LEHRSTUHL FÜR RECHNERARCHITEKTUR UND PARALLELE SYSTEME

Grundlagenpraktikum: Rechnerarchitektur

Fast Inverse Square Root (A300)
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 16. Juli 2023, 23:59 Uhr (CEST)**. 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 **21.08.2023 – 01.09.2023** 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 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 Fast Inverse Square Root

2.1 Überblick

Im Zuge Ihrer Projektaufgabe werden Sie sich mit einem berühmten Algorithmus aus der Computergrafik befassen und mithilfe eines Microbenchmarks einen Effizienzvergleich zu konventionellen Ansätzen durchführen. Sie werden außerdem im Zuge einer Evaluation die Genauigkeit der Ergebnisse weiter verbessern.

2.2 Der Algorithmus

Der *Fast Inverse Square Root* Algorithmus adressiert (wie der Name nahelegt) das Problem, zu einer gegebenen Zahl x die Zahl y zu finden, für die gilt:

$$y = \frac{1}{\sqrt{x}}$$

In der Computergrafik ist von Interesse, ein solches y möglichst schnell zu berechnen. Gleichzeitig spielt es aber für die menschliche Wahrnehmung keine große Rolle, dass die Zahl y sehr genau ist, ein ungefährer Wert genügt.
Der *Fast Inverse Square Root* Algorithmus erschien erstmals im Quelltext für das Spiel *Quake III Arena*. Die Hauptidee ist es, die IEEE754-Darstellung einer Fließkommazahl auszunutzen. Der Algorithmus lässt sich in einer Gleichung beschreiben als

$$y = \frac{1}{\sqrt{x}} \approx \text{MagicNumber} - (x \gg 1) \tag{1}$$

wobei $\gg$ einen logischen Rechtsshift auf Bitebene bezeichnet und MagicNumber eine (von Ihnen zu bestimmende) Konstante darstellt.
Damit ist der (zugegebenermaßen relativ triviale) Algorithmus grob umrissen. An dieser Stelle wird bewusst nicht genauer auf die Funktionsweise eingegangen, da das Verstehen derselben Teil der Aufgabenstellung ist (siehe unten). Ziehen Sie zum genauen Verständnis des Algorithmus geeignete Sekundärliteratur hinzu.

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 Ihren Code reflektiert.

2.3.1 Theoretischer Teil

- Begründen Sie die Motivation, das y mit den oben genannten Eigenschaften möglichst schnell zu berechnen. Denken Sie dabei an den Kontext der Computergrafik (insbesondere Vektorrechnung).
- Machen Sie sich mit dem IEEE-754 Standard zur Darstellung von Fließkommazahlen vertraut und begründen Sie daraus, warum die Näherung in Gleichung 1 überhaupt gelten kann.
- Entwickeln Sie ein Verfahren, mit welchem Sie „Ihre“ MagicNumber bestimmen. Das von Ihnen entwickelte Verfahren muss nicht den optimalen Wert liefern. Analysieren Sie die Güte der von Ihnen bestimmten MagicNumber.
- Entwickeln Sie ein Verfahren, mit welchem Sie Geschwindigkeit und Güte der Ergebnisse des *Fast Inverse Square Root* Algorithmus wissenschaftlich vergleichen können.
- Vergleich Sie die Geschwindigkeit und Güte Ihres Algorithmus mit Alternativen unter gleichen Hardwarebedingungen. Vergleichen Sie mit einer äquivalenten Version in C die eine Wurzelfunktion enthalten.
- Beschreiben Sie, wie die Software-Implementierung der Wurzelfunktion in einer aktuellen Version der C-Standardbibliothek *glibc* realisiert ist.

2.3.2 Praktischer Teil

- Implementieren Sie im Rahmenprogramm I/O-Operationen in C, mit welchen Sie Ihrer C-Funktion eine beliebig große Zahl von x_i Eingabewerten übergeben können. Sie sind frei in Ihrer Wahl der Mittel beim Einlesen und Generieren der Testwerte. Das Rahmenprogramm soll die berechneten Ergebnisse anschließend wieder ausgeben.
- Implementieren Sie in der Datei mit Ihrem C-Code die Funktion:

  ```
  void fastInvSqrt_flt(size_t n, float vals[n], float out[n])
  ```

  Die Funktion nimmt dabei einen Pointer auf die Fließkomma-Testwerte der Länge n als Argument und schreibt die mit den *Fast Inverse Square Root* Algorithmus berechneten Wurzelwerte in das Ausgabearray. Allokieren Sie hierzu in Ihrem Rahmenprogramm einen Buffer passender Größe.
- Die Berechnung soll ebenfalls mit 64-Bit Fließkommazahlen durchführbar sein. Implementieren Sie in der Datei mit Ihrem C-Code die Funktion:

  ```
  void fastInvSqrt_dbl(size_t n, double vals[n], double out[n])
  ```

Die Funktion nimmt dabei einen Pointer auf die Fließkomma-Testwerte der Länge n als Argument und schreibt die mit den *Fast Inverse Square Root* Algorithmus berechneten Wurzelwerte in das Ausgabearray. Allokieren Sie hierzu in Ihrem Rahmenprogramm einen Buffer passender Größe.

- Implementieren Sie in der Datei mit dem C-Code das von Ihnen entwickelte Verfahren, um die MagicNumber für 32- und 64-Bit Fließkommazahlen zu berechnen. Ermöglichen Sie in Ihrem Rahmenprogramm eine Ausgabe der berechneten Werte, damit diese anschließend manuell in die Funktion zur eigentlichen Berechnung übernommen werden kann.
- Evaluieren Sie Ihre Lösung im Hinblick auf etwaige Optimierungsmöglichkeiten. Können Sie eventuell Gebrauch von SIMD-Erweiterungen machen?

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<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.
- `<Dateiname>` — Positional Argument: Eingabedatei mit Fließkommazahlwerten
- `<Floating Point Zahl>` — Positional Argument: Beliebige Anzahl an Fließkommazahlwerten
- `-d` — Die Eingabewerte sollen als `double` interpretiert werden
- `-m` — Die Magic Number soll berechnet und ausgegeben werden
- `-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. 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 (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.