Tweede Deeltoets Concurrency

Donderdag 1 februari 2018, 8.30 – 10.30, Educ-Γ.

Licht je antwoorden *kort* toe. Schrijf niet teveel op: onzin toegevoegd aan een goed antwoord kan aftrek geven.
Cijfer: Vraag 3 is 2pt en de andere vragen elk 3pt. T2 is totaal plus 1, gedeeld door 1,4.

1. **Parallel Maximum:** We willen het grootste getal uit een array A van n verschillende getallen bepalen op een CRCW PRAM.

   (a) Laat zien hoe de CRCW PRAM de *conjunctie* van n condities c_i, dus $c_1 \wedge c_2 \wedge \ldots \wedge c_n$, kan bepalen in lineair (dus $O(n)$) werk en constante (dus $O(1)$) span.
   (b) Laat zien hoe het mogelijk is om het maximum van een array A van n getallen te bepalen met kwadratisch veel werk in constante span.
   (c) Geef een methode om het maximum te bepalen in $O(n\sqrt{n})$ work en constante span. **Hint:** Verdeel de invoer in $\sqrt{n}$ groepjes.

   Oplossing: (a) Dit is het duale van de OR (die op hoorcollege is voorgedaan):

   ```
   (1)  R = true;
   (2)  par.for(i, 1, n) if (!c[i]) R = false;
   ```

   Na afloop is R true wanneer alle c_i true zijn, en false wanneer minstens 1 c_i false is, dus de conjunctie.
   (b) Kom op het idee om (a) toe te passen. Getal $A[p]$ is het maximum van de array als geldt dat voor alle i: $A[p] \geq A[i]$, wat precies een conjunctie van n condities is. (Als je dit opmerkt, je beroept op (a), en voldoende duidelijk uitlegt dat de gewenste span en work gehaald worden, kreeg je hier al je punt.) Dat checken we voor alle mogelijke p parallel volgens de truuk van (a). Regel 1 stelt voor elke p de waarde op plek p kandidaat. Regel 2 slaat p weer knockout als er enig getal i groter is. Regel 3 schrijft de (enige) overgebleven kandidaat naar de uitvoer.

   ```
   (1)  par.for(p, 1, n) k[p] = true;
   (2)  par.for(p, 1, n) par.for(i, 1, n) if (A[i] > A[p]) k[p] = false;
   (3)  par.for(p, 1, n) if (k[p]) result = A[p];
   ```

   Het werkt ook prima trouwens als je de drie parallele loops tot eentje combineert.
   (c) Zo'n hint staat er vast niet voor niets. Als ik de invoer van n getallen verdeel in $s = \sqrt{n}$ groepjes, heeft elk groepje omvang n/s ofwel omvang s. In elk groepje kan ik, volgens de methode van (b), het maximum vinden in constante span met s^2, dus n, werk. Voor de $\sqrt{n}$ goepjes samen is dit dus $O(n\sqrt{n})$ werk en, omdat de groepjes onafhankelijk zijn, nog steeds constante span. Hierna zit ik nog met $\sqrt{n}$ groepswinnaars, ik heb dan nog n werk en constante span nodig om (weer met de methode uit (b)) het uiteindelijke maximum te bepalen.

   Beoordeling/Toelichting: Per deelvraag een punt. Je kunt (a) zien als een kennisvraag, (b) als een toepassingsvraag en (c) als doordenkvraag. Beoordelingscodes:

   A = Jij laat Alle threads de c_i schrijven naar R. Resultaat is dan niet per se de conjunctie, maar de waarde van de laatst schrijvende thread.
   B = Waarom elke c_i vergelijken met zijn Buur? Overbodig!
   C = Onduidelijk hoe de deelresultaten geCombineerd worden. Er bestaat geen Concurrent Return op de CRCW PRAM!
   L = Concurrent Read/Write slaat op operaties op dezelfde Locatie.
   OR = Er werd gevraagd om AND maar jij geeft OR, 1/2pt.
   R = Een recursieve oplossing kost logaritmische span. Elk ander schema dat volgens een boomplan paarsgewijze competities houdt ook.

U = Een Update zoals `R = R && c[i];` of `if (A[i] > max) max = A[i];` is inherent sequentieel en kan *niet* parallel, zelfs niet op een CRCW PRAM.

2. **Lengte van Greedy Schedule:** Een parallelle berekening met work w en span s wordt uitgevoerd op p cores volgens een Greedy Schedule. Om het aantal rondes te beredeneren, letten we speciaal op die rondes waarin *de span van de resterende berekeningsgraaf* afneemt; zulke rondes noemen we *krimpend*.

   (a) Beschrijf hoe een greedy schedule tot stand komt.
   (b) Bewijs dat er exact s krimpende rondes zijn.
   (c) Bewijs dat er hoogstens $\frac{w-s}{p}$ niet-krimpende rondes zijn en dat de lengte van de schedule begrensd is door $s + \frac{w-s}{p}$.

   Oplossing: (a) De Greedy Scheduler kiest in elke ronde een maximaal aantal *ready* stappen. Dit zijn er p in een "volle" (core-bound) ronde, of minder in een "lege" (ready-bound) ronde. Een lege ronde is krimpend omdat alle ready stappen worden uitgevoerd, dus alle ketens in lengte afnemen. Een niet-krimpende ronde is dus altijd vol.
   (b) In het begin is de langste keten in de graaf s operaties. Geen enkele ronde kan de langste keten meer dan 1 doen afnemen, omdat er van de keten maar 1 operatie kan worden uitgevoerd. Elke afname is dus een afname met 1, zodat het aantal afnames s is.
   (c) Omdat elke krimpende ronde minstens 1 operatie uitvoert, en er daar exact s van zijn, worden er hoogstens $w - s$ operaties in niet-krimpende rondes gedaan. Een niet-krimpende ronde kiest niet alle ready stappen, dus is vol en voert p operaties uit. Hun aantal is dus begrensd door $\frac{w-s}{p}$, en daarom het totaal aantal ronden (inc. krimpende) door $s + \frac{w-s}{p}$.

   Beoordeling/Toelichting: Te behalen 3pt, voor elk onderdeel 1.

   E = Benoem/bewijs dat de span *hoogstens* 1 kan afnemen per ronde.
   L = De Laatste conclusie van (c) volgt direct uit de andere twee dingen die je moest beredeneren, en alleen deze Laatste conclusie levert geen punt op.
   R = Een greedy scheduler maakt geen Random keuze uit de ready stappen, maar een nondeterministische.
   S = Je kunt een berekening niet Splitsen in een "span-pad" en "non-span knopen". In veel antwoorden werd er geredeneerd alsof er een sliert van s knopen is die in s ronden wordt verwerkt, plus $w - s$ knopen die dan vanzelf allemaal in ronden van p kunnen. Work en span zijn geen aparte delen van de berekeningsgraaf, maar twee eigenschappen van die graaf.
   U = Het langste pad in de graaf is niet Uniek!
   V = Benoem/beargumenteer dat een niet-krimpende ronde Vol is (dus p stapen doet).
   W = Maximaal aantal ready stappen klopt maar is erg summier; zeg Waardoor dit beperkt wordt (minder ready's dan cores, of alle cores bezet).
   Z = Greedy neemt Zoveel mogelijk ready stappen, maar het maakt niet uit welke. Ready verkiezen boven non-ready is onzinnig omdat non-ready stappen helemaal niet kunnen/mogen.

3. **Run-length decoding:** Leg uit (liefst met een diagram) hoe een run-length encoded string parallel *gedecodeerd* kan worden met behulp van de prefix sum. Gebruik hierbij de string: $A3B2F8D2$, die uitpakt naar $AAABBFFFFFFFFDD$. Neem aan dat de posities in de uitvoer Randomly Accessible zijn, dwz., in constante tijd geschreven kunnen worden.

   Oplossing: We starten per letter-cijfer paar een task. Iedere task weet welke letter in de output moet komen, en hoe vaak, maar niet waar. We berekenen dus eerst een (exclusive) prefix sum over de aantallen: $\{3, 2, 8, 2\} \Rightarrow \{0, 3, 5, 13\}$. De resulterende array geeft voor iedere task aan, vanaf waar de karakters geschreven moeten worden.

   Beoordeling/Toelichting: Twee punten voor een correct antwoord.

   Voor een string van n karakters met langste run R, decodeert bovenstaand algoritme in n work en $\lg n + R$ span. Deze span is dus een bottleneck als er vrij lange runs zijn.
   Het is mogelijk de output in $\lg n$ span op te bouwen door nogmaals prefix sum te gebruiken. Zie elke positie als een karakter-naar-karakter functie, initieel ingevuld als de identiteit. Als de ligging van een run bekend is, wordt op de eerste positie de constante functie van die letter gezet. Daarna worden alle posities ingevuld door een prefix-functiecompositie.

4. **Parallel inproduct:** Het inproduct van vectoren A en B in $\mathbb{R}^3$ is $A_xB_x + A_yB_y + A_zB_z$. Meer algemeen is het inproduct (engels: dotproduct) van twee reeksen A en B gedefinieerd is als $\sum_{i=0}^{n-1} A_iB_i$.
   (a) Laat (eventueel met een diagram) zien hoe het inproduct in parallel kan worden berekend.
   (b) Als we voor de elementen in A en B floating point getallen gebruiken, kan het gebeuren dat de uitkomst van het parallelle algoritme niet overeenkomt met de uitkomst van een seriële berekening. Geef een voorbeeld waarbij dit gebeurt.
   (c) In de slides wordt gesteld dat voor een grote input array, parallel reduction met vector operaties de hardware vrijwel optimaal benut. Waarom is dit "vrijwel optimaal", en niet gewoon "optimaal"?

   Oplossing: (a) We combineren steeds twee getallen totdat er 1 getal overblijft; zie slides.
   (b) De parallele code voert de additions in een andere volgorde uit, terwijl floating point addition niet associatief is. Je kunt hier dus een voorbeeld geven met drie getallen die opgeteld in twee volgorden een ander resultaat geven. Belangrijk is dat je antwoord laat zien dat de beperkte precisie van een floating point getal hier een cruciale rol speelt.
   (c) Er worden steeds groepen getallen gecombineerd, waarbij het aantal getallen in een groep gelijk is aan het aantal lanes in de SIMD units. De laatste groep moet nog gereduceerd worden tot een enkele waarde; dit gebeurt met scalar operaties, dus zonder gebruik te maken van SIMD.

   Beoordeling/Toelichting: Voor elk correct antwoord een punt.

5. **QuickSort:** QuickSort verdeelt de invoer rond één gekozen waarde, de pivot, in een deel elementen kleinergelijk de pivot en een deel grotergelijk dan de pivot. Deze twee delen worden met een recursieve aanroep gesorteerd. Het verdelen, de `Partition`, is een sequentieel proces dat lineair veel werk kost.

   (a) Neem aan dat de twee delen ongeveer even groot zijn en analyseer de tijd voor sequentiele QuickSort; dwz., geef een recurrente betrekking en los die op.
   (b) Analyseer de *span* van QuickSort wanneer de twee recursieve aanroepen parallel worden gedaan.
   (c) Het is mogelijk, `Partition` te parallelliseren tot $O(\lg n)$ span. Analyseer de Span van QuickSort als deze `Partition` wordt gebruikt.

   Oplossing: (a) Het sorteren kost lineaire tijd voor `Partition` plus twee recursieve aanroepen; als die ongeveer even groot zijn, voldoet de tijd $Q(n)$ voor het sorteren van n elementen aan $Q(n) = O(n) + 2Q(n/2)$.
   Gevalletje Master Theorem met $a = 2$, $b = 2$, $f(n) = O(n)$ dus n^1. Omdat ook ${}^b\log a = 1$ komt er een lg bij en is $Q(n) = O(n \lg n)$.
   (b) Noem de span $S(n)$ voor het sorteren van n elementen. Het verdelen is hier lineaire tijd en sequentieel en dit betekent span $O(n)$. De twee recursieve aanroepen verlopen parallel en geven daarom maar de span van eentje, zodat $S(n) = O(n) + S(n/2)$. Gevalletje Master Theorem met $a = 1$, $b = 2$, $f(n) = O(n)$ dus n^1. Omdat ${}^b\log a = 0$, domineert $f(n)$ polynomiaal dus $S(n) = n$.
   (c) Noem de span met deze parallelle `Partition` $T(n)$. De span bestaat uit de span voor `Partition` plus die van de recursieve aanroep, die we maar eenmaal tellen omdat de twee aanroepen parallel gaan. Dus $T(n) = O(\lg n) + T(n/2)$. Gevalletje Master Theorem met $a = 1$, $b = 2$, $f(n) = O(\lg n)$ dus $n^0 \lg n$. Omdat ${}^b\log a = 0$ wel $f(n)$ domineert, maar slechts met een $\lg n$ factor dus niet polynomiaal, een extra $\lg n$ factor erbij: $T(n) = \lg^2 n$.

   Beoordeling/Toelichting: Per deelvraag een punt.

   A = Als het antwoord ontbreekt of fout is, 0pt voor deze deelvraag.
   B = Als de recurrente Betrekking ontbreekt of fout is, hoogstens 1/2. Een veelgemaakte fout is het vergeten van de functie in de RHS, dus bv $Q(n) = 2.(n/2) + O(n)$. Als je de letter O gebruikt om de functie aan te duiden, wordt de vergelijking $O(n) = 2O(n/2) + O(n)$, wat heel onduidelijk is (fout eigenlijk) dus dat moet je niet doen!
   E = De log-factor bij Evenwicht is Extra, dus komt bovenop de grootste waarde. De gedachte "er is evenwicht dus de uitkomst is $O(\lg n)$" is onjuist!
   L = De Log factor bij (c) werd vaak vergeten. Op de MT-schaal liggen n^0 en $\lg n$, waarbij veel deelnemers wel zagen dat $\lg n$ domineert, maar dachten dat dit genoeg was om het antwoord te zijn. De dominantie is hier *slechts logaritmisch* en daarom is een extra log nodig.
   P = Het Product nemen van $f(n)$ en de recursiediepte is een foute methode, waarvan het resultaat alleen toevalligerwijs met de juiste waarde kan overeenstemmen.
   T = Bij (a) wordt om de Tijd van een sequentieel algoritme gevraagd, en moet je geen onderscheid maken tussen Work en Span.
   V = Voor en na de = moet je dezelfde letter hebben, dus *niet* zoiets als $S(n) = 2Q(n/2) + O(n)$, want dan is het geen recurrente betrekking!