Vector Processors Part 2

Performance

Vector Execution Time

- Convoys
  - Set of vector instructions that could potentially begin execution together in one clock period. Cannot contain structural or data hazards. One convoy must finish execution before another begins (do not overlap in time).

- Chime
  - Unit of time taken to execute a convoy. Approximate measure, ignores some overheads like issue limitations.
  - $m$ convoys of $n$ length vectors execute in $m$ chimes, approximately $m \times n$ cycles.

- Startup costs
  - Pipeline latency
Example: $D = aX + Y$

<table>
<thead>
<tr>
<th>Unit</th>
<th>Start-up overhead (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load and store unit</td>
<td>12</td>
</tr>
<tr>
<td>Multiply</td>
<td>7</td>
</tr>
<tr>
<td>Add unit</td>
<td>6</td>
</tr>
</tbody>
</table>

### Strip-mining loop

$$\text{low} = 1$$

$$\text{VL} = (n \mod \text{MVL}) /* \text{find the odd-size piece */}$$

$$\text{do 10 i = low, low + VL - 1} /* \text{runs for length VL */}$$

$$Y(i) = a \times X(i) + Y(i) /* \text{main operation */}$$

### Total running time of loop

$$T_n = \text{ceil}(n / \text{MVL}) \times (T_{\text{loop}} + T_{\text{start}}) + n \times T_{\text{chime}}$$

### Enhancing Performance

**Chaining**

- MULV.D V1,V2,V3
- ADDV.D V4,V1,V5

Instructions must execute in 2 convoys because of dependencies.

Chaining treats vector registers as a collection of individual registers. Forward individual registers.

### Flexible Chaining

Allows a vector instruction to chain to any other active vector instruction.
Conditionally Executed Statements

```fortran
do 100  i = 1, 64
   if (A(i).ne. 0) then
      A(i) = A(i) - B(i)
   endif
100   continue
```

Cannot vectorize loop because of conditional execution

Vector-mask control

Boolean vector of length MVL to control the execution of a vector instruction

When a vector-mask register is enabled, any vector instructions executed operate only on vector elements whose corresponding entries in the vector-mask register are 1. Entries corresponding to a 0 in the vector-mask are unaffected.

The previous loop can be vectorized using a vector-mask.

```
LV    V1,Ra        ;load vector A into V1
LV    V2,Rb        ;load vector B
L.D   F0,#0        ;load FP zero into F0
SNEVS.D V1,F0      ;sets VM(i) to 1 if V1(i) != F0
SUBV.D V1,V1,V2    ;subtract under vector mask
CVM   ;set the vector mask to all 1s
SV    Ra,V1        ;store the result in A
```

Vector instructions executed with vector-masks still take execution time for elements where the vector-mask is 0

However, even with many 0s in the mask performance is often better than scalar mode.
**Sparse Matrices**

Only non-zero elements of matrix are stored.

```
1 0 0 0
0 0 0 9
0 7 0 0
0 0 3 0
```

How do we access vectors in such a structure?

**Scatter-gather operations**

Allow retrieval and storage of vectors from sparse data structures

**Gather operation**

Takes an *index vector* and a *base address*. Fetches vector whose elements are at address given by adding the base address to the offsets in the index vector

```
LVI Va, (Ra+Vk)
```

**Scatter operation**

Stores a vector in sparse form using an index vector and a base address

```
SVI (Ra+Vk), Va
```

Most vector processors provide support for computing index vectors. In VMIPS we have an instruction CVI

```
SNEVS.D V1,F0
CVI V2,#8
```

Sparse matrices cannot be automatically vectorized by simple compilers. Compiler cannot tell if elements of index vector are distinct values and that no dependencies exist. Requires programmer directives.

<table>
<thead>
<tr>
<th></th>
<th>IBM RS6000</th>
<th>Cray YMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock</td>
<td>72 MHz</td>
<td>167 MHz</td>
</tr>
<tr>
<td>Cache</td>
<td>256 KB</td>
<td>0.25 KB</td>
</tr>
<tr>
<td>Linpack</td>
<td>140 MFLOPS</td>
<td>160 (1.1)</td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td>17 MFLOPS</td>
<td>125 (7.3)</td>
</tr>
<tr>
<td>(Cholesky Blocked)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Multiple Lanes**

- **Single lane**

- Adding extra lanes increases peak performance, but does not change start-up latency. As the number of lanes increases, start-up costs become more significant.

Can improve performance using multiple lanes

Adding extra lanes increases peak performance, but does not change start-up latency. As the number of lane increases, start-up costs become more significant.

Each lane operates on its elements independently of the others, so no communication between lanes needed.
Pipelined Instruction Start-Up

Allows the overlapping of vector instructions. Reduces start-up costs.

Effectiveness of Compiler Vectorization

<table>
<thead>
<tr>
<th>Benchmark name</th>
<th>Operations executed in vector mode, compiler-optimized</th>
<th>Operations executed in vector mode, hand-optimized</th>
<th>Speedup from hand optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDNA</td>
<td>96.1%</td>
<td>97.2%</td>
<td>1.52</td>
</tr>
<tr>
<td>MG3D</td>
<td>95.1%</td>
<td>94.5%</td>
<td>1.00</td>
</tr>
<tr>
<td>FLO52</td>
<td>91.5%</td>
<td>88.7%</td>
<td>N/A</td>
</tr>
<tr>
<td>ARC3D</td>
<td>91.1%</td>
<td>92.0%</td>
<td>1.01</td>
</tr>
<tr>
<td>SPEC77</td>
<td>90.3%</td>
<td>90.4%</td>
<td>1.07</td>
</tr>
<tr>
<td>MDG</td>
<td>87.7%</td>
<td>92.4%</td>
<td>1.49</td>
</tr>
<tr>
<td>TRFD</td>
<td>69.8%</td>
<td>73.7%</td>
<td>1.67</td>
</tr>
<tr>
<td>DYFESM</td>
<td>68.8%</td>
<td>65.6%</td>
<td>N/A</td>
</tr>
<tr>
<td>ADM</td>
<td>42.9%</td>
<td>59.6%</td>
<td>3.60</td>
</tr>
<tr>
<td>OCEAN</td>
<td>42.8%</td>
<td>91.2%</td>
<td>3.92</td>
</tr>
<tr>
<td>TRACK</td>
<td>14.4%</td>
<td>54.6%</td>
<td>2.52</td>
</tr>
<tr>
<td>SPICE</td>
<td>11.5%</td>
<td>79.9%</td>
<td>4.06</td>
</tr>
<tr>
<td>QCD</td>
<td>4.2%</td>
<td>75.1%</td>
<td>2.15</td>
</tr>
</tbody>
</table>

Performance of Vector Processors

- $R_{\text{inf}}$: MFLOPS rate on an infinite-length vector
  - upper bound
  - Real problems do not have unlimited vector lengths, and the start-up penalties encountered in real problems will be larger
  - ($R_n$ is the MFLOPS rate for a vector of length $n$)
- $N_{1/2}$: The vector length needed to reach one-half of $R_{\text{inf}}$
  - a good measure of the impact of start-up
- $N_V$: The vector length needed to make vector mode faster than scalar mode
  - measures both start-up and speed of scalars relative to vectors, quality of connection of scalar unit to vector unit
Example: $R_{\text{inf}}$

DAXPY loop: $D = aX + Y$

$T_n = \text{ceil}[n/MVL] \times (T_{\text{loop}} + T_{\text{start}}) + n \times T_{\text{chime}}$

Assuming chaining, we can execute loop in 3 chimes:

1. LV V1,Rx MULVS.D V2,V1,F0
2. LV V3,Ry ADDV.D V4,V2,V3
3. SV Ry,V4

Assume $MVL = 64$, $T_{\text{loop}} = 15$, $T_{\text{start}} = 49$, $T_{\text{chime}} = 3$

and a 500MHz processor

$T_n = \text{ceil}[n/64] \times (15 + 49) + n \times 3$

$T_n \leq (n + 64) + 3n = 4n + 64$

$R_{\text{inf}} = \lim_{n \to \infty} \frac{\text{Operations per iteration} \times \text{Clock rate}}{\text{Clock cycles per iteration}}$

$= \lim_{n \to \infty} \frac{(4n + 64)/n}{4} = 4$

$R_{\text{inf}} = 2 \times 500\text{MHz} / 4 = 250 \text{ MFLOPS}$

Example: $N_{\frac{1}{2}}$

MFLOPS = FLOPS executed in $N_{\frac{1}{2}}$ iterations $\times$ Clock cycles $\times 10^{-6}$

Clock cycles to execute $N_{\frac{1}{2}}$ iterations = second

$125 = 2 \times N_{\frac{1}{2}} \times 500$

$T_{N_{\frac{1}{2}}} = \frac{2 \times N_{\frac{1}{2}}}{500} = 0.25$

Simplifying this and then assuming $N_{\frac{1}{2}} \leq 64$, so that $T_{n=64} = 1 \times 64 + 3 \times n$, yields

$T_{n=64} = 8 \times N_{\frac{1}{2}}$

$1 \times 64 + 3 \times N_{\frac{1}{2}} = 8 \times N_{\frac{1}{2}}$

$5 \times N_{\frac{1}{2}} = 64$

$N_{\frac{1}{2}} = 12.8$

$N_{\frac{1}{2}} = 13$

Example: $N_v$

Estimated time to do one iteration in scalar mode is 59 clocks.

$64 + 3N_v = 59N_v$

$N_v = \text{ceil}[64/56] = 2$
Fallacies and Pitfalls

- Pitfall: Concentrating on peak performance and ignoring start-up overheads. Can lead to large $N_v > 100$!
- Pitfall: Increasing vector performance, without comparable increases in scalar performance (Amdahl's Law)
- Pitfall: You can get vector performance without providing memory bandwidth

Where to now?