Single Thread Parallelism
Granularity of Concurrency

* Intra-Instruction (Pipelining)

* Parallel Instructions (SIMD, VLIW)

* Tightly-coupled MP

* Loosely-coupled MP
Pipelining

Pipelined:

```
F_1 D_1 E_1 S_1
F_2 D_2 E_2 S_2
F_3 D_3 E_3 S_3
F_4 D_4
F_5
```

Superscalar:

```
F_1 D_1 E_1 S_1
F_2 D_2 E_2 S_2
F_3 D_3 E_3 S_3
F_4 D_4 E_4 S_4
F_5 D_5 E_5 S_5
F_6 D_6 E_6 S_6
F_7 D_7 E_7 S_7
F_8 D_8 E_8 S_8
F_9
```

Superpipelined:

```
F_1 D_1 E_1 S_1
F_2 D_2 E_2 S_2
F_3 D_3 E_3 S_3
F_4 D_4 E_4 S_4
F_5 D_5 E_5 S_5
F_6 D_6 E_6 S_6
```
**SIMD/MIMD**

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

and, Note:

![Diagram](image.png)

Pipelined
SISD

SIMD
SIMD

Vector Processors, Array Processors

<table>
<thead>
<tr>
<th>LD</th>
<th>*</th>
<th>@</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD₁</td>
<td></td>
<td>@₁</td>
<td></td>
</tr>
<tr>
<td>LD₂</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD₃</td>
<td>2</td>
<td>@₂</td>
<td>ST₁</td>
</tr>
<tr>
<td>LD₄</td>
<td>3</td>
<td>@₃</td>
<td>ST₂</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>@₄</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>@₄</td>
<td>ST₃</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ST₄</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LD₄</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

time
Fig. 5. Block diagram of registers.
An example: Vector processing

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

- The vector code:
  \( lvs 1 \)
  \( lvl 50 \)
  \( vld \ V0,B \)
  \( vld \ V1,C \)
  \( vadd \ V2,V0,V1 \)
  \( vshf \ V3,V2,1 \)
  \( vst \ V3,A \)
Vector processing example (continued)

- Scalar code (loads take 11 clock cycles):
- Vector code (no vector chaining):
- Vector code (with chaining):
- Vector code (with 2 load, 1 store port to memory):
VLIW

* Static Scheduling
  - Everything in lock step
  - Trace Scheduling

* Generic Model
Early Form of Decoupled-Access/Execute

* Andrew Plezskun, Univ. of Illinois
  SMA

* James E. Smith, Univ of Wisconsin
  DAE
HPS As Evolution

\[
\begin{array}{c}
\text{PE} \\
\downarrow \\
\alpha \\
\text{PE} \quad \text{PE} \quad \text{PE} \quad \ldots \quad \text{PE} \\
\downarrow \quad \downarrow \quad \downarrow \quad \ldots \quad \downarrow \\
\alpha \quad \alpha \quad \alpha \quad \ldots \quad \alpha \\
\text{PE} \quad \text{PE} \quad \text{PE} \quad \ldots \quad \text{PE} \\
\downarrow \quad \downarrow \quad \downarrow \quad \ldots \quad \downarrow \\
\alpha \quad \beta \quad \gamma \quad \ldots \quad \beta \\
\text{PE} \quad \text{PE} \quad \text{PE} \quad \ldots \quad \text{PE} \\
\downarrow \quad \downarrow \quad \downarrow \quad \ldots \quad \downarrow \\
\alpha \quad \beta \quad \gamma \quad \ldots \quad \beta
\end{array}
\]
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:

\[ \text{ADDL2} \ (R1) + , (R2) \]
HPS - What Is It?

I-Stream

Active Window

Machine Lang. Inst

Decoder

Data Flow Graph

 Mercer

Issue

All The Nodes

Level

Fire

F.U.

Dist

Restricted Data Flow
INSTRUCTIONS RETIRED IN ORDER MEANS WE CAN RECOVER THE STATE OF THE MACHINE JUST BEFORE THE INSTRUCTION. PRECISE EXCEPTIONS

NOT TRUE FOR TOMASULO AND 360/91
RAW HAZARD

\[(A + B) \times C]\n
Flow Dependency

DAVE LCVC

\[x = x + 1\]
\[y = x + 1\]

Dependencies (Introduced by Dave Lucci)

1. Flow Dependency: \((A + B) \times C\)
   You have to add \(A, B\) before you multiply by \(C\), i.e., write \((A + B)\) before read.
   RAW: Read After Write
   HAZARD

2. Anti-Dependency: ADD R1 R2 R3
   ADD R2 R3 R4
   You have to read \(R2\) before you overwrite it.
   RAW: Write After Read
   HAZARD

3. Output Dependency: ADD R1 R2 R3
   ADD R1 R4 R1
   RAW: Write After Write
   HAZARD

Static Assignment

Single Assignment
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**
Characteristics of Data Flow

* Data Driven Execution of Instruction-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 PC)

* Operations Execute Asynchronously

* Instructions Do Not Reference Memory
  (at least, memory as we understand it)

* Execution Triggered By Presence of Data
  --Safe vs. Queues
A Unit of Computation:
The Data Flow Node

OR,

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

Fires When Ready

| * | R | ARG1 | R | ARG2 | Dest. Of Result |
The Firing Rule:

When all Inputs Have Tokens

(Note: Safe vs. Queues)

*Conditional

*Relational

*Barrier Synch
An Example Data Flow Program:

**Factorial** (Done, Iteratively)