Kontrol afhængigheder i en Out-of-order pipeline

Modellering

Vi modellerer effekten af hop, kald og retur ved at forsinke `F`. For kald og retur skelner vi mellem korrekte og fejlagtige forudsigelser. For betingede hop skelner vi tillige mellem om hoppet tages eller ej.

Vi udtrykker effekten ved tildelinger til en ny tidsvariabel: NextF.time Og for enhver instruktion gælder altid at F.time >= NextF.time

Vi tilføjer en ny fase, `B`, specifik for betingede hop, kald og retur. `B` svarer til `X` for de aritmetiske instruktioner. `B` angiver det tidspunkt, hvor vi *afgør* om forudsigelser af instruktionen var korrekt eller ej. Vi tillader at `B` indtræffer out-of-order i forhold til andre typer instruktioner, men kræver det sker in-order i forhold til andre hop, kald eller retur.

```
call a,b:   F----QBC
ret  a:     F----QBC
cbcc a,b,x: F----QBC
inorder(B)
```

Her er nogle mulige regler for en out-of-order maskine som beskrevet ovenfor

```
Instruktion  Taget  Forudsagt    Effekt
Call         ja     ja           produce(F+2,pc)
Ret          ja     ja           produce(F+2,pc)
             ja     nej          produce(B+1,pc)
CBcc         nej    ja           - (ikke muligt)
             nej    nej          produce(B+1,pc)
             ja     ja           produce(F+2,pc)
             ja     nej          produce(F+2,pc)
```

Husk at `pc` er vores specielle register til at pege på næste instruktion, så ovenstående peger på hvor hurtigt næste instruktion kan være klar.

Herunder ses to gennemløb af en indre løkke, hvor hop forudsiges korrekt.

```
                     012345678901234567
loop: movq (r10),r11   F----QAM--C           -- produce(M+2,r11)
      addq $100,r11    F----Q----XC          -- depend(X,r11),
produce(X,r11)
      movq r11,(r10)    F----QAM--VC         -- depend(A,r10),
depend(V,r11), produce(V,r11)
      addq $8,r10       F----QX----C         -- produce(X,r10)
      cbl  r10,r12,loop  F----QB----C           -- produce(F+2, pc)
(forudsagt korrekt taget)
loop: movq (r10),r11       F----QAM--C          -- produce(M+2,r11)
```

```
      addq $100,r11        F----Q----XC          -- depend(X,r11),
produce(X,r11)
      movq r11,(r10)        F----QAM--VC         -- depend(A,r10),
depend(V,r11), produce(V,r11)
      addq $8,r10           F----QX----C         -- produce(X,r10)
      cbl  r10,r12,loop      F----QB----C
```

Ydeevne: 5/4 IPC. Læg mærke til at hoppet i denne eksekvering bliver taget korrekt og næste instruktion bliver derfor kun forsinket 2 clock perioder.

På grund af omkostningen ved hop vil en oversætter ofte rulle en løkke-krop ud en eller flere gange. Herunder ses effekten af en enkelt udrulning

```
                     0123456789012345678
loop: movq (r10),r11   F----QAM--C            -- produce(M+2,r11)
      addq $100,r11    F----Q----XC           -- depend(X,r11),
produce(X,r11)
      movq r11,(r10)    F----QAM--VC          -- depend(V,r11),
produce(V,r11)
      addq $8,r10       F----QX----C          -- produce(X,r10)
      cbgt  r10,r12,exit F----QB----C         -- forudsagt korrekt ikke taget
      movq (r10),r11     F----QAM---C         -- depend(A,r10),
produce(M+2,r11)
      addq $100,r11       F----Q----XC        -- depend(X,r11),
produce(X,r11)
      movq r11,(r10)      F----QAM--VC        -- depend(A,r10),
depend(V,r11), produce(V,r11)
      addq $8,r10          F----QX----C       -- produce(X,r10)
      cbl  r10,r12,loop    F----QB----C       -- produce(F+2, pc) (forudsagt
korrekt taget)
loop: movq (r10),r11         F----QAM--C      -- produce(M+2,r11)
```

Ydeevne: 10/6 IPC

En anden teknik til at skjule omkostningen ved tagne hop er at man dimensionerer forenden af mikroarkitektur (`F` til `Q`) lidt større end resten. Her er for eksempel et afviklingsplot for den ikke udrullede løkke på en maskine der kan håndtere 3 instruktioner samtidigt i `F` til `Q` :

```
                     0123456789012345678
loop: movq (r10),r11   F----QAM--C            -- produce(M+2,r11)
      addq $100,r11    F----Q----XC           -- depend(X,r11),
produce(X,r11)
      movq r11,(r10)   F----Q-AM--VC          -- depend(V,r11),
produce(V,r11)
      addq $8,r10       F----QX----C          -- produce(X,r10)
      cbl  r10,r12,loop F----Q-B----C         -- produce(F+2, pc) (forudsagt
korrekt taget)
loop: movq (r10),r11      F----QAM--C           -- depends(A, r10),
produce(M+2,r11)
      addq $100,r11       F----Q----XC          -- depend(X,r11),
produce(X,r11)
      movq r11,(r10)      F----Q-AM--VC         -- depend(A,r10),
depend(V,r11), produce(V,r11)
```

```
      addq $8,r10          F----QX----C          -- produce(X,r10)
      cbl  r10,r12,loop    F----Q-B----C
```

Ydeevne: 5/3 IPC

Her ses effekten af en forkert forudsigelse:

```
                     0123456789012345678
loop: movq (r10),r11   F----QAM--C            -- produce(M+2,r11)
      addq $100,r11    F----Q----XC           -- depend(X,r11),
produce(X,r11)
      movq r11,(r10)    F----QAM--VC          -- depend(V,r11),
produce(V,r11)
      addq $8,r10       F----QX----C          -- produce(X,r10)
      cbl  r10,r12,loop  F----QB----C         -- produce(B+1, pc)
loop: movq (r10),r11            F----QAM--C           -- depends(A, r10),
produce(M+2,r11)
      addq $100,r11             F----Q----XC          -- depend(X,r11),
produce(X,r11)
      movq r11,(r10)             F----QAM--VC         -- depend(A,r10),
depend(V,r11), produce(V,r11)
      addq $8,r10                F----QX----C         -- produce(X,r10)
```

Ydeevne 5/9 IPC

Jo længere pipeline, jo større omkostning ved forkerte forudsigelser.

spekulativ udførelse

Det kan ske at `B` fasen indtræffer meget sent i forhold til udførelsen af andre instruktioner. Betragt for eksempel nedenstående stump kode

```
    long v = tab[j];
    if (v > key) {
      *ptr = *ptr + 1; // found it!
    }
```

Oversat til x86prime kan det blive til følgende programstump:

```
        movq (r10,r11,8),r12
        cble r12,r13,.endif
        movq (r15),r14
        addq $1,r14
        movq r14,(r15)
.endif:
```

og lad os antage at variablen `tab` befinder sig i L2-cache, mens området udpeget af variablen `ptr` er i L1-cache. Lad os antage at L2-cache tilgang koster 10 cykler (oveni L1-tilgang).

```
                        01234567890123456789012
movq (r10,r11,8),r12    F----QAM------------C
cble r12,r13,else       F----Q--------------BC
movq (r15),r14           F----QAM------------C
addq $1,r14              F----Q----X----------C
movq r14,(r15)            F----QAM--V---------C
```

Det betingede hop afhænger af en instruktion der er nød til at hente en værdi i L2 og bliver således forsinket så det først kan afgøres i cyklus 20.

Hoppet er forudsagt "ikke taget" og før det afgøres kan de næste tre instruktioner læse fra L1-cachen, beregne en ny værdi og lægge den i kø til skrivning til L1.

Det går selvsagt ikke an faktisk at opdatere L1, før vi ved om hoppet er forudsagt korrekt, men alle øvrige aktiviteter kan gennemføres. De instruktioner som udføres tidligere end et eller flere hop, kald eller retur, som de egentlig afhænger af, siges at være spekulativt udført. Spekulativ udførelse fjerner en væsentlig begrænsning på hvor meget arbejde der kan udføres parallelt.