LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Aspekte der systemnahen Programmierung bei der Spieleentwicklung

Multiplikation dünnbesetzter Matrizen (A312)
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 Assembler-Code anzufertigen ist, sind für die 64-Bit ARMv8-Architektur (AArch64) zu schreiben.

Der **Abgabetermin** ist der **3. Februar 2019, 23:59 Uhr (MEZ)**. Die Abgabe erfolgt per Git in für Ihre Gruppe eingerichtete Projektrepository. Bitte beachten Sie die in der Praktikumsordnung angebene Liste von abzugebenden Dateien.

Der **erste Teil Ihrer Projektpräsentation** ist eine kurze Vorstellung Ihrer Aufgabe im Umfang von ca. 2 Minuten in einer Tutorübung in der Woche **10.12.2018 – 14.12.2018**. Erscheinen Sie bitte **mit allen Team-Mitgliedern** und wählen Sie bitte eine Übung, in der mindestens ein Team-Mitglied angemeldet ist.

Die **Abschlusspräsentationen** finden in der Woche vom **04.03.2019 – 08.03.2019** statt. Weitere Informationen zum Ablauf und die Zuteilung der einzelnen Präsentationstermine werden noch bekannt gegeben. Beachten Sie, dass die Folien für die Präsentation am obigen Abgabetermin im PDF-Format abzugeben sind.

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

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

Mit freundlichen Grüßen
Die Praktikumsleitung

PS: Vergessen Sie nicht, sich rechtzeitig in TUMonline zur Prüfung anzumelden. Dies ist Voraussetzung für eine erfolgreiche Teilnahme am Praktikum im laufenden Semester.

[1] `https://www.caps.in.tum.de/lehre/ws18/praktika/asp/`

2 Multiplikation dünnbesetzter Matrizen

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.
Das einfachste Format solcher dünnbesetzer Matrizen ist das so genannte *Coordinate Scheme*, welches neben jedem Wert $\neq 0$ noch die jeweilige Zeile und Spalte des Eintrags in der Matrix speichert. Man betrachte das folgende Beispiel:

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

Diese Matrix würde als *Coordinate Scheme* mit Tripeln (Zeile, Spalte, Wert) folgendermaßen aussehen:

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

Ein weiteres Format zum Speichern dünnbesetzter Matrizen ist das *Jagged Diagonal Storage* Schema.
Ihre Aufgabe wird es sein, die Multiplikation zweier Matrizen in beiden Formaten 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. Alle Antworten auf konzeptionelle

Fragen sollten in Ihrer Ausarbeitung erscheinen. Besprechen Sie nach eigenem Ermessen außerdem im Zuge Ihres Vortrags einige der konzeptionellen Fragen. Die Antworten auf die Implementierungsaufgaben werden durch Ihrem Code reflektiert.
Hinweis: Arbeiten Sie für diese Aufgabe mit 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 zweiten genannten Formats zu verstehen. Welche Werte werden gespeichert? Für welche Art von Matrizen bietet sich das jeweilige Format an?
- Überlegen Sie sich für beide Matrixformate, wie die Multiplikation aussehen muss.
- Entwerfen Sie ein eigenes, sinnvolles Format, in welchem der Nutzer beliebige Matrizen an Ihre Implementierung als Textdatei übergeben kann.
- Berechnen Sie die Peak-Performance der Matrixmultiplikation für zwei sehr große Matrizen in Floating Point Operationen pro Sekunde (FLOP/s). Vergleichen Sie diese mit der Performance wenn mit vollbesetzten Matrizen gerechnet wird.

2.3.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, mit welchen der Benutzer zwei zu multiplizierende Matrizen an Ihre Assemblerimplementierung übergeben kann.
- Implementieren Sie eine Funktion

  ```
  void matr_mult_coos(float *a, float *b, float *result)
  ```

  in Assembler, welche zwei Matrizen `a` und `b` im *Coordinate Scheme*, sowie einen Pointer auf freien Speicherbereich `result` übergeben bekommt, welche das Ergebnis der Multiplikation von `a` und `b` in `result` im entsprechenden Format speichert.
- Implementieren Sie eine Funktion

  ```
  void matr_mult_jds(float *a, float *b, float *result)
  ```

  in Assembler, welche zwei Matrizen `a` und `b` im *Jagged Diagonal Storage* Format, sowie einen Pointer auf freien Speicherbereich `result` übergeben bekommt, welche das Ergebnis der Multiplikation von `a` und `b` in `result` im entsprechenden Format speichert.

2.4 Checkliste

Die folgende Liste soll Ihnen als Gedächtnisstütze beim Bearbeiten der Aufgaben dienen. Beachten Sie ebenfalls die in der Praktikumsordnung angegebenen Hinweise. Sollten Sie eine Reverse-Engineering-Aufgabe erhalten haben, sind diese Punkte für Sie weitestgehend hinfällig.

- Verwenden Sie keinen Inline-Assembler.
- I/O-Operationen dürfen grundsätzlich in C implementiert werden.
- Sie dürfen die Signatur der in Assembler zu implementierenden Funktion nur dann ändern, wenn Sie dies (in Ihrer Ausarbeitung) rechtfertigen können.
- Denken Sie daran, das Laufzeitverhalten Ihres Codes zu testen (Sichere Programmierung, Performanz) und behandeln Sie alle möglichen Eingaben, auch Randfälle. Ziehen Sie ggf. alternative Implementierungen als Vergleich heran.
- Verwenden Sie SIMD-Befehle, wenn möglich.
- Fügen Sie Ihrem Projekt ein funktionierendes `Makefile` hinzu, welches durch den Aufruf von `make` Ihr Projekt kompiliert.
- Eingabedateien, welche Sie generieren, um Ihre Implementierungen zu testen, sollten mit abgegeben werden.
- Verwenden für die Ausarbeitung Sie die bereitgestellte TEX-Vorlage und legen Sie sowohl die PDF-Datei als auch sämtliche TEX-Quellen in das Repository.
- Stellen Sie Performanz-Ergebnisse grafisch oder tabellarisch dar.
- Vermeiden Sie unscharfe Grafiken und Screenshots von Code.
- Bonusaufgaben (sofern vorhanden) müssen nicht implementiert werden.
- 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 4:3 als Folien-Format.