The Cray I
Time line

• 1969 -- CDC Introduces 7600, designed by cray.

• 1972 -- Design of the 8600 stalls due to complexity. CDC can’t afford the redesign Cray wants. He leaves to start Cray Research

• 1975 -- CRI announces the Cray 1

• 1976 -- First Cray-1 ships
Vital Statistics

• 80Mhz clock
• A very compact machine -- fast!
• 5 tonnes
• 115kW -- freon cooled
• Just four kinds of chips
  • 5/4 NAND, Registers, memory, and ???
Vital Statistics

• 12 Functional units
• >4KB of registers.
• 8MB of main memory
  • In 16 banks
  • With ECC
• Instruction fetch -- 16 insts/cycle
Key Feature: Registers

- Lots of registers
  - T -- 64 x 64-bit scalar registers
  - B -- 64 x 24-bit address registers
  - B+T are essentially SW-managed L0 cache
  - V -- 8 x 64 x 64-bit vector registers
Fig. 5. Block diagram of registers.
Key Feature: Vector ops

- This is a scientific machine
- Lots of vector arithmetic
- Support it in hardware
Cray Vectors

- Dense instruction encoding -- 1 inst -> 64 operations
- Amortized instruction decode
- Access to lots of fast storage -- V registers are 4KB
- Fast initiation
  - vectors of length 3 break even. length 5 wins.
- No parallelism within one vector op!
Vector Parallelism: Chaining

**Source code**

```plaintext
for i in 1..64
    a[i] = b[i] + c[i] * d[i]
```

**Naive hardware**

```plaintext
for i in 1..64
    t[i] = c[i] * d[i]
for i in 1..64
    a[i] = t[i] + b[i]
```

**Cray hardware**

```
for i in 1..64
    t = c[i] * d[i]
for i in 1..64
    a[i] = t + b[i]
```

‘t’ is a wire

In lock step
Vector Tricks

ABS(A)
VI = A
V2 = 0-VI
VM = VI < 0
V3 = VI merge V2

Sort pair in A and B
VI = A
V2 = B
V3 = A-B
VM = V3 < 0
V2 = V2 merge VI
VM = V3 > 0
VI = VI merge V2

No branches!
Vector Parallelism: OOO execution

• Just like other instructions, vector ops can execute out-of-order/in parallel
• The scheduling algorithm is not clear
  • I can’t find it described anywhere
• Probably similar to 6600
Tarantula: A recent vector machine

- Vector extensions to the 21364 (never built)
- Basic argument: Too much control logic per FU (partially due to wire length)
- Vectors require less control.
Tarantula Architecture

- 32 Vector registers
- 128, 64-bit values each
- Tight integration with the OOO-core.
- Vector unit organized as 16 "lanes"
  - To FUs per lane
  - 32 parallel operations
  - 2-issue vector scheduler
Amdahl’s Rule

• 1 byte of IO per flops
• Where do you get the BW and capacity needed for vector ops?
• The L2!
Vector memory accesses.

- Only worry about unit-stride -- EZ and covers about 80% of cases.
- However... Large non-unit strides account for about 10% of accesses
  - Bad for cache lines
- 2-stride is about 4%
Vector Caching Options

• L1 or L2
  • L1 is too small and too tightly engineered
  • L2 is big and highly banked already
• Non-unit strides don’t play well with cache lines
  • Option 1: Just worry about unit-stride
  • Option 2: Use single-word cache lines (Cray SV1)
Other problems

• Vector/Scalar consistency
  • The vector processor accesses the L2 directly -- Extra bits in the L2 cache lines
  • Scalar stores may be to data that is then read by vector loads -- Special instruction to flush store queue
Tarantula Impact

- 14% more area
- 11% more power
- 4x peak Gflops (20 vs 80)
- 3.4x Gflops/W