Universität Hamburg

Fachbereich Informatik

Praktikumsbericht im Rahmen des Praktikums
Paralles Programmieren

Mastermind

vorgelegt von

Nicole Tänzer, Fabian Fiedler

Inhaltsverzeichnis

1 Einleitung

1.1 Das Spiel Mastermind

Abbildung 1
[wiki1]

Mastermind ist ein Logikspiel für 2 Personen und gehört zu den erfolgreichsten Spielen der 70er Jahre, da die grundlegenden Regeln einfach zu verstehen und schnell zu erlernen sind. Eine gute Strategie hingegen ist nicht schnell gefunden, sodass die Herrausforderung am Spiel lange erhalten bleibt. Das Spiel wurde 1973 in Paris von dem Israeli Marco Meirovitz erfunden. Die Grundform des Spiels wird mit 6 Farben über 10 Reihen gespielt.

1.1.1 Begriffe des Spiels

Lösungscode	Der Code ist die Folge von vier Farben, welche zu erraten ist.
Stift	Ein Teil dieses Codes ist ein Stift. Stifte gibt es in 6 unterschiedlichen Farben, und sie haben eine Position im Code.
Stecker	Beim Bewerten des Lösungsversuches werden weiße und schwarze Stifte gesetzt.

1.1.2 Spielprinzip

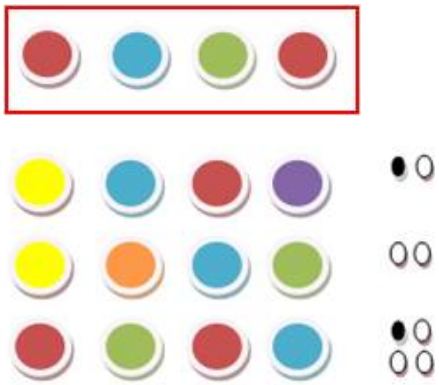

Abbildung 2

Ein Spieler setzt einen vierstelligen, durch einen Sichtschutz verdeckten Code, wobei die einzelnen Farben auch mehrfach vorkommen dürfen. Dieser Code ist in unserem Beispiel in Abbildung2 durch eine rote Umrandung gekennzeichnet. Dies ist die einzige aktive Tätigkeit des passiven Spielers während des gesamten Spiels. Bei vier Stellen und sechs Farben ergeben sich 6^4, also 1296 mögliche Ergebnisse. Die Aufgabe des anderen Spielers ist es, diesen Code zu erraten. Dazu beginnt er mit einer beliebigen Farbkombination. Der passive Spieler steckt nun für jede richtige Farbe an falscher Stelle einen weißen Stecker und für jede richtige Farbe an richtiger Stelle einen schwarzen Stecker. Der ratende Spieler weiß jedoch nicht, auf welche Stifte sich die Stecker beziehen. Das heißt welche Stifte richtig sind und welche Stifte zwar die richtige Farbe haben, sich allerdings an der falschen Position befinden. Nun kann der aktive Spieler unter Zuhilfenahme dieser weißen und schwarzen Stecker einen neuen Code raten, welcher wiederum bewertet wird. Dies wird in der Abbildung 2 an drei Beispielschritten verdeutlicht. Der aktive Spieler hat insgesamt 10 Versuche um den Code vollständig zu erraten, also vier schwarze Stecker zu erhalten. Schafft er dies, hat er gewonnen, sonst verloren.

1.2 Knuths-Algorithmus

Eine optimale Mastermind-Strategie ist für die meisten Menschen nicht im Kopf berechenbar. Dies ist unter anderem auch der Grund dafür, dass Mastermind und seine Strategie bereits Gegenstand zahlreicher wissenschaftlicher Arbeiten waren. Auch viele namhafte Forscher haben sich bereits mit dem Thema beschäftigt, so zum Beispiel Donald Ervin Knuth.

In unserem Programmcode benutzen wir den Algorithmus von Knuth.

Als ersten Schritt von Knuths Algorithmus haben wir ein Array mit allen 1296 möglichen Codes erstellt. Anschließend werden als zweiter Schritt nach dem ersten Rateversuch ((1, 1, 2, 2) bei Knuth), alle Codes aus dem Array gelöscht, welche nicht die gleiche Anzahl an schwarzen und weißen Steckern erhalten könnten, wie der Rateversuch. Erhält der Code zum Beispiel vier weiße Stecker, so bleiben nur Codes im Array enthalten, die die vier gesetzten Farben enthalten. Erhält der Code nur zwei schwarze Stecker, so bleiben alle Codes erhalten, die zwei der Stifte an selber Stelle haben, wie der Rateversuch. Die anderen beiden Stifte müssen komplett ausgetauscht werden, dürfen also nicht die Farben haben, die sie vorher hatten. Die Codes gehen wir mit Hilfe einer for-Schleife durch. Der dritte Schritt berechnet für jeden noch möglichen Code wie viele Lösungen bei welcher Konstellation von schwarzen und weißen Steckern noch übrig bleiben.

		Code				
		1111	1112	1122	1123	1234
Antwort	00	625	256	256	81	16
	01		308	256	276	152
	02		61	96	222	312
	03			16	44	136
	04			1	2	9
	10	500	317	256	182	108
	11		156	208	230	252
	12		27	36	84	132
	13				4	8
	20	150	123	114	105	96
	21		24	32	40	48
	22		3	4	5	6
	30	20	20	20	20	20
	40	1	1	1	1	1

Abbildung 3
[Fahrni09]

Abbildung 3 verdeutlicht, dass Knuth aufgrund dieser Vorgehensweise auf den Startcode von (1, 1, 2, 2) kommt. Bei dieser Konstellation hat die größte Teilmenge mit 256, im Vergleich zu den anderen möglichen Startcodes, den niedrigsten Wert. Bei (1, 2, 3, 4) ist die größte Teilmenge mit 312, wie bei allen anderen Möglichkeiten ebenfalls, größer. Bei der Möglichkeit

von 0 schwarzen und 4 weißen Steckern gibt es in diesem Fall zwar nur noch 9 übrige Möglichkeiten, aber das ist nicht die größte Teilmenge. Der vierte und letzte Schritt von Knuth besteht darin zu Schritt zwei zurückzukehren, bis man die Lösung gefunden hat.

1.3 Ziel

Unser Ziel ist es einen Löser zum Spiel zu programmieren. Hierbei haben wir aus Zeitgründen von einer graphischen Version abgesehen. Wir programmieren zwei Versionen unseres Lösers. Die serielle Version soll für eine gegebene Eingabe ausrechnen, wie viele Versuche sie benötigen würde. Die parallele Version soll für alle Codemöglichkeiten berechnen, wie lange der Knuth-Algorithmus braucht und gibt die Ergebnisse in einer Liste aus. Daraus kann man schließen, ob ein Code vorteilhaft für den passiven Spieler ist.

2 Implementierung des seriellen Lösers

2.1 Darstellung des Spiels im Programm

Der zu lösende Code wird vom Benutzer vorgegeben und unser Programm beginnt mit einer festgelegten Startformation (0, 0, 1, 1). Dies entspricht z.B. rot, rot, blau, blau. Dieser Code wird automatisch nach den Spielregeln bewertet und alle Lösungen, die nicht mehr möglich sind, werden entfernt.

Ein Beispiel hierfür ist der Code blau, grün, grün, rot. Wenn dieser zwei schwarze und einen weißen Stecker erhält, ist die Kombination grün, rot, grün, blau nicht mehr möglich, da alle Farben identisch mit dem gewählten Code sind. Wenn dies die richtigen Farben wären, hätte der Code mit vier Steckern bewertet werden müssen, anstatt mit drei Steckern.

Anschließend wird aus den übrigen Lösungen eine günstige nach Knuth berechnet und gewählt. Die Wiederholung dieser Schritte ist so umfangreich, dass kein Spieler mehr in der Lage ist, sie im Kopf zu bewältigen. Abschließend soll unser Programm die korrekte Lösung, welche mit 4 schwarzen Stecker bewertet wird, nach möglichst wenigen Schritten (auf jeden Fall weniger als 10) ausgeben.

Über die main Methode bekommt das Programm vom Benutzer den zu erratenden Code durchgegeben und wandelt ihn in ein Integerarray um. Außerdem haben wir ein zweidimensionales Array, in welchem wir alle 1296 möglichen Ergebnisse darstellen. Mit der Methode

‚erstelleListe‘ wird dieses Array nun initialisiert. Dadurch, dass wir die Arrays als Funktionsaufrufe übergeben, müssen wir den Speicherbereich mit dem Befehl „malloc“ dynamisch reservieren. So ist der Speicherbereich nicht an einen Block gebunden, sondern kann programmweit frei adressiert werden. Nun wird mit dem eigentlichen Lösungsvorgang und der Methode Loeser begonnen. Es werden die drei Arrays ‚bewertung‘, ‚naechsterTipp‘ und ‚loesung‘ erstellt. Danach wird die Variable naechsterTipp mit dem „Standard-Ersttipp“ von Knuth (0, 0, 1, 1) initialisiert und bewertet. Nun werden alle nicht mehr möglichen Farbkombinationen entfernt. Anschließend erfolgt das Berechnen des nächsten Tipps, darauf sind wir bereits bei der Erklärung des Algorithmus eingegangen. Dieser Tipp wird wiederum bewertet und anschließend wird erneut ein Tipp berechnet, dies wird so lange wiederholt, bis der Tipp 4 schwarze Stecker erhält, also mit der übergebenen Anfangslösung übereinstimmt.

Mit der Methode BerechnenaechstenTipp wird nach der Strategie von Knuth, der nächste Lösungsvorschlag berechnet.

2.2 Probleme

Schwierigkeiten hatten wir bei dem Versuch, Pointer auf Arrays zu setzen. Dieses Problem war ein Missverständnis, da wir nicht wussten, wie Pointer auf Arrays funktionieren. Wir haben den Speicher anschließend dynamisch mit malloc reserviert und das Problem so behoben. Weitere Probleme beim Erstellen des Quellcodes bezogen sich lediglich auf Verständnisprobleme durch das Programmieren mit C. Wie zum Beispiel die fehlerhafte Anwendung des „Oder-Zeichens“.

3 Parallelisieren

Das Berechnen einer Lösung im seriellen Code dauert weniger als eine Sekunde. Daher mussten wir das Problem erweitern. Unsere erste Idee war es, den Lösungsraum über mehr Stecker und mehr Farben zu vergrößern. Wir hatten dann 46656 statt 1296 Möglichkeiten, jedoch dauert auch dies lediglich 4,5 Sekunden. Das empfanden wir als nicht ausreichend zum Parallelisieren. Letztendlich kam uns eine Idee von Julian Kunkel wieder in den Sinn, zu berechnen, welche Tipps schwer und welche leicht zu knacken sind. Diese Idee haben wir daraufhin implementiert. Das Programm rechnet nun für jeden möglichen Code aus, wie viele Versuche es braucht und gibt dann eine nach Versuchen geordnete Liste aus. Die Berechnung dauert

nun 16 Minuten und war daher für eine Parallelisierung ausreichend. Wir begannen nun zu parallelisieren und verwendeten dafür MPI.

3.1 Strategie der Parallelisierung

Wir haben uns für eine Master-Slave-Architektur entschieden, dass heißt ein Prozessor ist der Master und kümmert sich um die Verteilung und Organisation der Aufgaben. Er bekommt und verarbeitet die Ergebnisse der Aufgaben. Die anderen Prozessoren sind Slaves, das heißt sie bekommen eine Aufgabe vom Master zugewiesen und berechnen das Ergebnis und geben es an den Master zurück. Wir haben uns dafür entschieden, dass ein Aufgabenpaket dafür zuständig ist zu Berechnen, wieviele Versuche unser Algorithmus für einen bestimmtes Codewort benötigt. Das MPIframework erlaubt nur das Senden von Nachrichten, deswegen ist die Struktur wie folgt: Der Masterserver sendet an jeden Klienten den Code, den dieser zu erraten hat. Dieser rechnet dann aus, wie viele Versuche er benötigen würde. Danach werden die Ergebnisse zurück geschickt. Nun sendet der Masterserver eine neue Aufgabe an den Clienten. Dies macht der Masterserver für alle zu erratenden Kombinationen. Nachdem alle Aufgaben verteilt sind, wartet er auf die restlichen Berechnungen. Zum Schluss gibt der Masterprozess die Ergebnisse sortiert aus.

3.2 Probleme

Ein Problem, welches in dem Programm auftritt, ist, dass Nachrichten nicht ankommen. Die Funktion MPI_WaitAll hält alle gesendeten Nachrichten für abgeschlossen. Die Error Status der Nachrichten sind alle 0 und stellen damit eigentlich eine korrekte Übertragung dar. Daher ist uns nicht bewusst, woher diese Fehler kommen. Aufgrund dieses Fehlers fehlen in der Parrallelen Version einige zufällige Werte.

Ein zweites Problem unserer parallelen Version ist, dass die Software abstürtzt sobald man weniger als 8 Kerne benutzt.

3.3 Speedup

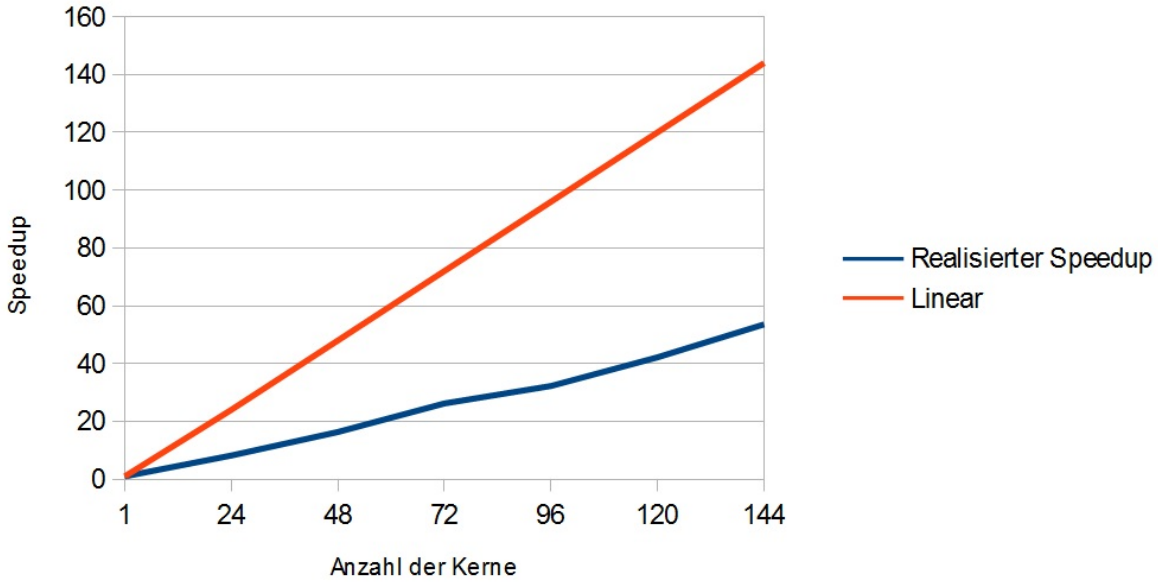

Abbildung 4

Wie man sieht, ist der Speedup weit unter dem linearen Optimum des Speedup. Dies entsteht zum einen dadurch, dass die sequenzielle Version kein MPI benutzt und daher den Overhead für das Initiallisieren nicht hat. Auch zeigt es, dass unsere Verteilung der Aufgaben keine optimale Lösung darstellt. Wenn man allerdings den SpeedUp nur im Vergleich paralleler Versionen zieht wie in Abbildung 5, ist der Speedup bedeutend besser. Daraus können wir schließen, dass der Großteil des Verlustes beim SpeedUp im Initialisieren des MPI liegt.

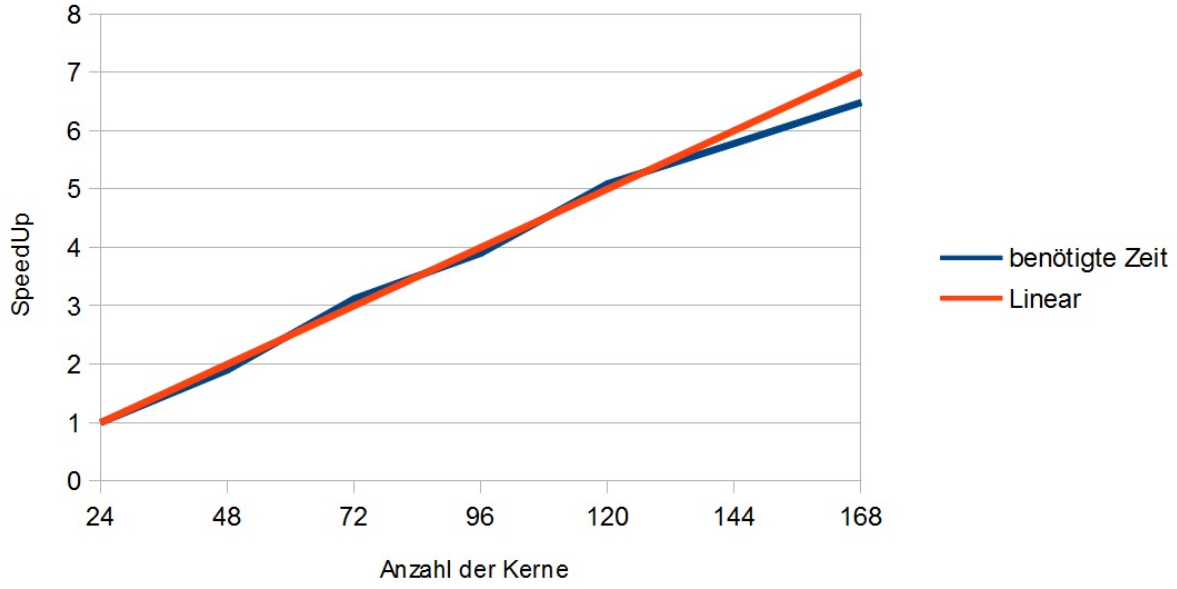

Abbildung 5

4 Fazit

Wir haben die Vorraussetzungen der Kenntnisse für die C-Programierung deutlich unterschätzt. Da wir keine Erfahrungen mit der C-Programmierung hatten, verloren wir viel Zeit durch das Suchen und Beheben von kleineren Fehlern. Durch die fehlenden C Kenntnisse kam hinzu, dass das Verstehen der Erklärungen am Anfang des Semsters für uns kaum möglich war. Gleiches gilt für die Bearbeitung der Hausaufgaben. Das Nacharbeiten war kaum praktikabel, weil wenig Materialien zu den Themen am Anfang ausgegeben wurden. Hier hätten wir uns ein Skript oder ähnliches gewünscht. Dies alles führte dazu, dass wir zu viel Zeit zu Beginn mit der ersten Version des seriellen Codes verbraucht haben und dadurch stark an Motivation verloren haben.

Literatur

[Bewersdorff12] Glück, Logik und Bluff, Dr, Jörg Bewersdorff, 2012

[Knuth67] J. Recreational Mathematics, Vol. 9(1), 1976/77: The Computer as Mastermind, Donald Knuth

[Stuckmann05] Mastermind is NP-Complete, Jeff Stuckman and Guo-Qiang Zhang

[Fahrni09] Maturaarbeit Reto Fahrni 2009: Mastermind – perfekte Strategie dank Computer

[wiki1] http://de.wikipedia.org/wiki/Mastermind

[wiki2] http://de.wikipedia.org/wiki/Donald_Ervin_Knuth