LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Aspekte der systemnahen Programmierung bei der Spieleentwicklung

Gruppe 113 – Abgabe zu Aufgabe A312
Wintersemester 2018/19

Raphael Penz/David Drothler/Joshua Weggarter

1 Einleitung

Im Praktikum "Aspekte der systemnahem Programmierung bei der Spieleentwicklung" wurde den Studenten das systemnahe Programmieren näher gelegt. Hierbei wurden in den Tutorstunden die Grundlagen der Linux Kommandozeile mit ihren Befehlen, so wie die Grundlagen von Assembler besprochen, wie die verschiedenen Datentypen, die unterschiedlichen Resgister, der Aufbau und die Funktionsweise vom Arbeitspeicher und des Stackpointers. Auch die Instruktionen für AArch64 ,wie Lade und Speicherinstruktionen oder Kontrollflussinstruktionen, wurden erklärt und teilweise einzelne wie Shiftinstuktionen, näher besprochen. Auch die Calling Conventions und Funktionsaufgrufe wurden in den Tutorien behandelt. Doch nicht nur Register für Ganzzahlarithmetik waren Thema des Paraktikums, sondern auch die Floating-Point Unit, mit der auch Gleitkommazahlen zur Berechnung benutzt werden können. Auch die verschiedenen Instruktionen für Floating-point-Werten und deren Unterschiede zu den StandardInstruktionen wurden erklärt. Im Laufe des Themas der Optimierung wurden auch SIMD Befehle eingeführt und ihre Verwendung mithilfe von Beispielen gezeigt. Auch das Debugging und Benchmarking wurden besprochen um das Optimieren der Programme zu ermöglichen. In den Tutoraufgaben und den Hausaufgaben wurden auch eineige Beisielprogramme wie "Quicksort" und "ToUpper" umgesetzt um das gelernte Wissen praktisch umzusetzen. Zum Schluss wurde auch noch nähergelegt, wie einfache Programmierfehlern in C die Integrität des ausgeführten Programms oder Systems beeinträchtigen können.

2 Problemstellung und Spezifikation

Unsere Aufgabe war es die Multiplikation von dünnbesetzten Matrizen zu implementieren. Hierbei sollten wir diese Multiplikation für zwei verschiedene Formate, die benutzt werden um dünnbesetzte Matrizen einfacher/effizienter darzustellen, umsetzen. Diese Formate sind zu einem das "Coordinate Scheme", welches die Matrix als eine Menge von Tupel darstellt. Jedes dieser Triplets enthält die Reihe,die Spalte und den Wert eines "Nicht-Null-Eintrages". Grundsätzlich lässt sich sagen, dass das "Coordinate Scheme" sich für sehr große Matrizen eignet, da es sehr schnell zu erstellen ist und gut geeignet um in andere Formate umgewandelt zu werden wie das "Compressed sparse row" oder das "Compressed sparse collum" Schema.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 1 & 3 & 0 & 1 \\ 0 & 0 & 3 & 0 \end{pmatrix}$$

Abbildung 1: Matrix 1

$$A = \{(0,1,1),(1,1,2),(1,2,1),(2,0,1),(2,1,3),(2,3,1),(3,2,3)\}$$

Abbildung 2: Matrix 1 im “Coordinate Scheme”

Das andere Format ist das “Jagged Diagonal Storage” Schema. Bei diesen wird die Matrix komprimiert, indem alle Nullen rausgenommen werden und die restlichen Einträge nach links verschoben werden. Danach werden die Zeilen nach der Anzahl an Einträgenin einer Zeile sortiert und diese Umstellung im Permutationsfeld “perm” notiert. Als nächstes werden die 1. Einträge der einzelnen Zeilen, beginned mit der längsten Zeile, danach die Zweitlängste,... , in das Feld “jdiag” übertragen und die ensprechende Spalte in der der Eintrag ist in das Feld “colInd” eingetragen. Danach werden die 2. Eintrage der Zeilen nach dem selben Schema behandelt und die restlichen ebenfalls. Zum Schluss wird in das Feld “jdPtr” die Stellen der Anfänge der so neu enstandenen “Jagged Diagonal” eingetragen, also der Index der Elemente, die in der längsten Zeile der Ausgangsmatrix standen. Beim Betrachten von dünnbesetzten Matrizen wird schnell klar, dass wenn man die Multiplikation dieser wie bei normalen Matrizen programmieren würde, dies sehr ineffizient wäre, da man viele Rechnungen, bei denen man mit 0 multipliziert, nicht durchführen müsste, sondern nur die Rechnungen, bei den zwei nichtnull Werte beteiligt sind. Auch das Speichern solcher Matrixen wäre aus ähnlichen Gründen nicht effizient. Deshalb gibt es diese Formate um dünnbesetze Matrizen darzustellen/abzuspeichen und deshalb müssen wir eine geeignete Implementierung finden, diese effizient zu multiplizieren.

$$\begin{pmatrix} 10 & -3 & 0 & 1 & 0 & 0 \\ 0 & 9 & 6 & 0 & -2 & 0 \\ 3 & 0 & 8 & 7 & 0 & 0 \\ 0 & 6 & 0 & 7 & 5 & 4 \\ 0 & 0 & 0 & 0 & 9 & 13 \\ 0 & 0 & 0 & 0 & 5 & -1 \end{pmatrix} \longrightarrow \begin{pmatrix} 10 & -3 & 1 & \\ 9 & 6 & -2 & \\ 3 & 8 & 7 & \\ 6 & 7 & 5 & 4 \\ 9 & 13 & & \\ 5 & -1 & & \end{pmatrix}$$

Abbildung 3: Matrix 2 und ihre komprimierte Variante

jdiag	6	9	3	10	9	5;	7	6	8	-3	13	-1;	5	-2	7	1;	4;
col_ind	2	2	1	1	5	5;	4	3	3	2	6	6;	5	5	4	4;	6;

perm	4	2	3	1	5	6

jd_ptr	1	7	13	17

Abbildung 4: Matrix 2 im "Jagged Diagonal Storage Scheme"

3 Lösungsfindung

Zunächst wurde über die gestellten Fragen im theoretischem Teil nachgedacht und die Grundvorrausetzungen für die Matrizenmultiplikation erarbeitet: Matrix 1 muss genau so viele Spalten haben wie Matrix 2 Zeilen. Für das in C geschriebene Ein- und Ausgabeprogramm wurde zunächst ein Format entwickelt in dem der User die beiden Matrizen übergeben kann. Damit der User nicht selbst per Hand die Matrizen in das jeweilige Schema umformen muss, übergibt er diese in unserem erdachten Format. In diesem Format sind die jeweiligen Einträge wie bei einer üblichen Matrix angeordnet. Nur sind die einzelnen Einträge mit einem Leerzeichen getrennt und die einzelnen Zeilen mit einem "," getrennt. Zwischen den beiden Matrizen wird ein ";" gesetzt um diese zu trennen. Nulleinträge müssen mit eingetragen werden. Dieses Format führt dazu, dass der Benutzer ohne zusätzliches Rechnen seine gewünschten Matrizen einfach übergeben kann. Nach dem Einlesen werden die Matrizen in ihr jeweiliges Format umgeformt, je nachdem was der User am Anfang des Programmes ausgewählt hat. Natürlich könnte man auch die Matrizen direkt im ensprechendem Format vom Benutzer eintragen lassen, doch dann wäre das viel aufwändiger für diesen. Bei der Implementierung der Multiplikation für das "Coordinate Scheme" wurde zunächst überlegt wie eine Multiplikation grundsätzlich in diesem Format aussieht.... Bei der Implementierung der Multiplikation für das "Jagged Diagonal Sceme" wurde ebenfalls überlegt wie eine Multiplikation in diesem Format grundsätzlich aussieht....Hierbei wurde schnell klar, dass für das effiziente implementieren die gegebene Signatur der Funktion angepasst werden muss. Anstatt einen Pointer auf die Matrix, die aus den verschiedenen Arrays besteht, werden für jede Matrix nun jedes Array einzeln übergeben. Dies führt dazu, dass man bei der Implementierung einfacher auf die Arrays zu greifen kann und am Anfang nicht erst suchen muss, wo die einzelnen Arrays anfangen bzw. aufhören. Des Weiteren wurden an den Enden der Arrays eine Null angehängt, damit das Ermitteln des Endens des Arrays ermöglicht wird, da diese keine feste Länge haben.

4 Dokumentation der Implementierung

Als Eingabe für die Matrixmultiplikation dient ein Textdokument namens “Matrices.txt”. In diesem trägt der Benutzer 2 Matrizen ein. Bei diesen Matrizen werden die einzelnen Einträge der Matrix mit Leerzeichen getrennt, die zeilen mit einem “,” und die beiden Matrizen mit einem “;”. Hierbei ist zu beachten, dass die Nulleinträge mit eingetragen werden müssen und nur Matrizen mit Ganz-, oder GleitkommaZahlen angenommen werden. Bei Matrizen mit Buchstaben, Sonderzeichen, oder ähnlichem wird die Eingabe nicht angenommen. Bei der Implementierung des C-Eingabeprogramm wurde wiefolgt vorgegangen: Zunächst wird der User solange nach der Art der Multiplikation gefragt, bis er eine gültige Eingabe macht. Danach wird das Eingabefile geöffnet und ausgelesen und Sachen wie Höhe, Breite und Anzahl der "Nicht-Null-Einträgen"bestimmt, um die größe der Felder zu bestimmen. Beim erneuten Öffnen werden nun die beiden “matrix” Felder mit den Werten aus den Eingabematrizen befüllt.Dies geschieht im “Coordinate Scheme”. Wenn nun der Benutzer die “Coordniate Scheme” Multiplikation am Anfang ausgewählt hat wird die Ergebnismatrix erstellt und die drei Matrizen der ensprechenden Funktion übergeben. Im Fall, dass die “Jagged Diagonal Storage” Multiplikation ausgewählt wurde, wird zuerst das Permutationsfeld für beide Matrizen berechnet und anschließend das Hilfsfeld “rows”. Dieses speichert wie viele Zeilen die Matrix mit der größten Anzahl an Einträgen bis zur niedrigsten Anzahl an Einträgen hat. Dieses wird benötigt um dann im dritten Schritt die Felder “jdag” und “colInd” zu befüllen, die den Wert und die Spalte der jeweiligen Einträge speichern. Zum Schluss wird dann noch ans Ende jedes Arrays eine Null angehängt, die ensprechenden Ergebnissfelder erzeugt und der Funktion als Parameter übergeben.

5 Ergebnisse

6 Zusammenfassung

7 Quellenverzeichnis

- http://www.netlib.org/linalg/html$_{t}emplates/node95.html$