anas.toma [☺] tu-dortmund.de

Übung zur Vorlesung
Rechnerarchitektur
Sommersemester 2017

Übungsblatt 4

(10 Punkte)

Besprechung ab Dienstag, 23. Mai 2017

4.1 Parallelisierung von Schleifen mit OpenMP (3 Punkte)

Das Programm Blatt4_1.c im GIT-Repository enthält mehrere for-Schleifen, welchen Sie mit Hilfe des Reduction-Statements parallelisieren sollen. Dabei darf abgesehen von den Pragmas kein Code hinzugefügt werden. Was welche for-Schleife genau bewirken soll, wird in den dazugehörigen Kommentaren definiert.
Bearbeiten Sie zusätzlich die folgenden Aufgaben:

- Lösen Sie nun die gleichen Probleme in Blatt4_1.c mit dem 'atomic pragma', also ohne Reduction-Statement. Diesmal dürfen Sie Code hinzufügen.
- Welche der beiden Lösungen ist performanter?
- Wie ändert sich die Performanz durch das Ändern der Anzahl der Threads?

4.2 Parallelisierung von Conways Spiel des Lebens (Game of Life) mit OpenMP (7 Punkte)

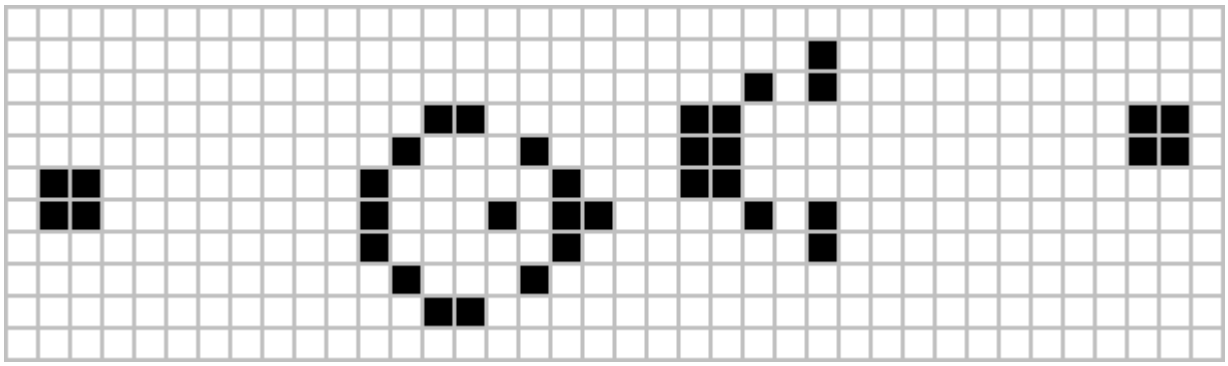

Abbildung 1: Spiel des Lebens Beispiel - Gun Slider Muster [www.wikipedia.org]

Das Spielfeld besteht aus $v \times h$ Elementen, wie in Abbildung 1 zu sehen ist. Jede Zelle heißt dabei lebendig wenn ihr Wert 1 (schwarzes Feld) ist, sonst tot (Wert 0, weißes Feld). Die Belegung des Spielfelds ändert sich iterativ, d.h. beginnend von einer Startbelegung wird in einer Iteration das gesamte Feld aktualisiert. Nach der Aktualisierung ist die aktuelle Iteration abgeschlossen und der Vorgang wiederholt sich bis zum Erreichen einer festgelegten Grenze. Es gibt Regeln, welche beschreiben wie sich die Belegung einer Zelle in Abhängigkeit von ihren direkten Nachbarzellen verändert.

Aktualisierungsregeln einer Zelle:

- Eine tote Zelle mit genau drei lebenden Nachbarn wird in der folgenden Iteration neu geboren.
- Lebende Zellen mit weniger als zwei lebenden Nachbarn sterben in der folgenden Iteration an Einsamkeit.
- Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt in der folgenden Iteration am Leben.
- Lebende Zellen mit mehr als drei lebenden Nachbarn sterben in der folgenden Iteration an Überbevölkerung.

Analysieren Sie den Sourcecode von Game of Life im GIT-Repository (“Blatt4_2.c” oder “Blatt4_2.cpp”).
Optional: Implementieren Sie Zuhause Ihre eigene Version vom Spiel des Lebens in C oder C++.

a. Stellen Sie sicher, dass jedes parallelisierte Programm semantisch korrekt ist und das selbe Ergebnis wie die sequentielle Version liefert.

b. Welche Programmabschnitte können parallelisiert werden? Implementieren Sie diese Parallelisierung.

c. Gibt es kritische Blöcke, die atomar ausgeführt werden müssen? Falls Sie zu dem Schluss kommen, dass es solche gibt, implementieren Sie diese Parallelisierung und analysieren Sie, ob es zu einer Verbesserung führt.

d. Wie viele parallele Threads führen zu einem besseren Ergebnis für das Beispiel RA und wie viele für das Beispiel BIG (RA und BIG sind zwei Beispiele in der Datei “muster.txt” im GIT-Repository).

e. Wenn Sie mit Ihrem Ergebnis zufrieden sind, tragen Sie Ihre Top 3 Zeiten, welche für BIG mit perf stat <Executable> gemessen wurden, in die Liste auf der Tafel ein. Beachten Sie, dass Konsolenausgaben sehr zeitintensiv sein können. Um einen unverfälschten Durchlauf zu messen, können Sie die Ausgaben auch deaktivieren.

Allgemeine Hinweise: Die Übungstermine und weitere Informationen finden Sie unter http://ls12-www.cs.tu-dortmund.de/daes/de/lehre/lehrveranstaltungen/sommersemester-2017/rechnerarchitektur-ss17.html. Die Übungsblätter werden in den Übungen bearbeitet und besprochen. Eine Abgabe vorher ist also nicht nötig, dennoch sollte man sich im Voraus mit den Übungen vertraut machen. Für die Teilnahme an der Klausur nach BPO 2013 / der Fachprüfung nach DPO 2001 ist der Übungsschein *nicht* erforderlich.