Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Dr. Thomas Worsch, Dr. Simon Gog
Demian Hespe, Yaroslav Akhremstev

5. Übungsblatt zu Algorithmen II im WS 2017/2018

http://algo2.iti.kit.edu/AlgorithmenII_WS17.php
{hespe,sanders,simon.gog,worsch,yaroslav.akhremtsev}@kit.edu

Musterlösungen

Weihnachtsblatt

Aufgabe 1 (*Pflichtaufgabe (*)*)

a) Machen Sie Ihren Übungsleitern ein schönes Weihnachtsgeschenk.

Musterlösung:

a) Kameras, Objektive, Ultrabooks, Pads, Smartphones, Fahrräder, ...

Aufgabe 2 (*Kleinaufgaben: Parallele Algorithmen*)

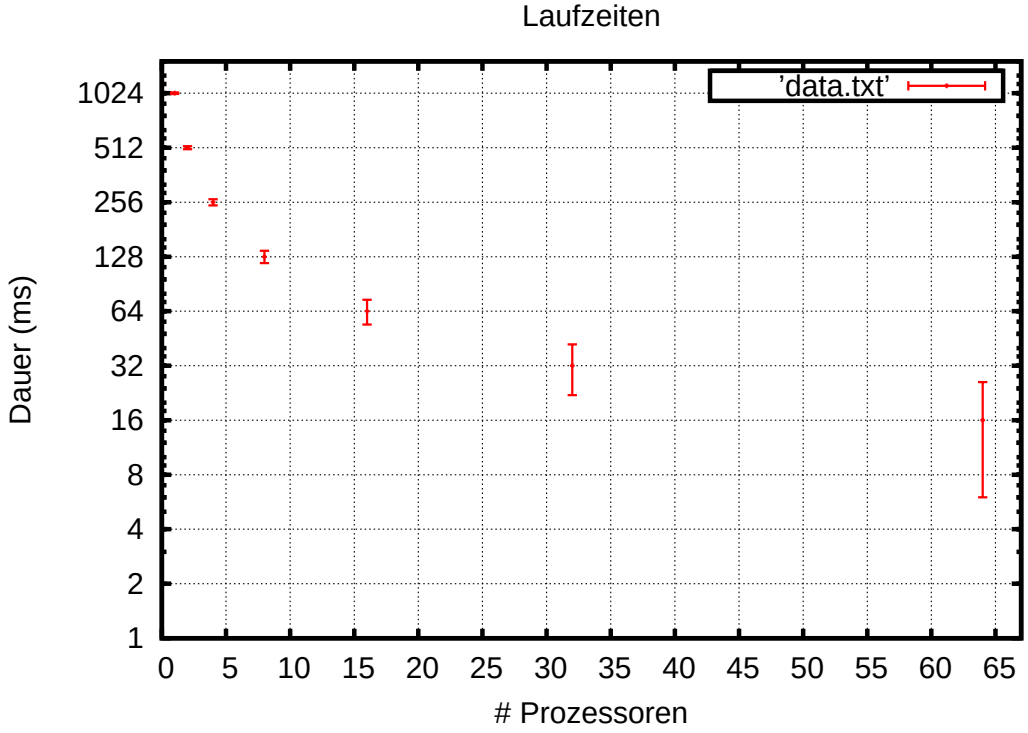
a) Gegeben sei ein paralleler vergleichsbasierter Sortieralgorithmus zum Sortieren von n Objekten auf p Prozessoren mit einer Laufzeit von

$$T(p) := \Theta(\frac{n^2 \log^2 n}{p^2}).$$

Geben Sie den absoluten *speed-up* und die *efficiency* an.

b) Wie muss in der vorherigen Teilaufgabe die Prozessorzahl p mit der Eingabegröße n wachsen, damit der absolute *speed-up* konstant bleibt?

c) Sie haben für einen parallelen Algorithmus folgende Laufzeiten bei unterschiedlicher Prozessorzahl gemessen. Was können Sie über die Skalierung dieses Algorithmus aussagen?



Musterlösung:

a) Die besten sequentiellen vergleichsbasierten Sortierverfahren benötigen $T_{seq} = \Theta(n \log n)$ Laufzeit. Damit ergibt sich für den absoluten *speed-up*

$$S(p) = \frac{T_{seq}}{T(p)} = \frac{\Theta\left(n \log n\right)}{\Theta\left(\frac{n^2 \cdot \log^2 n}{p^2}\right)} = \Theta\left(\frac{p^2}{n \log n}\right).$$

Mit der Definition der *efficiency* ergibt sich

$$E(p) = \frac{S(p)}{p} = \frac{\Theta\left(\frac{p^2}{n \log n}\right)}{p} = \Theta\left(\frac{p}{n \log n}\right).$$

b) Für einen konstanten *speed-up* unabhängig von der Prozessoranzahl muss gelten

$$S(p) = \Theta(\frac{p^2}{n \log n}) \stackrel{!}{=} \Theta(1)$$

und damit muss die Anzahl Prozessoren mit der Eingabegröße nach $p = \Theta(\sqrt{n \log n})$ wachsen.

c) Ignoriert man die Fehlerbalken, so weisst der Algorithmus eine Halbierung der Laufzeit bei Verdopplung der Prozessoren auf. Der relative *speed-up* ist also

$$S_{rel}(p) = \frac{T(1)}{T(p)} = p.$$

Der absolute *speed-up* skaliert auch linear mit p, über den genauen Faktor und über die Skalierung mit n kann man keine Aussage treffen. Die *efficiency* ist unabhängig von p.

Aufgabe 3 (*Entwurf+Analyse: CRCW und CREW Modelle*)

Gegeben sei ein Array $a[\cdot]$ im gemeinsamen Speicher, der n Zahlen hält.

a) Beschreiben Sie einen möglichst schnellen parallelen Algorithmus, der überprüft, ob eine der Zahlen durch 7 teilbar ist. Gehen Sie von $p = n$ Prozessoren und dem CRCW *common* Modell aus. Außerdem sei die Teilbarkeit in $O(1)$ prüfbar.

b) Wie ändert sich die Laufzeit, wenn Sie nur noch $p < n$ Prozessoren zur Verfügung haben?

c) Wieviel Zeit würde Ihr Algorithmus im CREW Modell benötigen (wieder $p = n$ Prozessoren)?

d) Geben Sie den absoluten *speed-up* und die *efficiency* für die vorherigen Teilaufgaben an.

e) Im Folgenden soll berechnet werden, wieviele der Zahlen in $a[\cdot]$ durch eine andere Zahl in $a[\cdot]$ (nicht durch sich selbst!) teilbar sind. Sie haben das CRCW Modell und $p = n^2$ Prozessoren zur Verfügung.

Musterlösung:

a) Die Variable zum Speichern des Resultats wird mit `result = false` initialisiert. Jeder Prozessor p_i überprüft für eine Zahl $a[i]$, ob diese durch 7 teilbar ist. Ist dies der Fall, setzt der Prozessor `result = true`. Ansonsten macht er nichts. Der Algorithmus läuft in $T(p) = O(1)$.

b) Jeder Prozessor muss sich seriell um n/p Speicherstellen kümmern. Ansonsten läuft der Algorithmus gleich ab und benötigt daher $T(p) = O(n/p)$ Zeit.

c) Im CREW Modell kann nicht mehr parallel auf eine Speicherstelle geschrieben werden. In einer einfachen Lösung würde für alle Prozessoren je ein Schreibzugriff auf `result` nacheinander ausgeführt werden. Dies würde $O(n)$ Zeit benötigen.

   Geschickter ist es, das Ergebnis für jede Zahl $a[i]$ in einem Array $b[\cdot]$ zu speichern - 0 für `false` und 1 für `true`. Dies geschieht in $O(1)$. Dann wird die Summe $\sum_{i=1}^{n} b[i]$ per Reduktion in $O(\log n)$ berechnet. Ist die Summe ungleich 0, so wird `result=true` gesetzt, sonst auf `false`. Die gesamte Laufzeit ist $T(p) = O(\log n) = O(\log p)$.

d) Der optimale sequentielle Algorithmus muss im schlimmsten Fall jede Zahl prüfen, benötigt also $T_{seq} = O(n)$. Damit ergibt sich für den absoluten *speed-up*

$$S(p) = \frac{T_{seq}}{T(p)} = O(n) = O(p) \text{ (a)}; \; O(p) \text{ (b)}; \; O(n/\log n) = O(p/\log p) \text{ (c)}.$$

   Mit der Definition von *efficiency* ergibt sich

$$E(p) = \frac{S(p)}{p} = O(1) \text{ (a)}; \; O(1) \text{ (b)}; \; O(1/\log p) \text{ (c)}.$$

e) Berechne Hilfsarray $b[\cdot]$. Es wird $b[i] =$ `true` gesetzt, wenn eine Zahl in $a[\cdot]$ von $a[i]$ geteilt wird, die nicht gleich $a[i]$ ist. Ansonsten wird $b[i] =$ `false` gesetzt. Dies geschieht analog zur ersten Teilaufgabe in $O(1)$ mit n Prozessoren für jedes i. Mit n^2 Prozessoren und n Einträgen in $b[\cdot]$ können alle Einträge in $O(1)$ berechnet werden. Anschließend wird über die Elemente in $b[\cdot]$ summiert, mit n Prozessoren und Reduktionsschema ist dies in $O(\log n)$ möglich. Die gesamte Laufzeit ist also $T(p) = O(\log n) = O(\log \sqrt{p})$.

Aufgabe 4 (*Entwurf+Analyse:* `findif`-*Anweisung*)

Gegeben sei ein Array $a[\cdot]$ im verteilten Speicher der n Objekte hält. Gesucht ist ein Algorithmus, der eine parallele `findif` Anweisung auf $a[\cdot]$ ausführt. Die Anweisung sortiert die Elemente von $a[\cdot]$ anhand eines Prädikats $pred(\cdot)$, so dass Elemente, die das Prädikat erfüllen, vorne stehen. Die relative Ordnung der Elemente untereinander soll dabei erhalten bleiben.

Bsp.: `findif( {1,4,9,7,3},` *is_bigger_than_3* `) = {4,9,7,1,3}`

a) Beschreiben Sie einen Algorithmus, der eine parallele `findif` Anweisung auf $a[\cdot]$ möglichst schnell ausführt. Sie haben $p = n$ Prozessoren zur Verfügung.

b) Untersuchen Sie die Laufzeit der Anweisung für den Fall, dass $p = n$ Prozessoren zur Verfügung stehen und das Prädikat in $T(n) = O(1),\ O(\log n)$ bzw. $O(n)$ ausgewertet werden kann.

c) Wie verhalten sich die Laufzeiten, wenn Sie nur noch $p < n$ Prozessoren zur Verfügung haben?

Musterlösung:

a) Prozessor p_i prüft Bedingung $pred(a[i])$ und schreibt das Resultat als 0 (falsch) oder 1 (wahr) in ein neues Array $s[\cdot]$ an Stelle i. Anschließend wird die Präfixsumme über $s[\cdot]$ gebildet. Aus diesen Werten kann jeder Prozessor p_i die neue Position für sein Datenelement $a[i]$ ableiten:

- $a[i] \rightarrow a[s[i] - 1]$, wenn $a[i]$ das Prädikat erfüllt,
- $a[i] \rightarrow a[s[n-1] + i - s[i]]$, wenn $a[i]$ das Prädikat nicht erfüllt.

Prozessoren werden von 0 durchnummeriert.

b) Die Ausführungszeit beträgt $O(T(n))$ für die Berechnung von $s[\cdot]$, $O(\log n)$ für die Berechnung der Präfixsumme und $O(1)$ für die Umsortierung. Gesamt ergeben sich die Laufzeiten in Abhängigkeit von $T(n)$ zu $O(\log n)$, $O(\log n)$ bzw. $O(n)$, wobei $p = n$ gilt.

c) Stehen weniger als n Prozessoren zur Verfügung, muss man die Arbeit für jeweils n/p Objekte von einem Prozessor ausführen lassen. Es ergeben sich folgende Laufzeiten für die drei Schritte $O(n/p \cdot T(n))$, $O(n/p + \log p)$ und $O(n/p)$. Insgesamt ergeben sich die Laufzeiten wieder in Abhängigkeit von $T(n)$ zu $O(n/p + \log p)$, $O(n/p \cdot \log p)$ bzw. $O(n^2/p + \log p)$.

Aufgabe 5 (*Entwurf+Analyse: Assoziative Operationen*)

Gegeben sei ein Array A im gemeinsamen Speicher bestehend aus n Objekten vom Typ X. Auf den Objekten sei eine Operator $\odot$ definiert. Es sei nach dem Ergebnis von $\odot_{i=1}^{n} a_i$ gesucht.

a) Sei $X = \mathbb{R}^2$ und der Operator definiert als

$$(x_1, x_2) \odot (y_1, y_2) := (x_1 y_1, x_2 + y_2)$$

Zeigen Sie, dass der Operator $\odot$ assoziativ ist.

b) Beschreiben Sie einen schnellen parallelen Algorithmus, der $\odot_{i=1}^{n} a_i$ berechnet und geben Sie dessen Laufzeit $T(n, p)$ an.

c) Nun sei $\odot$ wie folgt definiert: X beschreibe die Menge an möglichen Zeichenketten über dem Alphabet $\{(,)\}$. Die Operation $x \odot y$ verknüpfe beide Zeichenketten und schiebe alle öffnenden Klammern nach links, alle schließenden Klammern nach rechts (Bsp.: $()(() \odot (()) = ((((()))))$. Gehen sie davon aus, dass zu Begin die länge der Zeichenketten konstant ist.

Können Sie den selben Lösungsansatz wie in der vorherigen Teilaufgabe verwenden? Falls nein, geben Sie einen neuen parallelen Algorithmus an. Wie lange dauert die Ausführung?

Musterlösung:

a) Der Operator ist offensichtlich assoziativ, da beide Koordinaten unabhängig voneinander sind und die ausgeführte Addition bzw. Multiplikation auf reelen Zahlen assoziativ ist.

b) Der Operator $\odot$ ist assoziativ und in konstanter Zeit ausführbar. Daher kann $\odot_{i=1}^{n} a_i$ mit Hilfe des Reduktionsschemas in $T(n, p) = O(n/p + \log p)$ berechnet werden. Für $p = n$ liegt die Ausführungszeit in $T(n, p) = O(\log n)$.

c) Auch hier ist $\odot$ assoziativ, benötigt aber Zeit proportional zur Länge beider Operanden (ein Scan über beide Operanden liefert die Anzahl öffnender und schließender Klammern, die anschließend geschrieben werden können). Das Reduktionsschema ist weiterhin anwendbar, die Laufzeit erhöht sich allerdings insgesamt zu $T(n, p) = O\left(\sum_{i=0}^{\log p} \left(2^i \frac{n}{p} + 1\right)\right) = O(p \cdot n/p + \log p)$ bzw. zu $T(n, p) = O(n)$ für $p = n$.

Ein schnellerer Lösungsansatz wendet `findif` auf A an. Das Prädikat prüft hierbei, ob das Element eine öffnende Klammer ist. In diesem Fall ist die Laufzeit $T(n, p) = O(\frac{n}{p} + \log p)$ bzw. $T(n, p) = O(\log n)$ für $p = n$.

Aufgabe 6 (*Dijkstras Algorithmus auf dichten Graphen*)

Zeigen sie, dass Dijkstras Algorithmus mit einem binaren Heap eine erwartete Laufzeit in $O(m)$ hat, wenn $m \in \Omega(n \log n \log \log n)$.

Musterlösung:
Betrachte

$$\frac{m + n \log \frac{m}{n} \log n}{m} = 1 + n \log n \frac{\log \frac{m}{n}}{m}.$$

Wir zeigen, dass diese Verhältnis in $O(1)$ ist.
Wir wissen, dass $m = \Omega(n \cdot \log n \cdot \log \log n)$, damit $m = n \cdot \log n \cdot g(n)$, where $g(n) = \Omega(\log \log n)$.

$$1 + n \log n \frac{\log \frac{m}{n}}{m} = 1 + \frac{\log \log n + \log g(n)}{g(n)} = O(1)$$