FGI-2 – Formale Grundlagen der Informatik II

Modellierung und Analyse von Informatiksystemen

Aufgabenblatt 9 : PRAM und parallele Algorithmen

Präsenzaufgabe 9.1: Betrachte folgenden (hochsprachlich formulierten) Algorithmus, der auf einer RAM ausgeführt werden soll:

```
function addall(n) is
    v := 0
    for i := 1 to n do
        v := i + v
    endfor
    return v
```

1. Berechne addall(n) für $n = 0, 1, 2, 3$.
2. Gib eine geschlossenen Ausdruck für addall(n) an.
3. Schätzen Sie die Zeit- und die Platzkomplexität im uniformen und logarithmischen Maß ab.

Präsenzaufgabe 9.2: Der folgende Algorithmus berechnet die Funktion f auf den Prozessoren $P_0, \ldots, P_{n-1}$:

$$f(x_0, \ldots, x_{n-1}) = \prod_{i=0}^{n-1} x_i = x_0 \cdot x_1 \cdots x_{n-1}, \qquad n = 2^m$$

Die Argumente $x_0, \ldots, x_{n-1}$ werden in den globalen Registern $M(0), \ldots, M(n-1)$ übergeben. Der Output wird in $M(0)$ abgelegt.

Algorithmus für den Prozessor $P_i, 0 \leq i \leq n-1$:

```
for t := 1 to m do
    if i ist ein Vielfaches von 2^t then
        M(i) ← M(i) * M(i + 2^(t-1))
    endif
endfor
```

1. Zeige: Die uniforme Zeitkomplexität liegt in $O(\log n)$.
2. Zeige: Die Prozessorkomplexität liegt in $O(n)$.
3. Liegt der Algorithmus in $CRCW_+TimeProc(POL, POL) := \bigcup_{0 \leq m,n} CRCW_+TimeProc(n^m, n^n)$?
4. Ist der Algorithmus optimal?
5. Ist die Prozessor/Zeit-Komplexität asymptotisch gleich der sequentiellen Komplexität?
   Wenn nein: Können wir mit Brent's principle einen solchen Algorithmus bekommen?
6. Bonusmaterial: Was verändert sich für das uniforme Maß an der Komplexitätsbetrachtungen, wenn wir annehmen, dass in den Registern keine Werte größer als 2^n liegen? Und für das logarithmische Maß?

Übungsaufgabe 9.3: Sei $n = [b_{k-1} \dots b_1 b_0]_2 = \sum_{i=0}^{k-1} b_i 2^i$ mit $b_{k-1} = 1$. Der folgende Algorithmus berechnet die Potenz $r = a^n$ durch fortgesetztes Quadrieren.

von
6

function $\exp(a, n)$ **is**
1 $r := 1$
2 $p := a$ // $a = a^{2^0}$
3 **for** $i = 0$ **to** $k - 1$ **do**
4 **if** $odd(n)$ **then** $r := r \cdot p$ // $r := r \cdot \left(a^{2^i}\right)^{b_i}$
5 $n := n$ **div** 2
6 $p := p \cdot p$
7 **endfor**
8 **return** r

1. Das Verfahren basiert auf der folgenden Beziehung:
   Für $n = [b_{k-1} \dots b_1 b_0]_2$ ergibt sich:

$$r = a^n = a^{\sum_{i=0}^{k-1} b_i 2^i} = \prod_{i=0}^{k-1} a^{b_i 2^i} = \prod_{i=0}^{k-1} \left(a^{2^i}\right)^{b_i}$$

   Erläutere, wie dies in den Algorithmus einfließt!

2. Berechnen Sie die Zeitkomplexität im uniformen und im logarithmischen Maß. Nehmen Sie dabei Bezug auf die Zeilennummern des Algortihmus.

3. Berechnen Sie die Platzkomplexität im uniformen und im logarithmischen Maß.

Übungsaufgabe 9.4: Zeigen Sie die aus der Vorlesung bekannte Beziehung (Satz 4.21), dass Polynomialzeit für PRAMs genauso mächtig ist, wie Polynomialplatz für TMs oder RAMs:

von
6

$$\text{PRAM-Time}(POL) = \mathcal{PSPACE}$$

Dabei war PRAM-Time(t) die Klasse aller PRAMs mit der Zeitkomplexität $t(n)$ und der Prozessorkomplexität $proc(n) = 2^{t(n)}$:

$$\text{PRAM-Time}(t(n)) := CRCW_+ \; TimeProc(t(n), 2^{t(n)})$$

Hinweis. Sie können folgende Beziehungen verwenden:

1. Jede durch $s(n) \geq \log(n)$ platzbeschränkte TM kann durch eine PRAM mit $O(2^s)$ Prozessoren in der Zeit $O(s)$ simuliert werden:

$$DSpace(s(n)) \subseteq \text{PRAM-Time}(O(s(n)))$$

2. Umgekehrt kann man jede PRAM in PRAM-Time($t(n)$) auf einer TM simulieren kann, die $O(t^2(n))$ Platz benötigt:

$$\text{PRAM-Time}(t(n)) \subseteq NSpace(t^2(n))$$