, i + 2 \rangle \)
  - MIV if it contains more than one index - e.g. \( \langle i + j, j \rangle \)

Example classification

\[
A[5, i+1, j] = A[10, i, k] + c
\]

- Subscript in 1st dim contains zero index variables (ZIV)
- Subscript in 2nd dim contains single \((i)\) index variables (SIV)
- Subscript in 3rd dim
Dependence
Classification for simplification: Kennedy approach

- Test for each subscript in turn, if any subscript has no dependence - then no solution

ZIV, SIV, MIV

- Subscript is pair of expressions at same dimension
  - ZIV if it contains no index - e.g. \( \langle 2, 10 \rangle \)
  - SIV if it contains only one index - e.g. \( \langle i, i + 2 \rangle \)
  - MIV if it contains more than one index - e.g. \( \langle i + j, j \rangle \)

Example classification

\[ A[5, i+1, j] = A[10, i, k] + c \]

- Subscript in 1st dim contains zero index variables (ZIV)
- Subscript in 2nd dim contains single \((i)\) index variables (SIV)
- Subscript in 3rd dim contains multi \((j,k)\) index variables (MIV)
**Dependence**

**Separability**

- Indices in *separable* subscripts do not occur in other subscripts.
- If two different subscripts contain same index they are *coupled*.
- Separable subscripts and coupled groups handled independently.

**Example separability**

\[ a(i+1, j) = a(k, j) + c \]

First subscript is...
## Dependence
### Separability

<table>
<thead>
<tr>
<th>Separability</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Indices in <em>separable</em> subscripts do not occur in other subscripts</td>
</tr>
<tr>
<td>- If two different subscripts contain same index they are <em>coupled</em></td>
</tr>
<tr>
<td>- Separable subscripts and coupled groups handled independently</td>
</tr>
</tbody>
</table>

### Example separability

\[ a(i+1, j) = a(k, j) + c \]

First subscript is separable.
Second subscript is
Dependence

Separability

- Indices in *separable* subscripts do not occur in other subscripts.
- If two different subscripts contain the same index they are *coupled*.
- Separable subscripts and coupled groups handled independently.

Example separability

\[ a(i+1, j) = a(k, j) + c \]

First subscript is separable.
Second subscript is separable.

\[ a(i, j, j) = a(i, j, k) + c \]

Second subscript is
Dependence

Separability

Indices in *separable* subscripts do not occur in other subscripts

If two different subscripts contain same index they are *coupled*

Separable subscripts and coupled groups handled independently

Example separability

\[ a(i+1, j) = a(k, j) + c \]

First subscript is separable.  
Second subscript is separable.

\[ a(i, j, j) = a(i, j, k) + c \]

Second subscript is coupled.  
Third subscript is
Dependence
Separability

Separability
- Indices in *separable* subscripts do not occur in other subscripts
- If two different subscripts contain same index they are *coupled*
- Separable subscripts and coupled groups handled independently

Example separability

\[ a(i+1, j) = a(k, j) + c \]
First subscript is separable.
Second subscript is separable.

\[ a(i, j, j) = a(i, j, k) + c \]
Second subscript is coupled.
Third subscript is coupled.
Dependence

ZIV test

ZIV

\[ A[\ldots, c_w, \ldots] = A[\ldots, c_r, \ldots] \ldots \]

If \( c_w \) and \( c_r \) are constants or loop invariant and \( c_w \neq c_r \)
No dependence

ZIV example

\[ A[5, j+1, 10, k] = A[i, j, 12, k-1] + c \]

Third subscript has ZIV, and \( 10 \neq 12 \)
No dependence
Dependence
Strong SIV test

Constraint is:
\[ A[ai+c_w] = A[ai+c_r] \ldots \]
\[ ai_w + c_w = ai_r + c_r \Rightarrow \]
\[ i_w - i_r = (c_w - c_r)/a \]

Has solution is \((c_w - c_r)/a\) is integer and is in range.

Strong SIV example

```
for(i = 1 to 10)
```

Subscript is Strong SIV, \(a = 1, c_w = 1, c_r = 0\)
\[ (c_w - c_r)/a = 1 \in [1, 10] \]
Dependence
General SIV test or greatest common divisor

General SIV test

Constraint is:  \[ A[a_w i+c_w] = A[a_r i+c_r] \ldots \]
\[ a_w i_w + c_w = a_r i_r + c_r \]

If \( \text{gcd}(a_w, a_r) \) does not divides \( c_w - c_r \) then no solution.
Else possibly many solutions.

General SIV example

\begin{align*}
\text{for}(i = 1 \text{ to } 10) & \\
\end{align*}

Subscript is Strong SIV, \( a_w = 2, a_r = 4, c_w = 1, c_r = 0 \)
\[(c_w - c_r)/\text{gcd}(a_w, a_r) = 0.5\]

No dependence
Banerjee test

- GCD test does not consider range - only if integer solution possible somewhere
- Banerjee test for existence of real valued solution in range
- If no real solution in range, then no integer one either

### Banerjee test

\[ A[a_w i + c_w] = A[a_r i + c_r] \ldots \]

Constraint is:

\[ a_w i_w + c_w = a_r i_r + c_r \quad \Rightarrow \]

\[ h(i_w, i_r) = a_w i_w - a_r i_r + c_w - c_r = 0 \]

True by intermediate value theorem, if \( max(h) \geq 0 \land min(h) \leq 0 \)
Dependence
Banerjee test

Banerjee test

\[
\text{for}(i = 1 \text{ to } 100) \\
\]

- We have \(2i_w + 3 = i_r + 7\), \(h = 2i_wi_r\) and \(1 \leq i_w \leq i_r \leq 100\)
- \(\text{max}(h) = (2 \times 100 - 1 - 4) = 195\), \(\text{min}(h) = (2 \times 1 - 100 - 4) = -102\)
- \(\text{max}(h) = 195 \geq 0 \geq \text{min}(h) = 102\) – Hence solution
- Simple example can be extended. Technical difficulties with complex iteration spaces
- Performed sub-script at a time, Used for MIV
Dependence
Pugh’s Omega Test

- Exact solutions using integer linear programming
- Fast enough for most real programs
  - Worst case exponential time
  - Commonly low order polynomial
  - Can directly yield direction and distance
- Algorithm
  - Express constraints
  - Simplify constraints
  - Do Fourier-Motzkin elimination

Read the paper!
Geometric interpretation of constraints

- Constraints define $n$ dimensional, revalued volume
- If empty no possible integer solutions
- Project volume to one fewer dimensions - giving real shadow
- If no integer points in shadow, no integer points in volume
Integer points in real shadow do not imply integer points in volume

- Define ‘*dark shadow*’ where if dark shadow contains integer point then real volume must
- E.g. \(^7\) shadow wherever real volume is thicker than 1
- No integer points in dark shadow does not imply no integer points in real volume

\(^7\)Read paper for what they do
Dependence
Pugh’s Omega Test
Dependence
Pugh’s Omega Test
Dependence

Pugh’s Omega Test
Dependence
Pugh’s Omega Test
Summary

- Parallelism
- Types of dependence flow, anti and output
- Distance and direction vectors
- Classification of loop based data dependences
- Dependence tests: gcd, Banerjee and Omega
The biggest revolution in the technological landscape for fifty years
Now accepting applications!
Find out more and apply at: pervasiveparallelism.inf.ed.ac.uk

- 4-year programme: MSc by Research + PhD
- Research-focused: Work on your thesis topic from the start
- Collaboration between:
  ▶ University of Edinburgh’s School of Informatics
    ✴ Ranked top in the UK by 2014 REF
  ▶ Edinburgh Parallel Computing Centre
    ✴ UK’s largest supercomputing centre
- Research topics in software, hardware, theory and application of:
  ▶ Parallelism
  ▶ Concurrency
  ▶ Distribution
- Full funding available
- Industrial engagement programme includes internships at leading companies

The biggest revolution in the technological landscape for fifty years