Today's Plan

Division of Labor

- RISC vs. CISC
- Exposing parallelism

38

Two Key Performance Bottlenecks

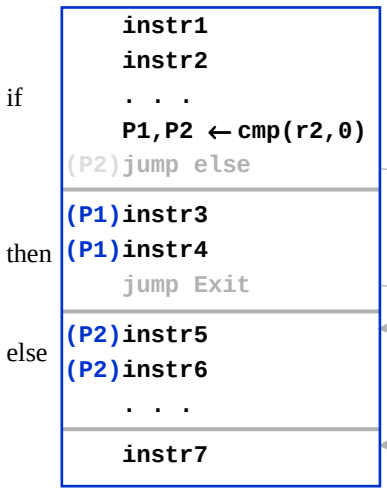
Branches

- Modern microprocessors perform good branch prediction
- But when they mispredict, the penalty is high and getting higher
  - Penalties increase as we increase pipeline depths
- Estimates: 20-30% of performance goes to branch mispredictions [Intel98]
- Branches also lead to small basic blocks, which restrict latency hiding opportunities

Memory latency

- CPU speed doubles every 18 months (60% annual increase)
- Memory speed increase about 5% per year

Predicated Execution



This is called *if-conversion*

Idea

- Add a predicate flag to each instruction
  - If predicate is true, the instruction is executed
  - If predicate is false, the instruction is not executed
  - Predicates are simply bits in a register
- Converts control flow into data flow
  - Exposes parallelism
  - With predicate flags, instr3 – instr7 can all be fetched in parallel

Benefits?

- Fewer branches (fewer mispredictions)
- Larger basic blocks
- More parallelism

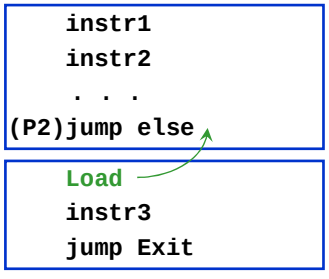
The Memory Latency Problem

Memory Latency

- Writes can be done out of order and can be buffered
- Loads are the problem: processor must wait for loads to complete before using the loaded value
- Standard latency-hiding trick: issue non-blocking load as early as possible to hide latency

The Problem

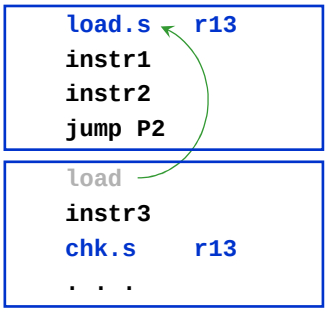
- Loads typically issued at beginning of basic block
- Can't move the Load outside the basic block
  - If the Load were to cause an exception when the basic block is not executed, then the early Load causes an erroneous exception



(Control) Speculative Loads

Split-phase operation

- Issue the load (load.s) as early as you wish
- Detect any exception and record it somewhere with the target of the load
- Can later check to see whether the load completed successfully: chk.s



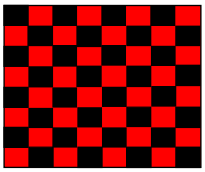
Benefits?

- More freedom to move code– can now move Loads above branches as long as the check is in the original basic block
- Complication: What happens if chk.s is issued without a corresponding load.s?
  - This is clearly an error, so we need to be careful about where we move the load.s

N-Queens Example

The Problem

- Place N Queens on a chessboard so they don't attack each other



The Solution

- March through columns with a recursive procedure
  - B array: check the row
  - A and C arrays: check the two diagonals
  - Code to test if the $(i,j)^{th}$ square is legal:

```
if ((b[i]==0) && (a[i+j-1]==0) && c[i+j+N]==0)
```

N-Queens Solution

```
if ((b[i]==0) && (a[i+j-1]==0) && c[i+j+N]==0)
```

```
1   r1 = &b[i]
    r3 = &a[i+j-1]
    r5 = &c[i+j+N]
2   load r2 = [r1]
4   P1,P2 ← cmp(r2,0)
5   <P2> jump Else

6   load r4 = [r3]
8   P3,P4 ← cmp(r4,0)
9   <P4> jump Else

10  load r6 = [r5]
12  P5,P6 ← cmp(r6,0)
13  <P5> jump Then
    Else . . .
```

Load the addresses into registers
(Assume we can issue 3 instructions per cycle)

De-reference the arrays
Test for legality

Parallelism boundary
Basic block boundary

Summary

3 basic blocks
12 instructions
13 cycles (assuming 1 cycle per instruction except 2 cycles per load)
3 conditional branches

Almost no parallelism!

N-Queens Solution: Adding Speculation

Original Code

```
1   r1 = &b[i]
    r3 = &a[i+j-1]
    r5 = &c[i+j+N]
2   load r2 = [r1]
4   P1,P2 ← cmp(r2,0)
5   <P2> jump Else

6   load r4 = [r3]
8   P3,P4 ← cmp(r4,0)
9   <P4> jump Else

10  load r6 = [r5]
12  P5,P6 ← cmp(r6,0)
13  <P5> jump Then
```

13 cycles, 3 branches

With Speculation

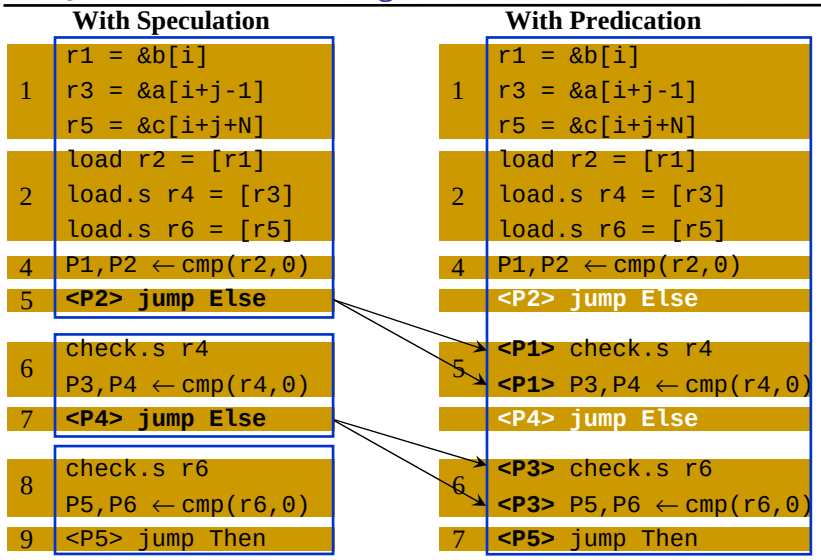
```
1   r1 = &b[i]
    r3 = &a[i+j-1]
    r5 = &c[i+j+N]
2   load r2 = [r1]
    load.s r4 = [r3]
    load.s r6 = [r5]
4   P1,P2 ← cmp(r2,0)
5   <P2> jump Else

6   check.s r4
    P3,P4 ← cmp(r4,0)
7   <P4> jump Else

8   check.s r6
    P5,P6 ← cmp(r6,0)
9   <P5> jump Then
```

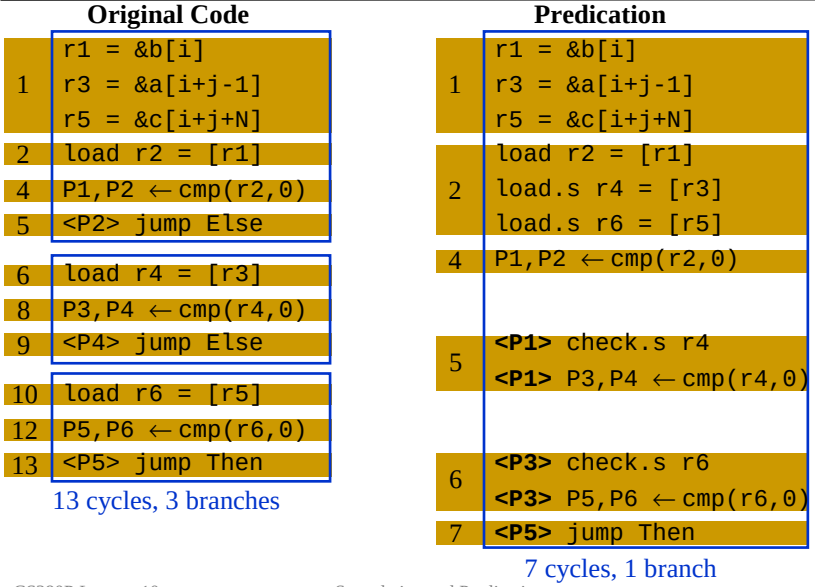
9 cycles, 3 branches

N-Queens Solution: Adding Predication



9 cycles, 3 branches

N-Queens Solution: Summary



13 cycles, 3 branches

7 cycles, 1 branch

Predication is an Old Idea

High performance computing

- SIMD machines (Single Instruction Multiple Data)
  - All processors operate in lock-step but operate on different data
  - What do you do with control flow?

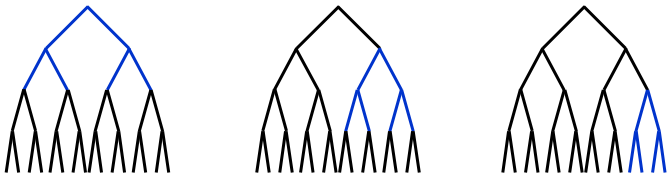
```
if (A[i][j] < 0)
    A[i][j] = -A[i][j]
```

  - Compute a mask of 0's and 1's
  - Execute both halves of the control flow using the appropriate mask
  - Can do this in either hardware or software

```
Mask[i][j] = (A[i][j] < 0)
A[i][j] -=  Mask[i][j] * 2 * A[i][j]
```


Is Predication a Good Idea?

Where should we perform predication?



Runtime information helps

- Branch behavior
- Load latencies

} Opportunities for profiling

Degree of predication depends on issue width

- The ISA can be implementation-independent
- But the compilers that emit code cannot be implementation-independent

Is Speculation a Good Idea?

What are the disadvantages of speculation?

- Wasted work

The real question: Who should perform speculation?

- The hardware can exploit runtime information
- The compiler can exploit a much larger scope

Speculation

- Another example of a split-phase operation

Implications

IA64

- The ideas are not new
- The willingness to change the ISA is new and significant

Implications for compilers

- Increased role of the compiler
- More control over sequencing, prefetching, stores, branch prediction
  - Hardware doesn't "undo" the compiler's work

Future systems

- What is the right division of labor between the compiler and the hardware?
- How else can compilers be used to simplify the hardware and make the hardware more effective?
- Can we improve the communication between the compiler and hardware?

Epilogue

Intel announces 64-bit IA-32

- The end of IA-64

What went wrong with IA-64?

What does the future hold for Dynamic Superscalar? VLIW?