TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR TECHNISCHE INFORMATIK
PROFESSUR FÜR RECHNERARCHITEKTUR
PROF. DR. WOLFGANG E. NAGEL

Komplexpraktikum Paralleles Rechnen

Daniel Körsten

Dresden, 14. Oktober 2021

Inhaltsverzeichnis

1 Aufgabe B

1.1 Beschreibung

In dieser Aufgabe wird Thread-parallele Ausführung von `Conways Game-of-Life` durchgeführt. Conways Game-of-Life kann man sich als $n \times m$ Matrix vorstellen, bei der in jedem Berechnungsschritt die nächste Generation berechnet wird. Die Spielregeln lassen sich hier nachlesen.

1.2 Vorbereitung des Codes für eine parallele Ausführung

Für die Berechnung der nächsten Generation muss nun jede Zelle einzeln betrachtet werden und ihr Zustand in der nächsten Generation gemäß den Spielregeln berechnet werden. Ein möglicher Ansatz ist über jede Zeile und anschließend jede Spalte zu iterieren. Realisieren lässt sich das über zwei geschachtelte `for`-Schleifen. Das Ergebnis dieser Berechnung muss in einer zweiten Matrix gespeichert werden um die Berechnungen der Nachbarzellen nicht zu verfälschen.
Dieser Ansatz bietet den Vorteil, dass er mit OpenMP relativ einfach parallelisiert werden kann, denn die Berechnung jeder einzelnen Zelle ist unabhängig von den Berechnungen anderer Zellen.
Besonderes Augenmerk muss man jedoch auf die Kanten und Ecken legen. Diese sollen, laut Aufgabenstellung, mit `periodic boundary conditions` implementiert werden. Jedoch kann auch hier OpenMP zur Parallelisierung der Kanten verwendet werden.

1.3 Ein und Ausgabe

Da die Messung später in verschiedenen Feldgrößen durchgeführt wird, habe ich mich für den Einsatz von `getopt` entschieden. Es ermöglicht die Anzahl der Schleifendurchläufe, die Feldgröße und eine optionale Fortschrittsanzeige über Argumente beim Programmstart einzustellen. Ebenso lassen sich Threadanzahl und OpenMP Schedulingverfahren einstellen.

1.4 Probleme und ihre Lösung

Die Erzeugung des 2D-Arrays soll dynamisch erfolgen, damit man, ohne Anpassung des Programmcodes, die Feldgröße festlegen kann. Gemäß der Aufgabenstellung umfasst das größte Feld 32768×32768 Zellen. Damit ist das Array zu groß für den Stack des Programms. Mein Lösungsansatz war die Allokation von Speicher mittels `malloc` und `double pointers`: Ein erstes Array wurde mit Pointern gefüllt, die jeweils wiederum auf die einzelnen Zeilen verweisen, welche ebenfalls mit `malloc` allokiert wurden. Später habe ich die `double pointers` durch ein einzelnes `malloc` ersetzt, indem die Anzahl von Spalten als Offset dient, um sich im 2D-Array zu bewegen. Dadurch vermeidet man einen Speicherzugriff und folglich eine Adressübersetzung bei jeder Datenabfrage und -manipulation.
Nach der Parallelisierung mit OpenMP änderte sich jedoch nichts an der Ausführungszeit, was eindeutig den Erwartungen widersprach. Durch Experimentieren fand ich heraus, dass die Parallelisierung der Funktion, welche das Array mit Zufallswerten füllt, die Performance massiv beeinflusst. Die Funktion `rand` ist nicht für die parallele Ausführung geeignet. Deshalb habe ich mich hier einfach für die sequentielle Ausführung entschieden.

1.5 Optimierung des Codes

Eine Möglichkeit zur Einsparung von Rechenressourcen ist, den Speicherverbrauch des Programms zu reduzieren. So verwendete ich für die Zellen der Felder den Datentyp `u_int8_t` statt `int`. Dadurch reduziert sich der Speicherverbrauch jeder Zelle von 4 auf 1 Byte. Bei einem Feld der Größe 32768×32768 entspricht dies einer stattlichen Einsparung von über 3 GiB.
Die Verwendung von von bitweisen Operatoren in den Berechnung bringen ebenfalls einen geringen Geschwindigkeitsvorteil, da bitweise Operationen in der Schaltungstechnik die Einfachsten darstellen. Alle höheren Operationen lassen sich auf sie zurückführen.