Habib Mubarak, Eraycan Gökcan

Aufgabenblatt 5

Hinweis: Für alle Ergebnisse wurde “make run-sse” verwendet.

Aufgabe 5.1

```
simdLen: 4

Time scalar:  79.69 ms
Time headers: 20.416 ms, speed up 3.90331
Time stdx:      20.4291 ms, speed up 3.9008
SIMD and scalar results are the same.
SIMD and scalar results are the same.
```

matrix.out

- Da simdLen == 4 und N == 1000 durch 4 teilbar => keine Restberechnung in der std::expertimental-Implementierung (bzw. stdx-Implementierung).
- Laufzeit von stdx und FVec sind fast identisch. Das liegt daran, dass die SIMD-Vektoren in beiden Implementierungen in diesem Fall die gleiche Größe haben.
- stdx und FVec haben einen Speedup von fast 4x, was der Theorie entspricht.

Aufgabe 5.2

```
Time scalar:     1693.3 ms.
Time stdx AOS:   1346.73 ms, speed up 1.25734.
Time stdx SOA:   595.487 ms, speed up 2.84356.
Time stdx AOSOA: 1216.08 ms, speed up 1.39243.
SIMD using AOS and scalar results are the same.
SIMD using SOA and scalar results are the same.
SIMD using AOSOA and scalar results are the same.
```

quadratic_equation.out

Beobachtung #1 - Keine Implentierung hat einen Speedup von 4x:

- Die CPU kann viel Zeit damit verbringen, auf Daten aus dem Arbeitsspeicher zu warten. SIMD kann Daten nur so schnell verarbeiten, wie sie geladen werden
- Nicht alle Anweisungen sind unbedingt SIMD
- Der Compiler kann den Skalar-Code optimieren, was den Unterschied verringern könnte

Beobachtung #2 - Von der Laufzeit her gilt hier SOA > AOSOA > AOS:

- AOS ist von der Laufzeit her am schlechtesten, denn z.B. kann reinterpret_cast nicht verwendet werden und stattdessen müssen Kopien gemacht werden => braucht Zeit (und Speicher)
- Es kann sein, dass SOA besser als AOSOA ist, da man hier keinen Pointer dereferenzieren muss (was Zeit benötigen könnte). Es kann auch sein, dass der Compiler besser darin ist, SOA zu optimieren.

Aufgabe 5.3

```
// Ergebnisse für Implementierung mit n = 1000 Iterationen:


Scalar part:
Results are correct!
Time: 7.19595 ms
SIMD part:
Results are NOT the same!
Time: 246.356 ms
Speed up: 0.0292096

The max difference is: 9.53674e-07

Scalar part:
Results are correct!
Time: 5.42439 ms
SIMD part:
Results are NOT the same!
Time: 251.695 ms
Speed up: 0.0215514

The max difference is: 4.76837e-07
```

newton.out

- Die größte Differenz bzw. der größte Fehler hier ist 4.76837e-07, also extrem klein und ist somit gut genug mit dem Ergebnis der Skalar-Version vergleichbar
- Die lange Laufzeit kann daran liegen, dass die Skalar-Version eventuell viel weniger Iterationen durchgehen muss als n = 1000.

```
// Ergebnisse für Implementierung mit Maske:

Scalar part:
Results are correct!
Time: 6.04061 ms
SIMD part:
Results are the same!
Time: 2.41798 ms
Speed up: 2.4982

The max difference is: 0
```

newton.out

- Man sieht einen Speedup von ungefähr 2.5x und nicht 4x. Außerhalb von den vorher genannten Gründen, kann dies an folgenden Gründen liegen:
- Die Funktionen für Masken wie where und any_of führen eventuell zu zusätzlichen Overhead
- Manche Lanes (bzw. manche Approximationen) sind schon fertig (bzw. präzise genug), während andere noch weiterrechnen müssen. Also werden somit nicht zu jeder Zeit alle SIMD-Lanes nützlich gemacht, denn die fertigen Lanes machen nichts.

Aufgabe 5.4

```
Indices: 24 13 94 33
Gather without masking: results are correct.
Gather with masking: results are correct.
Scatter with masking: results are correct.
```

random_access.out