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**

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
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
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
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. Maximal SCCs are largest possible.

What are the SCCs?

\[^{2}\text{Use Tarjan’s algorithm}\]
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?
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
**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 $(<,=,=)$, $\delta_{-1}^3$ for $(=,=,>)$.

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

```
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
```

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

\[ i_w + j_w = i_r \]

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

<table>
<thead>
<tr>
<th>Step</th>
<th>Statement</th>
</tr>
</thead>
</table>
| 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

\[
\begin{align*}
S_1 & : \text{Do } i = 1,100 \\
& \quad x(i) = y(i) + 10 \\
& \quad \text{Do } j = 1,100 \\
S_2 & : \quad b(j) = a(j,n) \\
& \quad \text{Do } k = 1,100 \\
S_3 & : \quad a(j+1,k) = b(j)+c(j,k) \\
& \quad \text{Enddo} \\
S_4 & : \quad y(i+j) = a(j+1,n) \\
& \quad \text{Enddo} \\
& \quad \text{Enddo}
\end{align*}
\]

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

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

Label and edge for this dependence?
Vectorisation algorithm

Example

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
</table>
| $S_1$ | 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
| $S_4$ | 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 $(\ast, <)$

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

Example

Example

| $S_1$ | 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
| $S_4$ | $y(i+j) = a(j+1,n)$
| | Enddo
| | Enddo

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

\begin{itemize}
\item \textbf{S}_1 \quad \text{Do } i = 1,100
\begin{align*}
    x(i) &= y(i) + 10 \\
    \text{Do } j &= 1,100
\end{align*}
\item \textbf{S}_2 \quad \text{Do } k = 1,100
\begin{align*}
    b(j) &= a(j,n) \\
    a(j+1,k) &= b(j)+c(j,k)
\end{align*}
\item \textbf{S}_3 \quad \text{Enddo}
\item \textbf{S}_4 \quad y(i+j) = a(j+1,n)
\item \text{Enddo}
\end{itemize}

Label and edge for this dependence?
## Vectorisation algorithm

### 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.
Vectorisation algorithm

Example

```
Example

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

Label and edge for this dependence?
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
Enddo
```

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

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

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| **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

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

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

All the edges
Vectorisation algorithm

Example

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

What are the SCCs?
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
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

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)

\[
\text{Vectorise}\left(\{S_2, S_3, S_4\}, 2\right)
\]

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
  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

**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)

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

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

```
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)
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 \(\text{CMA}\))
- 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

<table>
<thead>
<tr>
<th>Original</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do i = 1, N</td>
</tr>
<tr>
<td>t = a(i) + b(i)</td>
</tr>
<tr>
<td>c(i) = t + t</td>
</tr>
<tr>
<td>t = d(i) - b(i)</td>
</tr>
<tr>
<td>a(i+1) = t * t</td>
</tr>
<tr>
<td>Enddo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Renamed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do i = 1, N</td>
</tr>
<tr>
<td>t1 = a(i) + b(i)</td>
</tr>
<tr>
<td>c(i) = t1 + t1</td>
</tr>
<tr>
<td>t2 = d(i) - b(i)</td>
</tr>
<tr>
<td>a(i+1) = t2 * t2</td>
</tr>
<tr>
<td>Enddo</td>
</tr>
</tbody>
</table>

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

Original

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

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

Scalar expansion and renaming cannot eliminate all cycles

Split

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

Now accepting applications!
Find out more and apply at: pervasiveparallelism.inf.ed.ac.uk