This lecture:

- Vector loops - how to write loops in a vector format
- Loop distribution + statement reordering: basic vectorisation
- Dependence condition for vectorisation: Based on loop level
- Kennedy’s Vectorisation algorithm based on SCC and hierarchical dependences
- Loop Interchange: Move vector loops innermost
- Scalar Expansion, Renaming and Node splitting. Overcoming cycles
Vectorisation
What is vectorisation?

- Generalise operations on scalars to apply transparently to vectors, matrices, etc
- Architectures provide vector units, compute multiple elements at once
- Single instruction multiple data (SIMD)
Vectorisation
Vector code

- Use Fortran 90 vector notation to express vectorised loops.
- Triple notation used \( x(\text{start}:	ext{finish}:	ext{step}) \) to represent a vector in \( x \)
- Vectorisation depends on loop dependence

No loop carried dependence

\[
\begin{align*}
\text{Do } i &= 1, N \\
&x(i) = x(i) + c \\
\text{Enddo} \\
\text{Vectorisable} \\
x(1:N) &= x(1:N) + c
\end{align*}
\]

Loop carried dependence

\[
\begin{align*}
\text{Do } i &= 1, N \\
&x(i+1) = x(i) + c \\
\text{Enddo} \\
\text{Not vectorisable} \\
x(2:N+1) &= x(1:N) + c \\
\text{Reads } x \text{ at once}
\end{align*}
\]
Vector registers are a fixed size. Need to fit code to registers

**Original**

\[
\begin{align*}
\text{Do } & \ i = 1, \ N \\
& x(i) = x(i) + c \\
\text{Enddo}
\end{align*}
\]

**Strip-mined**

\[
\begin{align*}
\text{Do } & \ i = 1, \ N, \ s \\
& \text{Do } ii = i, \ i+s-1 \\
& \quad x(ii) = x(ii) + c \\
& \text{Enddo} \\
\text{Enddo}
\end{align*}
\]

**Vectorised**

\[
\begin{align*}
\text{Do } & \ i = 1, \ N, \ s \\
& x(i:i+s-1) = x(i:i+s-1) + c \\
\text{Enddo}
\end{align*}
\]
Vectorisation
Loop distribution + statement reordering

Standard approach to isolating statements within a loop for later vectorisation

**Original**

```
Do i = 1, N
  a(i+1) = b(i) + c
  d(i) = a(i) + e
Enddo
```

**Distributed**

```
Do i = 1, N
  a(i+1) = b(i) + c
Enddo
Do i = 1, N
  d(i) = a(i) + e
Enddo
```

**Vectorised**

```
a(2:N+1) = b(1:N) + c
d(1:N) = a(1:N) + e
```

Cyclic dependence prevent distribution and hence vectorisation.
Do $i = 1, N$
   Do $j = 1, M$
      $a(i+1,j) = a(i,j) + c$
   Enddo
Enddo

- Cannot vectorise as dependence $(1,0)$.
- If outer loop run sequential then can vectorise inner loop with dependence $(0)$.
- Generalises to nested loops.

Do $i = 1,N$
   $a(i+1,1:M) = a(i,1:M) + c$
Enddo
Vectorisation algorithm

Simple description of CMA algorithm. Read CMA!

1. Form dependence graph
2. Strongly Connected Component (SCC) identification (cycles)
3. Sort SCCs topologically
4. For each SCC
   - If weakly connected then
     - Vectorise using loop distribution
   - Else
     - Write loop start
     - Strip off outer dependence level
     - Goto 1 with SCC as program
     - Write loop end

\(^1\) Loop will be sequentialised
Vectorisation algorithm
Review: strongly connected components

**Strongly connected**
A graph is **strongly connected** if every vertex is reachable from every other vertex

**Strongly connected components (SCCs)**
SCCs partition a graph into strongly connected subgraphs²
Maximal SCCs are largest possible

What are the SCCs?

² Use Tarjan’s algorithm
Vectorisation algorithm
Review: strongly connected components

Strongly connected
A graph is **strongly connected** if every vertex is reachable from every other vertex

Strongly connected components (SCCs)
SCCs partition a graph into strongly connected subgraphs\(^2\)
Maximal SCCs are largest possible

What are the SCCs?

\(^2\)Use Tarjan’s algorithm
Vectorisation algorithm
Review: topological sort

**Topological sort of acyclic directed graph**

Linear ordering, $\prec_{\text{topo}}$ of nodes such that if there is edge $(u, v)$, then $u \prec_{\text{topo}} v$.

Maximal SCC graphs are acyclic directed

**What is the topological sort?**
Vectorisation algorithm
Review: topological sort

Topological sort of acyclic directed graph
Linear ordering, $<_{\text{topo}}$ of nodes such that if there is edge $(u, v)$, then $u <_{\text{topo}} v$.

Maximal SCC graphs are acyclic directed

What is the topological sort?
**Vectorisation algorithm**

Review: dependence graphs

---

**Flow (True)**

**RAW hazard**

\[ S_1: a = \]
\[ S_2: = a \]
Denoted \( S_2 \delta S_1 \)

**Anti WAR hazard**

\[ S_1: = a \]
\[ S_2: a = \]
Denoted \( S_2 \delta^{-1} S_1 \)

**Output WAW hazard**

\[ S_1: a = \]
\[ S_2: a = \]
Denoted \( S_2 \delta^0 S_1 \)

---

**Level of loop carried dependence**

**Level** of loop carried dependence is the index of the left-most non “=” in direction vector.

Written as subscript, e.g. \( \delta_1 \) for \((<,=,=)\), \( \delta_3^{-1} \) for \((=,=,>)\).

Infinity for in same loop, e.g. \( \delta_\infty \) for \((=,=,=)\).
Vectorisation algorithm

Example

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>Do $i = 1,100$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$x(i) = y(i) + 10$</td>
<td></td>
</tr>
<tr>
<td>$S_2$</td>
<td>Do $j = 1,100$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$b(j) = a(j,n)$</td>
<td></td>
</tr>
<tr>
<td>$S_3$</td>
<td>Do $k = 1,100$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$a(j+1,k) = b(j) + c(j,k)$</td>
<td></td>
</tr>
<tr>
<td>$S_4$</td>
<td>$y(i+j) = a(j+1,n)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enddo</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enddo</td>
<td></td>
</tr>
</tbody>
</table>

Label and edge for this dependence?
Vectorisation algorithm

Example

| $S_1$ | Do $i = 1,100$
|---|---|
|   | $x(i) = y(i) + 10$
|   | Do $j = 1,100$
| $S_2$ | $b(j) = a(j,n)$
|   | Do $k = 1,100$
| $S_3$ | $a(j+1,k) = b(j) + c(j,k)$
|   | Enddo
| $S_4$ | $y(i+j) = a(j+1,n)$
|   | Enddo
|   | Enddo

$1 \leq i_r \leq 100, \ 1 \leq i_w \leq 100, \ 1 \leq j_w \leq 100$

$i_w + j_w = i_r$

Has solutions and $j_w$ always positive, so $i_w < i_r \Rightarrow$ direction ($<$)

Loop carried flow dependence, level one ($\delta_1$)
Vectorisation algorithm
Example

Do i = 1,100
  x(i) = y(i) + 10
  Do j = 1,100
    b(j) = a(j,n)
    Do k = 1,100
      a(j+1,k) = b(j) + c(j,k)
    Enddo
  Enddo
y(i+j) = a(j+1,n)
Enddo
Enddo

Label and edge for this dependence?
Vectorisation algorithm

Example

```
Example

| S_1   | Do i = 1,100
      | x(i) = y(i) + 10
      | Do j = 1,100
      | S_2   | b(j) = a(j,n)
      | Do k = 1,100
      | S_3   | a(j+1,k) = b(j)+c(j,k)
          | Enddo
      | S_4   | y(i+j) = a(j+1,n)
          | Enddo
      | Enddo

Clearly direction for j loop is =. For i loop, i is not in either array subscript, so *.
So, direction is (*, =) or {(<, =), (=, =), (> ,=)} or δ_1, δ_∞, δ_1^{-1}
```
Example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| $S_1$ | Do $i = 1,100$
|     | $x(i) = y(i) + 10$
|     | Do $j = 1,100$
| $S_2$ | $b(j) = a(j,n)$
|     | Do $k = 1,100$
| $S_3$ | $a(j+1,k) = b(j)+c(j,k)$
|     | Enddo
| $S_4$ | $y(i+j) = a(j+1,n)$
|     | Enddo
|     | Enddo

Label and edge for this dependence?
Vectorisation algorithm

Example

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>
| \[ \begin{align*} &S_1 \quad \text{Do } i = 1,100 \\
& \quad x(i) = y(i) + 10 \\
& \quad \text{Do } j = 1,100 \\
& \quad \text{Enddo} \\
& S_2 \quad b(j) = a(j, n) \\
& \quad \text{Do } k = 1,100 \\
& \quad a(j+1, k) = b(j) + c(j, k) \\
& \quad \text{Enddo} \\
& S_3 \quad y(i+j) = a(j+1, n) \\
& \quad \text{Enddo} \\
& S_4 \quad \text{Enddo} \end{align*} \] |

\[1 \leq i_r, j_r, i_w, j_w, k_w \leq 100, n \in \mathbb{N}\]

\[j_w + 1 = j_r, k_w = n\]

Has solutions (assuming \( n \) in range) and \( j_w < j_r \) \( \Rightarrow \) direction \((*, <)\)

Directions \\{\((<, <), (=, <), (> , <)\)\} or \(\delta_1, \delta_2, \delta_1^{-1}\)
### Vectorisation algorithm

Example

| S₁ | Do i = 1,100  
|    | x(i) = y(i) + 10  
|    | Do j = 1,100  
| S₂ | b(j) = a(j,n)  
|    | Do k = 1,100  
| S₃ | a(j+1,k) = b(j)+c(j,k)  
|    | Enddo  
| S₄ | y(i+j) = a(j+1,n)  
|    | Enddo  
|    | Enddo

Label and edge for this dependence?
Vectorisation algorithm

Example

Do \( i = 1,100 \)
\[ \text{x}(i) = \text{y}(i) + 10 \]
Do \( j = 1,100 \)
\[ \text{b}(j) = \text{a}(j,n) \]
Do \( k = 1,100 \)
\[ \text{a}(j+1,k) = \text{b}(j) + \text{c}(j,k) \]
Enddo
\[ \text{y}(i+j) = \text{a}(j+1,n) \]
Enddo
Enddo

Directions \{(<,=),(=,=),(>,=)\} or \( \delta_1, \delta_\infty, \delta_1^{-1} \)
Vectorisation algorithm

Example

Do \( i = 1,100 \)
\[ x(i) = y(i) + 10 \]
Do \( j = 1,100 \)
\[ b(j) = a(j,n) \]
Do \( k = 1,100 \)
\[ a(j+1,k) = b(j)+c(j,k) \]
Enddo
\[ y(i+j) = a(j+1,n) \]
Enddo
Enddo

Label and edge for this dependence?
Vectorisation algorithm

Example

\[ \text{Do } i = 1,100 \]
\[ x(i) = y(i) + 10 \]
\[ \text{Do } j = 1,100 \]
\[ b(j) = a(j,n) \]
\[ \text{Do } k = 1,100 \]
\[ a(j+1,k) = b(j) + c(j,k) \]
\[ \text{Enddo} \]
\[ y(i+j) = a(j+1,n) \]
\[ \text{Enddo} \]
\[ \text{Enddo} \]

Output dependence on itself, at level 1 because \( i \) unconstrained.
Vectorisation algorithm

Example

| $S_1$ | Do $i = 1,100$
|-------|----------------|
|       | $x(i) = y(i) + 10$
|       | Do $j = 1,100$
| $S_2$ | $b(j) = a(j,n)$
|       | Do $k = 1,100$
| $S_3$ | $a(j+1,k) = b(j)+c(j,k)$
|       | Enddo
| $S_4$ | $y(i+j) = a(j+1,n)$
|       | Enddo
|       | Enddo

Label and edge for this dependence?
### Example

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| $S_1$ | Do $i = 1,100$
|   | $x(i) = y(i) + 10$
|   | Do $j = 1,100$
| $S_2$ | $b(j) = a(j,n)$
|   | Do $k = 1,100$
| $S_3$ | $a(j+1,k) = b(j) + c(j,k)$
|   | Enddo
| $S_4$ | $y(i+j) = a(j+1,n)$
|   | Enddo
|   | Enddo

Output dependence on itself, at level 1 because $i$ unconstrained.
Example

\( S_1 \)

\[
\text{Do } i = 1,100 \\
x(i) = y(i) + 10 \\
\text{Do } j = 1,100 \\
\]

\( S_2 \)

\[
b(j) = a(j,n) \\
\text{Do } k = 1,100 \\
\]

\( S_3 \)

\[
a(j+1,k) = b(j)+c(j,k) \\
\text{Enddo} \\
\]

\( S_4 \)

\[
y(i+j) = a(j+1,n) \\
\text{Enddo} \\
\text{Enddo} \\
\]

Label and edge for this dependence?
Vectorisation algorithm

Example

Output dependence on itself, at level 1 because $i$ unconstrained.
Vectorisation algorithm

Example

| $S_1$ | Do $i = 1,100$
|       | $x(i) = y(i) + 10$
|       | Do $j = 1,100$
| $S_2$ | $b(j) = a(j,n)$
|       | Do $k = 1,100$
| $S_3$ | $a(j+1,k) = b(j) + c(j,k)$
|       | Enddo
| $S_4$ | $y(i+j) = a(j+1,n)$
|       | Enddo
|       | Enddo

All the edges
Vectorisation algorithm

Example

| Example | Do $i = 1,100$
|---------|------------------|
| $S_1$   | $x(i) = y(i) + 10$
|         | Do $j = 1,100$
| $S_2$   | $b(j) = a(j,n)$
|         | Do $k = 1,100$
| $S_3$   | $a(j+1,k) = b(j)+c(j,k)$
|         | Enddo
| $S_4$   | $y(i+j) = a(j+1,n)$
|         | Enddo
|         | Enddo

What are the SCCs?
Vectorisation algorithm

Example

| $S_1$ | Do i = 1,100  
   | $\quad x(i) = y(i) + 10$  
   | Do j = 1,100  
   | $\quad b(j) = a(j,n)$  
   | Do k = 1,100  
   | $\quad a(j+1,k) = b(j) + c(j,k)$  
   | Enddo  
| $S_2$ | $\quad y(i+j) = a(j+1,n)$  
   | Enddo  
| $S_3$ | $\quad$  
| $S_4$ | $\quad$  

Diagram: 

- $S_1$ to $S_2$ with label $\delta_0$
- $S_2$ to $S_3$ with label $\delta_1$
- $S_3$ to $S_4$ with label $\delta^{-1}_1$
- $S_4$ to $S_1$ with label $\delta_0$
Vectorisation algorithm

Example

Vectorise(Region, LoopDepth, DDG)

Vectorise({S_1, S_2, S_3, S_4}, 1)

SCCs and topological sort gives 
{S_2, S_3, S_4}, {S_1}

Do i = 1, 100
  Vectorise({S_2, S_3, S_4}, 2)
Enddo

Vectorise({S_1}, 1)
Vectorisation algorithm

Example

Vectorise(Region, LoopDepth, DDG)

Vectorise(\{S_1, S_2, S_3, S_4\}, 1)

SCCs and topological sort gives
\{S_2, S_3, S_4\}, \{S_1\}

Do i = 1, 100
  Vectorise(\{S_2, S_3, S_4\}, 2)
Enddo

Vectorise(\{S_1\}, 1)
Vectorisation algorithm

Example

Vectorise(Region, LoopDepth, DDG)

Vectorise({S_1}, 1)

Distribute

Do i = 1, 100
   Vectorise({S_2, S_3, S_4}, 2)
Enddo

Do i = 1, 100
   x(i) = y(i) + 10
Enddo
Vectorisation algorithm

Example

Vectorise(Region, LoopDepth, DDG)

Vectorise(\{S_1\}, 1)

Vectorise

\begin{align*}
\text{Do } i & = 1, 100 \\
& \quad \text{Vectorise(}\{S_2, S_3, S_4\}, 2) \\
\text{Enddo} \\
x(1:100) & = y(1:100) + 10
\end{align*}
Vectorisation algorithm

Example

Vectorise(Region, LoopDepth, DDG)

Vectorise(\{S_1\}, 1)

Vectorise

Do i = 1, 100
    Vectorise(\{S_2, S_3, S_4\}, 2)
Enddo

x(1:100) = y(1:100) + 10
Vectorisation algorithm
Example

Vectorise(Region, LoopDepth, DDG)

Vectorise(\{S_2, S_3, S_4\}, 2)

SCCs and topological sort gives
\{S_2, S_3\}, \{S_4\}

Do i = 1, 100
  Do j = 1, 100
    Vectorise(\{S_2, S_3\}, 3)
    Enddo
  Vectorise(\{S_4\}, 2)
Enddo

x(1:100) = y(1:100) + 10

Level 1 dependences removed
Vectorisation algorithm

Example

Vectorise(Region, LoopDepth, DDG)

Vectorise(\{S_2, S_3, S_4\}, 2)

SCCs and topological sort gives
\{S_2, S_3\}, \{S_4\}

Do i = 1, 100
  Do j = 1, 100
    Vectorise(\{S_2, S_3\}, 3)
    Enddo
  Vectorise(\{S_4\}, 2)
Enddo

x(1:100) = y(1:100) + 10

Level 1 dependences removed
Vectorisation algorithm

Example

Vectorise(Region, LoopDepth, DDG)

Vectorise(\{S_2, S_3, S_4\}, 2)

SCCs and topological sort gives \{S_2, S_3\}, \{S_4\}

Do i = 1, 100
    Do j = 1, 100
        Vectorise(\{S_2, S_3\}, 3)
        Enddo
    Enddo

y(i+1:i+100) = a(2:101,N)

Enddo

x(1:100) = y(1:100) + 10

Level 1 dependences removed
Vectorisation algorithm

Example

Vectorise(Region, LoopDepth, DDG)

Vectorise(\{S_2, S_3, S_4\}, 2)

SCCs and topological sort gives
\{S_2, S_3\}, \{S_4\}

Do i = 1, 100
   Do j = 1, 100
      Vectorise(\{S_2, S_3\}, 3)
      Enddo
   Enddo
   y(i+1:i+100) = a(2:101,N)
Enddo

x(1:100) = y(1:100) + 10

Level 1 dependences removed
Vectorisation algorithm

Example

\[ \text{Vectorise}(\text{Region, LoopDepth, DDG}) \]

\[ \text{Vectorise}([S_2, S_3], 3) \]

SCCs and topological sort gives
\{S_2\}, \{S_3\}

Do i = 1, 100
  Do j = 1, 100
    Vectorise([S_2], 3)
    Vectorise([S_3], 3)
  Enddo
  y(i+1:i+100) = a(2:101,N)
Enddo

x(1:100) = y(1:100) + 10
Vectorisation algorithm

Example

Vectorise(Region, LoopDepth, DDG)

Vectorise(\{S_2, S_3\}, 3)

SCCs and topological sort gives
\{S_2\}, \{S_3\}

Do i = 1, 100
  Do j = 1, 100
    b(j) = a(j,n)
    a(j+1,1:100)=b(j)+c(j,1:100)
  Enddo
  y(i+1:i+100) = a(2:101,N)
Enddo
x(1:100) = y(1:100) + 10

Note $S_2$ not in depth 3 – leaves single statement
Dependency reducing transforms

- What happened if no vectorisable regions found?
- Try transformations
Dependency reducing transforms

Loop Interchange

Loop interchange: move loop carried dependences outermost

\[
\begin{align*}
\text{Do } j &= 1, M \\
\text{Do } i &= 1, N \\
&\quad a(i+1,j) = a(i,j) + c \\
&\quad \text{Enddo} \\
&\quad \text{Enddo} \\
\text{Enddo}
\end{align*}
\]

Distance \([0,1]\). Even if \(j\) run sequentially, loop carried dep \(i\) not vectorisable.

\[
\begin{align*}
\text{Do } i &= 1, N \\
\text{Do } j &= 1, M \\
&\quad a(i+1,j) = a(i,j) + c \\
&\quad \text{Enddo} \\
&\quad \text{Enddo} \\
\text{Enddo}
\end{align*}
\]

Now \([1,0]\) - inner loop vectorisable

\[
\begin{align*}
\text{Do } i &= 1, N \\
&\quad a(i+1,1:N) = a(i,1:N) + c \\
&\quad \text{Enddo}
\end{align*}
\]
Dependency reducing transforms
Scalar expansion

Convert a scalar in loop to array with one element per iteration

Example
Do i = 1, N
  t = a(i)
  a(i) = b(i)
  b(i) = t
Enddo

Where are the dependences?
(Ignore output dependences)
Dependency reducing transforms
Scalar expansion

Convert a scalar in loop to array with one element per iteration

**Example**

Do \( i = 1, N \)
  \( t = a(i) \)
  \( a(i) = b(i) \)
  \( b(i) = t \)
Enddo

Cycle in dependence graph prevents distribution and vectorisation
Dependency reducing transforms
Scalar expansion

Convert a scalar in loop to array with one element per iteration

Example
Do i = 1, N
   tt(i) = a(i)
   a(i) = b(i)
   b(i) = tt(i)
Enddo

Easily distributed and vectorised

Anti dependence removed
Dependency reducing transforms
Scalar expansion

May fail to remove dependence

Original
Do i = 1, N
t = t + a(i) + a(i+1)
a(i) = t
Enddo

Still cyclic
	tt(0) = t Do i = 1, N
tt(i) = t(i-1) + a(i) + a(i+1)
a(i) = tt(i)
Enddo
t = tt(N)

- Whether or not scalar expansion can break cycles depends on whether it is a covering definition (see CMA)
- In practise recurrence on the scalar is the biggest problem.

Covering definition
Definition X of scalar S covers the loop, if no earlier definition of S in the loop could reach a use after X
Dependency reducing transforms
Scalar renaming

Can be used to eliminate loop independent output and anti-dependences

Original
Do i = 1, N
    t = a(i) + b(i)
    c(i) = t + t
    t = d(i) - b(i)
    a(i+1) = t * t
Enddo

Renamed
Do i = 1, N
    t1 = a(i) + b(i)
    c(i) = t1 + t1
    t2 = d(i) - b(i)
    a(i+1) = t2 * t2
Enddo

Scalar expansion, loop distribution and vectorisation now possible
Scalar expansion and renaming cannot eliminate all cycles

Original
Do i = 1, N
  a(i) = x(i+1) + x(i)
  x(i+1) = b(i) + t
Enddo

- Renaming does not break cycle. Critical anti-dependence
**Dependency reducing transforms**

**Node splitting**

Scalar expansion and renaming cannot eliminate all cycles

**Split**

\[
\text{Do } i = 1, N \\
x_{x}(i) = x(i+1) \\
a(i) = xx(i) + x(i) \\
x(i+1) = b(i) + t
\]

Enddo

- Cycle broken. Vectorisable with statement reordering: $S_0, S_2, S_1$
- NP-Complete to find minimal critical dependences
Summary

- Vector loops
- Loop distribution
- Dependence condition for vectorisation
- Vectorisation algorithm based on SCC and hierarchical dependences
- Loop Interchange
- Scalar Expansion, Renaming and Node splitting
- Layout in memory important too!
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