Tweede Toets Concurrency

2 februari 2017, 8.30 – 10.30, Educ-β.

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: Elke *deelvraag* telt even zwaar. Vragen 1, 3 en 5 zijn 2pt, vragen 2 en 4 zijn 3pt en vraag 6 is 4pt. Te halen 16, T2 is totaal plus 1 gedeeld door 1,6.

1. **Inverse Prefix Sum:** Gegeven is een array B. Gevraagd wordt een array A te berekenen, zodanig dat B de PrefixSum is van A.
   (a) Geef een zo goed mogelijk parallel algoritme hiervoor.
   (b) Wat zijn work en span van je algoritme?

   Oplossing: (a) Omdat $B[i]$ de som is van de eerste i getallen, en $B[i-1]$ de som van de eerste $i-1$ getallen uit A, is $A[i]$ terug te vinden als $B[i] - B[i-1]$. En dat kan parallel:

   ```
   par.for(i, 0, n) A[i] = B[i] - B[i-1]
   ```

   (b) Deze n berekeningen kunnen allemaal tegelijk, dus superparallel, in work $O(n)$ en span $O(1)$.

   Beoordeling/Toelichting: Per deelvraag 1pt. Beoordelingscodes:
   B = Bij recursie moet je een recurrente Betrekking opstellen.
   C = In het work-span model omschrijf je het algoritme onafhankelijk van het aantal Cores. Je moet geen code maken waarin p, het aantal cores, expliciet voorkomt. Het verdelen over de cores laat je aan de scheduler over.
   F = Foutief algoritme gegeven (bv voor PrefixSum). Voor een fout algoritme geen punt, soms is bij (b) nog 1/2 gegeven voor een kloppende analyse.
   M = Masking voor jumpen!! Als je `iets(i)` wilt doen voor elke index die een veelvoud van j is (het zgn jumpen) dan moet je die j in je loop verwerken, bv. `pfor (i, 0, n/j) iets(i*j)`. Als je hem in een test verwerkt (masking) kost het je alsnog n tijd: `pfor (i, 0, n) if (i%j==0) iets(i)` doet hetzelfde maar is heel duur!
   P = Een parallelle loop heeft geen volgorde en loop-carried dependencies zijn niet mogelijk.
   R = Een Recursief algoritme met logaritmische span levert maar 1/2 (want het moest zo goed mogelijk).

2. **Work en Span vergelijken:** Arnold heeft een parallel algoritme ontworpen met *work* $\Theta(n^2)$ en *span* $\Theta(\lg^2 n)$. Bert zegt dat zijn eigen algoritme voor deze taak beter is. Bert heeft weliswaar een hogere span, namelijk $\Theta(\lg^3 n)$, maar zijn work is lager, namelijk $\Theta(n^{1,5})$.

   (a) Welk algoritme zal beter presteren bij een lineair aantal, dus $\Theta(n)$, processors? Leg uit!
   (b) Welk algoritme zal beter presteren bij een kwadratisch aantal, dus $\Theta(n^2)$, processors? Leg uit!
   (c) Voor welke aantallen van processors is Arnolds algoritme sneller?

   Oplossing: (a) Bij zo weinig processors (namelijk $p < w/s$ voor beide) zijn de algoritmen allebei Work-bound zodat je de tijd kunt schatten als w/p. Arnold loopt in n^2/n dus lineaire tijd. Bert loopt in $n^{1,5}/n$ dus $\sqrt{n}$ tijd en dat is veel (nl. polynomiaal) sneller.
   (b) Bij zoveel processors zijn beide algoritmen Span-bound zodat je de tijd kunt inschatten als s. Arnold gaat hier $\lg^2 n$ tijd gebruiken en Bert $\lg^3 n$, zodat hier Arnold de snellere is.
   (c) Het break even point ligt bij $p = n^2/\lg^3 n$ en bij hogere p is Arnold sneller. Bij dat aantal is Bert al Span-bound en gebruikt $\lg^3 n$ tijd, maar meer processors verlagen de tijd niet. Met dat aantal processors is Arnold nog Work-bound, hij gebruikt met $p = n^2/\lg^3 n$ cores ook $\lg^3 n$ tijd, maar hij kan het aantal processors nog (tot een factor $\lg n$) verhogen met evenredige speedup.

   Beoordeling/Toelichting: Per deelvraag een punt.

   C = Eerst (c) oplossen en daar (a) en (b) uit concluderen is natuurlijk heel slim! Maar (c) was hier wel een beetje als het moeilijkste stuk bedoeld, en veel kandidaten kwamen met een foutieve oplossing voor (c), of pasten de oplossing verkeerd toe. Uiteindelijk is het dan makkelijker om eerst (a) en (b) rechtstreeks op te lossen door de T_p expliciet uit te rekenen.
   G = Voor het break even point moet je Grootste work delen door Grootste span! Als je de formule w_2/s_1 gaat gebruiken (van p12 van de aantekeningen over work-span model) moet je wel de goeie kiezen voor 1 en 2!
   P = Als je (c) probeert uit te rekenen met de Plus (ipv de Max) kom je er vrijwel zeker niet uit.
   U = De Uitleg was te vaag. Heb je bij (a) Bert en bij (b) Arnold, maar zonder kwantitatieve onderbouwing, dan levert dit samen maar 1/2pt. Motiveer bv wanneer een algoritme work-bound is (en niet met: p is kleiner dan w, want dat klopt niet).
   V = Spreek niet van "het" omslagpunt want de omslag tussen work en span bound is per algoritme Verschillend.

3. **Heterogeen:** In een heterogeen systeem werken CPU en GPU samen om een probleem op te lossen.

   (a) In bepaalde gevallen kan code die sneller door een GPU uitgevoerd kan worden toch beter op de CPU uitgevoerd worden. Leg uit in welke gevallen, en waarom.
   (b) In een hypothetische heterogene renderer wordt het beeld opgedeeld in tiles. CPU en GPU gebruiken work stealing om deze set tiles te renderen. Waarom is dit een slecht idee?

   Oplossing: (a) Het kan zijn dat een van de processoren een deel van de tijd niets te doen heeft. Werk verplaatsen van de processor die de bottleneck is zal dan winst opleveren.
   (b) Dit vergt voortdurende communicatie tussen de CPU en de GPU: beide processoren zullen gebruik moeten maken van dezelfde atomic counter. Aangezien communicatie tussen de CPU en de GPU een hoge latency heeft zal dit systeem dus zeer traag werken.

   Beoordeling/Toelichting: Voor elk correct antwoord een punt.

4. **Parallel Reduction:** In de slides wordt uitgelegd dat voor een parallel reduction de combiner functie associatief moet zijn.

   (a) Leg uit waarom de combiner functie associatief moet zijn.
   (b) Laat zien dat floating point addition niet associatief is.
   (c) Laat zien hoe parallel reduction efficiënt met vector operaties uitgevoerd kan worden. Leg uit waarom de combiner functie bij gebruik van vector operaties, naast associatief ook commutatief moet zijn.

   Oplossing: (a) Meest eenvoudige antwoord: parallelle executie verandert de volgorde waarin we de operaties uitvoeren. Bij seriële uitvoering is dit bijv. (((A+B)+C)+D), terwijl parallel (A+B)+(C+D) zou kunnen worden uitgerekend. Een voorbeeld is hier fijn, maar niet strikt noodzakelijk.
   (b) Hier is een voorbeeld het duidelijkst. In het algemeen: floating point getallen worden met een beperkt aantal digits opgeslagen. Tussenresultaten worden afgerond. Dit kan leiden tot licht afwijkende antwoorden bij een verandering van de volgorde van de operaties.
   (c) Voorbeeld bij 4-wide SIMD: A+B+C+D+E+F+G+H wordt uitgerekend als (ABCD)+(EFGH); dit levert de vector {A+E,B+F,C+G,D+H} op. Drie seriële operaties maken de reductie af. Zoals te zien was. is nu de volgorde van de operands gewijzigd.

   Beoordeling/Toelichting: Voor elk correct antwoord een punt.

5. **Monte Carlo:** Deze toets bestaat uit zes opgaven.

   (a) Leg uit waarom deze vorm van toetsen gezien kan worden als een Monte-Carlo proces. Wat is de integraal in dit verband?
   (b) Is er sprake van stratificatie?

   Oplossing: (a) De kennis van de studenten wordt steekproefsgewijs getoetst: de vragen gaan over willekeurig gekozen onderdelen van de stof. Je kunt dus de pech hebben dat precies die kennis die jij hebt niet helpt, of geluk, als precies die paar feitjes die je kende gevraagd worden. De integraal is de kennis van de student, uit te drukken als kans dat hij een willekeurige vraag kan beantwoorden.
   (b) Ja; de stof is verdeeld in (ongeveer evenveel) vragen over de stof van Gerard en de stof van Jacco. Het kan niet gebeuren dat er toevallig alleen maar vragen over de stof van Gerard gesteld worden.

   Beoordeling/Toelichting: Een punt per correct antwoord.

6. **Parallelle Plip-King:** Plipper is een nieuw sociaal netwerk waar deelnemers maar 1 ding kunnen doen namelijk andere deelnemers *plippen* (je kunt *niet* jezelf plippen). In een groep A van deelnemers is x een *Plip-King* als hij door alle anderen in A geplipt is, maar zelf niemand in A plipt. Een verzameling van $n \geq 2$ deelnemers kan nul of een Kings bevatten, maar niet meer. Hoewel de informatie over onderling plipgedrag van n deelnemers in totaal bijna n^2 bits bevat, is het mogelijk om sequentieel in lineaire (dwz., $O(n)$) tijd te bepalen of die groep een Plip-King heeft.

   Deze opdracht kijkt naar een *parallel Plip-King algoritme*. Voor identifiers i en j kun je in $O(1)$ (dus constante) tijd opvragen of i j plipt. Om te bepalen of groep A van n personen een King heeft, splitsen we A in twee groepen A_1 en A_2, en bepalen recursief of die een Plip-King hebben.

   (a) Als A een King heeft, dan moet dat de King van A_1 of A_2 zijn. Wat moet je doen om te controleren of de deel-Kings inderdaad King van A zijn?
   (b) Stel vergelijkingen op voor de Work en Span van dit algoritme.
   (c) Wat is de uitkomst van de Work en Span?
   (d) Is dit algoritme efficient? Is het optimaal?

   Oplossing: (a) Uit de definitie van King volgt dat als x King is in A, hij dit ook is in een deelverzameling (waar hij in zit). Als x King van A_1 is, plipt hij daarin niemand en wordt door iedereen daar geplipt. Hij zal dus King van A zijn, als hij ook in A_2 niemand plipt en door alle leden van A_2 wordt geplipt. Dit controleren kost n handelingen, die allemaal parallel kunnen, en conjunctie van de uitkomst kan ook in constante span *op een CRCW PRAM*.
   (b) Na de twee deelaanroepen moet je nog lineair veel werk doen met constante span. De deelaanroepen kunnen volledig parallel. Dus je vindt voor het Work: $W(n) = 2{*}W(n/2){+}O(n)$. Je vindt voor de Span: $S(n) = S(n/2) + O(1)$.
   (c) Oplossing met de Master Theorem (benoem a, b en f) levert $W(n) = n \lg n$ en $S(n) = \lg n$.
   (d) De span is gelukkig logaritmisch, wat snel genoeg is om de methode efficient en optimaal te noemen. Helaas is er (omdat er een sequentieel lineair algoritme bestaat) sprake van een niet-triviale, namelijk logaritmische, overhead. Het algoritme is daarom *wel efficient*, maar *niet optimaal*.

   Beoordeling/Toelichting: Per deelvraag een punt tot een totaal van 4.

   A = Er is niet Altijd een plip-King, je moet dit echt wel echt checken!
   B = Bij (c) alleen 1 als je Beide vergelijkingen goed oplost, *en* het de goede vergelijkingen zijn!
   C = Als je van een EW (dus niet CW) PRAM uitgaat, kost conjunctie logaritmische span. ook kon je voor het combineren van de tests verwijzen naar de Reduce, die logaritmische span heeft. De spanvergelijking wordt dan $S(n) = S(n/2) + \lg n$ met oplossing $S(n) = \lg^2 n$. Het algoritme is nog steeds efficient maar niet optimaal.
   E = Kijk alleen of de deelKings Elkaar plippen; dit is niet genoeg, levert flase positives op.
   H = Je bent de Halve check vergeten (alleen of de King niemand plipt, of alleen dat hie door iedereen wordt geplipt). Omdat dit voor de complexiteit geen verschil maakt, geen aftrek.
   K = Redenering Klopt maar is gebaseerd op onjuiste conclusie van vorige deelvraag. Je *kon dus wel* deelpunten krijgen als je een fout maakte en daarmee goed verder werkte, maar meestal niet de volle punten. Je krijgt geen punten als je een onderdeel fout doet, en het volgende onderdeel weer fout doet maar "per ongeluk" op het juiste antwoord komt.
   L = Jouw work is Lager dan de tijd van het beste sequentiele algoritme (de overhead is significant kleiner dan 1). Dit kan helemaal niet! Want het sequentieel uitrollen van jouw algoritme zou dan een betere sequentiele oplossing geven.
   S = Niet gezien dat de checks parallel kunnen, dus in constante of log span, half aftrek.
   T = Je moet bij recursieve algoritmen de Mastertheorem Toepassen! De redenering "je doet

zoveel werk en er zijn zoveel recursienivo's" zou bij alle recurrente betrekkingen de uitkomst $f(n) \cdot \lg n$ geven, en dat is niet zo.
Z = King uit A_1 de Zelfde als uit A_2; kan niet, want de groepen overlappen elkaar niet.

Je krijgt een beter algoritme als je zoekt naar een *Plip-Prince*. Een Prince is een deelnemer, waarvan je weliswaar nog niet hebt vastgesteld dat hij King is, maar die de enige mogelijkheid is omdat je van alle anderen in de verzameling al hebt gezien dat ze iemand plippen, of geplipt worden. Als A een singleton is, lever de enige deelnemer op als Prince. Als A meer elementen heeft, splits in A_1 en A_2 en zoek Princes x_1 en x_2. Kijk hierna *alleen maar* of x_1, x_2 plipt. Zoja, dan is x_1 uitgesloten als King (want hij plipt x_2) en x_2 is Prince. Zonee, dan is x_2 uitgesloten (want x_1 plipt hem niet) dus x_1 is Prince. Dit recursieve algoritme geeft voor je hele verzameling A in logaritmische span en lineair work een Prince. Je moet dan aan het eind alleen checken (lineair work) of deze Prince inderdaad King is. De eindcheck kost constante span op een CW PRAM en logaritmisch op een EW PRAM. In beide gevallen levert dit logaritmische span voor het totaal, want je doet deze eindcheck maar 1 keer, en niet op elke laag van de recursie.