Abgabe zu Blatt 5, Aufgabe 1

Moritz Nöltner
Universität Heidelberg, ZITI

Index Terms—Pallel Programming, Relaxation

1 Erklärung der Einteilung

Ich nahm an, dass die größten Geschwindigkeitseinbusen von ungleich verteilter Arbeit und Cache-Fehlgriffen zu erwarten wären. Um also einen großen Durchsatz zu erzielen müssen diese beiden Fehlerquellen ausgeschalten werden. Gleichzeitig sollte die Aufteilung der Arbeit selbst nur geringen Aufwand bereiten. Daher entschied ich, die Arbeit ineinander verzahnt zeilenweise aufzuteilen. So benötigt ein einzelner Thread 3 Zeilen der alten Matrix sowie eine Zeile der neuen Matrix im Cache, für jeden weiteren Thread werden jedoch nur 2 weitere Zeilen (je eine von der alten und der neuen Matrix) im Cache benötigt. Des weiteren ist der Zugriff auf die Matrixelemente leicht vorhersehbar und sollte dem Cachealgorithmus so eine realistische Chance geben, die richtigen Zeilen im Cache vorzuhalten. Weiterhin habe ich die Threads, um den Aufwand der Threaderstellung zu sparen, über die gesammte Lebensdauer des Relaxation-Objekts am Leben gehalten.

2 Race Conditions

Die gewählte Einteilung vermeidet durch Konstruktion Race-Conditions in der Berechnungsphase, da jeder Thread auf einer eigenen Teilmenge der Matrixzeilen operiert. Lediglich zu Beginn und Ende der parallelen Berechnungsphase müssen die Threads syncronisiert werden, hierzu wurden Barrieren verwendet. Vor Beginn der Berechnung setzt der Hauptthread in je einer Klassenvariable die auszuführende Operation (Entweder füllen mit dem Initialwert, oder Berechnen der Werte eines neuen Zeitschritts) und die Zielmatrix und die anderen Threads verwenden diese Information um ihre Berechnung entsprechend auszuführen. Nach Ende der Berechnung werden die Prozesse synchronisiert, die Matrix mit den neuen Werten wird mit der Matrix mit den alten Werten vertauscht und die Matrix mit den alten Werten gelöscht. Das Relaxation-Objekt und seine Threads sind nun bereit für den nächsten Berechnungsschritt.

Zusätzliche Threads	# CPU-Ticks pro Durchlauf
Sequentiell	11884205445
1 Thread	17596180870
2 Threads	10678853046
3 Threads	10218917211
4 Threads	10112951190
5 Threads	10184051488
6 Threads	10250735061
7 Threads	10302345518
8 Threads	11433348153
9 Threads	11573821119
10 Threads	11758987826
11 Threads	12027272974
12 Threads	12148201196
13 Threads	12210375934
14 Threads	12241583642
15 Threads	12322422359
16 Threads	12430100829

Abbildung 1. Leistungsvergleich verschiedener Stufen der Nebenläufigkeit: N=1024, Simuliert wurden jeweile 10 Durchläufe zu 1000 Iterationen

3 Speedup

Der Geschwindigkeitszuwachs fällt entgegen der Erwartung deutlich geringer aus. Mögliche Erklärungsansätze wären:

- Einzelne Threads brauchen länger und bremsen damit die Gruppe aus. Dieses Problem würde sich noch verstärken, weil der hinterherhinkende Thread möglicherweise nicht mehr auf schon geladene Cachelines zugreifen kann, wenn die anderen Threads schon andere Zeilen in den Cache geladen haben.
- Um die begrenzten Parameter für die Threadfunktionen zu umgehen wird auf die Matrix durch eine weitere Indirektion zugegriffen. Möglicherweise bremst dies die Ausführung aus.

Aus Gründen der beschränkten Zeit sowie fehlender Diagnosemittel konnte diesen Ansätzen jedoch nicht nachgegangen werden.