Instruction-Level Parallelism
VLIW and CRAY-1

Jeffrey Bigham
Xu Miao
What is a VLIW Architecture?

- Single Long Instruction Per Cycle
- Each Instruction Contains Many Independent Operations
- Statically Predictable Number of Cycles Required for Each Operation
Trace Scheduling

- Limited to code with no back edges (i.e. all loops are unrolled) (a).
- Uses jump prediction to select execution streams (b), the trace.
- Scheduler treats trace as a single basic block (c). Unfortunately, relationships outside the trace are not maintained.
- Postprocessor repairs these connections by linking outside code to stream exits and entrances (d).
Problems/Solutions

- Memory Anti-Aliasing
  - Solution: Track most of them down

- Multiple Jumps
  - Clever method of reducing $2^n$ possible jumps to just $n+1$

- Multiple Memory References
  - Arrange so the common case is for memory references in single instruction refer to different, separately addressable memory banks
Critique

● No Real Results To Report, Only Speculation

● Without Building System Overlooked Important Considerations (exceptions, measurable performance on real code…)

● Author Invented Trace Scheduling and This Seems A Way for Him To Promote It. Why not publish it as a compiler technique?
Questions

- Why would a VLIW perform badly on dynamic code? What other types of code does trace scheduling not work on?
- In the second paper it is mentioned that they didn’t consider object code compatibility. How could this be remedied then and today?
- What goes wrong when handling exceptions and why does it happen?
- What are current applications of VLIW (IA-64 and x86-64)?
- What are the drawbacks of VLIW in terms of size of code and memory usage?
- Is Trace Scheduling or a derivative still used as a compiling technique?
- “…Bulldog ...(and prevent people from thinking it was written at Harvard).” What?
CRAY-1 Computer System

- The features:
  - The second generation of Vector machine
  - The fast memory hierarchy
  - Pipelining
  - Compact interconnection
  - Optimizing compiler for FORTRAN

- Applications:
  - Scientific computation
    - Weather forecasting
Smaller is faster

- Since the signal transmission distance is smaller, time needed is less. Actually fastest scalar machine at that time.
- C-shape design is to minimize the interconnection distance among the chips.
- In addition, an excellent cooling system makes it cool and fast.
Architecture of Cray-1

- 5 types of operating registers
- 12 functional units
- 64-bit word instruction buffer
- Other supporting registers
Fig. 5. Block diagram of registers.

VECTOR REGISTERS

MEMORY

SCALAR REGISTERS

ADDRESS REGISTERS

FUNCTIONAL UNITS

INSTRUCTION BUFFERS
Vector machine

- A vector is a bunch of identical operations done with independent objects. For example,
- Inner loop parallelism
- Short vector machine
  - In STAR100, ASC processors, >100 elements required to compete scalar processor
  - In Cray-1, >2~4 elements, actual using 64 elements
- Chaining

<table>
<thead>
<tr>
<th># C code</th>
<th># Scalar Code</th>
<th># Vector Code</th>
</tr>
</thead>
</table>
| for (i=0; i<64; i++)  
  C[i] = A[i] + B[i]; | li r4, #64                 | li vlr, #64       |
| loop:             | 1d f1, 0(r1)               | lv v1, r1, #1     |
|                   | 1d f2, 0(r2)               | lv v2, r2, #1     |
|                   | fadd f3, f1, f2            | faddv v3, v1, v2  |
|                   | st f3, 0(r3)               | sv v3, r3, #1     |
|                   | add r1, r1, #1             |                  |
|                   | add r2, r2, #1             |                  |
|                   | add r3, r3, #1             |                  |
|                   | sub r4, #1                 |                  |
|                   | bnez r4, loop              |                  |
More chaining

\[ \text{lv } v1 \]
\[ \text{vmul } v3, v1, v2 \]
\[ \text{vadd } v5, v3, v4 \]
Optimizing Compiler

- Maximize the usage of registers instead of saving the registers
- Minimize the data dependence
- Utilize vector mask instruction and vector merge instruction to implement vectorized GOTO, IF?
Pros & Cons

- **Pros:**
  - Compact and powerful
  - Good for scientific computation
- **Cons:**
  - At most 10 times faster, as claimed by VLIW people?
  - Not easy to build the actual system
  - No accurate division unit (reciprocal approximation unit is not accurate enough)
Questions

- Can the vectorization be done to non-loop part of the code?
- How to make the compiler perfect for vectorizing the code?
- How does CRAY-1 scale to long vectors?
- How can we deal with the physical problems mentioned in the paper?
- Can we use more complicated gates at all?
- Would it be in CISC?
- Interleaving is an old solution to slow/interfering HW. Any use today?
- How about the nowaday vector machine?
- What is an ERDA class? Does the designation still exist?
- What were the current time benefits of using emitter-coupled logic circuit technology?
- Why cylindrical instead of ball shape to lessen wiring distances?
- With the temperatures of CRAY going up to 130F, was there a burn concern for the individuals who would choose to access the interior of the machine?