Übungen zur Vorlesung "Architektur und Programmierung von Grafik- und Koprozessoren"
Übungsblatt 3

Sommersemester 2018

3 Parallele Algorithmen

Aufgabe 3.1

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? (5 Punkte)

Aufgabe 3.2

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. (5 Punkte)

Aufgabe 3.3

a.) Implementieren Sie den seriellen Algorithmus REDUZIERE() aus der Vorlesung mit C++11. Verwenden Sie als Vorlage die Datei `reduce.cpp`. (2 Punkte)

b.) Implementieren Sie den EREW Algorithmus REDUZIEREWORKTIME() aus der Vorlesung mit C++11. Dazu können Sie das *parallele for* Konstrukt aus der Datei `parallel_for.h` verwenden, oder alternativ ein paralleles for Konstrukt (und nur ein solches!) aus einer freien Bibliothek wie *OpenMP* oder Intels *Threading Building Blocks*. Sie dürfen annehmen, dass $n = 2^k$ Elemente reduziert werden. (8 Punkte)

Aufgabe 3.4

Die *Präfixsumme* der Folge $(a_n) = a_1, a_2, ..., a_n$ ist definiert als

$$\begin{aligned} p_1 &= a_1 \\ p_2 &= a_1 + a_2 \\ p_3 &= a_1 + a_2 + a_3 \\ p_n &= a_1 + ... + a_n \end{aligned}$$

a.) Formulieren Sie einen *seriellen* $O(n)$ Algorithmus, der als Eingabe ein Array mit n Elementen erhält und für diese die Präfixsumme berechnet. Das Ergebnis darf im selben Array wie die Eingabe gespeichert werden. (2 Punkte)

b.) Mit dem parallelen Algorithmus POINTERJUMPING aus der Vorlesung lassen sich ebenfalls Präfixsummen bestimmen. Formulieren Sie den Algorithmus entsprechend um. Wie in der Vorlesung erhält der Algorithmus als Eingabe ein Array, das für jeden Knoten in einem Wald wurzelgerichteter Bäume dessen Vorgänger speichert. Ihr Algorithmus erhält außerdem ein Array, das jedem wurzelgerichteten Baum eine Folge (a_n) zuordnet. Der Algorithmus berechnet die Präfixsummen dieser Folgen. Lautet die Eingabe etwa

	0	1	2	3	4	5	6	7	8	9
S	0	0	1	2	3	5	5	6	7	8
(a_n)	1	1	1	1	1	2	2	2	2	2

dann lautet die Ausgabe:

	0	1	2	3	4	5	6	7	8	9
S	0	0	0	0	0	5	5	5	5	5
(p_n)	1	1	2	3	4	2	2	4	6	8

Ist diese Methode zur Bestimmung der Präfixsumme kosteneffizient? (3 Punkte)

c.) Implementieren Sie den parallelen Algorithmus zur Bestimmung der Präfixsumme aus Aufgabenteil b.). Verwenden Sie dazu die Vorlage (`prefix_sum.cpp`) und wie in Aufgabe 3.3 die beigefügte parallele for Schleife oder eine Alternative aus einer freien Bibliothek. Denken Sie daran, dass POINTERJUMPING ein *CREW* Algorithmus ist! (10 Punkte)

Abgabe bitte bis zum 16.05.2018, 22:00h in Ilias.