Practicum 7

Practicum 8: De geheugenhiërachie (deel 2)

In dit practicum bouwen we verder op practicum 7. Het doel van dit practicum is inzicht te verwerven in de effecten van de keuze van cache-strategie en de bijbehorende parameters (blokgrootte, associativiteit) op de hit- en missrate van een programma. Tevens willen we het effect van het aanpassen van het algoritme in het programma op de efficiëntie van het cachegebruik aanschouwelijk maken.

In dit practicum zullen we gebruik maken van de cachesimulator en de verschillende cachetypes die daarvoor werden geïmplementeerd tijdens het vorige practicum. Je kunt hiervoor beschikken over de implementatie van de NWayAssociativeCache in een class bestand dat we ter beschikking stellen.

Het is eenvoudig te zien dat elk van de andere caches een speciaal geval is van een N-wegs set-associative cache. In de directory `pract08` vind je alle nodige class-bestanden, alsook de broncode van CacheSim.java. Daarnaast vind je er ook een spreadsheet, waarvan je gebruik kunt maken om adresstromen te visualiseren en de vragen op te lossen.

Opgelet! De LibreOffice spreadsheet die we gebruiken is vrij groot, als je Linuxomgeving hier traag op lijkt te werken heeft je virtuele machine allicht te weinig geheugen. Zet VirtualBox uit, en pas de Systeem-eigenschappen van je virtuele machine aan zodat deze meer geheugen toegewezen krijgt (gebruik minstens 1GiB, maar meer kan zeker geen kwaad).

7.1 Indienen

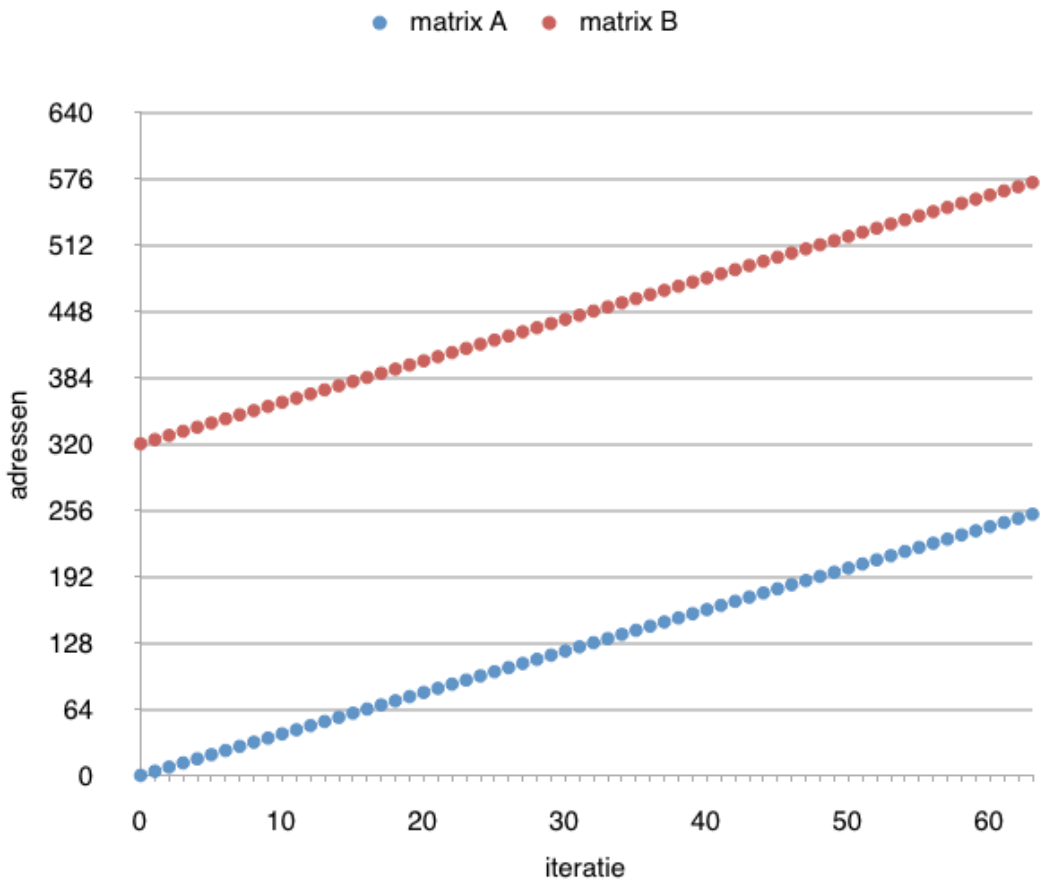
Dit practicum wordt gequoteerd. Je moet een verslag indienen voor **woensdag 21 mei 2014, 22:00:00** via `https://indianio.ugent.be`. Let op! We verwachten per groep één enkel zip bestand waarin de volgende bestanden zijn opgenomen.

- `practicum8_verslag.pdf`
- `practicum8_grafieken.ods`

7.2 Cachegedrag van een dubbele lus

In het eerste deel van het practicum beschouwen we een triviaal programma waarvoor je het cachegedrag zal analyseren. De essentie van het programma zit vervat in het volgende stukje code:

```
1 for (int i = 0; i < N; ++i) {
```



Figuur 7.1: Voorbeeldpatroon van geheugentoegangen bij het rowmajor doorlopen van het codefragment voor het verdubbelen van de elementen van een $N \times M$ matrix.

```
2       for (int j = 0; j < M; ++j) {
3           B[i][j] = 2 * A[i][j];
4       }
5   }
```

Het is duidelijk dat dit programma een matrix A doorloopt en elk element ervan verdubbelt alvorens de resulterende waarde te stockeren in de datastructuur die de matrix B voorstelt.

Veronderstel even dat we gebruik maken van een programmeertaal waarin de matrix A in *row-major-order* opgeslaan wordt (bvb. in C of Java), m.a.w. in het geheugen vinden we eerst de elementen (in volgorde) van rij 1, daarna de elementen van rij 2, enz. tot en met rij N. Je mag ervan uitgaan dat de variabelen i en j in een register worden bijgehouden, en bijgevolg geen geheugentoegangen vereisen om geïncrementeerd te worden. De toegang naar het element $A[i][j]$ vereist dat we het adres $\text{base} + iN + j$ gebruiken. Als de matrix A opgeslaan wordt vanaf adres $\text{base} = 0$ en dat $N = M = 8$, dan wordt het adres voor het element $[i, j]$ gegeven door $4(8i + j)$. Onderstel verder dat de matrix B opgeslaan wordt vanaf adres 288, dan wordt het adres voor element $[i, j]$ gegeven door $288+4(8i+j)$.

Als we de geheugentoegangen uitzetten in functie van de tijd voor de bovenstaande implementatie van dit vermenigvuldigingsalgoritme, verkrijgen we de grafiek gelijkaardig aan die in Figuur 7.1. Je kunt deze figuur maken in bvb. Excel, Numbers, of je favoriete spreadsheet. Bovendien kunnen de experimenten voor het bekomen van de data die getoond wordt in Figuur 7.1 op een eenvoudige

manier bekomen worden door het toegangspatroon rowMajor aan te leggen aan de simulator (voer "./compile.sh; ./run.sh" uit voor meer informatie over het correcte gebruik van de simulator). In de uitvoer van de simulator tonen we per iteratie informatie over de cache-toegangen: naam van de matrix, geheugenadres, en of de toegang een hit of miss is (cache-misses worden aangegeven met een "*").

7.2.1 Opgave

1. a) Duid op de grafische voorstelling van het rowMajor toegangspatroon de missers aan bij gebruik van een direct mapped cache met een grootte van 128 bytes en met een blokgrootte van 32 bytes. Geef tevens te kennen welke adressen bij elke misser geladen worden in de cache, door het geheugen dat in de cache aanwezig is anders in te kleuren. Maak hiervoor gebruik van het tabblad "invulblad opgave 1(row)" van de bijgevoegde spreadsheet. Maak gebruik van de cachesimulator om de adressen in te vullen in het tabblad "data opgave 1".

   b) Welke adressen zitten er in de cache (en in welk blok) bij iteratie 40.

2. Op basis van je analyse bij de vorige vraag, bepaal nu de theoretische missrate (#missers / #toegangen) in functie van de blokgrootte voor een direct mapped cache. 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. Verifieer uw formule door middel van de cachesimulator.

3. 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)? Onthoud dat de cachegrootte gegeven wordt door blokken $\times$ blokgrootte $\times$ associativiteit.

4. 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, maar ditmaal met toegangspatroon columnMajor. Gebruik een directmapped cache van 128 bytes met een blokgrootte van 32 bytes.

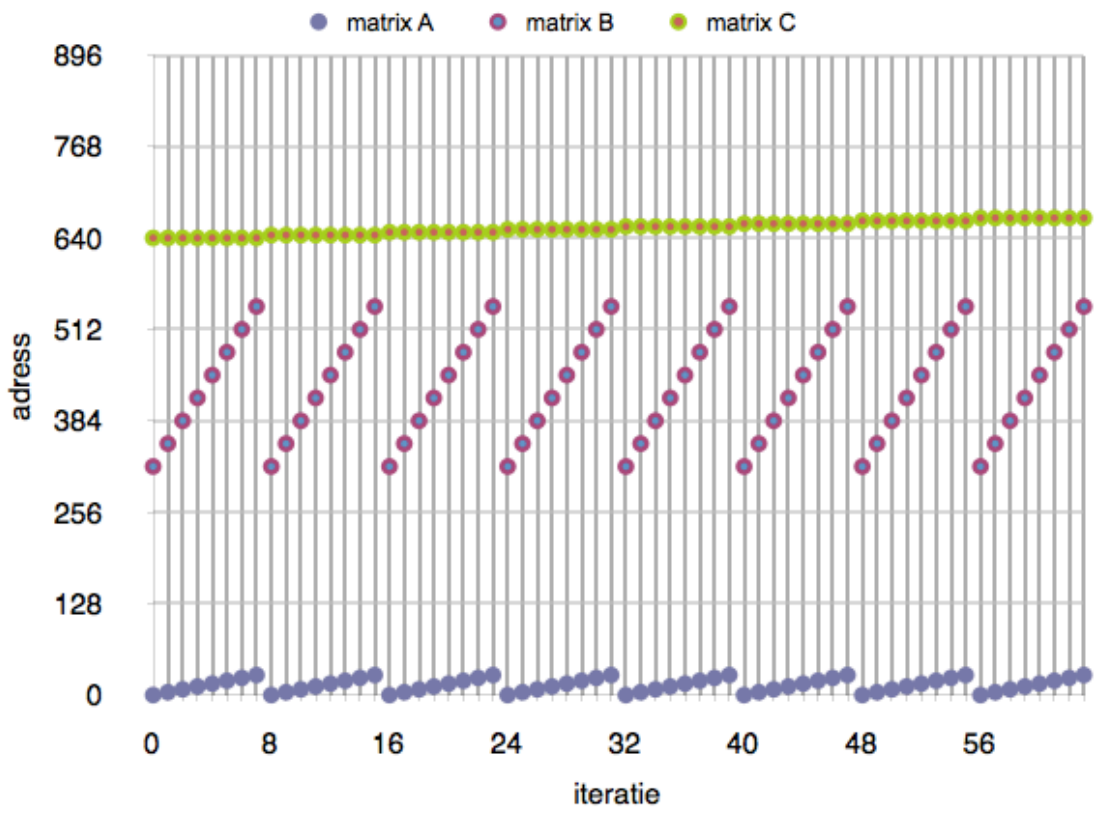
5. Uitgaande van de situatie geschetst in de vorige vraag, wat is de missrate nu? Maak opnieuw een overzichtelijke grafische voorstelling, gebruik makende van het rekenblad voor dit practicum.

6. Voor een vaste cachegrootte (128 bytes), varieer de blokgrootte en de associativiteit. Welke cacheconfiguratie verkies je te gebruiken om de vermenigvuldiging in column-major-order uit te voeren? Motiveer je antwoord in je verslag.

7.3 Matrixvermenigvuldiging C = AB

Beschouw twee matrices, A en B. De klassieke gekende werkwijze om het product te berekenen is voor elk element $[i, j]$ van het product in row-major-order het inproduct te bepalen van de i-de rij van A met de j-de kolom van B. Bijvoorbeeld als volgt:

```
1 for (int i=0;i<size;i++) {
2     for (int j=0;j<size;j++) {
3         for (int k=0;k<size;k++) {
4             C[i][j] = C[i][j]+ A[i][k]* B[k][j];
5         }
```



Figuur 7.2: Patroon van geheugentoegangen bij het doorlopen van het codefragment voor het vermenigvuldigen van twee matrices.

```
6         }
7 }
```

Hierbij onderstellen we C initieel de nulmatrix. Maak opnieuw gebruik van de cachesimulator om de volgende vragen te beantwoorden.

1. a) Beschouw Figuur 7.2, die de toegangspatronen toont om op de klassieke wijze het product $A \times B$ te bepalen en op te slaan in C, voor matrices die 8×8 elementen bevatten en waarbij er 4 toegangen zijn per berekening: lezen van $A[i,k]$, $B[k,i]$, $C[i,j]$ en schrijven van $C[i,j]$. 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? Beredeneer wat de missrate zal zijn voor de matrices A, B en C en verifieer, gebruik makend van de simulator.

   b) Geef aan welke toegangen cache misses zijn en geef de inhoud van de cache weer bij iteratie 40 van de binnenste lus (voor welke adressen zitten op dat moment in de cache?).

2. Welke matrix heeft de beste temporele lokaliteit? Hoe kun je dit afleiden uit de Java-code?

3. Als we een verdeel-en-heers-strategie toepassen op de matrixvermenigvuldiging, dan blijkt dat we ook via tegeling de vermenigvuldiging kunnen uitvoeren, waarbij er natuurlijk op gelet dient te worden dat de dimensies kloppen. Je kunt de effecten van deze strategie op het cachegedrag vinden door de `matrixTiledMultiply` optie mee te geven met `CacheSim`. Teken hiervan het toegangspatroon en duid aan wanneer de missers optreden.

4. Geef aan voor welke adressen de cache data bevat bij iteratie 40 van de binnenste lus. Ga ook hierbij uit van dezelfde cache als in vraag 1 van dit deel.
5. Leg uit waarom de getegelde versie een betere lokaliteit vertoont dan de klassieke vermenigvuldiging.
6. Leg het verband uit tussen de tegelgrootte en de blokgrootte.