P4 programok költségbecslése

Alexy Marcell

2023. december 22.

Kivonat

A félév során bekapcsolódtam egy projektbe, amely P4 programok költségbecslésével foglalkozott. A P4 programokat hálózati csomagok feldolgozására használják, és fontos kérdés, hogy a csomagok feldolgozása milyen sebességgel és erőforrásigénnyel történik. Én ezen belül azt vizsgáltam, hogy a programban a keresőtáblák használata mekkor memóriakésleltetéssel jár bizonyos gyorsítótárak használatával.

1. Bevezetés

A P4 (Programming Protocoll-independent Packet Processors)[1] egy hálózati csomagok feldolgozására használt nyelv, melyben különböző csomagok és különböző protokollok elemzését egy egységes programozási nyelven keresztül lehet megvalósítani. A egyéni kutatómunka során a csomagok elemzéséhez használt programok időigényével foglalkoztam. Ez az idő egyrészről a processzorban az számolásra használt ciklusok időigényéből áll, melyet a témavezetőm, Lukács Dániel és társai a cikkükben[5] elemeztek.

A P4 programok nagy része a különféle keresőtáblákban történő kereséssel történik. Ezeknek a tábláknak a mérete jelentősen nagyobb a program által használt változók méretéhez képest. Ezért a táblák mérete miatt egy teljes keresőtábla nem fog a processzor regisztereiben tárolódni, és itt jelentős lehet a memóriából való betöltés ideje.

A kerső táblák P!-en belül három típusba sorolhatóak: Egyrészről lehet a "ternary" típusú tábla, ahol egy vizsgált bitekek közül egy kulcs azt jelöli, hogy milyen biteket kell a cosmagból használni, és használt bitek minek kellenek, hogy megfeleljenek. Egy másik táblatípus az "lpm" (least-prefix matching), ahol a csomagból használt bitek mindig az első valahány bitnek felelnek

meg. A harmadik típusú táblában, az "exact" táblában, a beérkező csomagnak meg kell pontosan egyeznie a kulccsal. A kutatásom során én a harmadik típusú táblával foglalkoztam.

A P4 specifikációban [2] leírják, hogy az ilyen típusú táblákat általában hashtáblával szokták implementálni.

2. Processzor gyorsítótárak

Egy processzor működése során két fajta időigényes műveletet kell megkülönböztetni. Egyfelől az processzor regisztereiben tárolt adatokon való műveletvégzéshez, például regiszterek összehasonlításokhoz vagy memóriacím számolásához, bizonyos számú processzor ciklusnyi időt kell várni, hogy az processzor áramköre beálljon a megfelelő állapotba. Másfelől az adatnak a memóriából meg kell érkeznie a processzor megfelelő magjához tartozó regisztereibe. Ez utóbbi műveletnek az időigényével foglalkoztam mélyebben.

A számítógépek története során volt, hogy a processzor relatív sebessége gyorsabb volt a memória késleltetéséhez képest, és a fordított esetre is volt példa. Jelenlegi processzortechnológia esetén egy memóriából beérkező adatra várva a processzor 200 másik [3] műveletet is végre tudna hajtani, ha nem várna az adatra.

Emiatt a legtöbb mai nagyobb processzor rendelkezik valamilyen processzoron található gyorsítótárral. Ezek a gyorsítótárak gyorsabban tudnak reagálni, alacsonyabb a késleltetésük, mint a memóriának a késleltetése, ugyanakkor a memóriához képest kisebb mérettel rendelkeznek.

Az összetettebb processzorok általában többszintű gyorsítótár hierarchiával rendelkeznek. Általában három szint szokott lenni, és rzeket a gyorsítótárakat méret és késleltetés szerinti növekvő sorrendben L1, L2, L3 rövidítésekkel szokták jelölni. Az L1 és L2 szintek egy darab processzormaghoz köthetők, míg egy többmagos processzor esetében az L3-as gyorsítótár meg szokott osztva lenni az összes processzormag között.

Ha ezeket kibővítjük az elején a regiszterekkel és a végén a memóriával, akkor megkapjuk az adatokhoz szükséges teljes hierarchiát. Ebben a hierarchiában feltehetjük, hogy egy bizonyos szinten található adatot tartalmazni fog a hierarchiában felette lévő, nagyobb kapacitással rendelkező szintek is. Vagyis a regiszeterekben található adat megtalálható a többi gyorsítótárban is, vagy az L2-ben található adatot tartalmazza a memória is.

A cache vizsgálatakor fontos még azt vizsgálni, milyen módon kerül ki a cache-ből egy darab memóriarészlet. A cache a memóriát cacheline-ok szintjén keezelik, melynek mérete általában 64 bájt szokott lenni. Az x86-os processzorokban egy cache-ből kikerülő cacheline kiválasztására összetett stratégiák vannak, de egyszerűbb ARM-es processzorok esetében általában pszeudovéletlenszerűen választják ki a kikerülő cachelinet [4]. Én az utóbbi esettel foglalkoztam.

3. Egy hash tábla elérési sebessége

A hash tábla egy adatszerkezet, ahol kulcsok alapján értékekekt lehet keresni, egy kulcshoz egy darab érték tartozhat. A kulcsokhoz egy rajtuk elvégzett hash művelet segítségével hozzárendelünk egy indexet.

Lehetséges, hogy több kulcshoz ugyanazt az indexet rendeljük, ebben az esetben hash-ütközés lép fel. Ennek a feloldására több módszer van, ugyanakkor a memóriakésleltetés szempontjából meg kell vizsgálni, hogy egy elem lekérdezésekor várhatóan hány cachelinet szükséges betölteni. A P4 specifikiációban[**p4table**] írják, hogy lineáris próba használja a legkevesebb cacheline-t, ezért ennek az elemzésével foglalkoztam.

Jelöljük a hashtábla használati rátáját α-val, ami megfelelel a használt helyek számával osztva a teljes tábla méretével. Ha megfelelő a hash függvény, akkor uniform véletlenszerűen osztja be a kulcsokat a helyekre. Egy elem lekérdezésekor lehet, hogy az adott helyet már használja egy másik kulcs, ekkor rátérünk a következő vizsgálatára, és így tovább, amíg meg nem találjuk a kulcsot. Ez alapján felírhatjuk a vizsgált helyek számát:

$$\mathbb{E}(\text{Keresett helyek száma}) = 1 + \mathbb{P}(\text{Foglalt hely}) * \mathbb{E}(\text{Keresett helyek száma})$$

$$\mathbb{E}(\text{Keresett helyek száma}) = \frac{1}{1-\alpha}$$

A cache modell szerint először be kell tölteni a cache vonalat a processzorhoz közel, és csak utána lehet vizsgálni az egyes mezőket a táblában. Ehhez érdemes tudni a várhatóan használt cache vonalak számát. Ez függhet a cacheline méretétől, amit jelöljünk s_l-vel, a kulcs méretétől, amit jelöljünk s_k-val, valamint attól, hogy a keresést egy cacheline-on belül milyen eltolással indítjuk a keresést. Összességében ezt lehet úgy modellezni, hogy egy valós számtól elindítunk egy valamilyen hosszúságú szakaszt, és azt számoljuk meg,

hogy az egész számok közti egységhosszú szakaszokból várhatóan hányat metsz. Ezt a következő képlettel lehet meghatározni:

$$\mathbb{E}(\text{Használt cachelineok száma} = 1 + \mathbb{E}(\text{Keresett helyek száma}) \cdot \frac{s_k}{s_l}$$

$$\mathbb{E}(\text{Használt cachelineok száma} = 1 + \frac{1}{1-\alpha} \cdot \frac{s_k}{s_l} = k$$

A továbbiakban jelöljük az egy lekérdezés által használt cachevonalak számát k-val.

Valós esetben ez az érték várhatóan kettő alatt lesz. Az $s_l = 64$ bájt a legtöbb mai processzornál és $s_k = 4$ IPv4-es címek lekérdezésekor. Ha tábla töltöttségi szintje kevesebb, mint 90%, akkor ez az érték ≈ 1.63 lesz, ami nagyobb tábla és alacsonyabb töltöttségi szint esetén tovább csökkenhet. Amikor a P4 programot feltöltitik, akkor a program futása közben a tábla értékei nem változnak, és a tábla módosításánál az egész táblát újra feltöltik. Ezért a tábla mezőinek az elhelyezését előre meg lehet adni, amivel tovább lehet optimalizálni ezeket az értékeket.

A használt cachevonalak-at először be kell tölteni a processzorközeli cache-be a memóriából. A cache nagyságát jelöljük s_c-vel, és a tábla által használt memória nagyságát s_t-vel. Amennyiben $s_l << s_c$, feltehetjük, hogy a cachelineok sűrűn vannak egymás után, és folytonosnak vehetjük az elemzésben. Ha csak egy hashtáblát elemzünk, akkor feltehetjük a cache-ben a tábla értékein kívül más nem tartózkodik.

Először tegyük fel, hogy a beérkező csomagok uniform véletlenszerűen, egymástól függetlenül érkeznek vizsgálatra, és emiatt a táblából uniform véletlenszerűen vizsgálunk kulcsokat. Ekkor nézzük annak a valószínűségt, hogy a beérkező csomag vizsgálatakor használt cacheline benne van a cache-ben. Ha a cache nagyobb, mint a tábla mérete, akkor ez 1, ha $s_t > s_c$, akkor s_t/s_c, és $1 - s_t/s_c$ valószínűséggel kell az új cacheline-t betölteni.

Amennyiben az egymás utáni csomagok közt van összefüggés, például ugyanahhoz a helyhez menő csomagok érkeznek egymás után $p_s = \mathbb{P}(\text{Ugyanazt a mezőt vizsgáljuk egymás után})$, amit a csomag forgalom elemzésével lehet kiszámolni, akkor a következő csomag által használt cacheline-ok már benne vannak a cache-ben 1 valószínűséggel, és nem kell újra betölteni a cacheline-t. Legyen a t_{21} az L1 cache-be való betöltés ideje, ekkor a cacheline-ok az L1 cache-be való betöltésre várhatóan használt idő, amit t_{lc}-vel jelölünk, amikor a tábla mérete nagyobb a cache méreténél:

$$t_{lc} = k\left(1 - \frac{s_t}{s_c}\right)(1 - p_s) \cdot t_{21}$$
$$t_{lc} = \left(1 + \frac{1}{1-\alpha}\right)\left(1 - \frac{s_c}{s_t}\right)(1 - p_s) \cdot t_{21}$$

A $t_2 1$ időt megkapjuk, ha az egy darab L2-beli elem eléréséhez szükséges időből kivonjuk az egy darab L1-beli elem eléréséhez szükséges időt.

Ez alapján ki lehetne számítani egy táblához cacheline betöltés szempontjából optimális táblaméretet is. Amennyiben a tábla mérete nő, az telítettség (α) csökken, ugyanakkor a cache-ből kiszorult táblaméret ($1 - s_c/s_t$) nő. Valamint többszintű memória esetén, ahol az L1, L2, stb. cache mérete s_{c1}, s_{c2}, stb. és betöltési idejük az L1-be t_{21}, t_{32}, stb. ugyanezt ki lehet számolni minden egyes szintre. A fenti képlet az t_{lc1}-nek felel meg, az L1-be való betöltés várható idejéből az L2 cacheből, és ehhez hozzá kell adni az t_{lc2}-t, ami az L2 cachebe való betöltés ideje az L3-ból; $t_l c2$-t ugyanezzel a képlettel de más paraméterekkel kaphatjuk meg, és további szinteket is vizsgáhatunk így.

A teljes képhez hozzá kell venni az L1 cacheben vizsgált kulcsok számát, az L1-beli kulcsok a regiszterekbe való betöltésének az idejét, amit t_1-gyel lehet jelölni. Ezt jelöljük t_{lk}-val:

$$t_{lk} = \mathbb{E}(\text{Keresett helyek száma}) \cdot t_1$$
$$t_{lk} = \frac{1}{1-\alpha} \cdot t_1$$

Ezeket összeadva, kapjuk a memóriakezelésre használt teljes t_l időt (tegyük fel, hogy L3 tartalmazza a teljes táblát):

$$t_l = t_{lk} + t_{lc1} + t_{lc2}$$
$$t_l = \frac{1}{1-\alpha} \cdot t_1 + (1 - p_s) \cdot \left(1 + \frac{1}{1-\alpha}\right)\left(\left(1 - \frac{s_{c1}}{s_t}\right) \cdot t_{21} + \left(1 - \frac{s_{c2}}{s_t}\right) \cdot t_{32}\right)$$

4. Két hash tábla elérési sebessége

Amennyiben több tábla van, akkor egy csomag által mezők számát egy táblában a tábláktól függetlenül meg lehet határozni. Ugyanakkor a

cacheline-ok betöltésénél fontos tényező lesz, hogy az egyik vagy a másik tábla hogyan használja az adott cache-t. Ha az egyik tábla betölt egy cacheline-t, akkor az lehet azzal jár, hogy a másik táblának egy caheline-ja kikerül a cache-ből.

Vizsgáljuk meg azt az esetet, hogy két hashtábla van a programban használva, egy csomag elemzésekor mindkét hashtáblát használjuk, legyen az egyik tábla A-val, a másik B-vel jelölve. Az egy tábla mérete s_{tA}, a másik tábla mérete s_{tB} legyen. A csomagok elemzéséhez használt várható A cache-vonalak száma legyen k_A, és hasonlóan vezessük be k_B paramétert is.

A cache vizsgálatakor az is fontos, hogy az egyes táblák mekkora része van a cache-ben. Ezt jelöljük s_{cA} és s_{cB} változókkal. Habár csomagok feldolgozása közben ezek a válatozók nőhetnek/csökkenhetnek, ahogy egy táblá cahe-beli része nő vagy csökken, lesz egy egyensúlyi állapot. Ekkor, ha a táblákat egymás után vizsgáljuk, akkor a várhatóan cacheből kirakott A-beli cahceline-ok száma megegyezik az lekérdezés során cache-be berakott A-beli vonalak számával.

Egy A-beli cacheline akkor kerül ki, ha a B táblából való a lekérdezés során kikerül egy A-beli cacheline helyére egy B-beli kerül. Fordított esetben, A-beli lekérdezés során kikerül B-beli cacheline, s_{cA} növekedhet, és ez a két eset van. Ezeknek az esélye megegyezik azzal, hogy egy bizonyos táblából való lekérdezés során a kérdezett cacheline nincs benne a cacheben, mivel csak akkor raknak ki egy cachevonalat, ha egy másiknak érkeznie kell helyette. Az egyensúlyi helyzetben:

$$
\begin{aligned}
k_A \cdot \mathbb{P}(\text{A érkezik})\mathbb{P}(\text{B kikerül}) &= k_b \cdot \mathbb{P}(\text{B érkezik})\mathbb{P}(\text{A kikerül}) \\
k_A \left(1 - \frac{s_{cA}}{s_{tA}}\right)\left(\frac{s_{cB}}{s_{cA}+s_{cB}}\right) &= k_B \left(1 - \frac{s_{cB}}{s_{tB}}\right)\left(\frac{s_{cA}}{s_{cA}+s_{cB}}\right) \\
\frac{s_{cA}}{s_{cB}} &= \frac{k_A}{k_B}\frac{1-\frac{s_{cA}}{s_{tA}}}{1-\frac{s_{cB}}{s_{tB}}}
\end{aligned}
$$

Feltehetjük, hogy a cache mérete egy konstans s_c, és a két táblán kívül nincs más a memóriában. Ekkor $s_{cB} = s_c - s_{cA}$, és a fenti egyenletbe ezt behelyettesítve kapunk egy másodfokú egyenletet, amelyben csak s_a változó. Ennek két megoldása lesz, de csak a pozitív megoldásával kell foglalkozni. Ha foglaltságúak a táblák, akkor feltehetőleg $k_A \approx k_B$.

$$s_a = \frac{s_c(s_{cA} - s_{cB}) - 2s_{cA}s_{cB} \pm \sqrt{s_c^2(s_{cA} - s_{cB})^2 + 4s_{cA}^2 s_{cB}^2}}{2(s_{cA} - s_{c_B})}$$

Ennek az egyenletnek a pozitív megoldását kell választani.

Hivatkozások

[1] URL: https://opennetworking.org/p4/.

[2] URL: https://github.com/p4lang/p4-spec/blob/main/p4-16/spec/docs/p4-table-and-parser-value-set-sizes.md.

[3] URL: https://www.7-cpu.com/cpu/Cortex-A53.html.

[4] URL: https://developer.arm.com/documentation/ddi0500/j/Functional-Description/About-the-Cortex-A53-processor-functions/Data-side-memory-system?lang=en.

[5] Dániel Lukács, Gergely Pongrácz és Máté Tejfel. *Open Computer Science* 11.1 (2021), 70–79. old. DOI: doi:10.1515/comp-2020-0131. URL: https://doi.org/10.1515/comp-2020-0131.