LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Multiplikation dünnbesetzter Matrizen: CSC (A313)
Projektaufgabe – Aufgabenbereich Algorithmik

1 Organisatorisches

Auf den folgenden Seiten finden Sie die Aufgabenstellung zu Ihrer Projektaufgabe für das Praktikum. Die Rahmenbedingungen für die Bearbeitung werden in der Praktikumsordnung festgesetzt, die Sie über die Praktikumshomepage[1] aufrufen können.

Wie in der Praktikumsordnung beschrieben, sind die Aufgaben relativ offen gestellt. Besprechen Sie diese innerhalb Ihrer Gruppe und konkretisieren Sie die Aufgabenstellung. Die Teile der Aufgabe, in denen C-Code anzufertigen ist, sind in C nach dem C17-Standard zu schreiben.

Der **Abgabetermin** ist **Sonntag 21. Juli 2024, 23:59 Uhr (CET)**. Die Abgabe erfolgt per Git in das für Ihre Gruppe eingerichtete Projektrepository auf dem `main` Branch. Bitte beachten Sie die in der `README.md` angegebene Liste von abzugebenden Dateien.

Die **Abschlusspräsentationen** finden in der Zeit vom **19.08.2024 –30.08.2024** statt. Weitere Informationen werden noch bekannt gegeben. Beachten Sie, dass die Folien für die Präsentation am obigen Abgabetermin im PDF-Format abzugeben sind und keine nachträglichen Änderungen akzeptiert werden können.

Bei Fragen/Unklarheiten in Bezug auf den Ablauf und die Aufgabenstellung wenden Sie sich bitte an Ihre:n Tutor:in.

Wir wünschen Ihnen viel Erfolg und Freude bei der Bearbeitung Ihrer Aufgabe!

Mit freundlichen Grüßen
Die Praktikumsleitung

[1] `https://gra.caps.in.tum.de`

2 Multiplikation dünnbesetzter Matrizen: CSC

2.1 Überblick

Die Algorithmik ist ein Teilgebiet der Theoretischen Informatik, welches wir hier unter verschiedenen praktischen Aspekten beleuchten: Meist geht es um eine konkrete Frage- oder Problemstellung, welche durch mathematische Methoden beantwortet oder gelöst werden kann. Sie werden im Zuge Ihrer Projektaufgabe ein Problem lösen und die Güte Ihrer Lösung wissenschaftlich bewerten.

2.2 Einführung

Aus der linearen Algebra kennen Sie die Formel zur Multiplikation vollbesetzer Matrizen A und B, bei der sich jedes Element der Ergebnismatrix C folgendermaßen berechnen lässt:

$$c_{ik} = \sum_{j=1}^{m} a_{ij} \cdot b_{jk} \tag{1}$$

Es lässt sich leicht einsehen, dass diese Formel für Matrizen mit vielen 0-Einträgen (sogenannten dünnbesetzten Matrizen) ineffizient arbeitet ($0 \cdot 0 + 0 \cdot 0 + \cdots + 0 \cdot 0 = 0$). Man führt daher spezielle Formate für Matrizen ein, welche die Einträge, welche gleich 0 sind, nicht explizit speichern. Ein Format zum Speichern dünnbesetzter Matrizen ist das *Compressed Sparse Column* Schema.
Ihre Aufgabe ist es, die Multiplikation zweier Matrizen in diesem Format zu implementieren.

2.3 Aufgabenstellungen

Ihre Aufgaben lassen sich in die Bereiche Konzeption (theoretisch) und Implementierung (praktisch) aufteilen. Sie können (müssen aber nicht) dies bei der Verteilung der Aufgaben innerhalb Ihrer Arbeitsgruppe ausnutzen. Antworten auf konzeptionelle Fragen sollten an den passenden Stellen in Ihrem Vortrag in angemessenem Umfang erscheinen. Entscheiden Sie nach eigenem Ermessen, ob Sie im Rahmen Ihres Abschlussvortrags auch auf konzeptionelle Fragen eingehen. Die Antworten auf die Implementierungsaufgaben werden durch Ihren Code reflektiert.
Hinweis: Arbeiten Sie für diese Aufgabe mit Fließkommazahlen einfacher Genauigkeit.

2.3.1 Theoretischer Teil

- Was muss für die Breiten und Höhen zweier Matrizen gelten, sodass diese überhaupt miteinander multipliziert werden können?
- Ziehen Sie geeignete Sekundärliteratur hinzu, um die Funktionsweise des genannten Formats zu verstehen. Welche Werte werden gespeichert?

- Überlegen Sie sich, wie die Multiplikation aussehen muss.
- Untersuchen Sie die Performanz Ihrer Implementierungen anhand von hinreichend großen Matrizen unterschiedlicher Dichte.
- Nutzen Sie für Ein- und Ausgabedatei folgendes Textformat:

```
LINE  |  CONTENT
   1  |  <noRows>,<noCols>
   2  |  <values>
   3  |  <row_indices>
   4  |  <col_ptr>
```

Beispiel: Die zur Matrix

$$\begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 1 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

gehörige Textdatei sieht wie folgt aus:

```
4,4
5,0.5,6,1,3
0,2,1,1,3
0,2,3,4,5
```

2.3.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, die Ihr entworfenes Eingabeformat verarbeiten und mit welchen der Benutzer zwei zu multiplizierende Matrizen an Ihre C-Funktion übergeben kann.
- Implementieren Sie in der Datei mit Ihrem C-Code die Funktion:

```
void matr_mult_csc(const void* a, const void* b, void* result)
```

Die Funktion bekommt zwei Matrizen `a` und `b` im *Compressed Sparse Column* Format, sowie einen Pointer auf freien Speicherbereich `result` übergeben. Das Ergebnis der Multiplikation von `a` und `b` wird in `result` im gleichen Format gespeichert. Die Werte der Matrizen sollen im Datentyp `float` gespeichert und verarbeitet werden, die Dimensionen der Matrizen als `uint64_t` (d.h. die maximale Größe einer Matrix ist $\left(2^{64}-1\right) \times \left(2^{64}-1\right)$).

Hinweis: Sie können die Datenstruktur für die Matrizen frei bestimmen und dort auch Metadaten speichern, sofern die eigentlichen Daten im geforderten Format abgespeichert sind.

2.3.3 Rahmenprogramm

Ihr Rahmenprogramm muss bei einem Aufruf die folgenden Optionen entgegennehmen und verarbeiten können. Wenn möglich soll das Programm sinnvolle Standardwerte definieren, sodass nicht immer alle Optionen gesetzt werden müssen. Wird eine Option mit Argument gesetzt, so ist das entsprechende Argument zu benutzen. Die Reihenfolge der Optionen bei einem gültigen Aufruf muss irrelevant sein. Wir empfehlen Ihnen, die Kommandozeilenparameter mittels `getopt_long`[2] zu parsen.

- `-V <Zahl>` — Die Implementierung, die verwendet werden soll. Hierbei soll mit `-V 0` Ihre Hauptimplementierung verwendet werden. Wenn diese Option nicht gesetzt wird, soll ebenfalls die Hauptimplementierung ausgeführt werden.
- `-B<Zahl>` — Falls gesetzt, wird die Laufzeit der angegebenen Implementierung gemessen und ausgegeben. Das *optionale* Argument dieser Option gibt die Anzahl an Wiederholungen des Funktionsaufrufs an, z. B. `-B` oder `-B5` .
- `-a <Dateiname>` — Eingabedatei, welche die Matrix A enthält
- `-b <Dateiname>` — Eingabedatei, welche die Matrix B enthält
- `-o <Dateiname>` — Ausgabedatei
- `-h|--help` — Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet. Die Beschreibung soll mit "Help Message" anfangen.

Sie dürfen weitere Optionen implementieren, beispielsweise um vordefinierte Testfälle zu verwenden. Ihr Programm muss jedoch nur unter Verwendung der oben genannten Optionen verwendbar sein. Beachten Sie ebenfalls, dass Ihr Rahmenprogramm etwaige Randfälle korrekt abfangen muss und im Falle eines Fehlers mit einer Fehlermeldung, die auf den konkreten Fehler hinweist und einer kurzen Erläuterung zur Benutzung **terminieren** sollte. Fehlermeldungen jeglicher Art sollen auf `stderr` ausgegeben werden. Die Wertebereiche der Argumente dürfen nicht unnötig eingeschränkt werden.

2.4 Allgemeine Bewertungshinweise

Beachten Sie grundsätzlich alle in der Praktikumsordnung angegebenen Hinweise. Die folgende Liste konkretisiert einige der Bewertungspunkte:

- Diese PDF-Datei darf nicht verändert, gelöscht oder umbenannt werden.
- Die in `README.md` geforderte Struktur des Projektrepositorys muss eingehalten werden.

[2] Siehe `man 3 getopt_long` für Details

- Der Exit-Code Ihrer Implementierung muss die erfolgreiche Ausführung oder das Auftreten eines Fehlers widerspiegeln. Benutzen Sie dafür die Makros aus `stdlib.h`.
- Stellen Sie unbedingt sicher, dass Ihre Implementierung auf der Referenzplattform des Praktikums (`lxhalle`) kompiliert und vollständig funktionsfähig ist.
- Die Implementierung soll mit GCC/GNU as kompilieren. Verwenden Sie keinen Inline-Assembler. Achten Sie darauf, dass Ihr Programm keine x87-FPU- oder MMX-Instruktionen und SSE-Erweiterungen nur bis SSE4.2 verwendet. Andere ISA-Erweiterungen (z.B. AVX, BMI1) dürfen Sie nur benutzen, sofern Ihre Implementierung auch auf Prozessoren ohne derartige Erweiterungen lauffähig ist.
- Sie dürfen die angegebenen Funktionssignaturen, bis auf die Änderung des Funktionsnamens für die Benennung der unterschiedlichen Implementierungen (siehe nächster Punkt) *nicht* ändern.
- Verwenden Sie die angegebenen Funktionsnamen für Ihre Hauptimplementierung. Für alle weiteren Implementierungen gilt folgendes Schema: Hinter den Funktionsnamen wird das Suffix „`_V1`“, „`_V2`“, usw. angehängt.
- Denken Sie daran, das Laufzeitverhalten Ihres Codes zu testen (Sichere Programmierung, Korrektheit/Genauigkeit, Performanz) und behandeln Sie *alle möglichen Eingaben*, auch Randfälle. Ziehen Sie ggf. alternative Implementierungen als Vergleich heran.
- Eingabedateien, welche Sie generieren, um Ihre Implementierungen zu testen, sollten mit abgegeben werden; größere Eingaben sollten stattdessen stark komprimiert oder (bevorzugt) über ein abgegebenes Skript generierbar sein.
- Stellen Sie Performanz-Ergebnisse nach Möglichkeit grafisch dar.
- Vermeiden Sie unscharfe Grafiken und Screenshots von Code.
- Geben Sie die Folien für Ihre Abschlusspräsentation im PDF-Format ab. Achten Sie auf hinreichenden Kontrast (schwarzer Text auf weißem Grund!) und eine angemessene Schriftgröße. Verwenden Sie 16:9 als Folien-Format und fügen Sie Foliennummern hinzu.