Aufgabe Quadratzwurzel

Die Quadratwurzel $\sqrt{a}$ einer reellen positiven Zahl a ist die Nullstelle des Polynoms $x^2 - a$. Durch Anwendung des Newton-Verfahrens erhält man das folgende Näherungsverfahren für einen Startwert x_0 und $n \geq 0$:

$$x_{n+1} = 0,5 \cdot (x_n + \frac{a}{x_n})$$

Je näher der Startwert an der Quadratwurzel ist, desto schneller konvergiert das Verfahren.

Da die Berechnung der Quadratwurzel in Abhängigkeit der Genauigkeit und Präzision der Zahlencodierung relativ zeitaufwendig ist, lohnt es sich sie zu optimieren. Für Anwendungen in der Computergrafik können geringfügige Ungenauigkeiten in Kauf genommen werden.

Wir betrachten für diese Aufgabe nur Gleitkommazahlen im IEEE 754-Standard single precicions (float). Ein geeigneter Startwert wird durch folgende Berechnung ermittelt:[1]

```
// ap sei ein Zeiger auf den float Wert a
// rp sei ein Zeiger auf den Wert der Wurzel
int *ai = reinterpret_cast<int *>(ap);
int *initial = reinterpret_cast<int *>(rp);
*initial =  (1 << 29) + (*ai >> 1) - (1 << 22) - 0x4C000;
// initial zurueck casten zu float *
// und Newton Verfahren durchfuehren
```

Der Cast eines int-Zeigers auf ein float-Zeiger ist undefiniertes Verhalten. Es funktioniert allerdings beim der GCC.

Für die folgenden Aufgaben sollen eine konstante Anzahl der Iterationen des Newton-Verfahrens durchgeführt werden. Der Wert a wird als Zeiger übergeben. Der Wert auf dem dieser zeigt wird, darf durch Ihre Implementierung nicht überschrieben werden. Die Anzahl Iterationen ist ein Template-Parameter. Die zu implementierenden Funktionen befinden sich in der Datei `sqrt_opt.h`. Das Programm mit der main-Methode für die Zeitmessung ist in `sqrt_opt.cc` enthalten.

Beachten Sie, dass keine unnötigen Berechnungen oder Speicherverschiebungen durchgeführt werden. Vergessen Sie nicht die Funktionalität Ihrer Implementierung mit einigen Werten in einem eigenen Testprogramm zu überprüfen.

Die Funktionen sollen so programmiert werden, dass der Compiler gut die SIMD-Operationen für die AVX-Erweiterungen des Prozessors erzeugen kann. Die Programme dürfen keine GCC-Systemfunktionen enthalten, die direkt spezifische AVX-Befehle erzeugen.

Die Funktionen in den folgenden Teilaufgaben dürfen nicht Funktionen der vorherigen Teilaufgaben aufrufen.

[1] Einzelheiten zur Berechnung finden sich in `https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Approximations_that_depend_on_the_floating_point_representation`

Teilaufgabe 1

Implementieren Sie die Funktion

```
float sqrt_1(float * a)
```

, welche die Quadratwurzel näherungsweise für einen float-Wert nach obiger Methode berechnet und zurück gibt.

Teilaufgabe 2

Implementieren Sie die Funktion

```
void sqrt_2(float * __strict__ a, float * __strict__ sqrt)
```

, welche die Quadratwurzel näherungsweise für vier float-Werte nach obiger Methode berechnet und zurück gibt. a muss ein Zeiger auf vier float-Werte sein, sqrt ist ein Ausgabeparameter, der auf einen Speicherbereich mit vier float-Werten zeigt, in denen die Ergebnisse geschrieben werden. Die vier Berechnungen dürfen nicht mit einer Schleife gemacht werden. Damit der Compiler die SIMD-Befehle erzeugen kann, muss die Berechnung so programmiert sein, dass vier gleiche Befehle (für a[0], a[1], a[2], a[3]) hintereinander stehen.

__strict__ besagt, dass die Zeiger (durch den Programmierer garantiert) nicht auf die selben Speicherbereiche zeigen. Dies ist nötig, damit der Compiler mehr Optimierungsmöglichkeiten hat.

Teilaufgabe 3

Implementieren Sie die Funktion

```
void sqrt3(vec4 * __strict__ a, vec4 * __strict__ sqrt)
```

, welche analog zu Aufgabe 2 wieder vier Quadratwurzeln zieht, die sich gepackt in vec4 befinden.

Teilaufgabe 4

Übersetzen Sie das Programm für die Zeitmessung auf den Rechnern in E203 mit

```
g++ -Wall -pedantic -march=native -mfpmath=sse -mavx2 -O3 sqrt_opt.cc
```

Messen Sie die Ausführungszeiten 10 mal mit dem übersetzen Programm und bilden sie für jede Funktion den Durchschnittswert.

Teilaufgabe 5

Erstellen Sie einen Bericht zu dieser Aufgabe mit dem Quelltext Ihrer Funktionen, den zugehörigem Teilen des Assemblercode und einer tabellarischen Gegenüberstellung der Ausführungszeiten.

Interpretieren Sie die Resultate. Wenn die Zeiten nahezu gleich sind: Warum sind sie gleich? Wenn eine Version deutlich schneller ist: Warum sind ist sie schneller als eine andere?