Practicum 7

Practicum 7: De geheugenhiërarchie (deel 1)

In dit practicum worden enkele aspecten van de geheugenhiërarchie bestudeerd. Het is de bedoeling dat je de code die je hier ontwikkelt ook gebruikt voor het laatste (gequoteerde) practicum. Het practicum bestaat uit drie delen. In het eerste deel breid je een bestaande (eenvoudige) cachesimulator uit. In het tweede deel bekijk je de opbouw van de geheugenhiërarchie in de processor van de machine die je gebruikt voor het practicum of van je eigen machine thuis. Het laatste deel tenslotte bestaat uit een literatuuropdracht: na het lezen van een klassiek artikel i.v.m. caches los je enkele eenvoudige vragen op.

7.1 Ontwikkeling van een eenvoudige L1-cachesimulator

Bij het ontwerp van de cache(s) van een systeem wordt beroep gedaan op simulaties om de meest optimale cacheconfiguratie te achterhalen die binnen bepaalde beperkingen op de processor kan geïmplementeerd worden. In dergelijke simulaties worden typische sequenties van geheugentoegangen aan de cachesimulator meegegeven. Deze simulator houdt de adressen bij waarvoor de data zich in de cache bevindt en rapporteert het aantal cache hits en cache missers. Het is dus *niet* de bedoeling dat de simulator rekening houdt met de data in de cache, enkel de *toegangspatronen* (adressequenties) zijn belangrijk! De simulator hoeft dus helemaal niet te weten welke data zich op de adressen uit de adresstroom bevinden. Op deze manier kunnen verschillende cacheconfiguraties en werklasten (gemodelleerd als sequenties van geheugentoegangen) worden geëvalueerd, om uiteindelijk te komen bij een ontwerp dat goed presteert voor de meeste (liefst voor alle) werklasten.

De bedoeling van het tweede deel van dit practicum is om een eenvoudige simulator te schrijven die verschillende cachetypes kan simuleren: een direct mapped cache, een volledig associatieve cache met een LRU vervangingsstrategie, alsook een N-wegs set-associatieve cache. Heel wat van de functionaliteit van de simulator werd reeds geïmplementeerd. Bestudeer nauwkeurig de 2 Javabestanden `CacheSim.java` en `Cache.java`.

Om volledig te werken, vereist CacheSim nog de implementatie van drie ontbrekende klassen: DirectMappedCache, FullyAssociativeCache en NWayAssociativeCache. Elk van deze klassen implementeert de Cache interface en simuleert de cache die correspondeert met de bestandsnaam. In de opgave van dit deel zul je deze klassen implementeren en testen.

Zoals reeds aangegeven werd, bestaat de invoer voor de simulator uit een sequentie van *geheugenadressen*. In deze simulator worden de adressen gegenereerd op basis van simpele matrixbewerkingen. We zullen twee patronen gebruiken die je kan instellen door middel van de derde parameter. Voor het eerste patroon (patroon1) itereren we over alle waarden van een matrix A en schrijven we die weg in een tweede matrix B. Het tweede patroon (patroon2) bekomen we door voor elke A[i,j]

Direct mapped cache	Volledig associatieve cache
`Block 0: true 64`	`Block 0: true 519`
`Block 1: true 64`	`Block 1: true 255`
`Block 2: true 64`	`Block 2: true 247`
`Block 3: true 64`	`Block 3: true 239`
`Block 4: true 64`	`Block 4: true 231`
`Block 5: true 64`	`Block 5: true 518`
`Block 6: true 64`	`Block 6: true 223`
`Block 7: true 7`	`Block 7: true 215`
`Total Requests: 2048`	`Total Requests: 2048`
`Cache Hits: 672`	`Cache Hits: 768`
`Hit Rate: 0.328125`	`Hit Rate: 0.375`

Tabel 7.1: Uitvoer van de cachesimulator voor de sequentie geheugenadressen gegenereerd door `patroon1`

alle waarden van de rij i en de kolom j op te tellen en het resultaat weg te schrijven in een tweede matrix B. De details van deze patronen kan je terugvinden door naar de corresponderende code te kijken in CacheSim.java, doet dit ook! Het is belangrijk dat je goed snapt hoe deze patronen eruit zien.

Indien de simulator de waarde `-1` als geheugenadres krijgt, moet de toestand van de cache op het scherm worden geprint. Als je de DirectMappedCache klasse hebt geïmplementeerd (en gecompileerd), kan je de simulatie uitvoeren door middel van het commando:

```
./sim -cache DirectMapped -blocks 8 -linesize 16 -pattern patroon1
```

voor een direct mapped cache bestaande uit 8 blokken met een blokgrootte van 32 bytes en met toegangspatroon 1 als invoer; analoog

```
./sim -cache FullyAssociative -blocks 8 -linesize 16 -pattern patroon1
```

voor een volledig associatieve cache als FullyAssociativeCache is geïmplementeerd.

Nadat je de twee ontbrekende klassen correct hebt geïmplementeerd, krijg je de volgende uitvoer in Tabel 7.1 te zien voor de bovenstaande commandolijnen.

7.1.1 Opgaven

1. Schrijf en test de klasse DirectMappedCache die de Cache interface implementeert. De methode request simuleert een cache-aanvraag voor een specifiek 32-bit adres en geeft bij een cache miss false terug als resultaat; bij een hit is de teruggeven waarde natuurlijk true. Bij een cachemisser wordt verondersteld dat het overeenkomstige geheugenblok in de cache wordt geladen. Dit betekent dat de adressen voor dat blok vanaf dat ogenblik gekend zijn. Je mag ervan uitgaan dat het volledige blok ingeladen is voor het volgende adres aan de cache wordt aangelegd. Implementeer tevens de methode dump. Deze methode moet de inhoud van de cache op het scherm printen volgens het volgende formaat:

   ```
   Block 0: [valid] [tag]
   ```

   Gebruik `patroon1` als derde argument om de correctheid van jouw implementatie te testen: vergelijk deze met de uitvoer in Tabel 7.1.

   Tip: ga eerst na hoeveel bits er worden gebruikt voor de tag, de index en de offset (de adressen zijn 32 bits).

2. Probeer een antwoord te vinden op de volgende vragen.

   a) Wat is de minimale grootte van een Direct Mapped cache om een hit rate van 50% te bekomen (verander hiervoor zowel de blokgrootte als het aanta blokken)? Bespreek je oplossing.

   b) Is het mogelijk om een hit rate van 100% te bekomen? Waarom wel/niet? Motiveer je antwoord.

3. Doe nu hetzelfde experiment maar voor toegangspatroon 2, doe dit door `patroon2` als derde argument mee te geven. Ga de miss rates bij verschillende cachegroottes na aan de hand van jouw implementaties in de simulator. Vergelijk de cacheprestatie voor de verschillende toegangspatronen en cacheconfiguraties. Wat valt er op? Beschrijf de trends die je bekomt.

4. Implementeer nu ook de klasse FullyAssociativeCache. Gebruik het LRU-vervangingsalgoritme. Gebruik als invoer zowel `patroon1` als `patroon2`. Maak een vergelijkende studie tussen DirectMappedCache en FullyAssociativeCache a.d.h.v. twee grafieken voor elk van de adresstromen. Hou hierbij eenmaal het aantal blokken constant op 4, en eenmaal de blokgrootte constant op 32 bytes. Welke cache-implementatie is het meest efficiënt volgens jou?

5. Tot slot moet je een implementatie schrijven voor een N-wegs set-associatieve cache. Noem je klasse NWayAssociativeCache. *Let op: de parameter N wordt nog niet ingelezen*. Je moet bijgevolg de noodzakelijke wijzigingen aanbrengen in `CacheSim.java`.

6. Geef één belangrijk voordeel en één belangrijk nadeel van zowel een direct mapped als een volledig associatieve cache. Beschouw enkel effecten die een invloed hebben op de prestatie (dus niet op de chipoppervlakte of het energieverbruik).