Übungen zur Vorlesung "Architektur und Programmierung von Grafik- und Koprozessoren"
Übungsblatt 2

Sommersemester 2019

2 Performanz und Parallele Algorithmen

Aufgabe 2.1

Wir betrachten den nachfolgenden Ausschnitt des Befehlssatzes einer hypothetischen CPU:

Instruktion	**Beschreibung**	**Latenz**
`ADD $R1` C	Addiere den konstanten Wert C auf den Inhalt von Register `$R1`	4 TZ
`LD $R1` $[S]$	Lade den Inhalt an Speicherstelle $[S]$ in Register `$R1`	10 TZ
`ST` $[S]$ `$R1`	Speichere den Inhalt von Register `$R1` an Speicherstelle $[S]$	10 TZ
`JNZ $R1 label1`	Falls Inhalt von Register `$R1` $\neq 0$, springe zu Label `label1`	1 TZ

Bestimmen Sie die Performanz des nachfolgenden Programms, das auf einer CPU mit 100 Hz Taktfrequenz ausgeführt wird. Nehmen Sie an, dass der Inhalt der Register `$R1` und `$R2` zunächst 0 beträgt und dass initial im Speicher an der Adresse `0x80` der Wert 12 steht.

```
LD $R2 0x80
L1:
ADD $R1 3
ADD $R2 -4
JNZ $R2 L1
ST 0x80 $R1
```

Aufgabe 2.2

Sie haben als Aufgabe erhalten, ein C Programm für Multi-Core Prozessoren zu optimieren. Die meiste Programmausführungszeit wird für die folgende Funktion benötigt.

```
void func(double* x, double* y, double* a, double* b, int N)
{
    assert(N > 2);

    int i;
    for (i = 2; i < N; ++i)
        x[i] = x[i-1] + x[i-2];

    for (i = 0; i < N; ++i)
        y[i] = x[i] * a[i] / b[i];
}
```

Sie wissen, dass Speicherbereiche, auf die die an `func()` übergebenen Pointer zeigen, niemals überlappen. Mit diesem Wissen und nach sorgfältiger Analyse kommen Sie zu dem Ergebnis, dass sich die erste Schleife wegen einer *Inter Iterations*-Datenabhängigkeit nicht parallelisieren lässt. Die zweite Schleife lässt sich jedoch mit einer einfachen OpenMP Deklarative parallelisieren, sodass mehrere Threads auf einem Multi-Core Prozessor nebenläufig die Schleifeniterationen ausführen können. Sie haben das Programm mit einem Profiler und realistischen Eingabedaten analysiert. Ihre Analyse hat ergeben, dass bei serieller Programmausführung ebensoviel Zeit in der ersten wie in der zweiten Schleife verbracht wird.

Nehmen Sie vereinfachend an, dass eine Parallelisierung der zweiten Schleife *perfekt skaliert*: wenn Sie die Schleife mit N Prozessorkernen abarbeiten, erwarten Sie einen Speedup von N. Welche obere Schranke ergibt sich unter diesen Umständen für den *theoretischen* Speedup für die Funktion `func()` für jeweils $N = 2, N = 4$ und $N = 8$ Prozessorkerne?

Aufgabe 2.3

In der Vorlesung haben wir den Algorithmus ReduzierePRAM kennen gelernt. Hierbei sind wir vereinfachend von P = n ausgegangen. Entwerfen Sie nun einen parallelen Algorithmus ohne diese vereinfachende Annahme. Orientieren Sie sich dazu auch am Beispiel des SaxpyPRAM Algorithmus aus der Vorlesung.

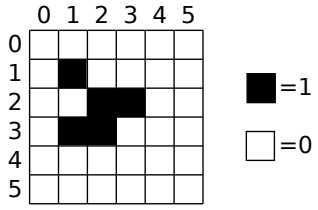
Aufgabe 2.4

Zelluläre Automaten sind gitterbasierte Modelle. Gitterzellen sind definiert durch ihren *Zustand*, der sich aufgrund von Nachbarzuständen im Verlauf der Zeit verändert. Zustandsveränderungen werden aufgrund eines Regelwerks beschrieben.

Gegeben sei ein Zellulärer Automat auf einem rechteckigen uniformen Gitter der Breite $N \in \mathbb{N}$ und Höhe $M \in \mathbb{N}$: jede Zelle ist eindeutig durch ihre Position $(x, y) \in [0, N) \times [0, M)$, $x, y \in \mathbb{N}$ bestimmt und speichert Zustände $s(x, y, t) \in \{0, 1\}$ für jeden Zeitschritt $t \in \mathbb{N}$. Bezeichne $\mathfrak{S}(x, y, t)$ die Anzahl der Nachbarzellen der Zelle an Position (x, y) zum Zeitpunkt t, für die $s = 1$ gilt. Um die $s(x, y, t)$ für alle Gitterzellen zu ermitteln, wendet man die nachfolgenden Regeln auf die Gitterzellen an:

$$s(x, y, t) = \begin{cases} 1 \text{ falls } (s(x, y, t-1) = 1 \wedge \mathfrak{S}(x, y, t-1) = 2) \vee \mathfrak{S}(x, y, t-1) = 3 \\ 0 \text{ sonst.} \end{cases}$$

a.) Entwickeln Sie die nächsten vier Zeitschritte für das folgende Gitter.



b.) Formulieren Sie einen PRAM Algorithmus, der in einer Schleife über unendlich viele Zeitschritte den Zustand des Zellulären Automaten berechnet. Ihnen stehen dabei $P = S \times T$ Prozessoren zur Verfügung, wobei $N \bmod S := 0$ und $M \bmod T := 0$ gelte. Nehmen Sie an, dass die Randzellen immer den Wert 0 haben. Verwalten Sie den Zustand für jeweils zwei aufeinanderfolgende Zeitschritte in zwei Gittern G_1 und G_2. In G_1 verwalten Sie immer den Zustand aus dem vorhergehenden Zeitschritt und speichern den neuen Zustand in G_2.

Welches ist das am wenigsten restriktive PRAM Speichermodell, auf dem der Algorithmus ausgeführt werden kann?

c.) Formulieren Sie den Algorithmus aus **b.)** mit Hilfe des *Arbeit vs. Zeit Paradigmas* für unendlich viele Prozessoren um.

Das Übungsblatt wird am 25.04.2019 besprochen.