Single Thread Parallelism
Outline

- Pipelining
- SIMD (both vector and array processing)
- VLIW
- DAE
- HPS
- Data Flow

Michael J. Flynn
SICCC 1967

MMX
SSE 1.2.3
HPS As Evolution

\[ \text{SIMD} \]

\[ \text{VLFW} \]

\[ \text{DA6} \]
SIMD/MIMD

SISO  The Typical Pentium-Pro, for example
MISD
SIMD  Array Processor, Vector Processor
MIMD  Multiprocessor

and, Note:

SYSTOLIC
ARRAY

HT KUNG
SIMD

Vector Processors, Array Processors
Figure 5: Block diagram of registers.
An example: Vector processing

- The scalar code:
  \[ A(i) = \frac{(B(i)+C(i))}{2}; \]
  \[ \text{for } i=1,50 \]

- The vector code:
  
  \[ \text{lv}1 \]
  \[ \text{lv}l \ 50 \]
  \[ \text{vld V0,B} \]
  \[ \text{vld V1,C} \]
  \[ \text{vadd V2,V0,V1} \]
  \[ \text{vshf V3,V2,1} \]
  \[ \text{vst V3,A} \]

\[ \frac{A}{\beta} = \frac{A \times (1 \div \beta)}{B} \]

```
\begin{array}{cccc}
B & B+1 & B+2 & B+4 \\
\end{array}
```

```
\begin{array}{cccc}
C & C+1 & C+2 & C+3 \\
C+4 & C+5 & \cdots & \cdots \\
\end{array}
```

Row Major Order

```
\begin{array}{cccc}
B & B+1 & B+2 & \cdots \\
\end{array}
```

\[ \text{stride} = 1 \]

\[ \begin{array}{cccc}
C & C+4 & C+8 & \cdots \\
\end{array} \]

\[ \text{stride} = 4 \]
TS Store Pipe
With 2 Load
79
With Chaining
182
(No Chaining)
285

Vector Chain

\[ 40 \times 50 = 2000 \]

\[ \frac{2}{3} + \frac{5}{4} = \frac{25}{4} \]

13
57
44
2

Square
2
Vector processing example (continued)

- Scalar code (loads take 11 clock cycles): 2000
- Vector code (no vector chaining): 28
- Vector code (with chaining): 182
- Vector code (with 2 load, 1 store port to memory): 75
INSTRUCTIONS RETIRED IN ORDER MEANS WE CAN RECOVER THE STATE OF THE MACHINE JUST BEFORE THE INSTRUCTION. EXCEPTIONS ARE NOT TRUE FOR TOMASULO AND 360/91.
Early Form of Decoupled - Access/Execute

Memory Access Unit

 Execute Unit

* Andrew Plezskun, Univ. of Illinois

SMA

* James E. Smith, Univ of Wisconsin

DAE
VLIW (Josh Fisher)

* Static Scheduling
  - Everything in lock step
  - Trace Scheduling
    - Good if Highly Skewed

* Generic Model

```
<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

PC → α β γ β

```

MULTIFLOW
TRACE

BOB RAU
CYDROMS
CYDRA-5

EPIC

128
41 41 41

TEMPLATE

S
The HPS Paradigm

- Incorporated the following:
  - Aggressive branch prediction
  - Speculative execution
  - Wide issue
  - Out-of-order execution
  - In-order retirement

- First published in Micro-18 (1985)
  - Patt, Hwu, Shebanow: Introduction to HPS
  - Patt, Melvin, Hwu, Shebanow: Critical issues
HPS
(RESTRICTED DATA FLOW)

For example, the VAX instruction:

```
ADDL2 (R1)+, (R2).
```

![Diagram of VAX instruction flow]
The Firing Rule:

When all inputs have tokens

(Note: Safe vs. Queues)

* Conditional

* Relational

* Barrier Synch
HPS - WHAT IS IT?
RAW: HAZARD

\[(A + B) \times C\]

Flow Dependency

Dave Kuck

\[\begin{align*}
x &= x + 1 \\
y &= x + 1
\end{align*}\]

Dependencies (introduced by Dave Kuck)

1. Flow Dependency: \((A + B) \times C\)

2. Anti-Dependency: \(ADD R1, R2, R3, R4\)
   You have to read \(R2\) before you

3. Output Dependency: \(ADD R1, R2, R3, R4\)
   You have to read \(R2\) before you

\(\text{OVERWRITE IT}\)

\(\text{WAR: WRITE AFTER READ HAZARD}\)

\(\text{RAW: READ AFTER WRITE HAZARD}\)
Data Flow

• **Data Driven execution of inst-level Graphical code**
  – Nodes are operators
  – Arcs are I/O

• **Only REAL dependencies constrain processing**
  – Anti-dependencies don’t (write-after-read)
  – Output dependencies don’t (write-after-write)
  – NO sequential I-stream (No program counter)

• **Operations execute ASYNCHRONOUSLY**

• **Instructions do not reference memory**
  – (at least memory as we understand it)

• **Execution is triggered by presence of data**
A Unit of Computation:
The Data Flow Node

OR,

<table>
<thead>
<tr>
<th>*</th>
<th>R</th>
<th>ARG1</th>
<th>R</th>
<th>ARG2</th>
<th>Dest. Of Result</th>
</tr>
</thead>
</table>

The Operation
(In Larger Granularity Systems, "The Compound Function")

Fires When Ready
An Example Data Flow Program:

Factorial (Done, iteratively)