SIMD Parallel Computers

- BSP: Classic SIMD number cruncher
- CM-2: Started as AI bit-twiddler -- turned toward classic SIMD number cruncher
- Terasys: bit-twiddler integrated into conventional computer system.
Burroughs BSP

- Developed during the "supercomputer wars" of the late '70s, early '80s.
- Taken to prototype stage, but never shipped
- Draws distinct division between vector and scalar processing
  - control and parallel processors have totally different memories (for both insts. and data)

Fig. 1. BSP system diagram.
Burroughs BSP

- **Control (scalar) processor**
  - processes all instructions from control memory
  - 80 ns clock => up to 1.5 MFLOPS
- **Parallel (vector) processor**
  - 16 processors
  - 160 ns clock
  - 2 cp latency for major FP operations
  - pipelined at a high level

BSP FAPAS Pipeline

- stages: Fetch, Align, Process, Align, Store (FAPAS)
Example

- **Example:** \( D = A \times B + C \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>A1</td>
<td>B1</td>
<td>C1</td>
<td>A2</td>
<td>B2</td>
<td>C2</td>
<td>A3</td>
<td>B3</td>
<td>C3</td>
<td>A4</td>
<td>B4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Align</td>
<td>A1</td>
<td>B1</td>
<td>C1</td>
<td>A2</td>
<td>B2</td>
<td>C2</td>
<td>A3</td>
<td>B3</td>
<td>C3</td>
<td>A4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>*</td>
<td>*</td>
<td>+</td>
<td>*</td>
<td>*</td>
<td>+</td>
<td>*</td>
<td>*</td>
<td>+</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Align</td>
<td>D1</td>
<td>D2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Store</td>
<td>D1</td>
<td>D2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Note that memory bandwidth and fp bandwidth are both fully committed for triad.**

---

**FAPAS pipeline contd.**

- **Throughput:** 1 FP op/320ns * 16 ops in ||
  
  => 50 MFLOPS peak
  
  => big difference between performance in scalar and vector modes
  
  also, many scalar arith operations were done in the vector unit
  because of partitioning of memory
Architecture

- Memory to Memory
- Unlimited vector lengths
  - (stripmining is automatic in hardware)
- Up to 5 input operands per instruction (pentads)
  - takes into account all evaluation trees
  - refer to table 1 for list of forms
  - Any fixed stride
- Supports two levels of loop nesting

Example:

\[
\begin{align*}
\text{VFORM} & \quad \text{TRIAD, op1, op2} \\
\text{OBV} & \quad \text{(operand bit vector)} \\
\text{RBV} & \quad \text{(result bit vector)} \\
\text{VOPERAND} & \quad A \\
\text{VOPERAND} & \quad B \\
\text{VOPERAND} & \quad C \\
\text{VRESULT} & \quad Z \\
\Rightarrow & \quad \text{RBV, Z} = A \text{ op1 B op2 C, OBV}
\end{align*}
\]
FORTRAN Example

- Livermore Fortran Kernel 1, Hydro Excerpt
  - inner loop: \[ x(k) = u(k) \times (r \times z(k+10) + t \times z(k+11)) \]
  - VLEN = "one level of nesting, length 100"
  - VFORM PENTAD2, *, +, *, *, no bit vectors
  - no OSV
  - no RBV
  - VOPERAND \( r \) (broadcast)
  - VOPERAND \( z+10 \) (stride 1)
  - VOPERAND \( t \) (broadcast)
  - VOPERAND \( z+11 \) (stride 1)
  - VOPERAND \( u \) (stride 1)
  - VRESULT \( x \) (stride 1)

Architecture, contd.

- Strided loads/stores
  - Implemented with prime memory system to avoid conflicts
- Sparse matrices
  - compress
  - expand
  - random fetch
  - random store
- IF statements
  - bit vectors embedded in vector insts.
- Recurrences/reductions
  - special instructions
- Chaining
  - built into instructions via multi-operands
  - saves loads and stores in a mem-to-mem arch.
  - 10 temp registers per arithmetic unit
Template Processing

120 bits
16 entries
Vector Data Buffer
Vector input and Validation Unit
16 entries
Template Descriptor Memory

Template Control Unit

Connection Machines: Data Parallel Algorithms

- Fine-grain parallelism
- Looks like intelligent RAM to host (front-end)
- Front-end dispatches "macro" instructions to sequencer
- Macro instructions decoded by sequencer and broadcast to bit-serial parallel processors
CM Basics, contd.

- All instructions are executed by all processors
- Subject to context flag
- Context flags
  - Processor is selected if context flag = 1
  - saving and restoring of context is unconditional
  - AND, OR, NOT operations can be done on context flag
- Operations
  - Can do logical, integer, floating point as a series of bit serial operations

CM Basics, contd.

- Front-end can broadcast data
  - (e.g. immediate values)
- SEND instruction does communication
  - within each processor, pointers can be computed, stored and re-used
- Virtual processor abstraction
  - time multiplexing of processors
Example: Count

- Count (to count active processors)
  - Each processor uncond. tests context flag
  - compute integer 1 if flag set; else 0
  - perform unconditional summation of integers

for \( j := 1 \) to \( \log_2 n \) do
  for all \( k \) in parallel do
    if \( (k + 1) \mod 2^j = 0 \) then
      \( x[k] := x[k - 2^j] + x[k] \)
    fi
  od
od

Example: Enumerate

- Enumerate
  - (give each active processor a sequence number)
  - Similar to above, except use sum-prefix computation

for \( j := 1 \) to \( \log_2 n \) do
  for all \( i \) in parallel do
    if \( k \geq 2^j \) then
      \( x[k] := x[k - 2^j] + x[k] \)
    fi
  od
od
Example: Radix Sort

for \( j := 1 \) to \( 1 + \lfloor \log_2 \text{maxInt} \rfloor \) do
  for all \( k \) in parallel do
    if \( \lfloor x[k] \rfloor \mod 2^j < 2^{j-1} \) then
      comment The bit with weight \( 2^{j-1} \) is zero.
      \( y[k] := \text{enumerate} \)
      \( c := \text{count} \)
    fi
    if \( \lfloor x[k] \rfloor \mod 2^j \geq 2^{j-1} \) then
      comment The bit with weight \( 2^{j-1} \) is one.
      \( y[k] := \text{enumerate} + c \)
    fi
    \( x[y[k]] := x[k] \)
  od
od

Example: Find End of Linked List

for all \( k \) in parallel do
  \( \text{chain}[k] := \text{next}[k] \)
  while \( \text{chain}[k] \neq \text{null} \) and \( \text{chain}[	ext{chain}[k]] \neq \text{null} \) do
    \( \text{chain}[k] := \text{chain}[	ext{chain}[k]] \)
  od
od
Connection Machine Architecture

- **Nexus: 4x4, 32-bits wide**
  - Cross-bar interconnect for host communications
- **16K processors per sequencer**
- **Memory**
  - 4K mem per processor (CM-1)
  - 64K mem per processor (CM-2)
- **CM-1 Processor**
- **16 processors on processor chip**
Connection Machine Arch., contd.

- **ALU**: 3-inputs (2 memory + 1 flag),
  - 2 outputs (1 to memory, 1 to flag)
  - operates on "nano-instructions"
  - LOAD A
  - LOAD B
  - STORE result

- **4K RAM**
- **8 flag registers** (conditions, carries, etc.)
- **router interface**
- **2-D mesh interface**

**Performance:**
- 32-bit add => 24 microseconds
- times 64K procs. => 2 Billion/sec
Communications

- Broadcast: sequencer -> processors
- Global OR for terminal conditions and exceptions
- Hypercube communications
  - 12-cube; 16 procs at each vertex
- NEWS net
  - about 6 times faster than hypercube;
  - for neighbor communication

Instruction Processing

- HLLs: C* and FORTRAN 8X
- Paris virtual machine instruction set
- Virtual processors
  - Allows some hardware independence
  - Time-share real processors
  - V virtual processors per real processor
    => 1/V as much memory per virtual processor
- Nexus contains sequencer
  - AMD 2900 bit-sliced micro-sequencer
  - 16K of 96-bit horizontal microcode
- Inst. processing:
  - 32 bit virtual machine insts (host)
    -> 96-bit microcode (nexus sequencer)
    -> nanocode (to processors)
CM-2

- re-designed sequencer; 4x microcode memory
- New processor chip
- FP accelerator (1 per 32 processors)
- 16X memory capacity (4K-> 64K)
- SEC/DED on RAM
- I/O subsystem
- Data vault
- Graphics system
- Improved router

Performance

- Computation
  - 4000 MIPS 32-bit integer
  - 20 GFLOPS 32-bit FP
  - 4K x 4K matrix mult: 5 GFLOPS
- Communication
  - 2-d grid: 3 microseconds per bit
    » 96 microseconds per 32 bits
    » 20 billion bits /sec
  - general router: 600 microseconds/32 bits
    » 3 billion bits /sec
- Compare with CRAY Y-YP (8 procs.)
  - 2.4 GFLOPS
    » But could come much closer to peak than CM-2
  - 246 Billion bits/sec to/from shared memory