Architektury systemów komputerowych

Projekt 3: "Symulator pamięci podręcznej i predyktora skoków"

Celem implementacji poniższych zadań jest zrozumienie w jaki sposób procesor zarządza pamięcią podręczną i przewiduje skoki na podstawie danych zbieranych w trakcie wykonania programu (ang. *runtime analysis*)

Do rozwiązania zadań skorzystaj ze szkieletu rozwiązania dostępnego w katalogu `skeletons` w repozytorium https://github.com/cahirwpz/ii-ask. Do testowania swoich rozwiązań wykorzystaj pliki danych, które można ściągnąć ze strony prowadzącego. Po implementacji każdego z podpunktów wykonaj pomiary na dostarczonych śladach programów. Narzędzia służące do wygenerowania tych danych są dostępne w katalogu `pintools`.

Symulator pamięci podręcznej

Poniższe rozwiązania muszą działać zarówno w wariancie, gdy mamy tylko pamięć podręczną L1 jak i obie.

1. **[3]** Zaimplementuj pamięć podręczną z mapowaniem bezpośrednim i politykami zapisu: *write-back*, *write-through* oraz przydziału przy zapisie *write-allocate* i *write-no-allocate*. Wybór polityki ma być konfigurowalny z linii poleceń. Upewnij się, że dostępy do danych położonych na granicy dwóch bloków są prawidłowo obsługiwane.
2. **[3]** Dodaj organizację sekcyjno-skojarzeniową z polityką wymiany *random*. Poziom asocjacyjności ma być konfigurowalny z linii poleceń.
3. **[2]** Dodaj politykę wymiany *LRU*.

Na wejściu standardowym program ma akceptować rekordy, w zapisie tekstowym, reprezentujące kolejne dostępy do pamięci. Poniżej przedstawiono możliwe formaty rekordów:

```
w 7ffecf594948 8
r 7facb2f2ee70 4
```

W pierwszej kolumnie jest rodzaj operacji (odczyt lub zapis), w drugiej 64-bitowy adres dostępu, w trzeciej rozmiar danych operacji (1,2,4 lub 8). Zauważ, że nie dbamy o utrzymywanie zawartości danych w bloku.

Na wyjściu Twojego programu powinno pojawiać się:

- w trybie gadatliwym (opcja linii poleceń `--verbose`) dla każdego dostępu jedna linia, w której będzie para "`index:set`" dla trafienia w pamięć podręczną, lub "`miss`" dla chybienia; dla dwupoziomowej pamięci podręcznej odpowiednio z prefiksem "`L1:`" i "`L2:`";
- łączna ilość trafień i chybień dla poszczególnych poziomów pamięci podręcznej, ilość dostępów do RAM.

Dodatkowo Twój program musi akceptować z linii poleceń opis organizacji symulowanej pamięci podręcznej, który dla każdego poziomu cache będzie zawierać informacje o:

- polityce zapisu: `write-back`, `write-through`;
- polityce przydziału przy zapisie: `write-allocate`, `write-no-allocate`;
- polityka wymiany: `lru`, `random`;
- wielkość bloku (potęga dwójki od 8 do 256): `block`;
- liczbę wierszy (potęga dwójki): `size`;
- asocjacyjność (1 = mapowanie bezpośrednie, od 1 do 16): `associativity`;

Przykład wywołania Twojego programu z linii poleceń:

```
./cache-sim --l1-write-back --l1-write-allocate --l1-associativity=4 \
            --l1-block=64 --l1-size=512 --l1-replace=random \
            < data0.in > data0.out
```

UWAGA: Obchodzi nas jedynie symulowanie zachowania metadanych w poszczególnych blokach pamięci podręcznej – dlatego należy zdeklarować typ reprezentujący rekord metadanych. Cache będzie reprezentowany jako tablica (tablic) tychże rekordów.

Symulator predyktora skoków

Od połowy lat 90 procesory implementują różne techniki przewidywania skoków w związku z powiększającą się głębokością potoku – w chwili obecnej między 14 a 20 etapów. Oznacza to, że od momentu, w którym procesor zna licznik programu dla danej instrukcji skoku[1] do momentu jej wykonania może upłynąć wiele cykli. Jeśli w potoku przed instrukcją skoku znajdują się instrukcje ze złej ścieżki programu to procesor będzie musiał wykonać kosztowną operację przeładowania potoku. Zatem dla danego skoku chcielibyśmy możliwie jak najwcześniej wiedzieć adres, do którego przeniesie się licznik programu, a dla skoku warunkowego czy się wykona. Dlatego procesory starają się jak najwięcej wywnioskować z historii wykonania programu i wyłącznie licznika rozkazów – dzięki temu można spekulować ścieżką wykonania kodu jeszcze przed zdekodowaniem instrukcji!

1. **[2]** Zaimplementuj lokalny predyktor bazujący na 2-bitowym liczniku nasycającym, który startuje w stanie *strongly not-taken*. Gdy w tabeli brak jest informacji nt. bieżącej instrukcji wykorzystaj strategię *forward branch always not-taken, backwards branch always taken*.
2. **[2]** Dodaj adaptacyjny predyktor Yale-Patt'a opisany na ćwiczeniach.
3. **[1]** Dodaj możliwość przewidywania adresu docelowego skoków bezwarunkowych i wywołań procedur.
4. **[2]** Zauważ, że bez dodatkowych danych bardzo ciężko jest przewidzieć adres powrotu z podprocedury. Można to zrobić dodając np. 16 elementowy cykliczny bufor do spamiętywania adresów powrotnych.

Na wejściu standardowym program ma akceptować rekordy, w zapisie tekstowym, reprezentujące kolejne zmiany licznika instrukcji w trakcie przetwarzania programu. Zauważ, że instrukcje w architekturze `x86-64` są zmiennej długości, zatem mogą się zaczynać pod dowolnym adresem. Poniżej przedstawiono możliwe formaty rekordów:

```
7fda505e4803 b 7fda505e47e8 0
7fda505e480f b 7fda505e4a50 1
7fda505e4bf7 j 7fda505e47f3
7fda505e4bd7 J 7fda505e4c0b
7fda505e49b1 c 7fda505ea6a0 7fda505e49b6
7fda505ea702 r 7fda505e49b6
7fda505f52b3 C 7fda505e1680 7fda505f52b5
```

Pierwsza kolumna koduje 64-bitowy licznik programu, pod którym leży instrukcja skoku. Druga kolumna to rodzaj instrukcji zmieniającej licznik programu. Trzecia kolumna to miejsce w programie, pod które procesor skacze po wykonaniu instrukcji (skok warunkowy może się nie wykonać). Reszta pól jest zależna od typu instrukcji:

- `b` – skok warunkowy; ostatnia kolumna koduje, czy skok się wykonał czy nie;
- `j` – bezpośredni skok bezwarunkowy (adres docelowy zakodowano w instrukcji);
- `J` – pośredni skok bezwarunkowy (adres docelowy wczytano z rejestru lub pamięci);
- `c` – bezpośrednie wywołanie procedury; ostatnia kolumna koduje adres powrotu odłożony na stos;
- `C` – pośrednie wywołanie procedury;
- `r` – powrót z podprocedury; adres powrotu zdjęto ze stosu.

Na wyjściu Twojego programu powinno pojawiać się:

- dla każdej instrukcji, jedna wartość boolowska mówiąca czy prawidłowo przewidzieliśmy docelowy adres skoku;

[1] Wszystkie instrukcje zmieniające licznik programu – tj. skok warunkowy, bezwarunkowy, wywołanie i powrót z podprocedury.

- łączna ilość dobrze przewidzianych: skoków bezpośrednich / pośrednich, wywołań procedur bezpośrednich i pośrednich, powrotów z podprocedur, skoków warunkowych.

Uwaga: Działanie predyktora składa się z dwóch etapów. W pierwszym predyktor zgaduje gdzie skoczy program – wyłącznie na podstawie licznika programu. W drugim etapie, odpowiadającym wykonaniu się instrukcji, procesor weryfikuje pracę predyktor – aktualizuje dane w BHT i BTB – i ew. musi wycofać się ze spekulacji i zapłacić odpowiednią karę (ang. *misprediction penalty*).

Dodatkowo Twój program musi akceptować z linii poleceń opis organizacji symulowanego predyktora:

- liczba wierszy w tabeli BHT / BTB (potęga dwójki): `size`;
- typ predyktora: `static`, `2bit`, `adaptive`.

Przykład wywołania Twojego programu z linii poleceń:

```
./bpred-sim --size=1024 --type=adaptive < data0.in > data0.out
```