University of BRISTOL

Speculative Execution

Joseph Hallett

Coming up…

Next lecture will talk about the Spectre and Meltdown vulnerabilities

- But to understand them you need a little bit of context and a little bit of Computer Architecture knowledge

This isn't going to be *super detailed*

- For more information take the *Advanced Computer Architecture* unit ;-)

How do CPUs work?

What happens when you execute a program on a CPU?

- It *fetches* the instruction from memory
- It *decodes* the instruction into its own internal microcode
- It *executes* that microcode
- And it *writes back* the result of that execution to the surrounding hardware (e.g registers, memory)

All this takes time…

- Instructions x (4 x Cycles)

bristol.ac.uk

How do CPUs work quickly?

So lets parallelize (*pipeline*) the whole process!

- As one instruction has finished fetching...
- Start fetching the next whilst the first decodes…
- And the next as the first executes and the second decodes...

So long as no instruction depends on the results of a prior one this will work fine

- See *out of order* execution for this getting even more tricksy!

In the *best* case

- Instructions x Cycles + 3

What about branches?

Which instruction do you fetch after a conditional jump?

- In other words, do you know whether the program will take the branch or not?

Could block until you know for sure…

- But this would stall the pipeline and make things slow…

…or you could guess

Branch prediction

When fetching instructions after a branch make a guess as to what the next instruction will be

If you get it right:

- then do nothing...

If you get it wrong:

- then *so long as you catch it before writeback* flush the pipeline and run the correct instruction sequence

Branch prediction algorithms

So how do you guess whether a branch will be taken?

Simplest approach:

- Assume always no (80486)
- Backwards taken, forwards not taken (Pentium 4)
- Or use programmer hints

  `__builtin_expect` (does nothing on modern x86 chips)

(godbolt.org x86-64 GCC Trunk, -O2 -march=native)

Branch prediction algorithms

```
1   sum:
2           xor     r8d, r8d
3           test    edi, edi
4           jle     .L1
5           xor     eax, eax
6   .L3:
7           add     r8d, eax
8           inc     eax
9           cmp     edi, eax
10          jne     .L3
11  .L1:
12          mov     eax, r8d
13          ret
```

```
1   int sum(register int upto) {
2       register int i;
3       register int sum = 0;
4
5       for (i = 0; i < upto; i++)
6           sum += i;
7
8       return sum;
9   }
10
```

Branch prediction algorithms

Modern branch prediction is much more complex!

- X86-64 chips have dedicated hardware for this
- Basic idea is that it keeps a record of what the branch has done before and predicts on the basis of that…

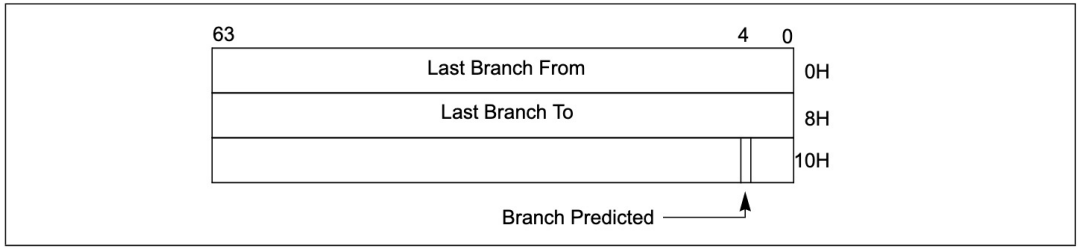

Figure 17-9. 64-bit Branch Trace Record Format

(In reality, the behaviour varies chip by chip and is fairly obscure)

So what about the caches?

What happens if executing an instruction speculatively executed causes data to be loaded into the cache?

Should you flush the caches on a branch misprediction?

Next time…

- Spectre and Meltdown
  (or why computers got 5–10% slower depending on workload)