Cheaper Adjoins by
Reversing Address Computations

Jean Utke, Uwe Naumann, Laurent Hascoët

ANL, RWTH, INRIA

Euro-AD workshop, Aachen
December 2006
Motivation

\[
\begin{align*}
\text{do } i &= 1, N \\
PUSH(j) \\
j &= IND(i) \\
... \\
...(j) ... \\
......(j) ... \\
...(j) ...... \\
end \ do
\end{align*}
\]

\[
\begin{align*}
\text{do } i &= 1, N \\
... \\
PUSH(j) \\
j &= j+1 \\
... \\
PUSH(j) \\
j &= j+2 \\
... \\
end \ do
\end{align*}
\]

⇒ Obviously, we mustn’t store \textit{j}!
Some related questions

- Detection and expansion of Induction Variables, to make a loop parallel.
- Forward recomputation (Recompute-All reverse AD)
- Pointer arithmetic, i.e. index \( j \) can be replaced by a pointer.
- Avoiding storage of control flow decisions.
Elementary tactics, to be combined

- **inversion** of address computation:
  \[ i = j + k + 2 \] can be inversed for \( j \) given \( i \) and \( k \), or for \( k \) given \( i \) and \( j \), (Any use of an address can be helpful for inversion!)

- **forward address recomputation:**
  \[ i = j + k + 2 \] can recompute \( i \) given \( j \) and \( k \),

- when everything else fails, **storage** is still an option!
Scheduling inversion of address computations

Reversing the original order does not work!

<table>
<thead>
<tr>
<th>Orig. loop:</th>
<th>wrong backward sweep:</th>
</tr>
</thead>
<tbody>
<tr>
<td>j = 3</td>
<td>.</td>
</tr>
<tr>
<td>do while(j&lt;100)</td>
<td>3,? 6,4</td>
</tr>
<tr>
<td>k = j+1</td>
<td>3,4 6,7</td>
</tr>
<tr>
<td>j = k+2</td>
<td>6,4 9,7</td>
</tr>
<tr>
<td>S(j,k)</td>
<td>6,4 9,7</td>
</tr>
<tr>
<td>end do</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>do while(j&gt;=5)</td>
</tr>
<tr>
<td></td>
<td>k = j-2</td>
</tr>
<tr>
<td></td>
<td>S(j,k)</td>
</tr>
<tr>
<td></td>
<td>j = k-1</td>
</tr>
<tr>
<td></td>
<td>end do</td>
</tr>
<tr>
<td></td>
<td>9,7 6,7</td>
</tr>
<tr>
<td></td>
<td>9,7 6,7</td>
</tr>
<tr>
<td></td>
<td>9,7 6,4</td>
</tr>
<tr>
<td></td>
<td>6,7 3,4</td>
</tr>
</tbody>
</table>

Find a correct scheduling of control reversal and adjoint statements!

Utke, Naumann, Hascoet (ANL, RWTH, INRIA)
Move *tokens* representing active values on the DDG:

\begin{verbatim}
  do while(j>=5)
    S(j,k)
    j = k-1
    k = j-2
  end do
\end{verbatim}
Data-Dependence Cycles

If forward recomputation is not an option, looking for SCC’s and cycles of true dependences and invertible value dependences is a good idea.

If forward recomputation is an option, then we must play the “token game” on the data-dependence graph of the loop. Problem becomes combinatorial!

- prefer nodes that can be recomputed by inversion
- else compute nodes that can be recomputed forward
- when blocked, use storage for one node (. . . but which one is best?)
do while(...) 
  l = i-j-3 
  m = l+3 
  i = l+2 
  j = j+m+4 
  i = i+j+1 
end do
do while(...) 
  l = i-j-3 
  m = l+3 
  i = l+2 
  j = j+m+4 
  i = i+j+1 
end do
do while(...)
  l = i-j-3
  m = l+3
  i = l+2
  j = j+m+4
  i = i+j+1
end do
Example

```
  do while(...)
    l = i-j-3
    m = l+3
    i = l+2
    j = j+m+4
    i = i+j+1
  end do
```

```
  do while(...)
    i = i-j-1
```

Utke, Naumann, Hascoet (ANL, RWTH, INRIA)
Rev. Address Comp.
Aachen 8 / 1
do while(...)  
l = i-j-3  
m = l+3  
i = l+2  
j = j+m+4  
i = i+j+1  
end do

\[
\begin{align*}
\text{do while(...)} & \quad \text{do while(...)} \\
i = i-j-1 & \quad i = i-j-1 \\
l = i-2 & \\
m = l+3 & \\
j = j-m-4 & \\
i = l+j+3 &
\end{align*}
\]
Example

```
do while (...)  
  l = i - j - 3  
  m = l + 3  
  i = l + 2  
  j = j + m + 4  
  i = i + j + 1  
end do
```

```
  i  
    j  
      l

  i  
    j  
      m

  i  
    j

  i  
    j
```

```
do while (...)  
  i = i - j - 1  
  l = i - 2  
  m = l + 3  
  j = j - m - 4  
  i = l + j + 3  
end do
```

Utke, Naumann, Hascoet (ANL, RWTH, INRIA)
Rev. Address Comp. Aachen 8 / 1
do while(...) 
  l = i-j-3
  m = l+3
  i = l+2
  j = j+m+4
  i = i+j+1
end do

do while(...)
  i = i-j-1
  l = i-2
do while(...) 
  l = i-j-3 
  m = l+3 
  i = l+2 
  j = j+m+4 
  i = i+j+1 
end do 

do while(...) 
  i = i-j-1 
  l = i-2 
  m = l+3 

do while(...)  
l = i-j-3  
m = l+3  
i = l+2  
j = j+m+4  
i = i+j+1  
end do  

do while(...)  
i = i-j-1  
l = i-2  
m = l+3
Example

\[
\begin{align*}
\text{do while(...)} & \quad \text{do while(...)} \\
\quad l &= i-j-3 & \quad i &= i-j-1 \\
\quad m &= l+3 & \quad l &= i-2 \\
\quad i &= l+2 & \quad m &= l+3 \\
\quad j &= j+m+4 & \quad j &= j-m-4 \\
\quad i &= i+j+1 & \\
\text{end do} & \quad \text{end do}
\end{align*}
\]
Example

\[
\text{do while(...)}
\]

\[
l = i-j-3
\]

\[
m = l+3
\]

\[
i = l+2
\]

\[
j = j+m+4
\]

\[
i = i+j+1
\]

\[
\text{end do}
\]
do while(...) 
  l = i-j-3  
  m = l+3  
  i = l+2  
  j = j+m+4  
  i = i+j+1  
end do

---

dc while(...) 
  i = i-j-1  
  l = i-2  
  m = l+3  
  j = j-m-4  
  i = 1+j+3

---

Utke, Naumann, Hascoet (ANL, RWTH, INRIA)  
Rev. Address Comp.  
Aachen 8 / 1
Example

do while(...)
  l = i-j-3
  m = l+3
  i = l+2
  j = j+m+4
  i = i+j+1
end do

Utke, Naumann, Hascoet (ANL, RWTH, INRIA)
Rev. Address Comp. Aachen 8 / 1
Many open questions...

- Data-Dependence graphs may become too large: need **hierarchical** methods that don’t lose too much detail.
- What is the **complexity** of the search for the inversion/recomputation/storage schedule?
- When in a blocked state, which is the **best node to store**?
- **Aliasing** blurs the way tokens can move up and down **true** dependences.