Applying vectorization methods to a Stochastic Dynamic Programming model for scheduling of off-shore fields

by

*Kjetil Haugen*

The Foundation for Scientific and Industrial Research at The Norwegian Institute of Technology, N-7034 Trondheim NORWAY.

E-mail: kh@moses.iok.unit.no
SUPERCOMPUTERS

* Fast numerical processing

* Parallel processing properties

- Vector computers
- Parallel computers
\[ C = A \cdot B \]
Vector processing "trade-off"

In certain situations, it may be optimal to introduce more calculations in order to utilize vector processing facilities.

Hence, a code written for a scalar and a vector computer may differ significantly as to the number of calculations done.

Still, the vectorization code may prove faster overall on a vector computer.
**Code Structure**

**Base Code**

For all states do:

1. Compute values $X(1:N)$
2. Find optimal value $x^* = \text{Max}\{x_1, \ldots x_N\}$ and decision $i^*$
3. Check if $i^*$ is legal

- Yes:
- No: Set $X(i^*) = -M$

**Alternative Code**

For all states do:

1. Initialize $X(1:N) : -M$
2. Loop over all $i$ (1:N) and compute $X(i)$ if $i$ is legal
3. Find optimal value $x^* = \text{Max}\{x_1, \ldots x_N\}$ and decision $i^*$
Comparing reference code/alternative code

Reference code vectorizes, alternative code does not vectorize expected value calculations.
Comparing Vector/Novector Performance

*Only ten first observations used*

- **Secs**
  - 1.0
  - 1.2
  - 1.4
  - 1.6
  - 1.8
  - 2.0
  - 2.2
  - 2.4
  - 2.6
  - 2.8
  - 3.0

- **# Innerloops**
  - 1
  - 2
  - 3
  - 4
  - 5
  - 6
  - 7
  - 8
  - 9

- **Lines**
  - **No-vector**
  - **Vector-version**
Constraint dependency

Performance as a function of "size" (how constrained the problem is)

Ratio

1.95
1.90
1.85
1.80
1.75
1.70
1.65
1.60

size /1000
## Description of Cases

<table>
<thead>
<tr>
<th>Case</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C(_1)</td>
<td>2026</td>
<td>Basic case</td>
</tr>
<tr>
<td>C(_2)</td>
<td>2092</td>
<td>All deterministic constraints removed</td>
</tr>
<tr>
<td>C(_3)</td>
<td>2224</td>
<td>Lowest demand level = second lowest</td>
</tr>
<tr>
<td>C(_4)</td>
<td>5104</td>
<td>All demand levels = largest demand level</td>
</tr>
<tr>
<td>C(_5)</td>
<td>6364</td>
<td>All values in demand level = max value of largest level</td>
</tr>
<tr>
<td>C(_6)</td>
<td>13753</td>
<td>All values in demand level = &quot;infinity&quot;, i.e. non-binding</td>
</tr>
</tbody>
</table>
Scalar Performance
*Examples executed on a 286-Pc*

Vector Performance
*Examples executed on a CRAY XMP-2/16*
# OBSERVED SPEED-UPs

<table>
<thead>
<tr>
<th>CASE</th>
<th>VEC.</th>
<th>NO-VEC.</th>
<th>RATIO</th>
<th>NO-VEC.</th>
<th>RATIO</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0.4468</td>
<td>1.3522</td>
<td>3.01</td>
<td>0.7338</td>
<td>1.64</td>
</tr>
<tr>
<td>C2</td>
<td>0.4583</td>
<td>1.3676</td>
<td>2.98</td>
<td>0.7529</td>
<td>1.64</td>
</tr>
<tr>
<td>C3</td>
<td>0.4987</td>
<td>1.4279</td>
<td>2.86</td>
<td>0.8008</td>
<td>1.61</td>
</tr>
<tr>
<td>C4</td>
<td>0.9238</td>
<td>3.0945</td>
<td>3.35</td>
<td>1.5941</td>
<td>1.73</td>
</tr>
<tr>
<td>C5</td>
<td>1.1178</td>
<td>3.7708</td>
<td>3.37</td>
<td>1.9074</td>
<td>1.71</td>
</tr>
<tr>
<td>C6</td>
<td>2.0100</td>
<td>7.8207</td>
<td>3.89</td>
<td>3.8556</td>
<td>1.92</td>
</tr>
</tbody>
</table>

*) This code does not inhibit vectorization*
VECTORIZATION CONCLUSIONS

1. SDP-code "relatively" simple to vectorize

2. Speed-up less than a factor 2

3. Severe I/O boundness. In fact, a simple prediction model for estimating maximal speed-up, gave a factor of 2.1