Practicum 8: De geheugenhiërarchie

Tibo De Peuter, Mathis De Witte

21 Mei 2023

Cachegedrag van een dubbele lus

2. Houd de cachegrootte constant op 128 bytes, en bedenk voor deze grootte een formule die de theoretische missrate (#missers / #toegangen) in functie van de blokgrootte voor een direct mapped cache kan bepalen. Je mag ervan uitgaan dat (i) de matrices geen conflictmissers veroorzaken binnen eenzelfde iteratie en (ii) het aantal toegangen een veelvoud is van de blokgrootte van de cache. Leg deze formule uit.

$$\text{missrate} = \frac{4}{\text{blokgrootte}} \frac{\text{missers}}{\text{toegangen}}$$

Als de blokgrootte groter wordt, dan duurt het langer voordat het programma tot buiten de grenzen van een blok terecht komt, dus duurt het langer voordat er een mis voorkomt. 4 bytes.

3. Verifieer je formule door middel van de cachesimulator. Klopt je formule bij elke blokgrootte? Zoniet, leg uit. Denk hier in termen van koude missers, capaciteitsmissers, en conflictmissers.

We willen zien dat de missrate halveert wanneer we de blokgrootte verdubbelen.

```
Blokgrootte: 32 bytes  => Verwachte missrate = 1/8  <-> Effectieve missrate = 0.1250 => OK!
Blokgrootte: 64 bytes  => Verwachte missrate = 1/16 <-> Effectieve missrate = 0.0625 => OK!
Blokgrootte: 128 bytes => Verwachte missrate = 1/32 <-> Effectieve missrate = 1.0000 => NIET OK!
```

Als er maar 1 blok is, dan treden er **conflictmissers** op doordat het programma afwisselend de eerste matrix en de tweede matrix opvraagt. Het programma zal eerste de eerste matrix in de cache steken, daarna de cache overschrijven met de tweede matrix, daarna weer overschrijven met de eerste matrix, …

4. Voor een vaste cachegrootte, wat is het effect van een toenemende associativiteit op de missrate in dit geval (dus als we van een direct mapped cache zouden overgaan naar een n-wegs associatieve cache)? Verklaar opnieuw in termen van de verschillende types van misser.

Als we overschakelen op een n-wegs associatieve cache en de cachegrootte gelijk laten, dan daalt de missrate, doordat er minder conflictmissers zijn (door de natuur van n-wegs).

Koude missers (de matrices staan nog altijd op dezelfde plaatsen in het geheugen en moeten nog altijd eens ingeladen worden) en capaciteitsmissers (de blokgroottes blijven gelijk) blijven gelijk.

5. In sommige talen (Fortran) worden matrices niet rijgewijs, maar kolomsgewijs in het geheugen opgeslaan. Als we een dergelijke matrix overlopen met het bovenstaande programma, hoe zou de toegangsgrafiek er dan uit zien? Je kunt hiervoor opnieuw gebruik maken van de cachesimulator met dezelfde parameters als voorheen, maar ditmaal met toegangspatroon ColumnMajor. Wat is de missrate nu? Verklaar deze missrate.

In plaats van 1 enkele rechte lijn, zien we op de grafiek meerdere stijlere lijnen die zich herhalen. De cacheblokken kunnen nu maar nuttig gebruikt worden voor 1 geheugentoegang, in vergelijking met 8 in het geval van rijgewijs.

We verwachten missrate = 1 en kunnen dit ook bevestigen aan de hand van de simulator.

Uitleg:

Wanneer je een element uit een matrix haalt, dan wordt heel de rij van dat element in de cache geplaatst. Als je kolomgewijs elementen uit de matrix haalt, zal je telkens wanneer je een nieuw element uit de kolom haalt, een nieuwe rij in de cache moeten plaatsen.

Specifieker; Bij de eerste iteratie wordt de hele eerste rij ingeladen. Bij de tweede iteratie wordt de tweede rij ingeladen enzovoort. Als het programma dan overgaat naar het verwerken van de tweede kolom, dan herhaalt zich dit opnieuw en moet de cache steeds opnieuw gevuld worden.

6. Hoe verschilt de lokaliteit tussen de toegangspatronen RowMajor en ColumnMajor?

De spatiale lokaliteit van RowMajor is best goed: De geheugenlocaties van een rij staan vlak naast elkaar, en het programma overloopt de matrix rij per rij.

ColumnMajor heeft een bijzonder slechte spatiale lokaliteit: Het programma overloopt de matrix kolom per kolom, maar de matrix is rij per rij opeslagen, dus de geheugenlocaties liggen steeds op een rij-afstand van elkaar verwijderd.

De temporele lokaliteit is voor beide toegangspatronen gelijk: Het programma berekent meteen wat het moet berekenen, en hoe dat gebeurt is niet afhankelijk van het toegangspatroon.

Matrixvermenigvuldiging C = AB

1.a) Maak in deze vraag gebruik van een 4-wegs set-associatieve cache, met een blokgrootte van 32 bytes en een totale cachegrootte van 256 bytes. Hoeveel sets heeft deze cache? Leg je berekening uit.

Formule: $\frac{\text{aantal blokken}}{n}$

We weten: aantal blokken $= \frac{\text{totale cachegrootte}}{\text{blokgrootte}}$

$$\begin{aligned}&\frac{\frac{\text{aantal blokken}}{n}}{\frac{\text{totale cachegrootte}}{n * \text{blokgrootte}}}\\ =&\frac{265}{4 * 32}\\ =&2\end{aligned}$$

1.b) Beredeneer de afzonderlijke missrates (#missers / #toegangen) voor de matrices A, B en C (met gegeven adressen). Bereken de totale missrate die hier uit volgt, en verifieer deze vervolgens, gebruik makend van de simulator. Om het toegangspatroon van Figuur 8.2 te verkrijgen (met de matrices op de gegeven adressen), maken jullie deze keer niet gebruik van jullie groepsnummer als argument.

Omdat we matrix A rijgewijs overlopen zal de lokaliteit voor A goed zijn. We beredeneren

$$\begin{aligned}\text{missrate} =&\frac{4}{\text{blokgrootte}}\frac{\text{missers}}{\text{reads}}\\ =&\frac{4}{32}\frac{\text{missers}}{\text{reads}}\\ =&\frac{1}{8}\frac{\text{missers}}{\text{reads}}\end{aligned}$$

Omdat we matrix B kolomsgewijs overlopen hebben we hier een slechte lokaiteit. We beredeneren missrate = 1 volgens dezelfde redenering als in vraag 5 in 8.2 (over Fortran).

2. Welke matrix heeft de beste temporele lokaliteit? Hoe kun je dit afleiden uit het toegevoegde stukje code?

We zien in de code dat matrix C de enigste matrix is die enkel anders uitgelezen wordt in de buitenste twee loops van de code. Voor de binnenste loop zal dus steeds hetzelfde element gelezen en geschreven worden. C heeft de beste temporele lokaliteit, want zowel A en B worden anders uitgelezen afhankelijk van de binnenste loop.

4. Bespreek hoe klassieke en getegelde vermenigvuldiging verschillen van elkaar qua lokaliteit.

De spatiale lokaliteit van matrices A en C zijn bij getegelde vermenigvuldiging iets minder goed dan bij klassieke vermenigvuldiging. Dat 'verlies' valt sterk mee in vergelijking met de opvallend betere spatiale lokaliteit van matrix B bij getegelde vermenigvuldiging.

5. Indien er geen caches aanwezig zouden zijn, welke van de twee methodes van vermenigvuldiging zou het snelste uitvoeren? Hierbij moet je er van uitgaan dat het geheugen een uniforme, constante toegangstijd heeft voor alle geheugentoegangen. Verklaar je antwoord. Je kan er hierbij van uitgaan dat er voor beiden evenveel instructies uitgevoerd zullen worden.

De implementatie van getegelde vermenigvuldiging gebruikt dubbel zo veel lussen om de berekening uit te voeren. We weten dat het programma steeds vanuit geheugen zal moeten lezen, want er zijn geen caches. Als we er dan vanuit gaan dat de toegangstijd constant is, dan verliest de getegelde vermenigvuldiging enkel nog tijd door de extra procestijd die nodig is voor het dubbele aantal lussen in de code. De getegelde vermenigvuldiging zal dus trager zijn dan de klassieke vermenigvuldiging.