Eerste Toets Concurrency

17 december 2015, 11.00–13.00, Educ-Γ.

Motiveer je antwoorden *kort*! Zet je mobiel uit. Stel geen vragen over deze toets; als je een vraag niet duidelijk vindt, schrijf dan op hoe je de vraag interpreteert en beantwoord de vraag zoals je hem begrijpt.
Cijfer: Opgaven 1 t/m 4 zijn elk 2pt en opgaven 5 t/m 7 zijn elk 3pt. T1 is totaal plus 1, gedeeld door 1,6. Maak opgaven 1, 2 en 3 op bladzijde 1, opgaven 4 en 5 op bladzijde 2, en opgaven 6 en 7 op bladzijde 3.

1. **Hardware:** Een processor met vier fysieke cores doet zich naar het operating system voor als een processor met acht cores.

   (a) Welke technologie wordt hier beschreven?
   (b) Een zekere applicatie heeft een 20% hogere throughput op deze processor dan op een verder gelijkwaardige processor zonder de bedoelde technologie (dus met "gewoon" vier cores). Hoe valt deze snelheidswinst te verklaren?

   Oplossing: (a) Hyperthreading.
   (b) Wanneer er een vertraging optreedt in een van de hardware threads, schakelt de processor naar de andere hardware thread en maskeert zo de vertraging door werk uit te voeren van de andere thread.

   Beoordeling: Voor elk correct deelantwoord een punt. Beoordelingscodes:

   A = Verkeerde vraag beAntwoord. Geen punt.
   M = Bij (b) Maskeren van stalls niet expliciet genoemd, stalls zelf wel: 1pt.
   Goede Methode of formule maar verkeerd antwoord, halve punt.
   P = Verwart hyperthreading met Pipelining.
   V = Bij (b) niet verwezen naar Vertragingen of stalls als reden voor het wisselen van threads. Geen punt.

2. **Parallellisme:** Wat is het verschil tussen potential en mandatory concurrency?

   Oplossing: Mandatory concurrency: toewijzing van parallelle taken aan threads.
   Potential concurrency: opdeling van werk in taken die in parallel uitgevoerd kunnen worden, waarbij de daadwerkelijke verdeling over threads overgelaten wordt aan een task scheduler.

   Beoordeling: Wanneer beide begrippen correct beschreven worden: 2 punten. Beoordelingscodes:

   A = Verkeerde vraag beAntwoord.
   C = Cores en threads verward.
   H = Alleen herhaling van de vraag.
   M = Alleen Mandatory concurrency is correct beschreven, 1pt.
   P = Alleen Potential concurrency is correct beschreven, 1pt.
   R = Potential/mandatory en Regular/irregular verward.
   T = Taken OS en taskmanager verward, geen pt.

3. **Vectorizatie:** In een simulatie wordt de volgende data gebruikt:
   class Particle, met de properties 'argb' van type Vector4, en 'size', van type float.
   En array particles met 16384 instances van de Particle class.

   (a) Hoe wordt, in de context van vectorizatie, deze data layout genoemd?
   (b) Herstructureer de data layout zodat deze geschikt is voor vectorizatie. (Ga uit van 4-wide SIMD.)

   Oplossing: (a)Array of Structures of AoS.
   (b) class Particle4, met de properties a4, r4, g4, b4 en size4 van type Vector4, en array particle2 van type Particle4 en lengte 4096.

   Beoordeling: Voor elk correct deelantwoord een punt. Beoordelingscodes:

   A = Verkeerde vraag beAntwoord. Geen punt.
   F = Bij (b) een afwijkende structuur. Geen punt.
   G = Geen volledige structuur met eenduidige layout gegeven, geen pt.
   I = Verwarring tussen termen SIMD en AoS.
   S = Bij (a) Structure of Arrays of andere foute naam. Geen punt.
   V = Bij (b) andere naamgeving van de properties, dit is ook goed.

4. **Speculative Execution:** Leg (in circa 10 regels) uit hoe speculative execution kan leiden tot een super-linear speedup bij vectorizatie.

   Oplossing: Speculative execution is het loskoppelen van de uitvoer van conditionele code en het evalueren van de conditie (met andere woorden: het verwijderen van de dependency tussen de evaluatie van de conditie en de uitvoer van de conditionele code). De effecten van de onvoorwaardelijk uitgevoerde code worden genegeerd of tenietgedaan als de conditie onwaar is.
   Het onvoorwaardelijk uitvoeren van code die wellicht niet uitgevoerd had hoeven worden heeft een belangrijk voordeel:
   er kan geen branch misprediction voorkomen. Een branch misprediction leidt tot het opnieuw vullen van de pipeline van de processor. De kosten hiervan kunnen hoger zijn dan de kosten van de uitvoer van de conditionele code, waardoor speculative execution sneller kan zijn.
   Het verband met vectorizatie zit in de vector instructieset: deze bevat instructies die specifiek gericht zijn op speculative execution.

   Beoordeling: Kernbegrippen in het antwoord zijn: dependency, conditionele code, brach misprediction, pipeline.
   Wanneer de begrippen correct in verband zijn gebracht: 2 punten. Beoordelingscodes:

   A = Verkeerde vraag beAntwoord. Geen punt.
   B = Individuele Begrippen goed uitgelegd, verband niet: 1pt.
   C = "Conditional" niet meegenomen in antwoord.
   G = Enkel gevolgen van het concept uitgelegd maar niet het concept zelf.
   L = Super-Linear verkeerd geinterpreteerd. Geen punt.
   P = Speculative execution verward met Pipelining.
   S = Speculative execution verkeerd geinterpreteerd. Geen punt.
   V = Vectorizatie verkeerd geinterpreteerd. Geen punt.
   X = Speculative eXecution verward met branch prediction.

5. **Het TaS-lock:** Met een enkel TaS-register kun je een multi-threaded lock maken.

   (a) Geef de code voor dit TaS-lock (`lock` en `unlock` methode).
   (b) Welke van deze eigenschappen heeft het TaS-lock *niet*: **Mutal Exclusion**, **NoDeadlock**, **NoStarvation** en leg uit waarom niet.
   (c) Welke instructie(s) in C# kun je gebruiken om de TaS-operatie te implementeren?

   Oplossing: (a) De pseudocode voor `lock()` is `while(L.TaS == 1) { }` en de `unlock()` is `L.write(0)`.
   (b) Het TaS-lock lijdt aan Starvation: een thread die wacht op het lock kan eindeloos lang pech hebben doordat steeds andere threads doorkunnen nadat het lock is vrijgegeven. Hij heeft dus *niet* **NoStarvation**.
   (c) Je hebt de klasse `Interlocked` nodig, en daaruit de methode `Exchange`.

   Beoordeling: Per deelvraag een punt dus max 3. Beoordelingscodes:

   C = Je hebt geen CompareExchange nodig, de Exchange is genoeg.
   D = Oneindig lang wachten op een thred die het lock niet vrijgeeft is wel een Deadlock op applicatieniveau, maar geen schending van de Deadlock eis op het lock.
   E = Voor de Eis No Starvation geldt de premisse dat alle threads het lock ooit weer vrijgeven. Blijven wachten op een thread die het lock vasthoudt, is dus geen schending van NoStarvation.
   I = De Increment niet gebruiken! Die verandert een 1 in een 2.

6. **Register Implementaties:** Voor het implementeren van wacht-vrije registers bestaan o.a. deze technieken: (A) Copying, (B) Silent Write, (C) Sequence Numbers, (D) Unary Representation.

   (a) Welke hiervan gebruik je om (i) van een Safe Bit een Regular Bit te maken; (ii) van een Regular Register een Atomic Register te maken; (iii) van een Regular Bit een Regular m-Valued Register te maken?
   (b) Leg uit hoe de Silent Write werkt.
   (c) Kun je deze technieken combineren om met registers een TaS-instructie te implementeren?

   Oplossing: (a) i = B, ii = C, iii = D.
   (b) De Silent Write wordt gebruikt bij een *write* op bits. Als de te schrijven waarde gelijk is aan de vorige waarde, wordt de write niet uitgevoerd (is Silent). Hierdoor kan een read op de bit *alleen* overlappen met een write die de waarde ook echt verandert, en in dat geval is zowel een 0 als een 1 goed als antwoord.
   (c) Dat zal niet gaan, want het is onmogelijk om met registers wachtvrij een TaS te implementeren. Die heeft namelijk Consensusgetal 2.

   Beoordeling: Max 3, een per deelvraag. Bij (a) is 2 goede antwoorden 1/2, 1 goed is 0pt. Codes:

   C = Beredeneer het met Consensusgetallen!
   M = Geen Motivering of onzinnig verhaal.
   L = Silent Write zal *niet* het shared register uitlezen, maar checkt tegen een locale kopie van de laatstgeschreven waarde.
   S = Silent Write werkt alleen voor iets dat een Single Writer heeft. Bij meerdere writers werkt het niet meer, daarom kun je er geen TaS mee implementeren.

7. **Consensus Number:** (a) Wat wordt bedoeld met het *Consensus Number* van een object?
(b) Wat is het Consensus Number van (i) Het Snapshot Object; (ii) Een Register; (iii) Een Compare-and-Swap; (iv) de Test-And-Set?
(c) Is het mogelijk om, uitgaande van alleen registers, een wachtvrije Stack te implementeren? Bewijs je antwoord!

Oplossing: (a) Het object heeft Consensus Getal n als het mogelijk is om er Consensus mee te implementeren voor n threads, wachtvrij.
(b) Antwoord: i=1, ii=1, iii=∞, iv=2.
(i) Snapshot is met Registers te bouwen dus heeft CN=1. (ii) Register heeft CN=1, bewezen op college. (iii) Met de Compare-and-Swap (C#: `Compareexchange`) kun je Consensus implementeren voor elk aantal threads, dus CN is ∞. (iv) De taS heeft CN=2.
(c) Dat zal niet gaan want Registers hebben CN=1 en Stack heeft CN$\geq$2. Het laatste kun je inzien met een arbiter-constructie; zet op de stack een 1 en een 0, dan zal de eerste Popper de 0 krijgen en de tweede Popper de 1.

Beoordeling: Tot 3pt, eentje per deelvraag. Bij (b) is 3 goed 1/2, 2 is 0pt. Codes:

C = Beredeneer het met Consensusgetallen.
K = Onjuist is "Nee want je kunt geen Kritieke Secties maken" want dan moet je nog aantonen dat de functionaliteit van de Stack een CS vereist.
T = Onjuist dat je met registers nooit iets zou kunnen maken wat werkt voor Twee threads.
W = Consensus bereikt overeenstemming over een Waarde die als argument is meegegeven, niet over een thread id (verwart doel met middel).