Precizni prekidi kod Tomasulo algoritma

Kod bazičnog Tomasulo algoritma nije razmatrano šta se događa ako dođe do exception-a (izuzetaka) ili greške u predikciji grananja. U daljem tekstu će se spominjati samo izuzetak, jer je način reagovanja sličan i za greške u predikciji kod instrukcija grananja i za izuzetak kod ostalih instrukcija, u pogledu oporavka mašine. Ubacivanje instrukcija bazičnog Tomasulo algoritma je bilo in order, a završavanje izvršavanja i upis rezultata koji je ujedno bio završetak instrukcije – completion, out of order. Ako se dogodi exception, verovatnije je da će operacija koja ga je generisala još biti spekulativna po kontroli i/ili spekulativna zato što se mogao dogoditi exception na bilo kojoj prethodnoj instrukciji koja nije doživela commit po programskom redosledu. Da bi se pravilno obradili izuzeci, neophodno je izuzetak koji nastaje tokom izvršavanja operacije odložiti do trenutka commit-a operacije, jer je jedino tada sigurno da taj izuzetak zaista treba i prihvatiti. Tada sve instrukcije koje su prethodile na predviđenom dinamičkom tragu (programskom redosledu) treba da imaju izračunate rezultate, a efekte svih operacija iza operacije koja je generisala izuzetak, izvršenih ili u raznim fazama izvršavanja, treba poništiti. To znači da kod izuzetaka treba da zapamtimo izračunate rezultate za in order stanje u registarskom fajlu koji odgovara vrednostima arhitekturalnih registara u trenutku kada se događa izuzetak za isti programski kôd, a prilikom sekvencijalnog izvršavanja.

Taj registarski fajl će morati da se unese u registarski fajl nepreimenovanih *Data Ready* vrednosti kod Tomasulo algoritma, kako bi mašina mogla da nastavi nakon prekida! Naravno, mogu postojati dva takva registarska fajla u modelu dvostrukog baferisanja koji naizmenično imaju uloge registarskog fajla nepreimenovanih *Data Ready* vrednost i registarskog fajla koji čuva trenutno in order stanje.

Faze izvršavanja instrukcija kod Tomasulo algoritma sa preciznim prekidima.

Osnovna izmena u fazama izvršavanja instrukcija je zbog odvajanja trenutka upisa rezultata od trenutka završetka instrukcije. Razlog za to je zakašnjena obrada izuzetaka dok se ne uradi commit instrukcije. Zato se i uvodi in order completion kao nova faza u izvršavanju instrukcija. Distribucija rezultata (write) se češće obavlja više ciklusa pre commit-a (in order completion-a) za potrebe IrDDSet instrukcija za njihove argumente, jer se samo agresivnim spekulativnim izvršavanjem može dobiti visok nivo paralelizma. Mora da postoji složeni mehanizam utvrđivanja in order stanja, kako bi se utvrdilo kada instrukcija ulazi u fazu commit-a. Taj posao radi Reorder buffer.

Reorder buffer

Sam naziv govori o tome da posle izmene redosleda izvršavanja i završetaka instrukcija treba da se nekako uvede red u instrukcije, uvođenjem originalnog redosleda zbog commit-a instrukcija. Kako je originalni (programski) redosled bio ustvari predviđeni dinamički trag od strane prediktora grananja, reorder buffer treba da nekako sačuva programski (predviđeni) redosled za sve instrukcije na dinamičkom tragu za potrebe commit-a. Taj isti redosled je kod Tomasulo algoritma korišćen prilikom ubacivanja instrukcija u instrukcijski prozor, da bi se pravilno odredile zavisnosti po podacima. Nameće se ideja da se prilikom ubacivanja instrukcija u instrukcijski prozor, one u nekom obliku ubace u neku kompleksnu FIFO strukturu koja će pamtiti originalan programski redosled, a FIFO redosled bi generalno pomagao da se odredi dokle su commit-ovane instrukcije.

Ta kompleksna FIFO struktura se realizuje kao kružni bafer, a njegova veličina mora da bude veća od broja instrukcija u instrukcijskom prozoru. Kako takva struktura mora da ima pokazivače head i tail, postavlja se pitanje uloge svakog od ovih pokazivača. Head pokazuje najstariju instrukciju (najranije ubačenu) u instrukcijskom prozoru za koju još nije izračunat rezultat i čijim izračunavanjem se pomera Head. Tail pokazuje gde treba da se ubaci u Reorder buffer instrukcija koja se upravo ubacuje u instrukcijski prozor.

Head se u principu pomera u skokovima, jer kada se izračuna rezultat instrukcije na koju je pokazivao head, češće će se dogoditi da je nekoliko sledećih instrukcija po programskom redosledu već izračunato pre nego što se pojavi instrukcija sa neizračunatim rezultatom. Ta neizračunata instrukcija će postati nova pozicija head-a reorder buffer-a. Tail se takođe pomera u skokovima, ako se više instrukcija u istom ciklusu ubacuje u instrukcijski prozor. Veličina ovog skoka je jednaka broju instrukcija koje se kao paket ubacuju u jednom ciklusu u instrukcijski prozor.

Detekcija promene in order stanja

U trenutku izračunavanja rezultata najstarije instrukcije u Reorder buffer-u, potrebno je prvo detektovati taj događaj, a zatim je potrebno odrediti za koliko mesta se mora pomeriti pokazivač za Head. Pri ubacivanju u ROB, svaka instrukcija dobija pridruženi executed bit postavljen na 0, dok rezultat instrukcije nije izračunat. Kada se rezultat instrukcije izračuna, taj bit se postavlja na 1. Kada je u pitanju instrukcija (lokacija u ROB) na koju pokazuje Head, njenim setovanjem na 1 se započinje posao određivanja nove tačke Head-a, koja treba da bude na najstarijoj instrukciji u ROB koja ima vrednost bita executed na 0. Za to postoji odgovarajući hardver koji u jednom ciklusu obavlja pomeranje Head pokazivača.

Kada je već detektovana nova lokacija Head-a, potrebno je odrediti koje od instrukcija iz skupa, koje su ovim događajem upravo commit-ovane, menjaju in order stanje u registarskom fajlu koga ćemo ovde nazvati history buffer. Ako su sve originalno upisivale u različite arhitekturalne registre, tada se svi njihovi rezultati upisuju u odgovarajuće lokacije registarskog fajla zaduženog za čuvanje in order stanja (history buffer-a). Ako dve ili više instrukcija iz tog skupa upravo commit-ovanih instrukcija upisuju u isti originalni arhitekturalni registar, pamti se samo poslednji upis, na osnovu definicije in order stanja. Kako se potreba za velikim brojem upisa javlja na nivou ciklusa, potrebno je da ta registarska memorija ima veliki broj ulaznih portova i njeno ažuriranje se mora u nekim slučajevima raditi u više ciklusa. Od trenutka kada se dogodi greška u predikciji grananja ili exception, potrebno je najmanje 3 ciklusa (fetch, decode i provera raspoloživosti resursa pre ubacivanja u rezervacionu stanicu i ROB) da se dođe do trenutka kada je sadržaj history buffer-a potreban, pa to i ne mora da predstavlja veliku manu.

Kada instrukcija ažurira registarski fajl koji čuva in order stanje, ona više nije potrebna u ugrađenoj data flow mašini i pomeranjem tail-a se oslobađaju lokacije u ROB za nove instrukcije koje ulaze u instrukcijski prozor.

U prethodnim pasusima smo spomenuli upis rezultata iz reorder buffer-a. To znači da reorder buffer kod Tomasulo algoritma mora da bude povezan na zajednički data bus i mora da prepozna u koju lokaciju treba da upiše taj rezultat. Kada bi se za identifikaciju rezultata koristila adresa rezervacione stanice, to bi značilo da sve lokacije ROB moraju paralelno da čitaju taj identifikator i da prepoznaju da li je rezultat za njih. To bi znatno uticalo na kompleksnost ROB.

Razmotrimo odnos između instrukcija, preimenovanog rezultata, rezervacione stanice u kojoj će se nalaziti instrukcija i lokacije u ROB u kojoj se nalazi instrukcija. Očigledno postoji korespodencija 1:1:1:1 između instrukcija, preimenovanog rezultata, rezervacione stanice u kojoj će se nalaziti instrukcija i lokacije u ROB. Sada se javljaju dva kandidata za jedinstveni identifikator na CDB. To mogu da budu adrese rezervacione stanice sa instrukcijom koja generiše rezultat, a može da bude i adresa te instrukcije u ROB.

Sa stanovišta rezervacionih stanica koje očekuju argumente, svejedno je koji će se identifikator koristiti, jer one komapariraju identifikatore koje su dobili pri ubacivanju u prozor za argumente sa identifikatorom na CDB. Sa stanovišta rezervacione stanice emitera, ona mora da ima dodatno polje za adresu lokacije u ROB, ako se koristi ta adresa za identifikator. Međutim, kod ROB se korišćenjem adrese u ROB obavlja klasičan upis umesto preslikavanja iz adrese rezervacione stanice u adresu u ROB. Zato je i prihvaćeno da adresa rezervacione stanice bude jedinstveni identifikator na CDB.

Minimalan sadržaj ROB za određivanje in order stanja kod Tomasulo algoritma

Da bi se prethodna logika za ROB u slučaju Tomasulo algoritma realizovala, neophodno je u svakoj lokaciji ROB imati odgovarajuća polja. Ta polja su: Done, Tip instrukcije, Rd, Vd, PC i Except?

Done polje je jedan bit. Vrednost 1 označava da je rezultat izračunat. Vrednost tog polja se postavlja na 1 kada se sa CDB-a pokupi vrednost rezultata odgovarajuće instrukcije i upiše u polje Vd. Za upis se koristi Identifikator sa CDB koji je ujedno adresa lokacije u ROB! Pri unošenju instrukcije u instrukcijski prozor se ovo polje postavlja na 0, jer instrukcija sigurno nije izračunala rezultat u tom trenutku.

Tip instrukcije je polje koje treba da razdvoji instrukcije koje generišu rezultat za neki od arhitekturalnih registara od instrukcija koje ne upisuju u neki arhitekturalni registar. Takve instrukcije su pre svega store, uslovni skokovi i CMP. Ovo je neophodno da bi se te instrukcije itostavile iz procesa ažuriranja in-order stanja. **Rd** označava adresu arhitekturalnog registra u koji instrukcija upisuje rezultat. **VD** je polje u koje se upisuje rezultat instrukcije, a može biti potrebno za ažuriranje in order stanja. **PC** označava adresu programskog brojača (instruction pointer) instrukcije u ROB koji omogućava da se odredi mesto u programu na kome se desio izuzetak, ako se on dogodi, da bi se pokrenuo odgovarajući mehanizam obrade izuzetka. **Except** označava da li se na toj instrukciji dogodio exception (ili branch misprediction). Ako dodje do commit-a takve instrukcije, pokreće se proces brisanja svih efekata spekulativnih (lookahead) instrukcija.

Oporavak od exception-a

U trenutku kada je urađen commit instrukcije koja je imala setovan bit za izuzetak, neophodno je dovršiti ažuriranje registarskog file-a arhitekturalnih registara koji sadrži in-order vrednosti. Tada on sadrži stanje

koje treba da posluži za novo pokretanje ugrađene dataflow mašine. To se obavlja jednostavnim kopiranjem sadržaja tog registarskog file-a u Registarski fajl nepreimenovanih Data Ready vrednosti. Nakon toga, može da se pokrene izvršavanje nekog drugog kôda u odnosu na odbačeni spekulativni kôd. Istovremeno, ROB se prazni, jer u tom trenutku ugrađena dataflow mašina ima 0 operacija. Ubacivanjem instrukcija istovremeno u ROB I rezervacione stanice se brzo puni ugrađena dataflow mašina i pokreće paralelno izvršavanje instrukcija.