Vectorisation

Michael O’Boyle

February, 2011
Course Structure

• Course work deadline today. New coursework today - see website

• 4/5 lectures on high level restructuring for parallelism and memory

• Dependence Analysis

• Program Transformations

• **Automatic vectorisation** ch2 and 5 of Allen and Kennedy

• Automatic parallelisation

• Speculative Parallelisation
Lecture Overview

• 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
Vector code

- 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

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

No loop carried dependence [0] \hspace{1cm} \text{Loop carried dependence [1]}

Vectorisable \hspace{1cm} \text{Not vectorisable}

\begin{align*}
x(1:N) = x(1:N) + c
\end{align*}
Vector code: varying vector length

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

Do i = 1,N,s

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

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

Enddo

Original

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

Strip-mine

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

Vectorise

M. O'Boyle

Vectorisation

February, 2011
Loop Distribution + Statement reordering

Standard approach to isolating statements within a loop for later vectorisation

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

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

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

Cyclic dependence prevent distribution and hence vectorisation. Examine techniques to overcome this.
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 dep (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 Ch2 algorithm. Look at Ch2 for more details

- Form dependence graph

- Strongly Connected Component (SCC) identification (cycles)

- Separate out weakly connected and vectorise using loop distribution and statement reordering

- Strip off outer dependence level (loop will be sequentialised) and repeat
Running 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

Use $d$ notation where $d^y_{x}$ is a dependence of type y at loop level x.

Loops numbered from outermost x=1 ... Infinity means within a loop, not loop
carried. y=0 output, y=-1 anti else flow.

Loop carried flow dependence from s4 to s1 on y. $d_1$
Running Example with S1 dependences

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

Loop carried flow dependence from s4 to s1 on y. \( d_1 \)
No other dependences reach s1
Running Example S2 dependences

\[ d_1^o \]

\[ s_2 \]

\[ d_2 \]

\[ d_1 \]

\[ d_1^{-1} \]

\[ s_3 \]

b(j) in s2 has two flow dependences with s3. Loop carried and loop independent \( d_1, d_{inf} \).

Corresponding loop carried anti dep from s3 to s2 \( d_1^{-1} \). Finally loop carried output dependence in s2

a(j+1,k) in s3 has a level one and two flow dep with s2. Corresponding loop carried antidep from s2 to s3.
Example S3 dependences

Loop carried and independent flow dependence from $a(j+1,k)$ in $s3$ to $s4$

Corresponding loop carried anti-dep from $s4$. Output dependence
Example S4 dependences

- Trivial loop carried output dependence in s4 at level 1 on write to \( y(i+j) \).
- Other dependence with s1 already shown
• Analysing connected components using Tarjan’s algorithm. Two separate
groups (s1),(s2,s3,s4).

• Separate out s1 by loop distribution. Statement reordering required here.
Vectorisation algorithm

Apply on outer level vectorise \((s1,s2,s3,s4,1)\).

\(s1\) not part of SC. Loop distribution and statement reordering and vectorised gives

\[
\begin{align*}
\text{Do } i &= 1,100 \\
\text{vectorise } \{s2,s3,s4\},2 & \quad \text{vectorise } \{s2,s3,s4\},2 \\
\text{Enddo} & \quad \text{Enddo} \\
\text{Do } i &= 1,100 \\
\text{s1 } x(i) &= y(i) +10 & \quad \text{x(1:100)} = y(1:100) +10 \\
\text{Enddo} & \quad \text{Enddo}
\end{align*}
\]

Apply algorithm at next level stripping of level 1 dependences
Vectorise(\{s2,s3,s4\},2) level 1 dependences stripped off

- Analysing connected components using Tarjan’s algorithm. Two separate groups \((s4),(s2,s3)\).

- Separate out \(s4\) by loop distribution. No Statement reordering required here.
Vectorisation algorithm

Apply on level2 vectorise (s1,s2,s3,2).

s4 not part of SCC. Isolated, distributed and vectorised giving

\[
\begin{align*}
\text{Do i} & = 1,100 \\
\text{Do j} & = 1,100 \\
& \quad \text{vectorise}([\{s2,s3\},3]) \\
& \quad \text{Enddo} \\
\text{s4 } & \ y(i+1:i+100) = a(2:101,N) \\
& \quad \text{Enddo} \\
\text{s1 } & \ x(1:100) = y(1:100) + 10
\end{align*}
\]

- Apply vectorise again striping off next level
Vectorise({s2,s3},3) level 1,2 dependences stripped off

- Analysing connected components using Tarjan’s algorithm. Two separate groups (s2),(s3).
- Separate out by loop distribution. No Statement reordering required here.
Vectorisation algorithm : Final code

Do i = 1,100
    Do j = 1,100
        b(j) = a(j,n)
    Enddo
    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

As s2 has no loop of depth 3, distribution leaves a single statement.

• 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

[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
Scalar expansion

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

Cycle in dependence graph prevents distribution and vectorisation (output not shown)

Try to eliminate anti-dependence with scalar expansion
Anti-dependence removed eliminating cycle

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

Can now be easily distributed and vectorised
Scalar expansion : may fail with subsequent uses

\[ \text{tt}(0) = t \]

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

\[ \text{tt}(i) = \text{tt}(i-1) + a(i) + a(i+1) \]
\[ a(i) = \text{tt}(i) \]
\[ \text{Enddo} \]
\[ t = \text{tt}(N) \]

- Whether or not scalar expansion can break cycles depends on whether it is a covering definitions

- A covering definition for a use means that there are subsequent later uses.

- In practise recurrence on the scalar is the biggest problem.
Scalar Renaming

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

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

Do i = 1, N
  t1 = t + 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
Node Splitting

• Scalar expansion and renaming cannot eliminate all cycles

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

• Renaming does not break cycle. Critical anti-dependence
Node Splitting

• Make copy of node where anti-dep starts

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: s0, s2, s1

• NP-C to find minimal critical deps!
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
- Used in Media SIMD instructions/ GPUs