Compiler Optimisation

10 – Vectorisation

Hugh Leather
IF 1.18a
hleather@inf.ed.ac.uk

Institute for Computing Systems Architecture
School of Informatics
University of Edinburgh

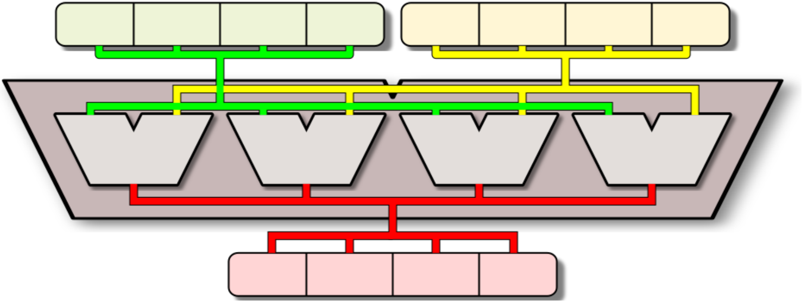
2019

Introduction

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

- 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)



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

No loop carried dependence

```
Do i = 1, N
  x(i) = x(i) + c
Enddo
```

Vectorisable

```
x(1:N) = x(1:N) + c
```

Loop carried dependence

```
Do i = 1, N
  x(i+1) = x(i) + c
Enddo
```

Not vectorisable

```
x(2:N+1) = x(1:N) + c
```

Reads x at once

Vectorisation

Varying vector length

Vector registers are a fixed size. Need to fit code to registers

Original

```
Do i = 1, N
  x(i) = x(i) + c
Enddo
```

Strip-mined

```
Do i = 1, N, s
  Do ii = i, i+s-1
    x(ii) = x(ii) + c
  Enddo
Enddo
```

Vectorised

```
Do i = 1, N, s
  x(i:i+s-1) = x(i:i+s-1) + c
Enddo
```

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

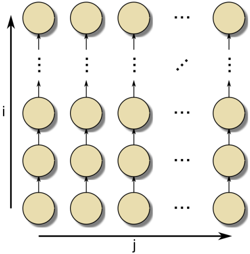
```
a(2:N+1) = b(1:N) + c
d(1:N) = a(1:N) + e
```

Cyclic dependence prevent distribution and hence vectorisation.

Vectorised

Inner loop 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 [1]
     - Goto 1 with SCC as program
     - Write loop end

[1] loop will be sequentialised

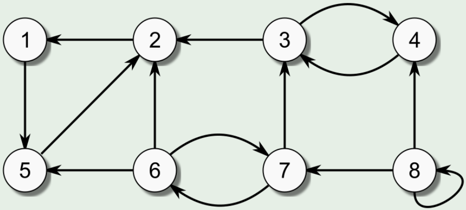
Strongly connected

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

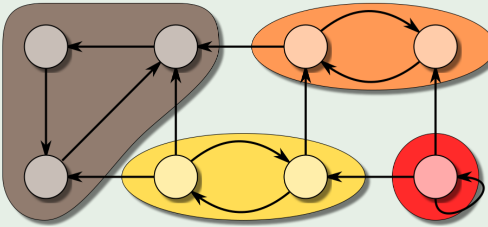
Strongly connected

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?



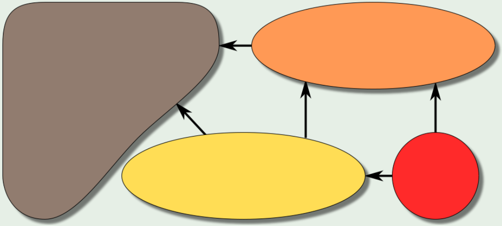
[2]Use Tarjan's algorithm

Topological sort of acyclic directed graph

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

Maximal SCC graphs are acyclic directed

What is the topological sort?



Vectorisation algorithm

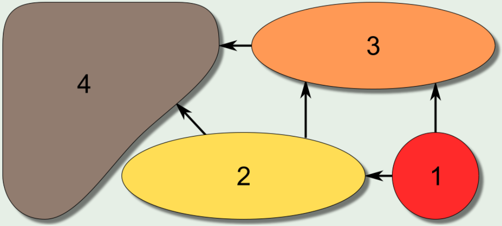
Review: topological sort

Topological sort of acyclic directed graph

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

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

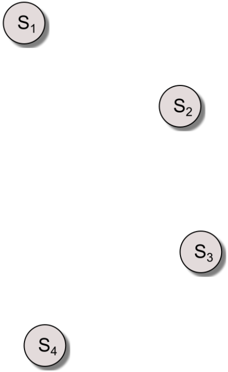
Level of loop carried dependence is the index of the left-most non "$=$" in direction vector.
Written as subscript, e.g. δ_1 for $(<, =, =)$, δ_3^{-1} for $(=, =, >)$.
Infinity for in same loop, e.g. δ_∞ for $(=, =, =)$

Vectorisation algorithm
Example

Example

```
     Do i = 1,100
S1     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?

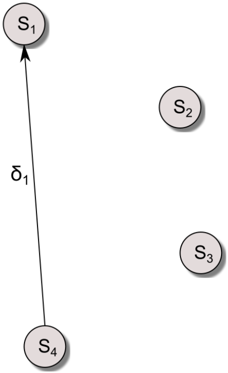


Vectorisation algorithm

Example

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`



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

$$i_w + j_w = i_r$$

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

Loop carried flow dependence, level one (δ_1)

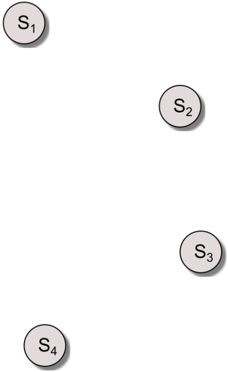
Vectorisation algorithm

Example

Example

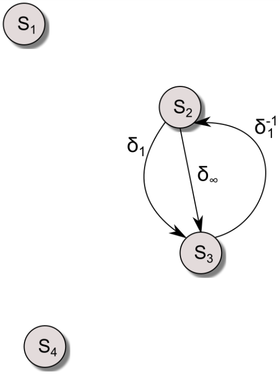
```
      Do i = 1,100
S1      x(i) = y(i) + 10
        Do j = 1,100
S2        b(j) = a(j,n)
          Do k = 1,100
S3          a(j+1,k) = b(j)+c(j,k)
          Enddo
S4        y(i+j) = a(j+1,n)
        Enddo
      Enddo
```

Label and edge for this dependence?



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
```



Clearly direction for j loop is $=$.

For i loop, i is not in either array subscript, so *.

So, direction is $(*, =)$ or $\{(<, =), (=, =), (>, =)\}$ or $\delta_1, \delta_\infty, \delta_1^{-1}$

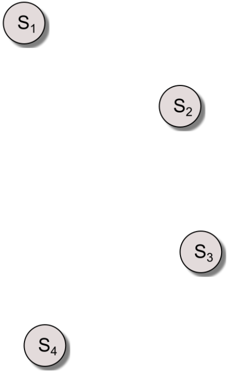
Vectorisation algorithm

Example

Example

```
     Do i = 1,100
S1     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
```

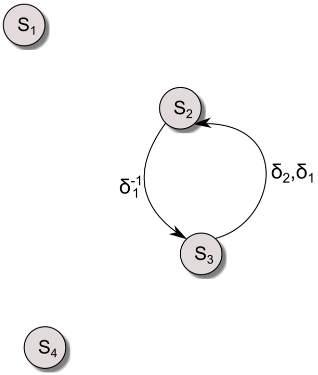
Label and edge for this dependence?



Vectorisation algorithm
Example

Example

```
      Do i = 1,100
S1      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
```



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

$$j_w + 1 = j_r, k_w = n$$

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

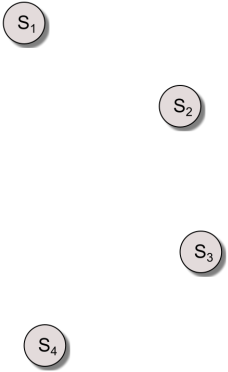
Directions $\{(<,<), (=,<), (>,<)\}$ or $\delta_1, \delta_2, \delta_1^{-1})$

Vectorisation algorithm
Example

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
```

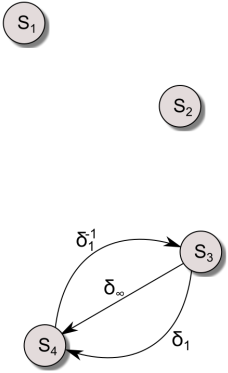
Label and edge for this dependence?



Vectorisation algorithm
Example

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`



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

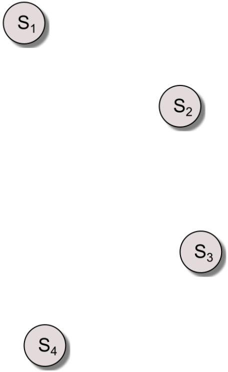
Vectorisation algorithm

Example

Example

```
     Do i = 1,100
S1     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?

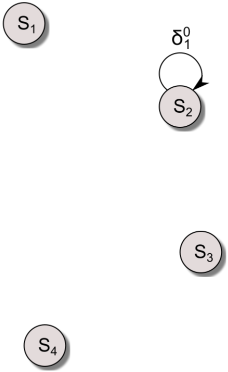


Vectorisation algorithm

Example

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`

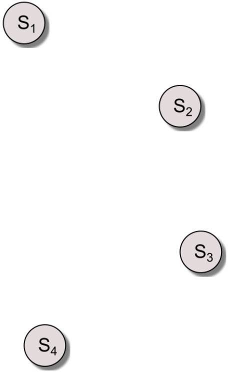


Output dependence on itself, at level 1 because *i* unconstrained.

Vectorisation algorithm
Example

Example

```
     Do i = 1,100
S1     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
```

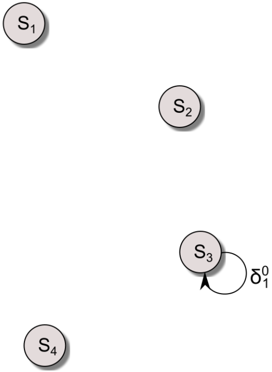


Label and edge for this dependence?

Vectorisation algorithm
Example

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`



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

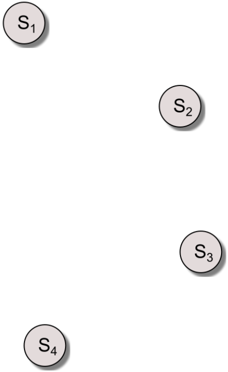
Vectorisation algorithm

Example

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?



Example

```
      Do i = 1,100
S1      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
```

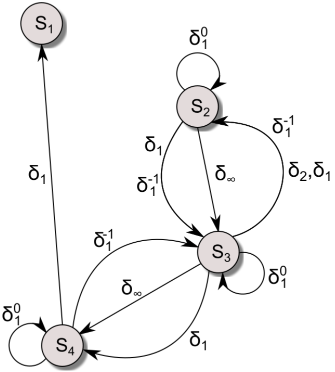
Output dependence on itself, at level 1 because *i* unconstrained.

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`

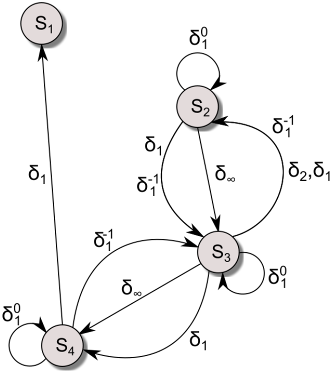


All the edges

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`

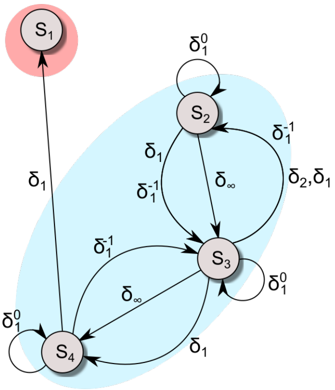


What are the SCCs?

Vectorisation algorithm
Example

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



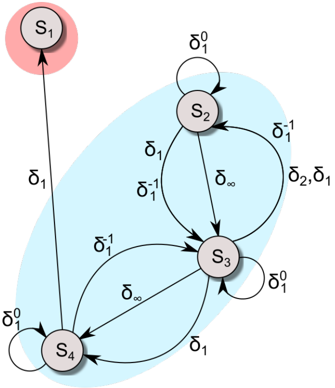
Vectorisation algorithm
Example

`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($\{S_1\}, 1$)



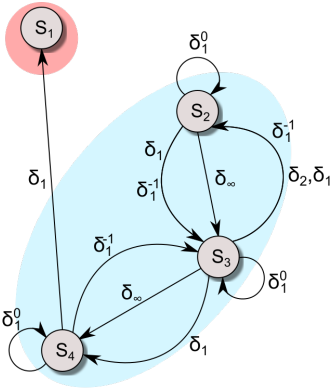
Vectorisation algorithm
Example

`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($\{\mathbf{S_1}\}, \mathbf{1}$)



Vectorisation algorithm
Example

```
Vectorise(Region, LoopDepth, DDG)
```

Vectorise($\{S_1\}, 1$)

Distribute

```
Do i = 1, 100
```
$\mathit{Vectorise}(\{S_2, S_3, S_4\}, 2)$
```
Enddo
Do i = 1, 100
  x(i) = y(i) + 10
Enddo
```

Vectorisation algorithm
Example

```
Vectorise(Region, LoopDepth, DDG)
```

Vectorise($\{S_1\}$, 1)

Vectorise

```
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($\{S_1\}$, 1)

Vectorise

```
Do i = 1, 100
```
  Vectorise($\{S_2, S_3, S_4\}$, 2)
```
Enddo
x(1:100) = y(1:100) + 10
```

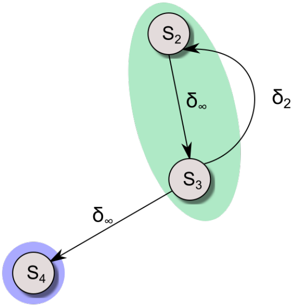
Vectorisation algorithm
Example

```
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({S2, S3}, 3)
  Enddo
  Vectorise({S4}, 2)
Enddo
x(1:100) = y(1:100) + 10
```



Level 1 dependences removed

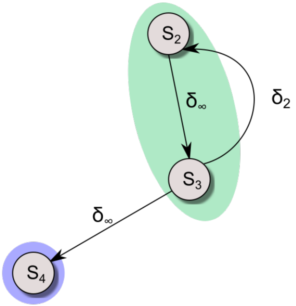
Vectorisation algorithm
Example

`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($\{\mathbf{S_4}\}, \mathbf{2}$)
Enddo
`x(1:100) = y(1:100) + 10`



Level 1 dependences removed

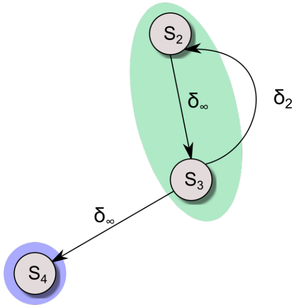
Vectorisation algorithm
Example

```
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({S2, S3}, 3)
  Enddo
  y(i+1:i+100) = a(2:101,N)
Enddo
x(1:100) = y(1:100) + 10
```



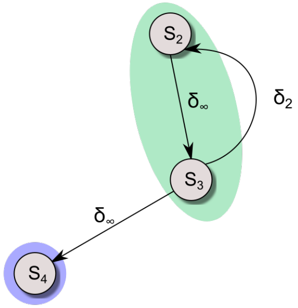
Level 1 dependences removed

```
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({S2, S3}, 3)
  Enddo
  y(i+1:i+100) = a(2:101,N)
Enddo
x(1:100) = y(1:100) + 10
```



Level 1 dependences removed

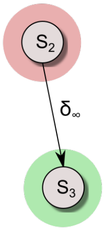
Vectorisation algorithm
Example

```
Vectorise(Region, LoopDepth, DDG)
```

Vectorise($\{S_2, S_3\}$, 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



```
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

Loop interchange: move loop carried dependences outermost

```
Do j = 1, M
  Do i = 1, N
    a(i+1,j) = a(i,j) + c
  Enddo
Enddo
```

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

```
Do i = 1, N
  Do j = 1, M
    a(i+1,j) = a(i,j) + c
  Enddo
Enddo
```

Now [1,0] - inner loop vectorisable

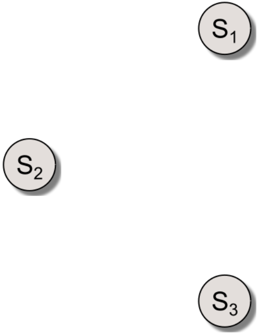
```
Do i = 1, N
  a(i+1,1:N) = a(i,1:N) + c
Enddo
```

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
```



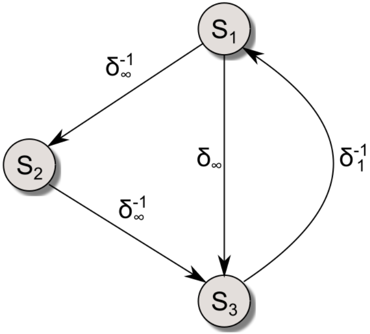
Where are the dependences?
(Ignore output dependences)

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

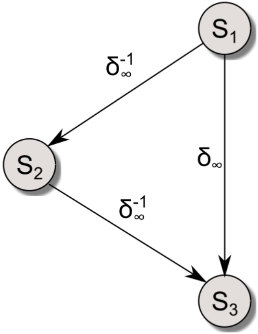


Convert a scalar in loop to array with one element per iteration

Example

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

Easily distributed and vectorised



Anti dependence removed

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

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
```

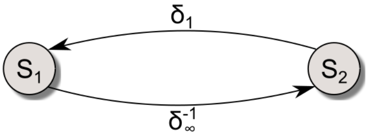
Scalar expansion, loop distribution and vectorisation now possible

Dependency reducing transforms
Node splitting

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
```

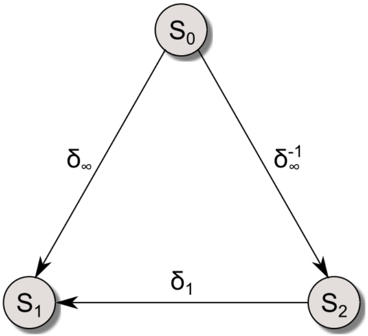


- Renaming does not break cycle. Critical anti-dependence

Scalar expansion and renaming cannot eliminate all cycles

Split

```
Do i = 1, N
  xx(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!

PPar CDT Advert