LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Bildkompression (A203)
Projektaufgabe – Aufgabenbereich Bildverarbeitung

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 x86-Architektur (x86-64) unter Verwendung der SSE-Erweiterungen zu schreiben; alle anderen Bestandteile der Hauptimplementierung sind in C nach dem C11-Standard anzufertigen.

Der **Abgabetermin** ist der **07. Februar 2022, 11:59 Uhr (MEZ)**. Die Abgabe erfolgt per Git in das für Ihre Gruppe eingerichtete Projektrepository. Bitte beachten Sie die in der `README.md` angegebene Liste von abzugebenden Dateien.

Die **Abschlusspräsentationen** finden in der Zeit vom **07.03.2022 – 11.03.2022** statt. Weitere Informationen werden noch bekannt gegeben. Beachten Sie, dass die Folien für die Präsentation am obigen Abgabetermin im PDF-Format abzugeben sind.

Sofern die Rahmenbedingungen (Hygiene-, Abstandsregeln etc.) für eine Präsenzprüfung nicht vorliegen, kann gemäß §13a APSO die geplante Prüfungsform auf eine virtuelle Präsentation (Videokonferenz) oder eine kurze schriftliche Fernprüfung umgestellt werden. Die Entscheidung über diesen Wechsel wird möglichst zeitnah, spätestens jedoch 14 Tage vor dem Prüfungstermin nach Abstimmung mit dem zuständigen Prüfungsausschuss bekannt gegeben.

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

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

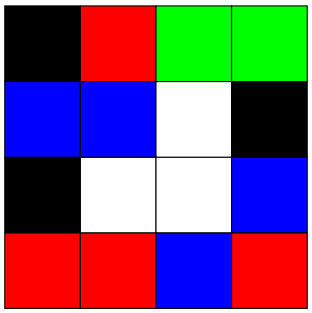
2 Bildkompression

2.1 Überblick

Im Zuge Ihrer Projektaufgabe werden Sie theoretisches Wissen aus der Mathematik im Anwendungszusammenhang verwenden, um einen Algorithmus in Assembler zu implementieren. Sie konzentrieren sich dabei auf das Feld des *Image Processing*, in welchem Pixelbilder, wie sie typischerweise Digitalkameras produzieren, als Eingabe für bestimmte Algorithmen verwendet werden und mathematische Überlegungen dadurch sichtbar gemacht werden.

2.2 Funktionsweise

Die Lauflängencodierung (Run-Length-Encoding) ist ein einfaches Verfahren zur Datenkompression. Betrachten Sie beispielsweise die 16 Pixel des folgenden Bildes:



Anstatt die Pixel einzeln abzuspeichern, kann man auch *Folgen* von gleichen Pixeln speichern. Im konkreten Fall scannt man das Bild zeilenweise von links oben nach rechts unten und speichert Tupel $T = (L, V)$ mit L hintereinander auftretenden Werten $V \in \{R, G, B, W, S\}$:

$$RLE = ((1,S),(1,R),(2,G),(2,B),(1,W),(2,S),(2,W),(1,B),(2,R),(1,B),(1,R)) \tag{1}$$

Die Werte lassen sich dann wiefolgt lesen:

- $(1, S)$ Ein Pixel Schwarz
- $(1, R)$ Ein Pixel Rot
- $(2, G)$ Zwei Pixel Grün
- ...

So lässt sich aus den komprimierten Daten das Bild Stück für Stück wieder rekonstruieren.
Ihre Anwendung findet die Lauflängencodierung beispielsweise im BMP-Format für Pixelgrafiken. Ihr Ziel ist es, ein Programm zum Komprimieren von BMP-Grafiken zu erstellen.

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 Ihrer Ausarbeitung 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 Ihrem Code reflektiert.

2.3.1 Theoretischer Teil

- Erarbeiten Sie anhand geeigneter Sekundärliteratur die genaue Funktionsweise der Lauflängenkodierung.
- Das Bitmap-Format ist gut dokumentiert. Erarbeiten Sie mithilfe einer geeigneten Quelle die Dateiformate für unkomprimierte und RLE-komprimierte Bitmaps mit indizierten 8 Bit pro Pixel. (Eine mögliche Quelle für das komprimierte Datenformat ist folgender MSDN-Artikel: `http://msdn.microsoft.com/en-us/library/windows/desktop/dd183383.aspx`) Ihr Programm muss explizit nur das 8bpp-BMP-Format abdecken! Sie haben das beim Image-Processing üblicherweise verwendete Testbild in komprimierter und unkomprimierter Fassung zur Dokumentation erhalten.
- Finden Sie eine möglichst genaue obere Schranke für die Größe der komprimierten Daten.
- Berechnen Sie Kompressionsrate und Laufzeit Ihres Algorithmus. Verifizieren Sie dessen Korrektheit, indem Sie das Ergebnis in einem Bildbetrachter (z.B. GIMP) öffnen.

2.3.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, mit welchen Sie eine BMP-Datei einlesen können und Ihrer C-Funktion einen Pointer auf die sequentiellen Bilddaten sowie die relevanten Metadaten (z.B. Höhe und Breite) übergeben können. Das Rahmenprogramm sollte dann ein neue BMP-Datei erstellen und die berechneten Werte nach dem Aufruf dort hineinschreiben. Das Ergebnis sollte mit den üblichen Bildbetrachtern lesbar sein.
- Implementieren Sie in der Datei mit Ihrem C-Code die Funktion:

```
size_t bmp_rle(const uint8_t* img_in, size_t width, size_t height,
                        uint8_t* rle_data)
```

Die Funktion nimmt dabei einen Pointer auf die unkomprimierten Pixelwerte, sowie weitere Metadaten als Parameter, und schreibt die nach BMP-Spezifikation lauflängenkodierten Daten in das dafür vorgesehene Array und gibt deren Länge zurück. Allokieren Sie hierzu in Ihrem Rahmenprogramm einen Buffer passender Größe.

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.

- `-V<int>` — Die Implementierung, die verwendet werden soll. Hierbei soll mit `-V0` Ihre Hauptimplementierung verwendet werden. Wenn diese Option nicht gesetzt wird, soll ebenfalls die Hauptimplementierung ausgeführt werden.
- `-B<int>` — Falls gesetzt, wird die Laufzeit der angegebenen Implementierung gemessen und ausgegeben. Das optionale Argument dieser Option gibt die Anzahl an Wiederholungen des Funktionsaufrufs an.
- `<file>` — Positional Argument: Eingabedatei
- `-o<file>` — Ausgabedatei
- `-h` — Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet.
- `--help` — Eine Beschreibung aller Optionen des Programms und Verwendungsbeispiele werden ausgegeben und das Programm danach beendet.

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 aussagekräftigen Fehlermeldung auf `stderr` und einer kurzen Erläuterung zur Benutzung terminieren sollte.

2.4 Allgemeine Bewertungshinweise

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

- Stellen Sie unbedingt sicher, dass *sowohl* Ihre Implementierung *als auch* Ihre Ausarbeitung auf der Referenzplattform des Praktikums (`lxhalle`) kompilieren und vollständig korrekt bzw. funktionsfähig sind.

- Die Implementierung soll mit GCC/GNU as kompilieren. 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 (nur dann) ändern, wenn Sie dies (in Ihrer Ausarbeitung) begründen.
- Verwenden Sie die angegebenen Funktionsnamen für Ihre Hauptimplementierung. Falls Sie mehrere Implementierungen schreiben, legen wir Ihnen nahe, für die Benennung der alternativen Implementierungen mit dem Suffix „`_V1`“, „`_V2`“ etc. zu arbeiten.
- 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.
- 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.