PARALLEL COMPUTING PLATFORMS, PIPELINING, VECTOR COMPUTING

- Terminology and Flynn’s taxonomy
- The PRAM model
- Pipelining
- Vector computers
- Easy parallelism: Multiple functional units
- Superscalar computers and VLIW
Flynn’s Taxonomy of parallel computers

Distinguishes architectures by the way processors execute their instructions on the data.

Four categories:

1. Single Instruction stream Single Data stream (SISD)
2. Single Instruction stream Multiple Data stream (SIMD)
3. Multiple Instruction stream Single Data stream (MISD)
4. Multiple Instruction stream Multiple Data stream (MIMD)
**SISD Architecture**  This is the standard von Neuman organization

**MISD Architecture**  Same data simultaneously exploited by several processors or computing units that operate on it. Pipelined processors, systolic arrays, ...,

**SIMD Architecture**
Same instruction is executed simultaneously on different “data streams”. Single Control Unit (CU) dispatches a single stream of instructions.

- Includes pipelined vector computers + many classical machines (e.g., ILLIAC IV in the 60s, the CM2 in early 90s)
MIMD Architecture MIMD machines simultaneously execute different instructions on different data streams.

- Processing elements have their own control units - but coordinate computational effort with the other processors.

- MIMD computers further classified in two important subcategories
  
  - **Shared memory models**: processors have very little local or ‘private’ memory; they exchange data and co-operate by accessing a global shared memory.
  
  - **Distributed memory models**: No shared global address space. Each processor has its own local memory. Interconnections between processors allow exchange of data and control information.
Typical SIMD (left) and MIMD (right) organizations.
**The PRAM model**

- **PRAM** = Parallel Random Access Memory. Model used mostly for theory.
- Extends the Von-Neumann model.
- \( P \) processors having access to a large memory

Question: How to manage conflicts? [2 processors wanting to read & write to same location]
- Four different models used
**EREW** - Exclusive Read, Exclusive Write

**CREW** - Concurrent Read, Exclusive Write

**ERCW** - Exclusive Read, Concurrent Write

**CRCW** - Concurrent Read, Concurrent Write

Concurrent reads are harmless. Concurrent writes require arbitration. Four methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common</td>
<td>Writes successful if data is same</td>
</tr>
<tr>
<td>Arbitrary</td>
<td>One PE only succeeds in the write</td>
</tr>
<tr>
<td>Priority</td>
<td>PEs prioritized. Highest priority PE writes</td>
</tr>
<tr>
<td>Combine results</td>
<td>e.g., add results then write</td>
</tr>
</tbody>
</table>
Architecture options for exploiting parallelism

1. Multiple functional units *, +, ..
2. Pipelining, Superscalar pipelines
3. Vector processing
4. Multiple Vector pipelines
5. Multiprocessing
6. Distributed computing
**Pipelining**

Idea of the *assembly line*. Assume an operation (e.g., FP adder) takes $s$ stages to complete. Then we pass the operands through the $s$ stages instead of waiting for all stages to be complete for first two operands.

**Example:** an arithmetic adder:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unpack operands</td>
<td>4</td>
<td>Add fractions;</td>
</tr>
<tr>
<td>2</td>
<td>Compare exponents</td>
<td>5</td>
<td>Normalize fraction</td>
</tr>
<tr>
<td>3</td>
<td>Align significands</td>
<td>6</td>
<td>Pack operands</td>
</tr>
</tbody>
</table>

➢ To operate on a stream of data (a pair of vectors), no need to wait for all the stages to complete on each pair. Instead, “pipeline” the stream through the 6 stages.
Assume that each stage takes one clock period. After $s$ clocks, the pipe is filled, i.e., all stages are active. Then an operation is produced at each clock cycle.

If each stage takes time $\tau$, then, operation with $n$ numbers will take

$$s\tau + (n - 1)\tau = (n + s - 1)\tau$$ seconds

Instead of

$$ns\tau$$ seconds

Gain a factor of

$$\frac{ns}{n+s-1}$$
Vector pipelines

Simplest example: \( s1 = s2 + s3 \)

Machine operations (informal assembler notation)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Register</th>
<th>Stage</th>
</tr>
</thead>
<tbody>
<tr>
<td>load ( s2 ) --&gt; R1</td>
<td>1</td>
<td>% R1-R3 = registers</td>
</tr>
<tr>
<td>load ( s3 ) --&gt; R2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>add ( R1 + R2 ) --&gt; R3</td>
<td>6</td>
<td>% (6 stages)</td>
</tr>
<tr>
<td>store R3 --&gt; ( s1 )</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

9 cycles to complete one operation.
On a sequential computer (scalar mode) the loop

\[
\text{do } i=1,64 \\
\quad s1(i)=s2(i)+s3(i) \\
\text{enddo}
\]

would take \(64 \times 9 = 576\) cycles + loop overhead.

If operations are pipelined: one operation delivered at each cycle once pipe is filled. Total time: \(63 + 9\) cycles. (!)

Vector computers: (1) exploit ideas of pipelining at all levels (loads, store, vector adds, ...) to reduce cost and (2) provide “vector” instructions
Assume that $r$ is the ratio of the peak vector speed to peak scalar speed (the ‘speed-up’) and that the fraction of vector to scalar operations in your calculation is $f$ (if 5% of operations are purely scalar then $f = 0.95$). Show that the speed-up for your calculation will be

$$S(f) = \frac{r}{(1 - f) \ast r + f}$$

What speed up do you get when vector processing is very fast ($r = \infty$) and $f = 0.9$
Vector computers - continued

- Hardware component: vector registers
- Vector registers are used in vector instructions.

**Example:** The loop \[ a(1:64) = b(1:64) + c(1:64) \]

May be translated as:

- `vload b(1), 64, 1 --> V1 % Vector load to V1`
- `vload c(1), 64, 1 --> V2`
- `vadd V1 + V2 --> V3 % Vector addition`
- `vstore a(1), 64, 1 <-- V3`

- `b(1) = start address, "64" = vector length, "1" = stride`
Chaining

Take the result of one pipe (e.g. a multiplier) as an input to another pipe (e.g. adder).

t(1:64)=z(1:64)+x(1:64)*y(1:64)

Main assembly instructions:

\[
\begin{align*}
\text{vmul } & V1*V2 \rightarrow V3 \\
\text{vadd } & V3+V4 \rightarrow V5
\end{align*}
\]


**Advanced Vector Extensions (AVX)**

- Modern versions of vector instructions added to common chips.
- Additions to the x86 instruction set architectures for Intel and AMD processors.
- AVX augments the “Streaming SIMD extensions” (SSE) introduced in '99 with Pentium III.
- Started with the ‘Sandy-bridge’ Family from Intel in 2011 followed by AMD.
- Borrowed from vector processing by adding vector instruction sets.
AVX2

AVX introduced in 2008 (Intel) - modified from legacy SSE and SSE2 (circa 2000)

All Intel Processors in the category core i3/i5/i7 support AVX. Also AMD Zen/Zen2-based processors

Uses 16 registers (of 2 Bytes each) to perform SIMD computations on data

The 16 registers can hold: 8 Sing. Prec. (32b each)

or 4 Doub. Prec. (64b each)

Instructions can perform FP operation (+,*,/,...,) on data in registers in parallel.

Most modern processors combine some form of vector processing (multiple data) with superscalar processing (multiple instructions)

Discussed next
An $m$-way superscalar processor can issue $m$ instructions per cycle. Parallel execution of instructions is scheduled at run time (hardware). Note: $m$ is typically small (2 or 3).

“Instruction level parallelism” – different from pipelining – executes $m$ instructions in parallel.
Superscalar processors: An example

The assembly code:

1. load R1, @1000
2. load R2, @1008
3. add R1, @1004
4. add R2, @100C
5. add R1, R2
6. store R1, @2000

will be executed as:
In-order execution: Order of instructions is not changed - This leads to limited parallelism

Out-of-order execution: instructions are rearranged (i.e., executed in an order different from initial order) to increase parallelism.

**Example:** The code:

1. load R1, @1000
2. add R1, @1004
3. load R2, @1008
4. add R2, @100C
5. add R1, R2
6. store R1, @2000

Has a problem: (2) depends on (1)

It can be rearranged by swapping (2) and (3) [Gives earlier code fragment]
VLIW and the Multiflow experience

- Above idea is a reduced version of ’trace scheduling’ –
- A compiler technique based on unraveling parallelism ‘automatically’
- Main idea: used ‘Very Long Instruction Words’ [VLIW]. Idea of ’speculation’ is exploited.
- ’Multiflow, Inc’ commercialized such computers in mid-1980s.
- Had a great impact on design –

Read more about the Multiflow:
http://en.wikipedia.org/wiki/Multiflow
## History: A few milestones

<table>
<thead>
<tr>
<th>Name (year)</th>
<th>PEs, topology</th>
<th>Prog. model</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILLIAC IV ('72)</td>
<td>64, mesh</td>
<td>SIMD</td>
</tr>
<tr>
<td>DAP ('74)</td>
<td>4096, mesh</td>
<td>SIMD</td>
</tr>
<tr>
<td>MPP ('80)</td>
<td>16384, h-cube</td>
<td>SIMD</td>
</tr>
<tr>
<td>CM-1 ('83)</td>
<td>4096, h-cube</td>
<td>SIMD</td>
</tr>
<tr>
<td>CM-2 ('85)</td>
<td>16384, h-cube</td>
<td>SIMD</td>
</tr>
<tr>
<td>BBN Butterfly ('81)</td>
<td>256, Butterfly</td>
<td>MIMD</td>
</tr>
<tr>
<td>Ncube ('85)</td>
<td>1024, h-cube</td>
<td>MIMD</td>
</tr>
<tr>
<td>IPSC-1 ('85)</td>
<td>128, hypercube</td>
<td>MIMD</td>
</tr>
<tr>
<td>Meiko ('85)</td>
<td>40, reconf.</td>
<td>MIMD</td>
</tr>
<tr>
<td>Sequent Symmetry ('86)</td>
<td>30, bus</td>
<td>Shared Mem</td>
</tr>
<tr>
<td>Encore multimax ('87)</td>
<td>11, bus</td>
<td>Shared Mem</td>
</tr>
<tr>
<td>Name (year)</td>
<td>PEs, topology</td>
<td>Prog. Model</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>iPSC/2 (’87)</td>
<td>256, h-cube</td>
<td>MIMD</td>
</tr>
<tr>
<td>CRAY Y-MP (’88)</td>
<td>16, multiport</td>
<td>MIMD</td>
</tr>
<tr>
<td>Ultracomputer (’88)</td>
<td>4096, Omega</td>
<td>MIMD</td>
</tr>
<tr>
<td>Cedar (’89)</td>
<td>32, Multistage</td>
<td>MIMD</td>
</tr>
<tr>
<td>iPSC/860 (’90)</td>
<td>512 h-cube</td>
<td>MIMD</td>
</tr>
<tr>
<td>CM-5 (’91)</td>
<td>1024 Fat-Tree</td>
<td>MIMD</td>
</tr>
<tr>
<td>Paragon XP (’91)</td>
<td>4096 2-D mesh</td>
<td>MIMD</td>
</tr>
<tr>
<td>Cray T3D-T3E (’96)</td>
<td>1024, 3-D torus</td>
<td>MIMD</td>
</tr>
<tr>
<td>IBM SP series (’97 –)</td>
<td>8192, Switch</td>
<td>MIMD</td>
</tr>
<tr>
<td>SGI Origin/2000 ser (’98)</td>
<td>h-cube + crossbar</td>
<td>MIMD</td>
</tr>
<tr>
<td>SGI Origin/Altix ser (’03)</td>
<td>h-cube + crossbar</td>
<td>MIMD</td>
</tr>
<tr>
<td>IBM Blue Gene (’04)</td>
<td>100,000+, 3D mesh + tree</td>
<td>MIMD</td>
</tr>
<tr>
<td>CRAY XT3 (’04)</td>
<td>10,000+, 3D torus</td>
<td>MIMD</td>
</tr>
</tbody>
</table>

2-23 – machines
**Recent top 500s**

<table>
<thead>
<tr>
<th>Year</th>
<th>Name</th>
<th>PFlop/s</th>
<th>cores</th>
<th>interconne.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008n</td>
<td>IBM Roadrunner</td>
<td>1.105</td>
<td>129,600</td>
<td>Voltaire Infiniband</td>
</tr>
<tr>
<td></td>
<td>Cray Jaguar XT5</td>
<td>1.759</td>
<td>298,592</td>
<td>Cray Gemini</td>
</tr>
<tr>
<td>2010</td>
<td>Tianhe-IA</td>
<td>2.566</td>
<td>186,368</td>
<td>Proprietary</td>
</tr>
<tr>
<td></td>
<td>Fujitsu K computer</td>
<td>10.51</td>
<td>705,024</td>
<td>Custom</td>
</tr>
<tr>
<td>2011</td>
<td>IBM Sequoia</td>
<td>16.32</td>
<td>1,572,864</td>
<td>Custom</td>
</tr>
<tr>
<td>2012j</td>
<td>Cray Titan</td>
<td>17.59</td>
<td>560,640</td>
<td>Cray Gemini</td>
</tr>
<tr>
<td>2012n</td>
<td>NUDT Tianhe-2</td>
<td>33.86</td>
<td>4,981,760</td>
<td>TH Express-2</td>
</tr>
<tr>
<td>2013</td>
<td>Sunway TaihuLight</td>
<td>93.01</td>
<td>10,649,600</td>
<td>Sunway</td>
</tr>
<tr>
<td>2016</td>
<td>IBM Summit</td>
<td>122.3</td>
<td>2,282,544</td>
<td>Mellanox Infiniband</td>
</tr>
</tbody>
</table>

---

2-24  – machines

2-24
How about the Minnesota Supercomputer Institute?

- MSI was once in the top 10 of the top 500 (1996?)
- No longer in top 500.

Two main machines – One of which is *brand-new*:

**Mesabi Cluster** [https://www.msi.umn.edu/content/mesabi](https://www.msi.umn.edu/content/mesabi)

- HP Linux cluster with 741 nodes – Different types of nodes.
  
  - Total of 17,784 compute cores Intel Haswell E5-2680v3 processors
  - 616 Nodes have 64GB of RAM, 24 nodes have 256 GB, and 16 nodes have 1 TB of RAM each
  - 40 nodes equipped with NVIDIA K20X GPUs [peak of 2.63 TFlop/s each] 128GB RAM each
  - 32 Nodes have 480 GB SSD memory (each) for High Perf. I/O
Recently added as a replacement to Ithasca [Jan 2020]

* HPC cluster with 128 AMD ’Rome’ compute nodes (EPYC family, Zen-2 arch.)

* Total of 20,992 compute cores

* 12 of the nodes include 2-way Nvidia V100 GPUs + 2 nodes with 4 way GPUs, and one with 8-way GPU.

* Memory: 144 nodes $\times$ 256 GB; 10 nodes $\times$ 512 GB; 10 nodes $\times$ 2TB; or RAM.

* Interconnect: HDR100 (100 Gb/second) infiniband network

Note on Terminology: Infiniband is a standard [like ‘ethernet’] for HPC with high bandwidth and low latency. The term is also used to designate a type of network. Competing networks: Mellanox, Fibre Chanel. HDR stands for High-DataRate. There is also: Enhance Data Rate (EDR), FDR, all different speeds in IB technology.