Message Passing Programmierung

Projektaufgabe 1

Rechteckmustererkennung

Andrej Lisnitzki und Max Winkler
Hochschule für Technik, Wirtschaft und Kultur Leipzig
Fakultät Informatik, Mathematik und Naturwissenschaften

Professor: Herr Prof. Dr.-Ing. Axel Schneider

Leipzig, 27. Januar 2017

Inhaltsverzeichnis

1 Programmierumgebung.....1
2 Aufbau des Programms.....1
2.1 Aufrufparameter.....1
2.2 Beschreibung des Rechteckmustererkennung - Algorithmus.....3
2.2.1 Beschreibung des Rechteckfindung - Algorithmus.....4
2.2.2 Beschreibung des Algorithmus über das Vorliegen des Rechtecks.....5

1 Programmierumgebung

Das Projekt wurde mit der Programmiersprache C und C++ geschrieben, wobei für die Realisierung der parallelen Berechnungen wurden C-Prozeduraufrufe der MPI - Bibliothek benutzt. C++ wurde hauptsächlich für komfortablen Umgang mit Zeichenketten (Strings) und Dateioperationen benutzt.

Da im Programm C++ Code benutzt wurde, soll das Programm mit dem **mpicxx** - Compiler kompiliert. Es werden auch Funktionen aus dem **C++11** - Standard benutzt, deswegen beim Kompilieren soll entsprechendes Flag gesetzt werden. Vollständiger Kompilierbefehl sieht wie folgt aus:

```
mpicxx -std=c++11 main.cpp -o project_c
```

2 Aufbau des Programms

Das Programm ist logisch auf zwei Abschnitte unterteilt:

1. Auswerten der Aufrufparameter und
2. Ausführung der gewählten Option

Diese werden im Folgendem detaillierter Beschrieben.

2.1 Aufrufparameter

Dieses Projekt wurde auf einem Rechner mit 4 physikalischen (8 logischen) Rechenkernen entwickelt und größtenteils auch auf diesen Rechner getestet. Um die Anzahl der Recheneinheiten zu steuern wurde ein Parameter **-np n** benutzt, wo **n** die Anzahl der Recheneinheiten bedeutet.

Das Programm hat mehrere Aufrufparameter. Es gibt erforderliche, optionale und miteinander kombinierte Parameter. Diese werden in der folgenden Tabelle aufgelistet.

		Anforderung	
Aufrufparameter	**Parameterargument**	**Parameter**	**Parameterargument**
-f	Dateiname	erforderlich	erforderlich
-g	n	optional	erforderlich
	n_x_y_h_w		erforderlich
-p		optional	
-r		optional	
-d	µSec	optional	optional
-o		optional	
-t	Versuche	optional	optional
-w	Teilstring	optional	optional

Tabelle 1: Alle im Programm existierende Aufrufparameter

In der ersten Spalte in der Tabelle 1 ist zu sehen, dass die Aufrufparameter unterschiedlich nach rechts eingerückt sind. Das bedeutet, dass die Aufrufparameter voneinander abhängen. Zum Beispiel -o hängt von **-r** ab und **-r** hängt von **-f** ab. In der dritten und vierten Spalte wird

angegeben ob der Aufrufparameter bzw. deren Parameterargument optional oder erforderlich ist. Im Folgendem werden die Aufrufparameter näher beschrieben.

-f Dateiname

Dieser Parameter gibt an, welche **Datei** zum Testen des Programms oder welcher **Dateiname** zum Generieren der Testdatei verwendet werden soll.

-g n oder **-g n_x_y_h_w**

Mit diesem Parameter wird eine **n*n** Matrix generiert. Es gibt eine Möglichkeit einen einfachen Rechteck in die Matrix einzulegen. Dabei soll man eine obere linke Ecke mit **x_y** – Koordinaten[1] sowie dessen Breite und Höhe mit **h_w** angeben. Alle vier Parameterargumente sollen mit einem Unterstrich getrennt werden. Die generierte Datei kann mit einem beliebigen Texteditor geöffnet und editiert werden. Weißen Felder werden mit dem Punkt(.) und schwarzen Felder mit dem großen **X** dargestellt. Auf diese weise kann man einzelne Matrixfelder editieren.

-p

Damit kann man die generierte Matrix auf der Konsole ausgeben.

-r

Dieser Parameter startet ein Rechteckmustererkennung - Algorithmus.

-o

Mit diesem Parameter werden einige sinnvolle Informationen, welche beim Ablauf des Algorithmus entstehen, auf der Konsole ausgegeben. Diese sind die aufgeteilten Blöcke, welche den einzelnen Prozessoren zugesandt wurden und die Ergebnisse der Rechteckmustererkennung, welche dem Master - Prozess zugesendet werden.

-d oder **-dµSec**

Da die einzelnen Prozessoren nicht in der richtigen Reihenfolge die Information, welche mit dem **-o** - Parameter ausgegeben wird, auf der Konsole ausgegeben wird, soll diese Ausgabe zeitversetzt geschehen. Die einzelnen Prozesse werden einfach mit der bestimmten Zeitlänge schlafengelegt. Die Zeitversetzung wird mit der folgenden Formel berechnet: $delay = rank \cdot \mu Sec$. Der Standardwert der Verzögerung liegt bei 1000µs.

1 Die Indexierung fängt bei 0 an.

-t oder **-tVersuche**

Mit diesem Parameter wird die reine Rechteckfindung auf den Prozessoren mehrmals wiederholt um die Fehlerquote der Zeitmessung, welche durch Beanspruchen der Rechenzeit durch die Anderen verursacht werden kann, zu reduzieren. Der Standartwert beträgt 1.

-w oder **-wTeilstring**

Mit diesem Parameter werden die Ergebnisse der Zeitmessung in eine csv - Datei geschrieben. Ein **Teilstring** wird am Ende der Dateiname angehängt.

Zu Beachten: Alle optionale Parameterargumente sollen **ohne Leerzeichen** hinter dem Aufrufparameter geschrieben werden!

Ausführung der gewählten Option geschieht im Prozess mit dem Rang 0 (im Folgendem - Masterprozess) um die sequentielle Aufgaben nicht parallelisieren zu können. Falls die Aufrufparameter falsch mit einander kombiniert angegeben werden, wird entsprechendes Fehlercode auf der Konsole ausgegeben und das Programm wird beendet.

2.2 Beschreibung des Rechteckmustererkennung - Algorithmus

Als erstes werden die Daten aus der Datei, welche mit dem **-f** - Flag angegeben wurde, gelesen und in ein eindimensionales char - Feld geschrieben. Danach wird die Dimension der Matrix mit der Anzahl der verfügbaren Prozessoren verglichen. Falls die Anzahl der Prozessoren größer als die Dimension der Matrix ist, wird der entsprechender Fehlercode auf der Konsole ausgegeben und das Programm wird beendet. Falls das Dimensionsproblem nicht nicht auftritt, wird die Matrix horizontal auf Datenblöcke aufgeteilt. Dimension und die Anzahl der Zeilen im einen Datenblock wird vom Masterprozess an die Slaveprozesse mit der MPI_Bcast() - Funktion gesendet. Die Matrix wird so aufgeteilt, dass der Rest immer auf alle - 1 Prozessoren verteilt wird. Zum Beispiel: Eine 10*10 Matrix wird auf 3 Prozessoren verteilt → da die Division (10/3) nicht aufgeht, bekommen 2 Prozessoren je 4 Zeilen und der letzte bekommt nur 2 Zeilen der Matrix.

Als Nächstes werden die Datenpakete an die einzelne Prozessoren mit der MPI_Send() - Funktion gesendet. Dabei wurde festgelegt (in unserem Fall), dass die maximale Matrix-dimension den Wert $\lfloor\sqrt{2147483647}\rfloor = 46340$ [2] annehmen kann. Es liegt an der MPI_Send() - Funktion, welcher ein **int count** - Parameter übergeben werden soll. Da der Parameter vom Typ **int** ist, kann auch dann nur eine Matrix der Dimension 46340 gesendet werden. Mann kann auch eine größere Matrix senden in dem man als Datentyp **long long** nimmt und die Pixelinformation im einzelnen Bit kodiert.

Anschließend wird eine MPI_Recv() - Funktion gestartet und auf die einzelne Ergebnisse gewartet.

2 2147483647 – maximaler int - Wert

2.2.1 Beschreibung des Rechteckfindung - Algorithmus

Das Rechteckfindung - Algorithmus besteht aus zwei for - Schleifen. Äußere Schleife adressiert Zeilen und innere Schleife adressiert Zeichen in der Zeile (also Spalten). An der Ersten stelle wird geprüft ob ein Zeichen dem 1 - Wert[3] entspricht. Falls dies der Fall ist, werden alle vier Variablen (links, rechts, oben, unten) mit dem entsprechenden Wert initialisiert und der inRect - Flag, welcher das Befinden im Rechteck signalisiert, wird auf 1 gesetzt. Dies bedeutet, dass man sich in einem Rechteck befindet. Am Ende der Rechteckzeile wird die Position der letzten Spalte in dieser Zeile in eine ir1First - Variable gespeichert. Die ir1First - Variable ermöglicht in den nächsten Zeilen zu prüfen, ob der Rechteck seinen rechten Rand immer in der gleichen Spalte hat. Dies schließt folgende Fälle aus:

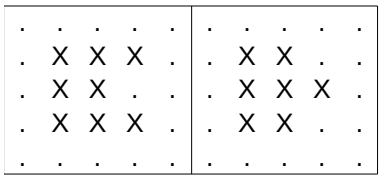

Abbildung 1: Fälle, welche mit der ir1First - Variable ausgeschlossen werden

Wenn man auf einer Zeile im Rechteck ist und danach eine weiße Fläche trifft, dann wird die inRect - Variable auf 0 gesetzt. Falls man auf der selben Zeile wieder eine schwarze Fläche Trifft, dann wird ein zweites Rechteck erkannt. Beim Übergang auf die nächste Zeile wird immer geprüft, ob der linke Rand immer in der gleichen Spalte bleibt. So werden folgende Fälle erkannt:

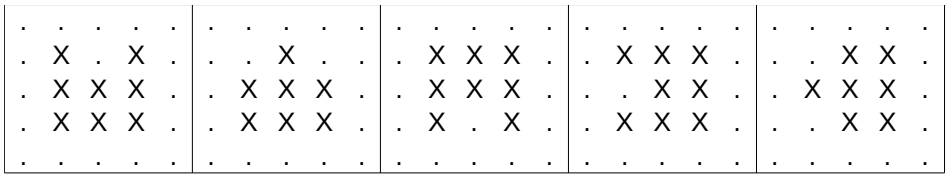

Abbildung 2: Fälle, welche mit der oben beschriebenen Verfahren ausgeschlossen werden

Die Resultate der einzelnen Prozessoren werden in ein int - Feld mit 6 Einträgen (Rang, R, i0, i1, j0, j1)[4] geschrieben und an den Masterprozess gesendet. Die Koordinate der gefundenen Rechtecke werden auf den jeweiligen Datenblock (ein Teil der gesamten Matrix) bezogen.

3 1 – Schwarz, 0 – Weiß

4 R, i0, i1, j0, j1 – siehe Aufgabestellung

2.2.2 Beschreibung des Algorithmus über das Vorliegen des Rechtecks

Nachdem der Masterprozess alle Ergebnisse von den Slaveprozessoren in ein Ergebnisvektor gesammelt hat, rechnet er die Koordinate der gefundenen Rechtecke der einzelnen Prozessoren auf die Koordinate der ganzen Matrix um. Diese Operation geschieht sehr schnell und die dabei entstehende Resultate können mit dem **-o** - Aufrufparameter auf der Konsole ausgegeben werden.

Nun werden die Ergebnisse ausgewertet. Da der obere und der untere Rand des Rechtecks schon bei den einzelnen Slaveprozessoren geprüft werden, es sollen nur wenige Bedingungen geprüft werden. Als Erstes wird die Bedingung geprüft, ob die Rechtecke der einzelnen Prozessoren zusammenhängend[5] sind. Weitere Bedingung[6] ist, dass der linker und der rechter Rand die selbe i - Koordinate haben. Falls im Ergebnisvektor der R-Wert den Wert 0 aufweist, wird der Vorgang mit der Meldung „Es gibt kein zusammenhängendes Rechteck!“ abgebrochen. Falls ein zusammenhängendes Rechteck gefunden wurde, es werden die Koordinaten des Rechtecks zusammen mit der Meldung „Es gibt ein zusammenhängendes Rechteck!“ auf der Konsole ausgegeben.

5 Siehe 2. Bedingung in der Aufgabestellung.
6 Siehe 1. Bedingung in der Aufgabestellung.