Präsentation zum Praktikum “Einführung in die Rechnerarchitektur“

Moore Kurven

Anton Baumann Felix Brandis Michal Cizevskij

Problemstellung

Lösungsansatz

Performanz

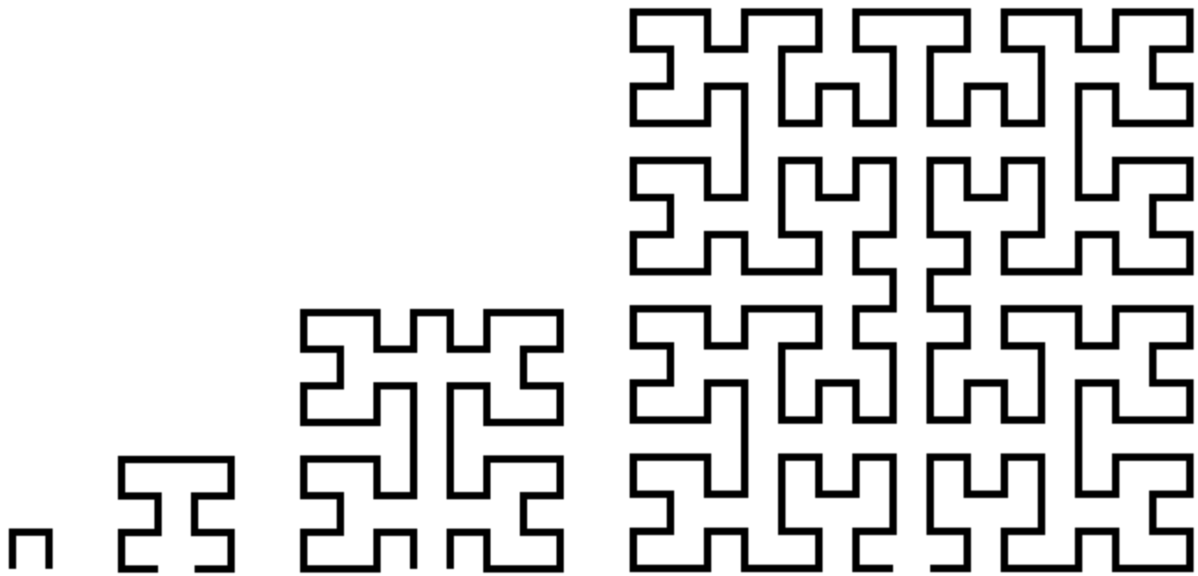
Problemstellung

Lösungsansatz

Performanz

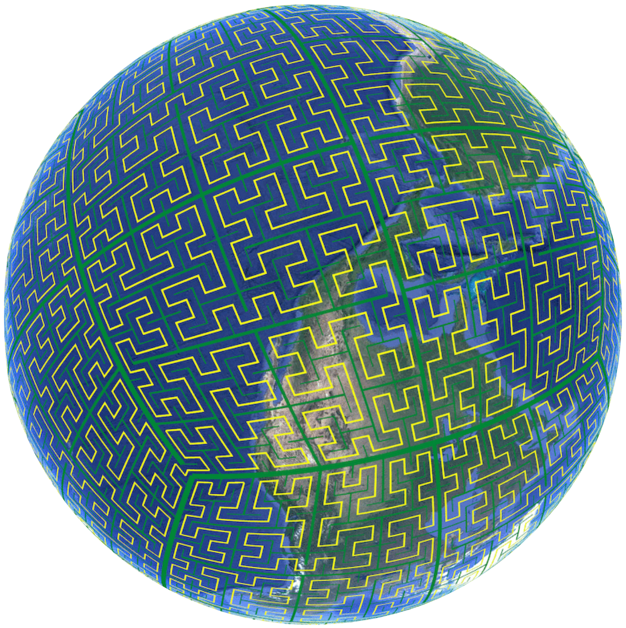
Moore Kurven

- Eliakim Hastings Moore (1862 – 1932)
- raumfüllend für Grad $n \to \infty$
- Seitenlänge: 2^n
- Anzahl Punkte : 4^n



Anwendungen

- bijektive Mappings zwischen $\mathbb{R}^1 \longleftrightarrow \mathbb{R}^2$ (auch für höhere Dimensionen)
- Dithering in der Bildverarbeitung
- Anordnung von Daten im Speicher
- spatial indexing (Google's S2 library)



Problemstellung

Lösungsansatz

Performanz

Lösungsansatz

Berechnung eines einzelnen Punkts:

Idee:

1. Hilbert Kurve des Grads n-1 ausrechnen
2. entsprechenden Punkt verschieben:

Lösungsansatz

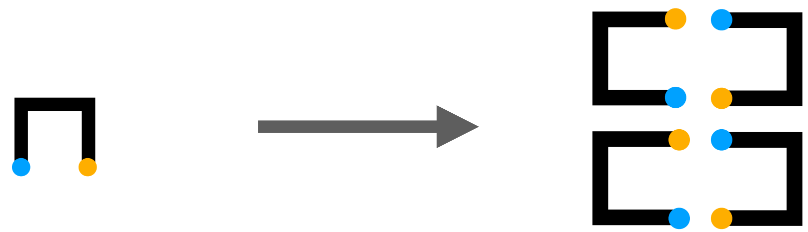
Berechnung eines einzelnen Punkts:

Idee:

1. Hilbert Kurve des Grads n-1 ausrechnen

2. entsprechenden Punkt verschieben:

- Startpunkt
- Endpunkt



Hilbert Kurve Grad n-1

Moore Kurve Grad n

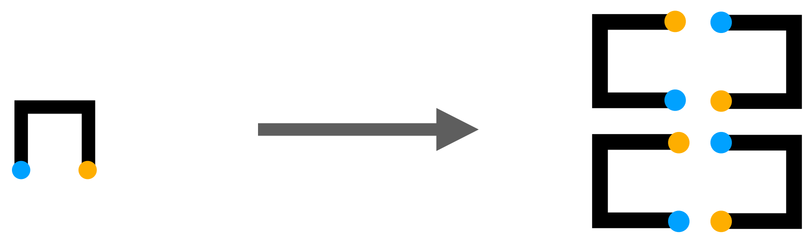
Lösungsansatz

Berechnung eines einzelnen Punkts:

Idee:

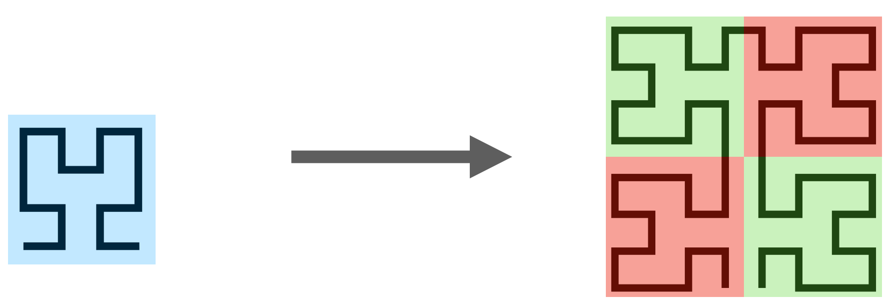
1. Hilbert Kurve des Grads n-1 ausrechnen
2. **entsprechenden Punkt verschieben:**

- Startpunkt
- Endpunkt



Hilbert Kurve Grad n-1 Moore Kurve Grad n

Beispiel:



Hilbert Kurve Grad 2 Moore Kurve Grad 3

Lösungsansatz

Berechnung eines einzelnen Punkts:

Idee:

1. Hilbert Kurve des Grads n-1 ausrechnen:

2. entsprechenden Punkt verschieben ✓

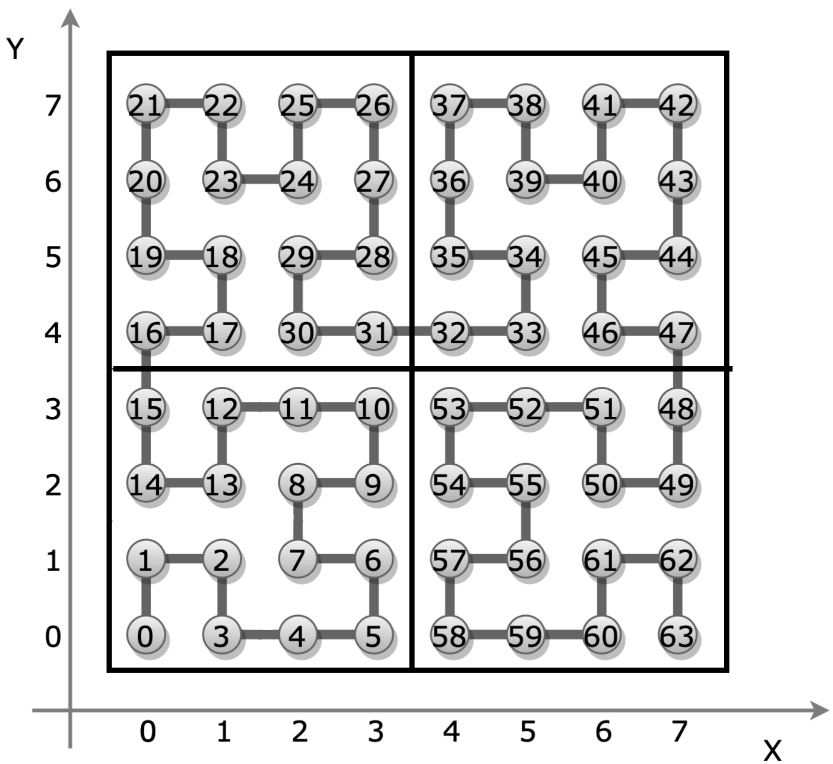
Lösungsansatz

Berechnung eines einzelnen Punkts:

Idee:

1. Hilbert Kurve des Grads n-1 ausrechnen:

2. entsprechenden Punkt verschieben



Hilbert Kurve Grad 3

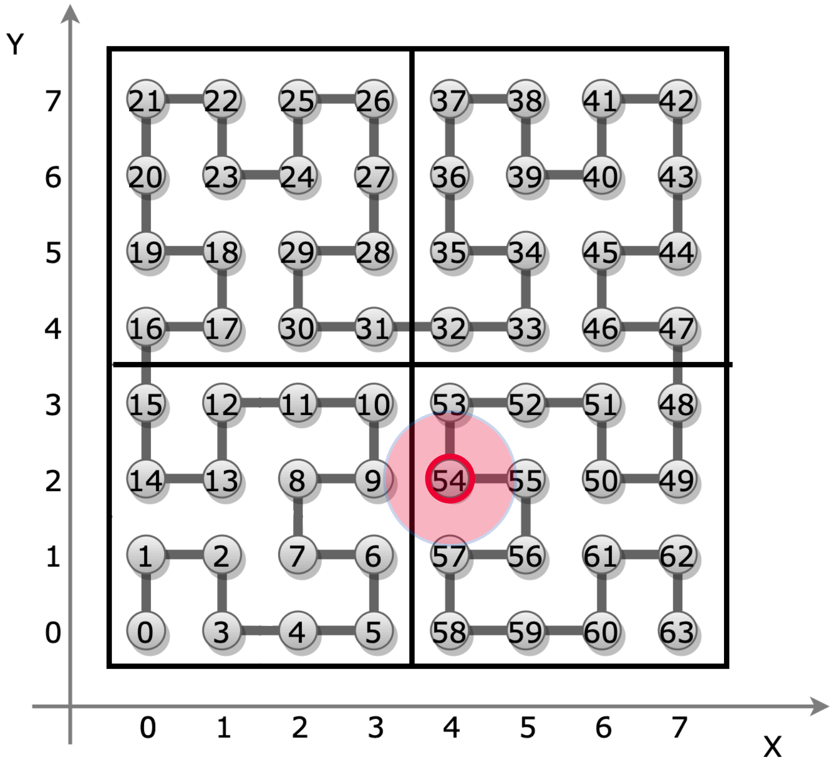
Lösungsansatz

Berechnung eines einzelnen Punkts:

Idee:

1. Hilbert Kurve des Grads n-1 ausrechnen

2. entsprechenden Punkt verschieben

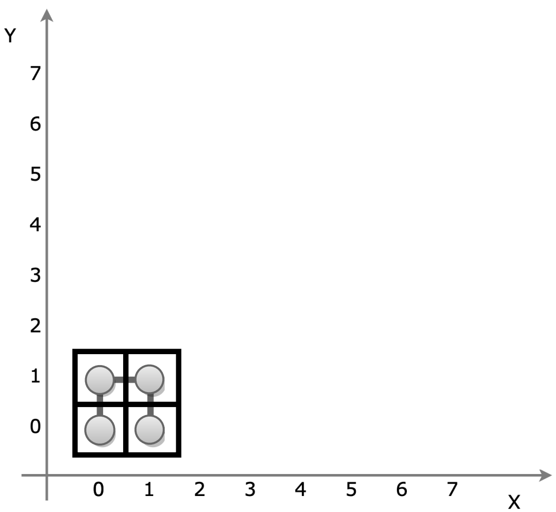


$54_{10} = 110110_2$

Hilbert Kurve Grad 3

Lösungsansatz

$$54_{10} = 110110_2$$

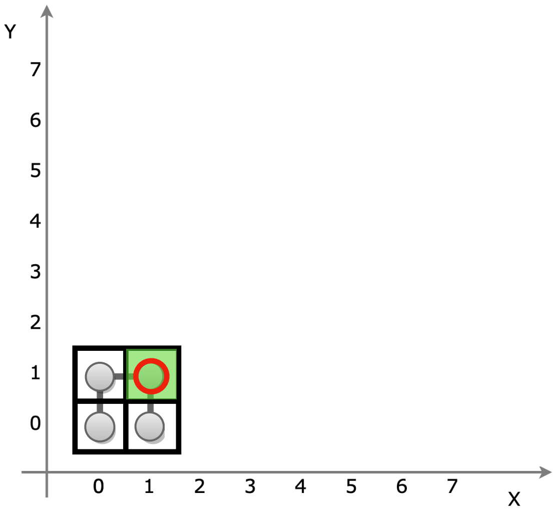


110110_2

01	10
00	11

Lösungsansatz

$$54_{10} = 110110_2$$

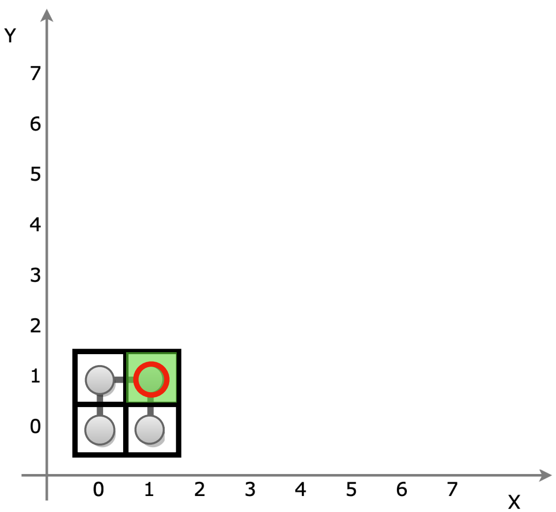


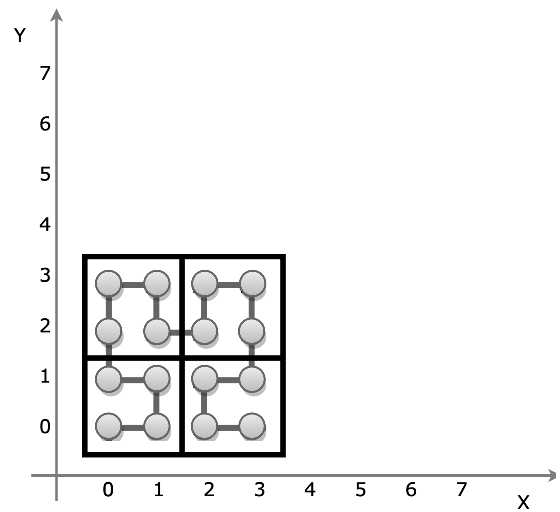
$1101\boxed{10}_2$

01	10
00	11

Lösungsansatz

$54_{10} = 110110_2$





110110_2

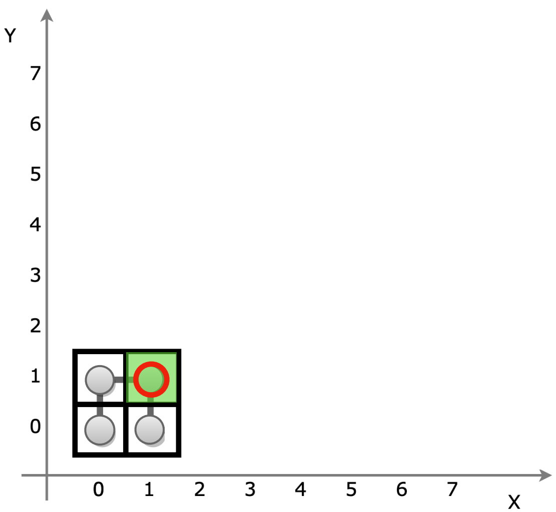
01	10
00	11

110110_2

01	10
00	11

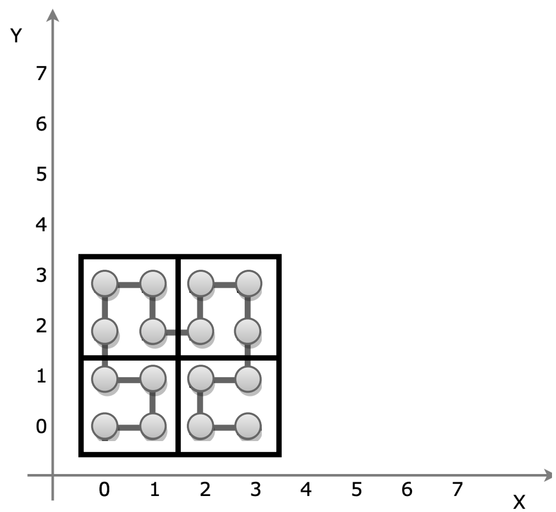
Lösungsansatz

$$54_{10} = 110110_2$$



$1101\mathbf{10}_2$

01	10
00	11

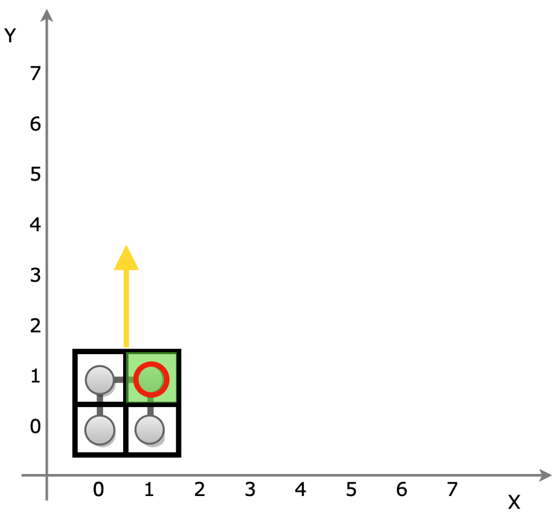


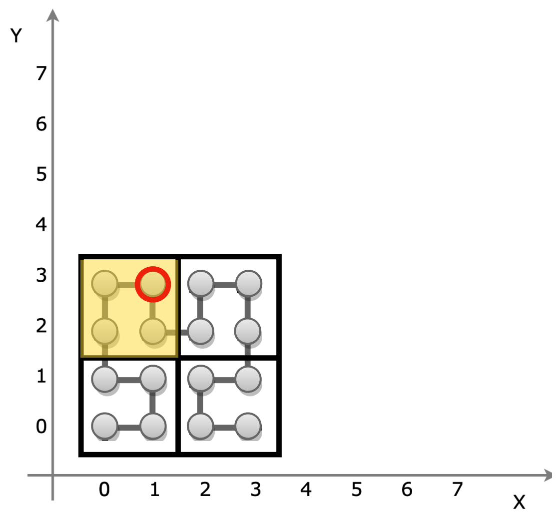
$11\mathbf{01}10_2$

01	10
00	11

Lösungsansatz

$$54_{10} = 110110_2$$





$1101\boxed{10}_2$

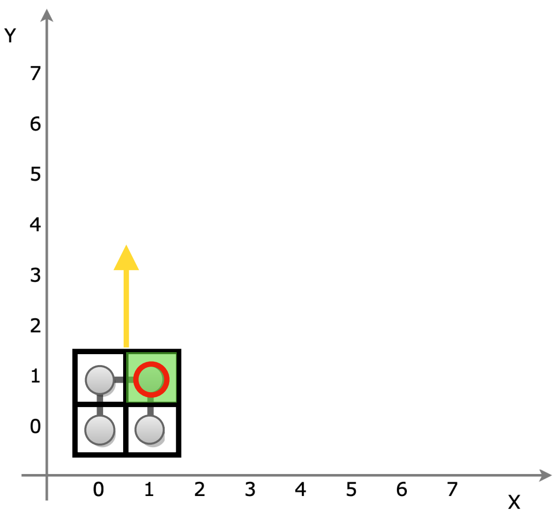
01	10
00	11

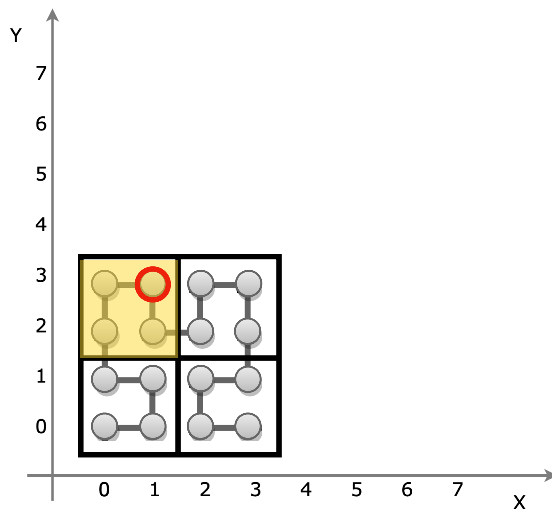
$11\boxed{01}10_2$

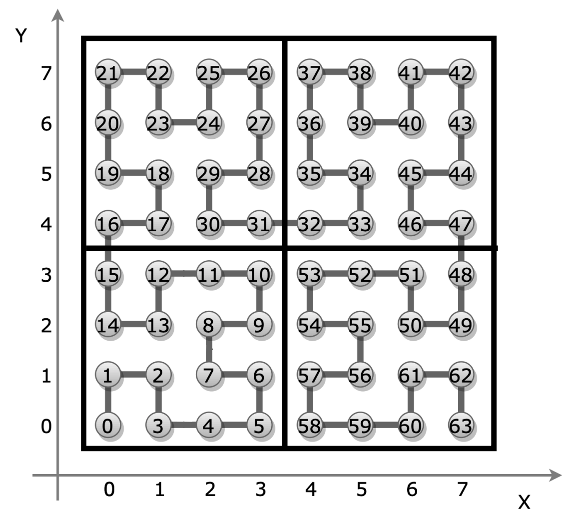
01	10
00	11

Lösungsansatz

$$54_{10} = 110110_2$$







110110_2

01	10
00	11

110110_2

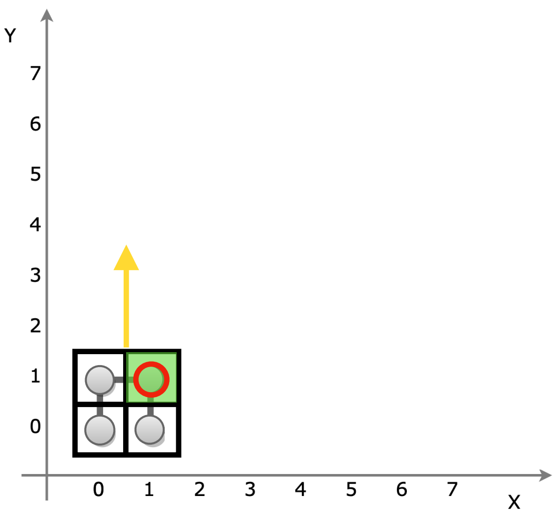
01	10
00	11

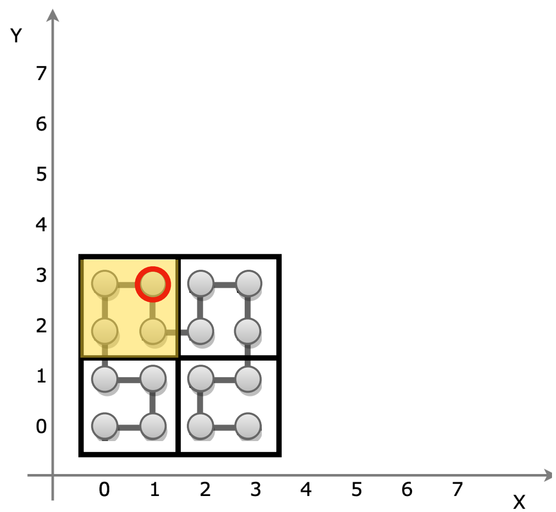
110110_2

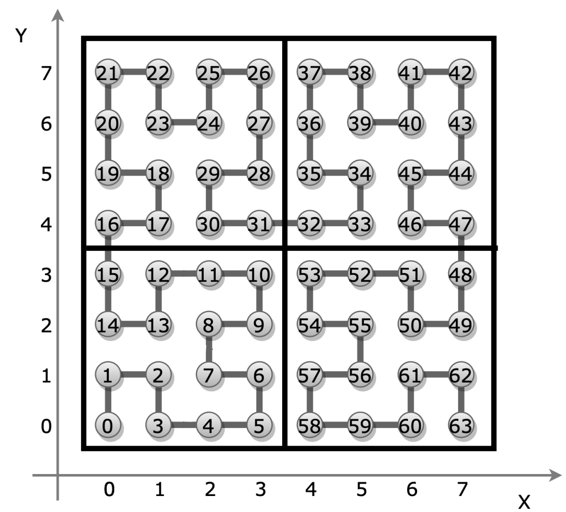
01	10
00	11

Lösungsansatz

$$54_{10} = 110110_2$$







110110_2

01	10
00	11

110110_2

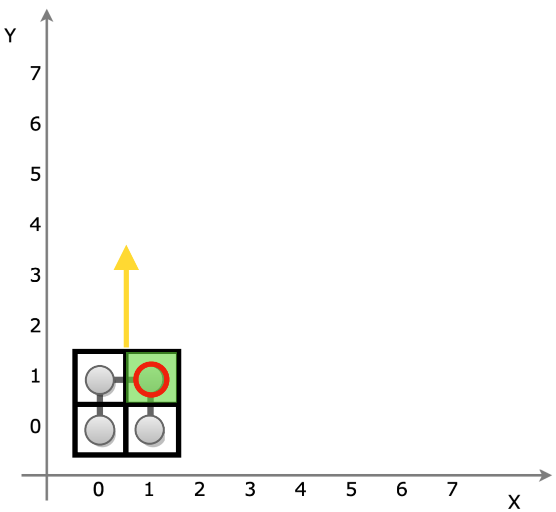
01	10
00	11

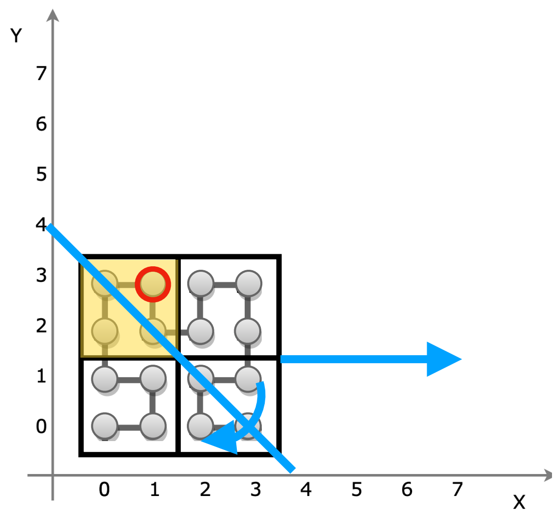
110110_2

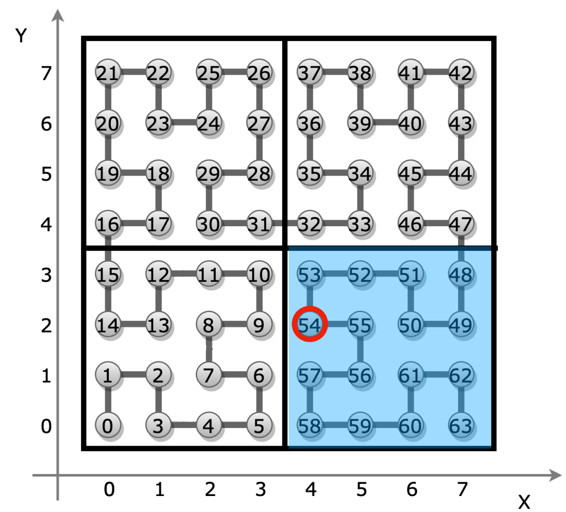
01	10
00	11

Lösungsansatz

$$54_{10} = 110110_2$$







$1101\,10_2$

01	10
00	11

$11\,01\,10_2$

01	10
00	11

$11\,0110_2$

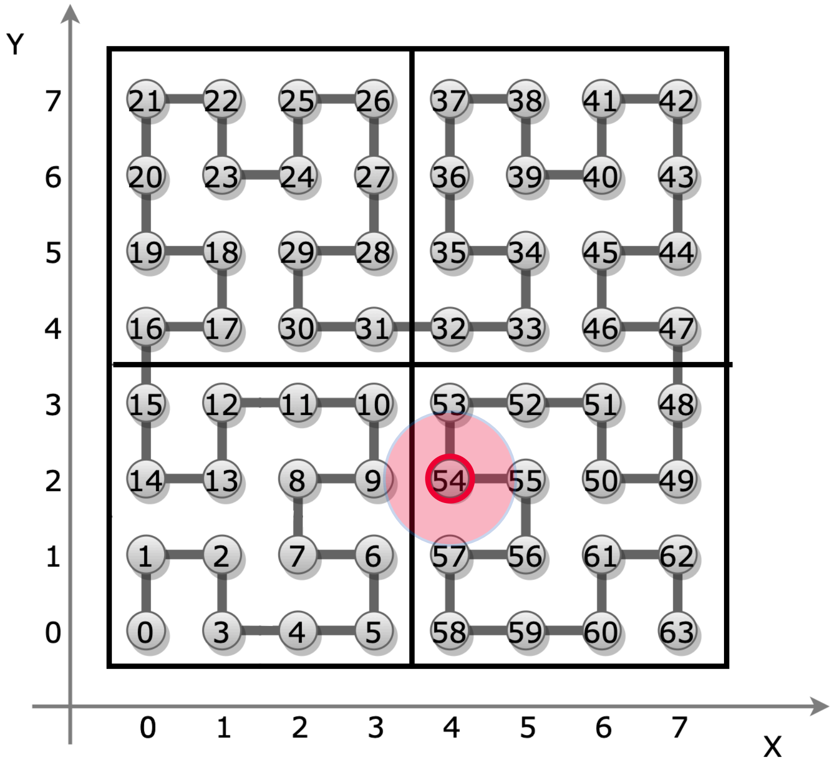
01	10
00	11

Lösungsansatz

Berechnung eines einzelnen Punkts:

Idee:

1. Hilbert Kurve des Grads n-1 ausrechnen
2. entsprechenden Punkt verschieben



$54_{10} = 110110_2$

Hilbert Kurve Grad 3

Lösungsansatz

Berechnung der gesamten Kurve:

Naiver Ansatz:

Wiederhole obigen Algorithmus für jeden Punkt

Lösungsansatz

Berechnung der gesamten Kurve:

Naiver Ansatz:

Wiederhole obigen Algorithmus für jeden Punkt

Nachteile:

- Neuberechnung für jeden der 4^n Punkte
- keine Wiederverwendung alter Rechenergebnisse
- Aufwand hängt von Grad der Kurve ab

Lösungsansatz

Berechnung der gesamten Kurve:

Naiver Ansatz:

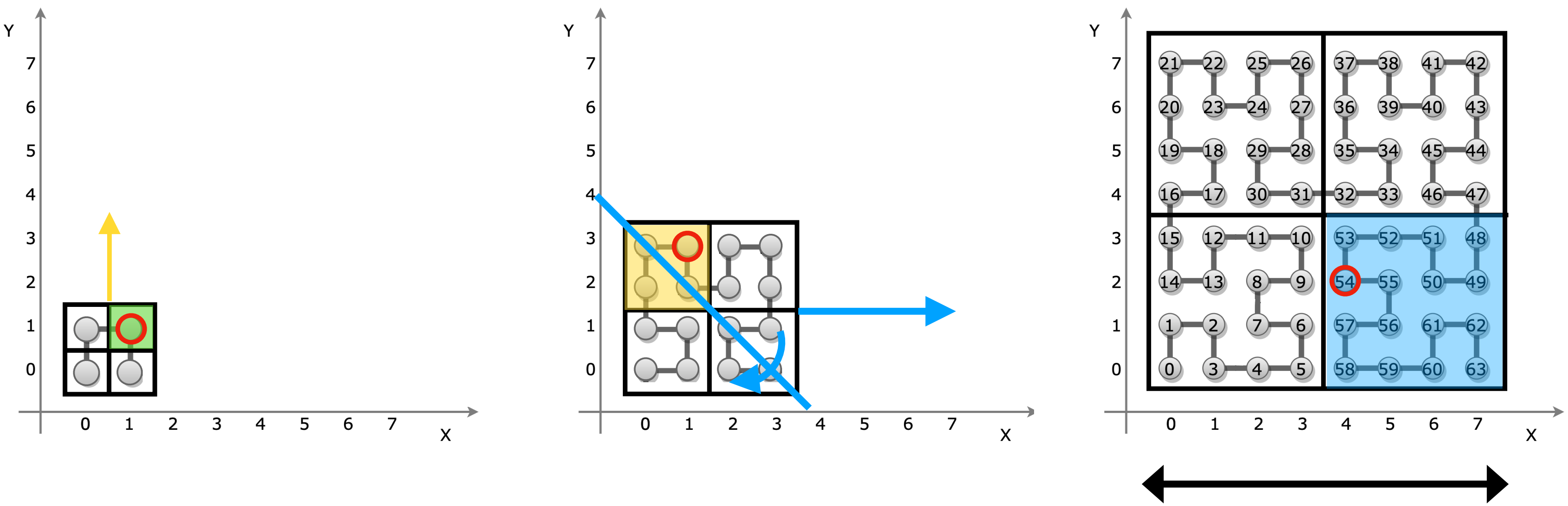
Wiederhole obigen Algorithmus für jeden Punkt

Nachteile:

- Neuberechnung für jeden der 4^n Punkte
- keine Wiederverwendung alter Rechenergebnisse
- Aufwand hängt von Grad der Kurve ab:

Aufwand pro Punkt:

Lösungsansatz



Seitenlänge = 2^n

Lösungsansatz

Berechnung der gesamten Kurve:

Naiver Ansatz:

Wiederhole obigen Algorithmus für jeden Punkt

Nachteile:

- Neuberechnung für jeden der 4^n Punkte
- keine Wiederverwendung alter Rechenergebnisse
- Aufwand hängt von Grad der Kurve ab:

Aufwand pro Punkt:

Lösungsansatz

Berechnung der gesamten Kurve:

Naiver Ansatz:

Wiederhole obigen Algorithmus für jeden Punkt

Nachteile:

- Neuberechnung für jeden der 4^n Punkte
- keine Wiederverwendung alter Rechenergebnisse
- Aufwand hängt von Grad der Kurve ab:

Aufwand pro Punkt: $log_2(2^n) \cdot c = n \cdot c$

Lösungsansatz

Berechnung der gesamten Kurve:

Naiver Ansatz:

Wiederhole obigen Algorithmus für jeden Punkt

Nachteile:

- Neuberechnung für jeden der 4^n Punkte
- keine Wiederverwendung alter Rechenergebnisse
- Aufwand hängt von Grad der Kurve ab:

Aufwand pro Punkt: $log_2(2^n) \cdot c = n \cdot c$

$\Rightarrow$ steigt linear mit Grad der Kurve

Lösungsansatz

Berechnung der gesamten Kurve:

Dynamischer Ansatz:

Lösungsansatz

Berechnung der gesamten Kurve:

Dynamischer Ansatz:

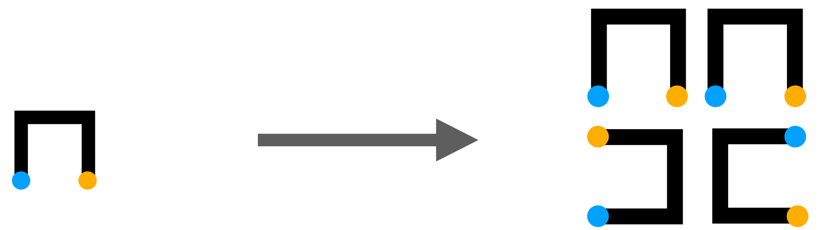
Idee: verwende bereits errechnete Zwischenergebnisse, baue Hilbert Kurve iterativ auf, wie Hilbert $\longrightarrow$ Moore

Lösungsansatz

Berechnung der gesamten Kurve:

Dynamischer Ansatz:

- Startpunkt
- Endpunkt



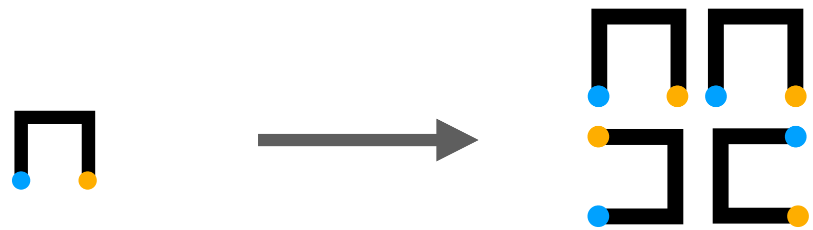
Hilbert Kurve Grad n-1 Hilbert Kurve Grad n

Lösungsansatz

Berechnung der gesamten Kurve:

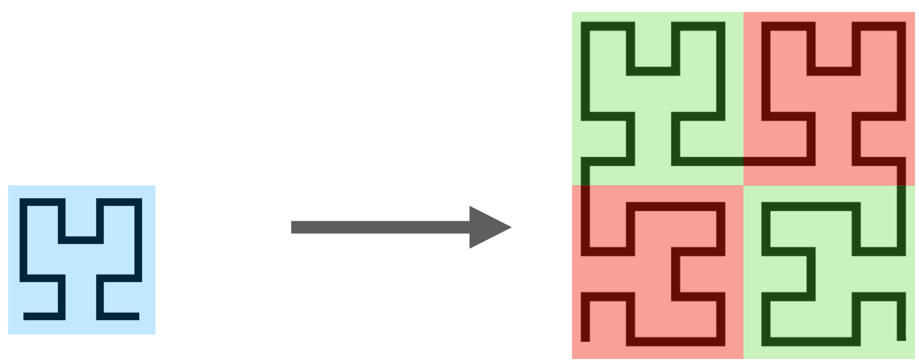
Dynamischer Ansatz:

- Startpunkt
- Endpunkt



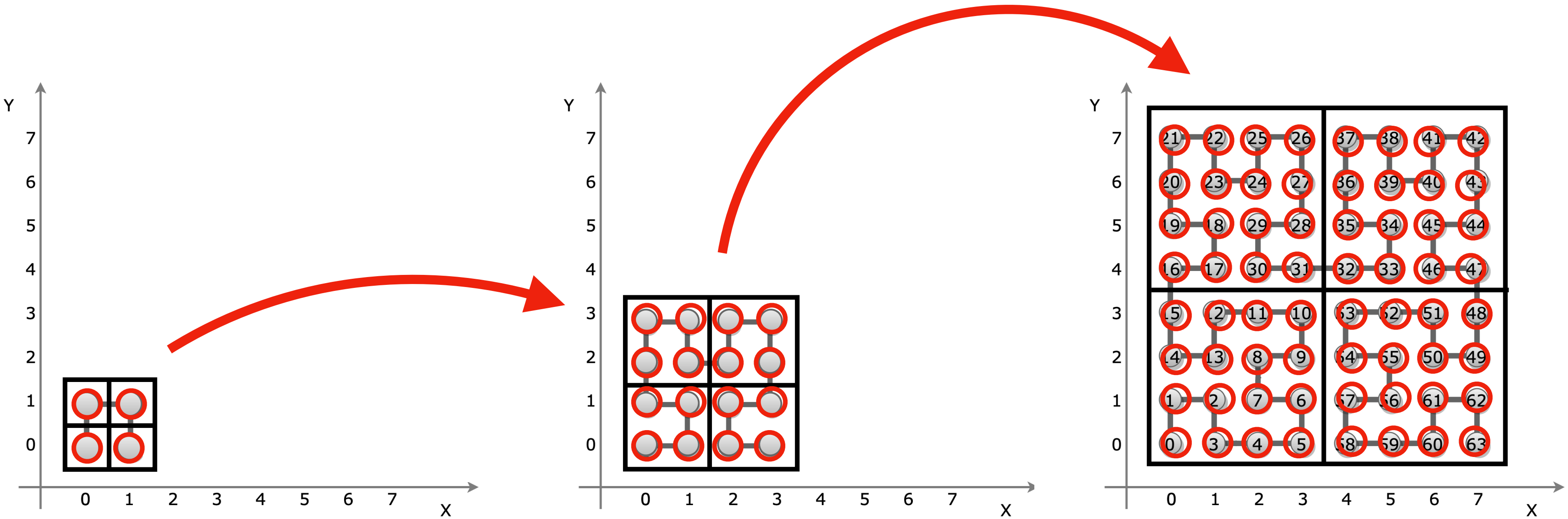
Hilbert Kurve Grad n-1 Hilbert Kurve Grad n

Beispiel:



Hilbert Kurve Grad 2 Hilbert Kurve Grad 3

Lösungsansatz



Lösungsansatz

Berechnung der gesamten Kurve:

Dynamischer Ansatz:

Idee: verwende bereits errechnete Zwischenergebnisse, baue

Hilbert Kurve iterativ auf, wie Hilbert $\longrightarrow$ Moore

Lösungsansatz

Berechnung der gesamten Kurve:

Dynamischer Ansatz:

Idee: verwende bereits errechnete Zwischenergebnisse, baue Hilbert Kurve iterativ auf, wie Hilbert $\longrightarrow$ Moore

Nachteile:

- kein einzelner Punkt an gegebenem Index berechenbar
- mehr Speicherzugriffe

Lösungsansatz

Berechnung der gesamten Kurve:

Dynamischer Ansatz:

Idee: verwende bereits errechnete Zwischenergebnisse, baue Hilbert Kurve iterativ auf, wie Hilbert $\longrightarrow$ Moore

Nachteile:

- kein einzelner Punkt an gegebenem Index berechenbar
- mehr Speicherzugriffe

Vorteil:

- durchschnittlich konstanter Rechenaufwand pro Punkt

Lösungsansatz

Berechnung der gesamten Kurve:

Dynamischer Ansatz: konstanter Rechenaufwand pro Punkt

$$I: Op(1) = c$$

$$\begin{aligned} II: Op(n) &= Op(n-1) + 4^{n-1} \cdot c \\ &= Op(n-2) + 4^{n-2} \cdot c + 4^{n-1} \cdot c \\ &= c \cdot (4^0 + \ldots + 4^{n-2} + 4^{n-1}) \\ &= c \cdot \sum_{i=0}^{n-1} 4^i \\ &= c \cdot \frac{1}{3}(4^n - 1) \end{aligned}$$

Lösungsansatz

Berechnung der gesamten Kurve:

Dynamischer Ansatz: avg konstanter Rechenaufwand pro Punkt

$$I : Op(1) = c$$

$$\begin{aligned} II : Op(n) &= Op(n-1) + 4^{n-1} \cdot c \\ &= Op(n-2) + 4^{n-2} \cdot c + 4^{n-1} \cdot c \\ &= c \cdot (4^0 + \ldots + 4^{n-2} + 4^{n-1}) \\ &= c \cdot \sum_{i=0}^{n-1} 4^i \\ &= c \cdot \frac{1}{3}(4^n - 1) \end{aligned}$$

$$\begin{aligned} avg\_Cost(n) &= \frac{Op(n)}{4^n} = c \cdot \left(\frac{4^n}{3 \cdot 4^n} - \frac{1}{3 \cdot 4^n} \right) \\ &= \frac{1}{3}c - c \cdot \frac{1}{3 \cdot 4^n} \end{aligned}$$

Lösungsansatz

Berechnung der gesamten Kurve:

Dynamischer Ansatz: konstanter Rechenaufwand pro Punkt

$$I : Op(1) = c$$

$$\begin{aligned} II : Op(n) &= Op(n-1) + 4^{n-1} \cdot c \\ &= Op(n-2) + 4^{n-2} \cdot c + 4^{n-1} \cdot c \\ &= c \cdot (4^0 + \ldots + 4^{n-2} + 4^{n-1}) \\ &= c \cdot \sum_{i=0}^{n-1} 4^i \\ &= c \cdot \frac{1}{3}(4^n - 1) \end{aligned}$$

$$\begin{aligned} avg\_Cost(n) &= \frac{Op(n)}{4^n} = c \cdot \left(\frac{4^n}{3 \cdot 4^n} - \frac{1}{3 \cdot 4^n} \right) \\ &= \frac{1}{3}c - c \cdot \frac{1}{3 \cdot 4^n} \end{aligned}$$

$$\boxed{\Rightarrow \lim_{n \to \infty} avg\_Cost(n) = \lim_{n \to \infty} \frac{Op(n)}{4^n} = \frac{1}{3}c}$$

Lösungsansatz

Berechnung der gesamten Kurve:

Dynamischer Ansatz:

Idee: verwende bereits errechnete Zwischenergebnisse, baue

Hilbert Kurve iterativ auf, wie Hilbert $\longrightarrow$ Moore

Nachteile:

- kein einzelner Punkt an gegebenem Index berechenbar
- mehr Speicherzugriffe

Vorteil:

- konstanter Rechenaufwand pro Punkt

Problemstellung

Lösungsansatz

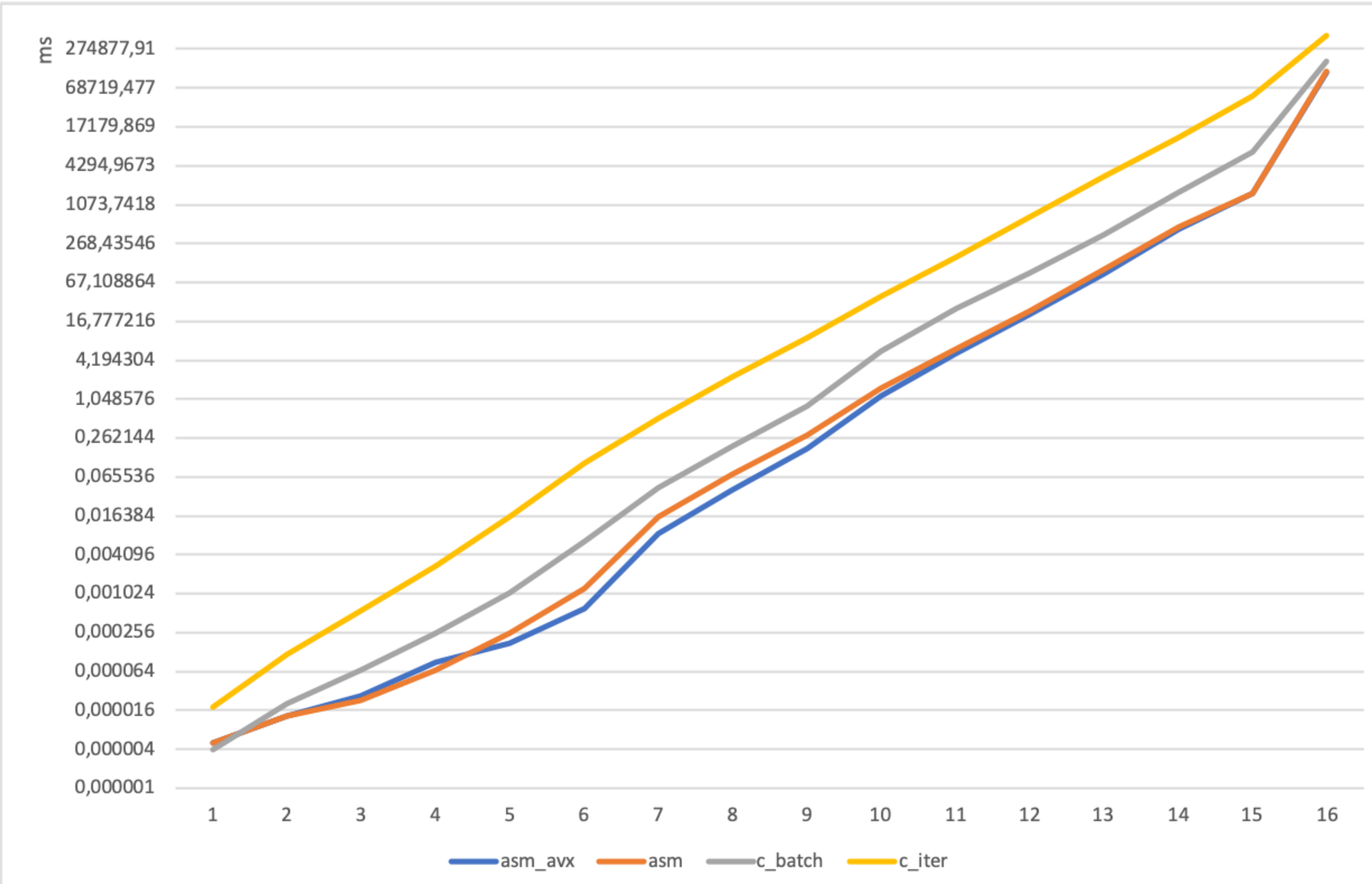
Performanz

Problemstellung

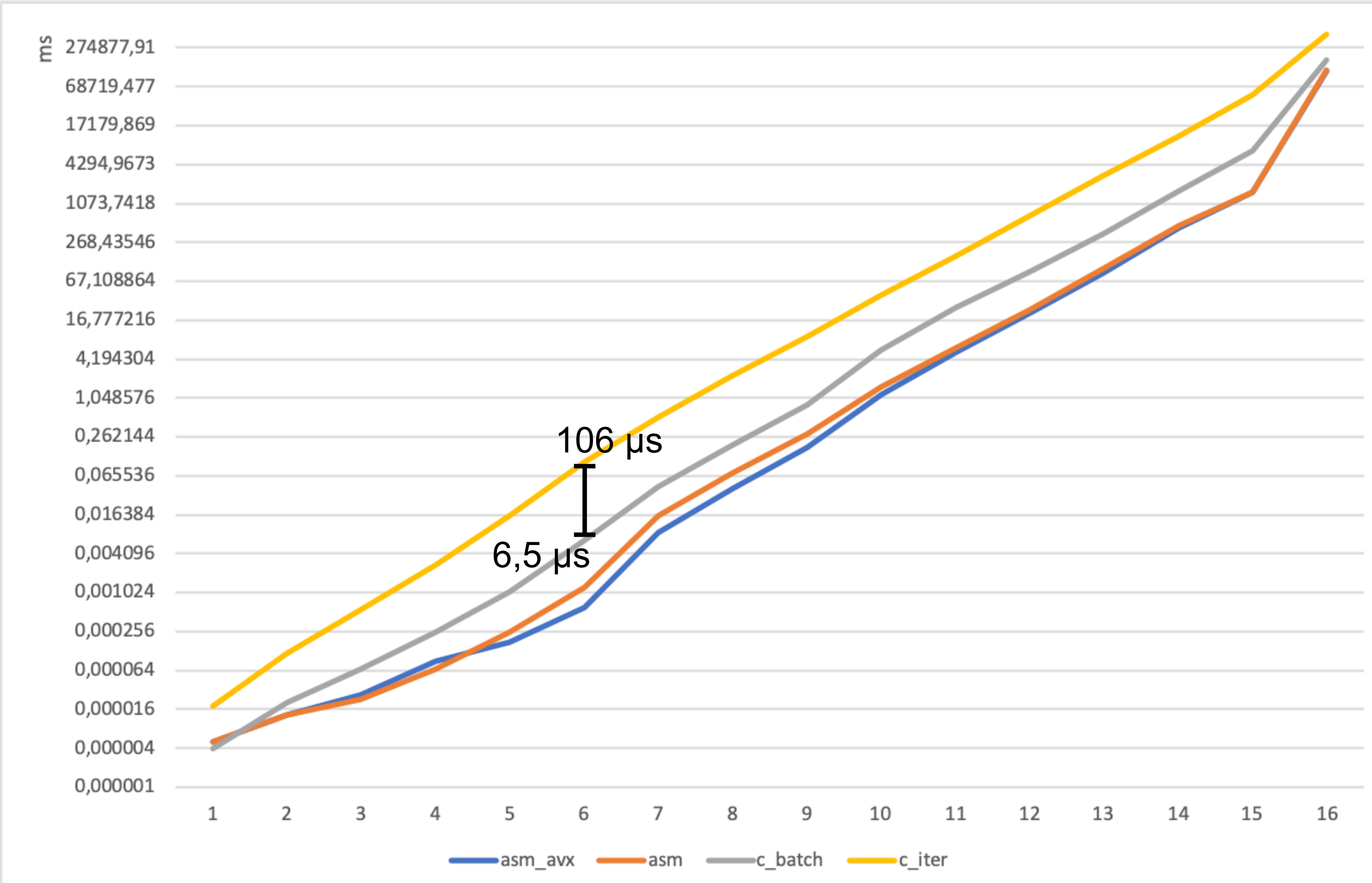
Lösungsansatz

Performanz

Performanz

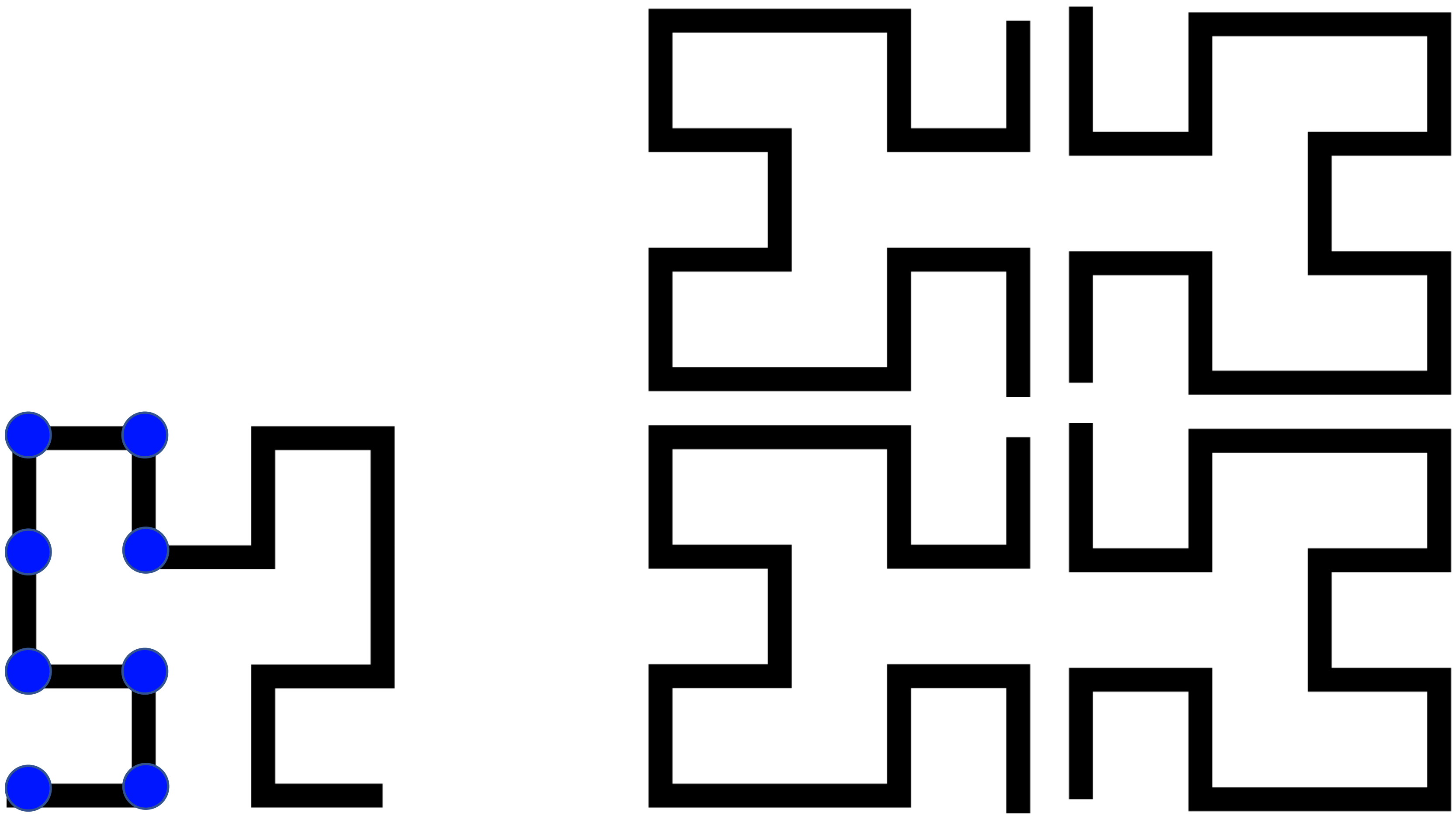

ms
274877,91
68719,477
17179,869
4294,9673
1073,7418
268,43546
67,108864
16,777216
4,194304
1,048576
0,262144
0,065536
0,016384
0,004096
0,001024
0,000256
0,000064
0,000016
0,000004
0,000001
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
asm_avx
asm
c_batch
c_iter

Performanz

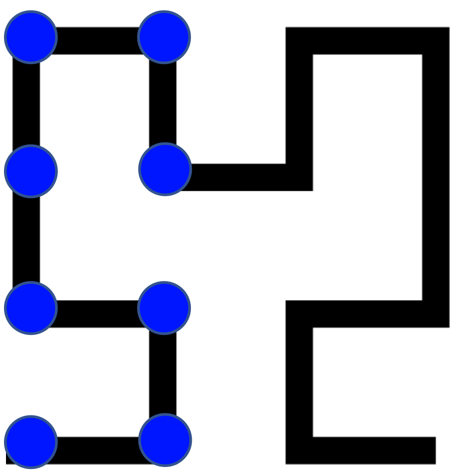

ms
274877,91
68719,477
17179,869
4294,9673
1073,7418
268,43546
67,108864
16,777216
4,194304
1,048576
0,262144
0,065536
0,016384
0,004096
0,001024
0,000256
0,000064
0,000016
0,000004
0,000001
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
106 µs
6,5 µs
asm_avx
asm
c_batch
c_iter

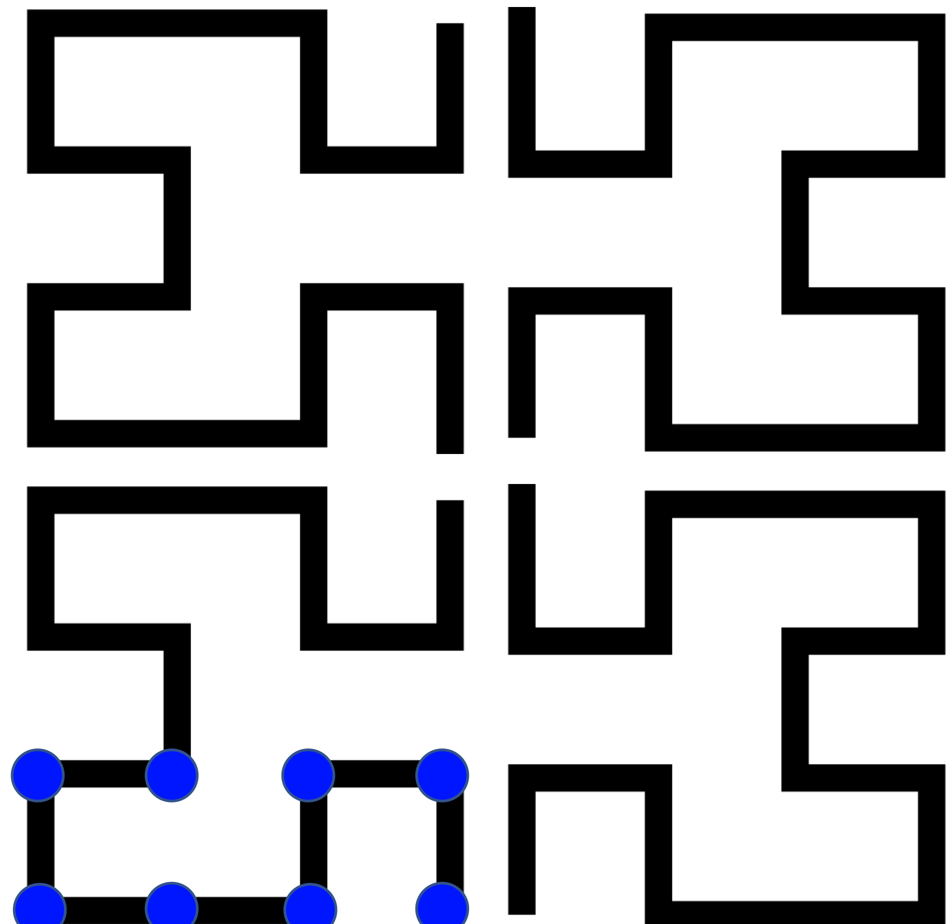
Performanz / SIMD

Performanz / SIMD

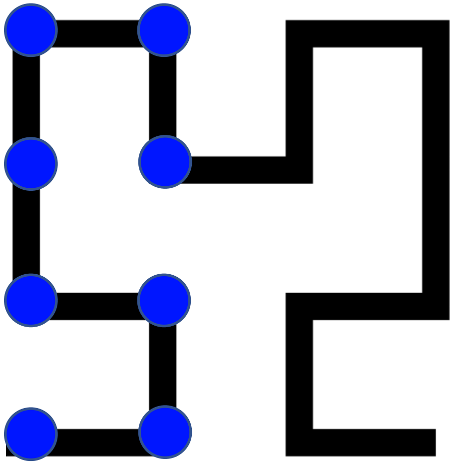


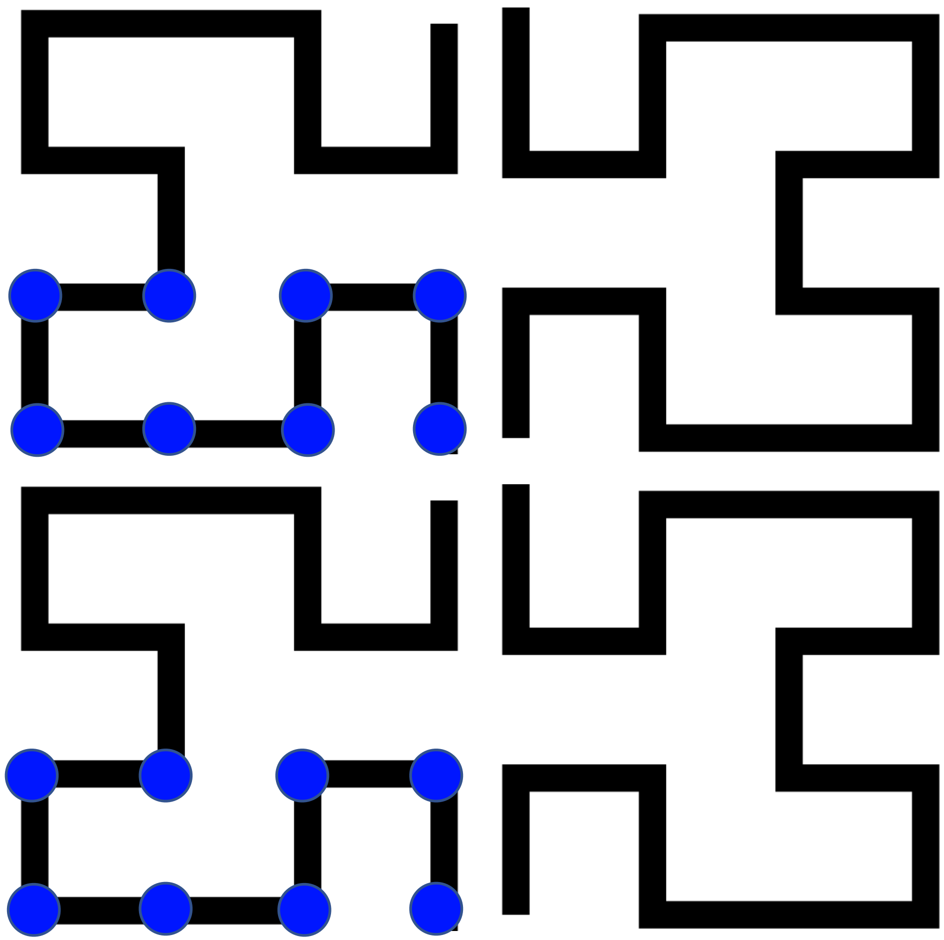
Performanz / SIMD



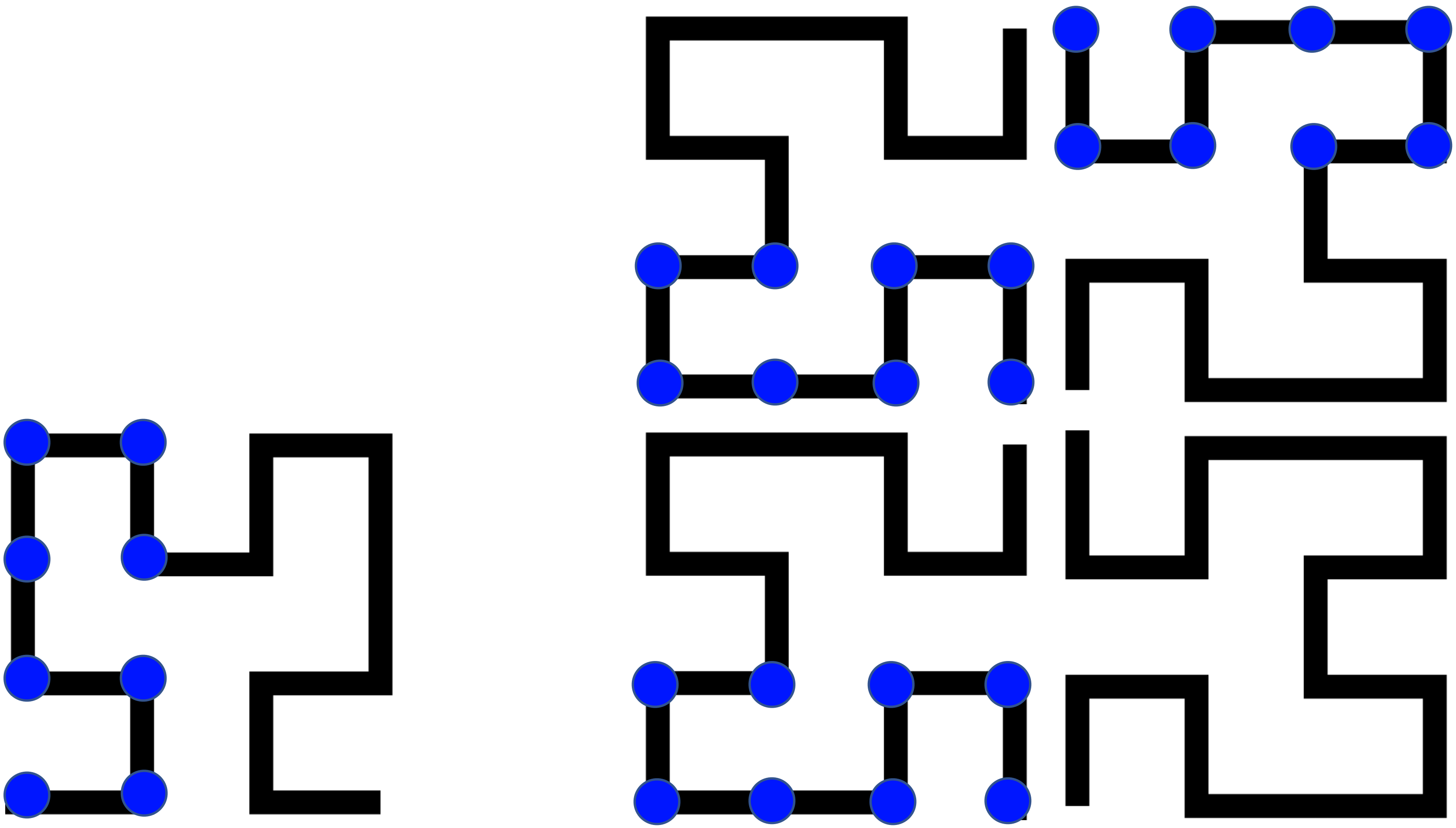


Performanz / SIMD

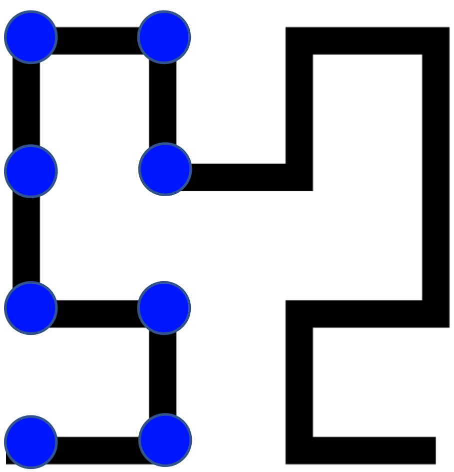


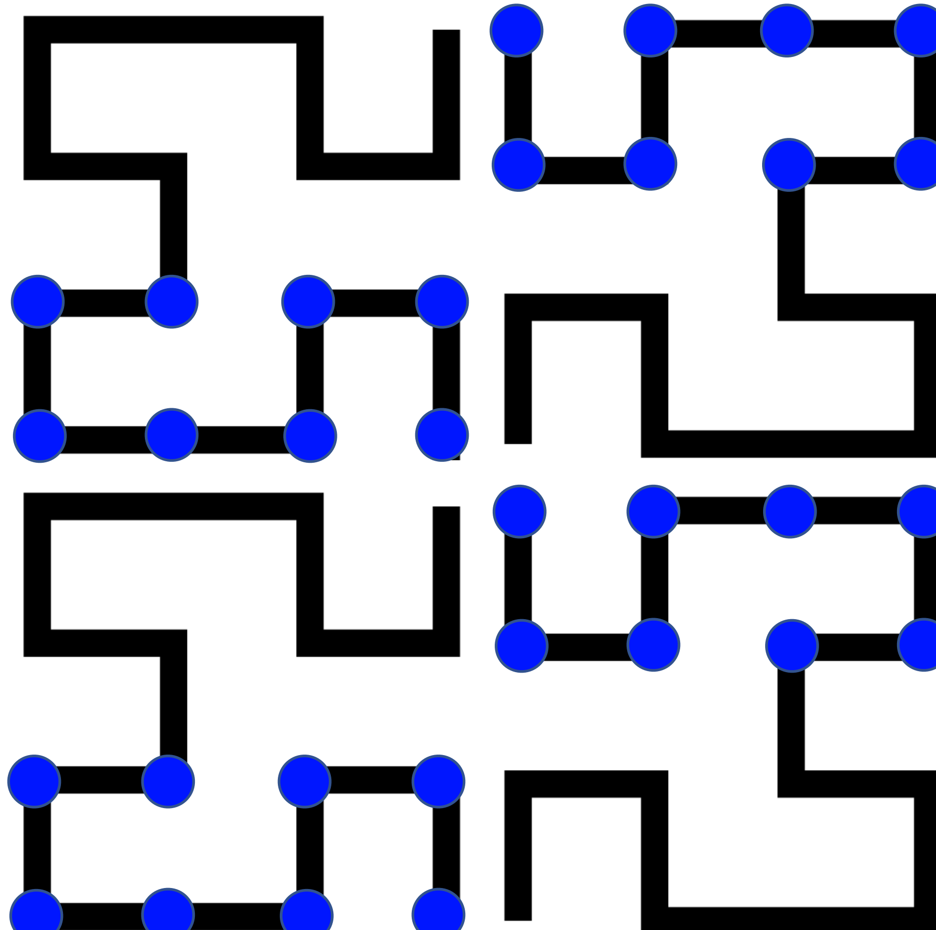


Performanz / SIMD

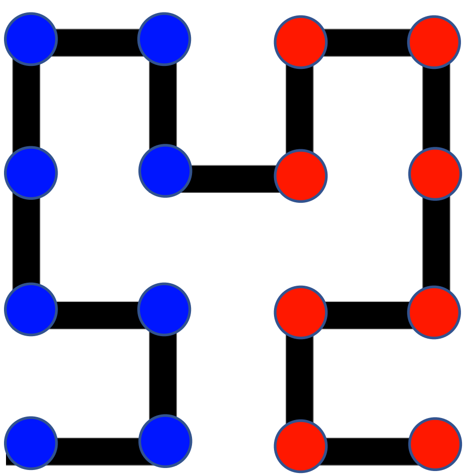


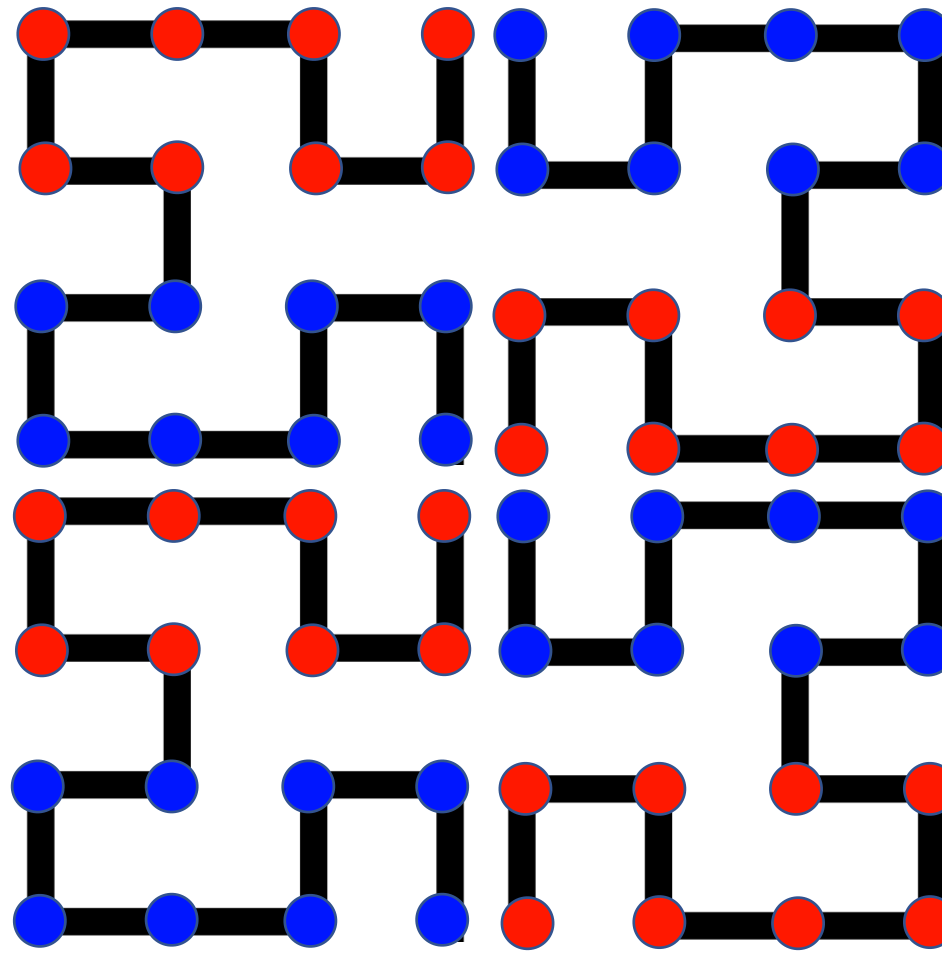
Performanz / SIMD



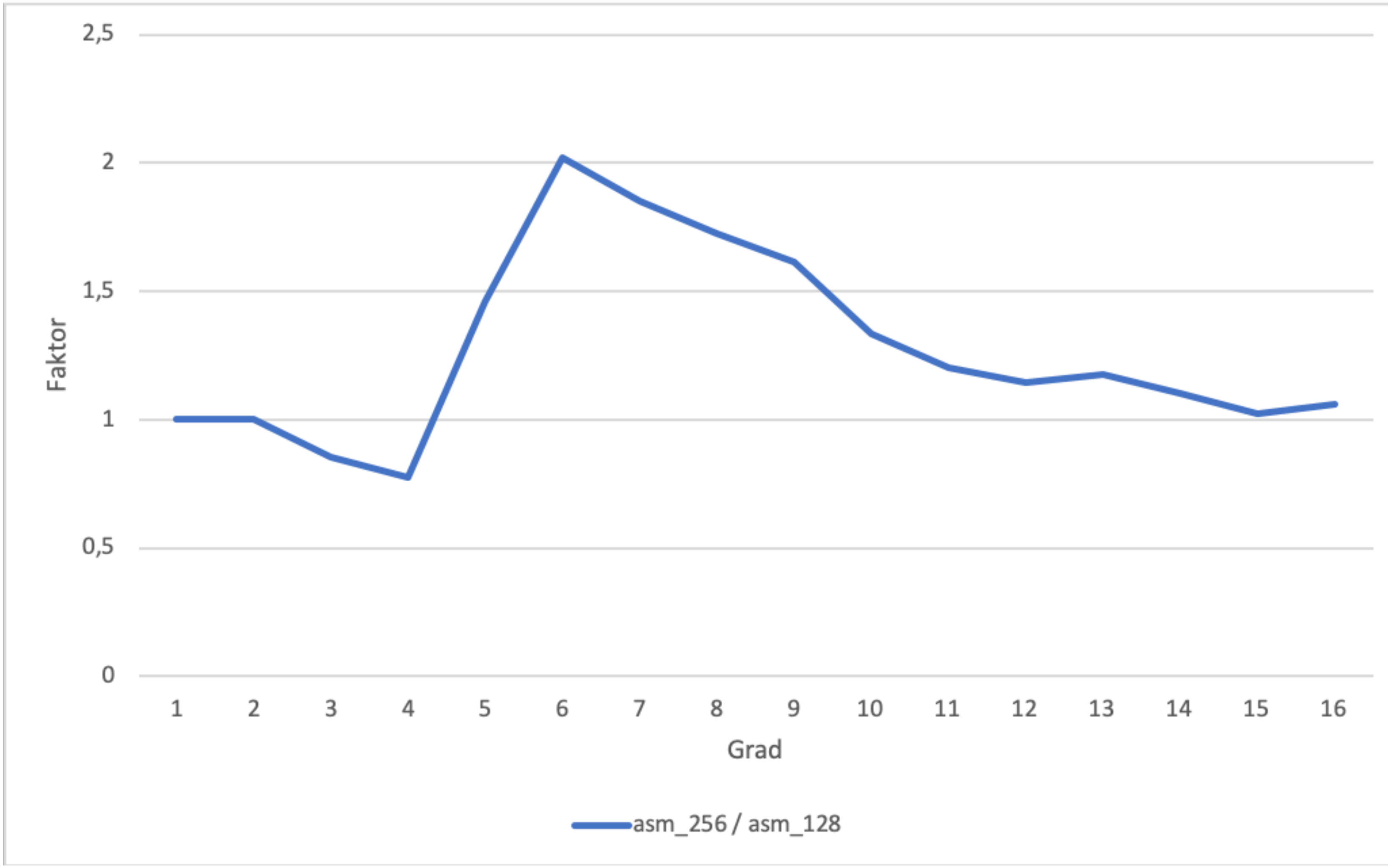


Performanz / SIMD

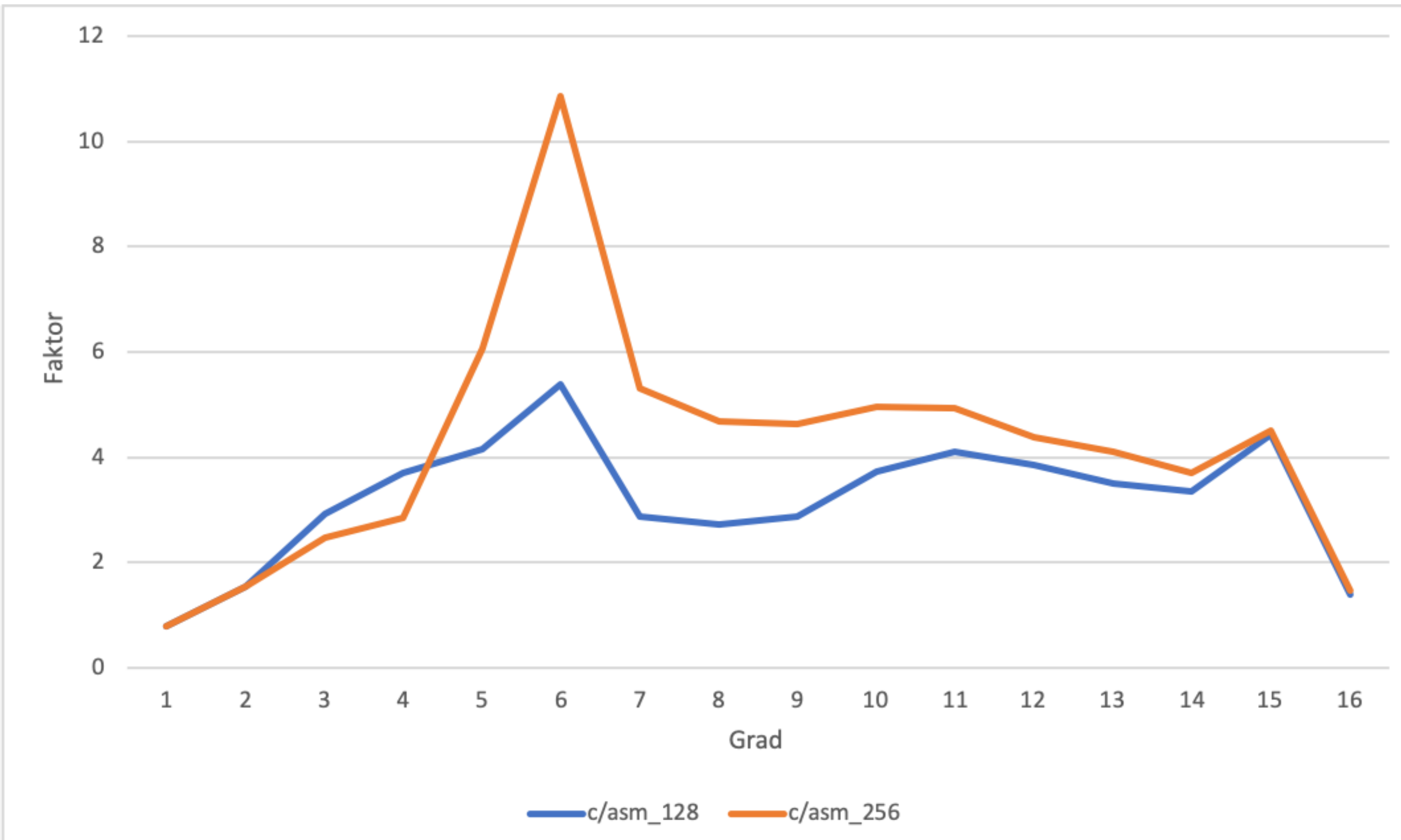




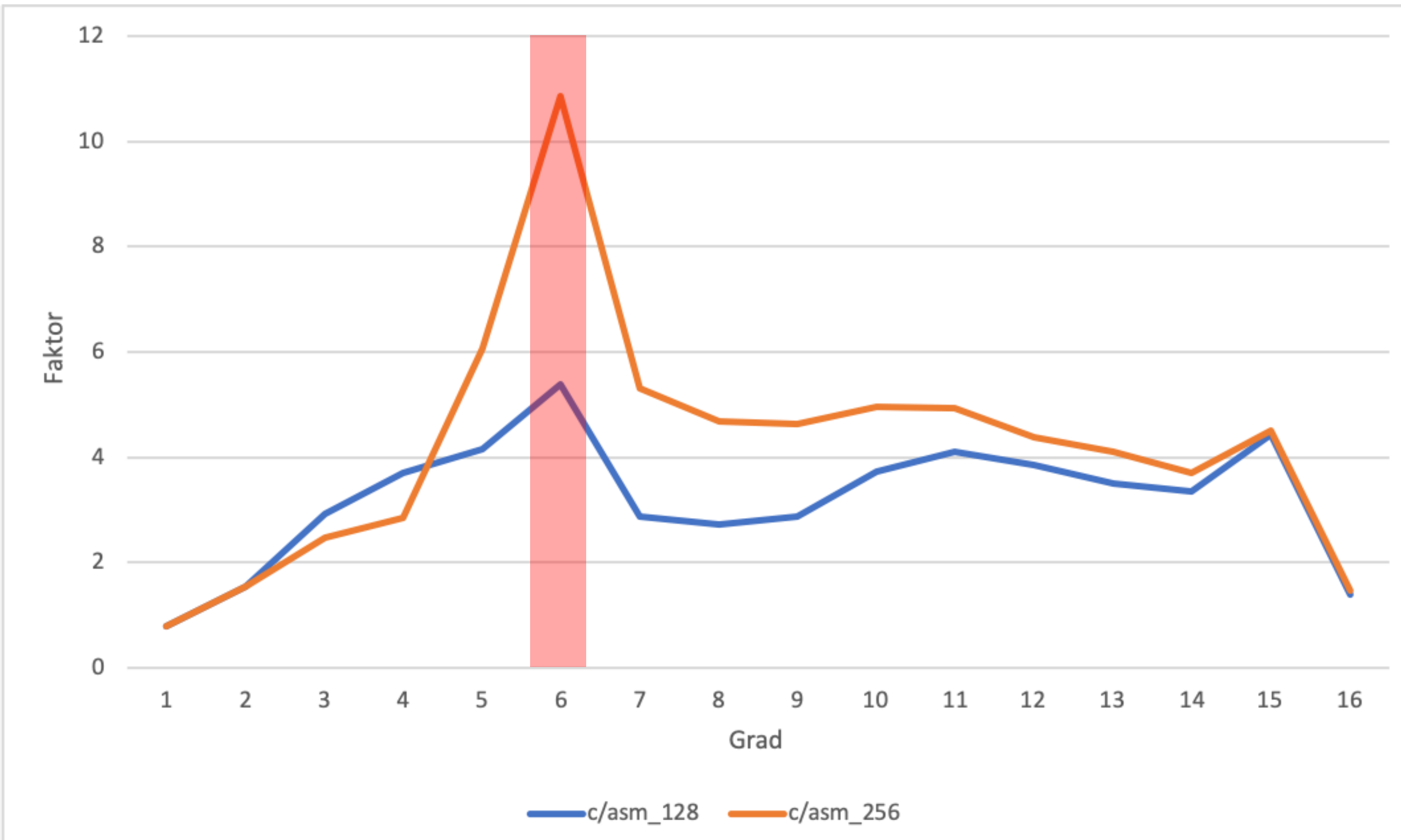
Performanz / SIMD



Performanz / SIMD



Performanz / Cache



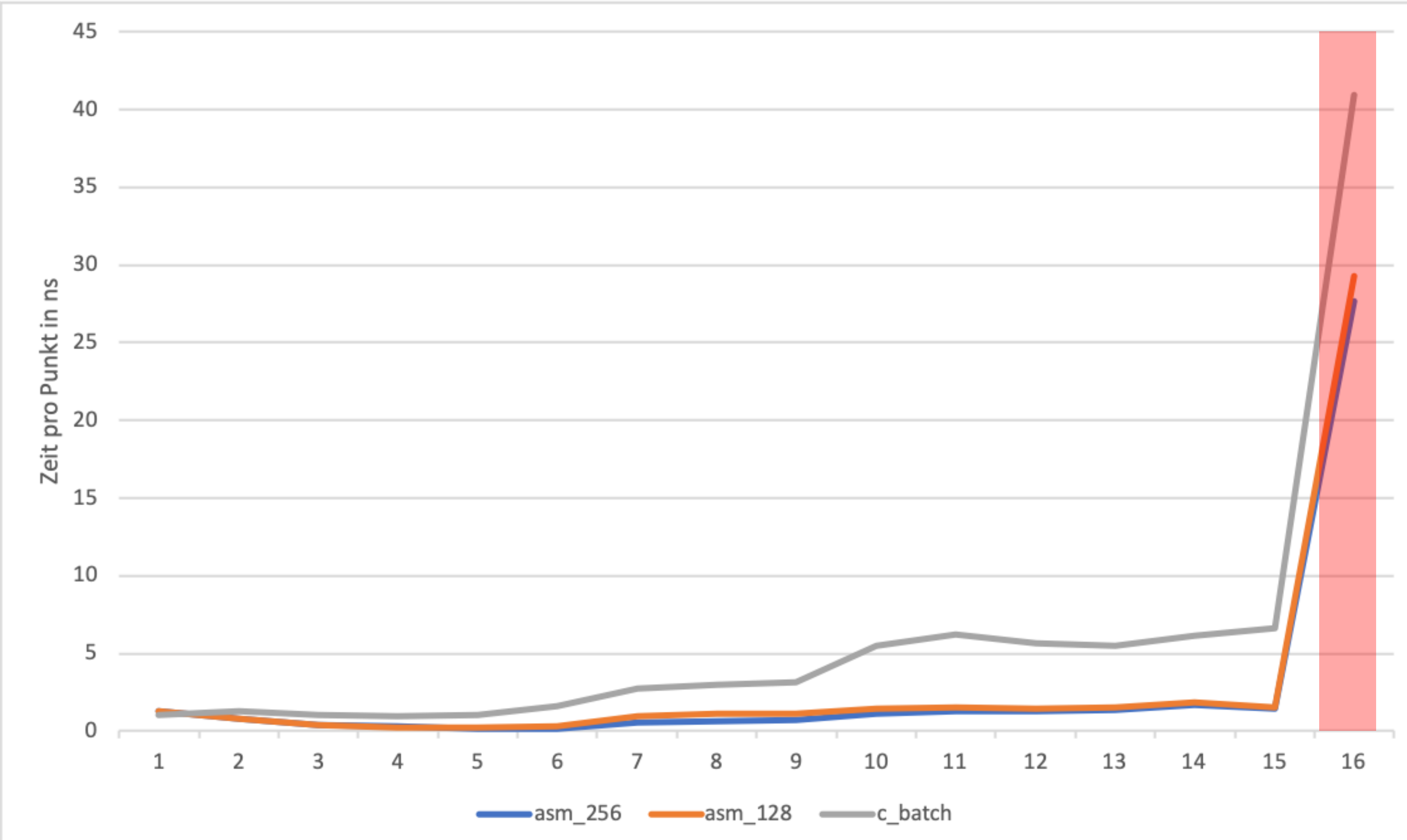
Performanz / Cache

- Performance-Spike bei Grad 6
- Annahme: Programm bis Grad 6 sehr Cache-freundlich
  - mit *perf*-Tool sehr wenige L1-Misses gemessen
  - L1-Cache auf Maschine: 32 KiB
  - Moore Kurve mit Grad 6 benötigt genau 32 KiB an Speicher

Performanz / Cache

- Performance-Spike bei Grad 6
- Annahme: Programm bis Grad 6 sehr Cache-freundlich
  - mit *perf*-Tool sehr wenige L1-Misses gemessen
  - L1-Cache auf Maschine: 32 KiB
  - Moore Kurve mit Grad 6 benötigt genau 32 KiB an Speicher
- Effizienzeinbruch bei Grad 7:
  - deutlich mehr L1-Cache-Misses
  - Speicherzugriffe haben nun großen Einfluss auf Laufzeit

Performanz / RAM-Größe



Performanz / RAM-Größe

- enormer Performance Einbruch bei Grad 16
- Moore Kurve (Grad 16) benötigt *32 GiB* an Speicher
- RAM auf Testmaschine nur *16 GiB*
- *s*tarke Zunahme an **Pagefaults**

Zusammenfassung

- Zwei Algorithmen:
  - Punkt für Punkt
  - Dynamischer Ansatz
- Weitere Optimierungsmöglichkeiten
  - dynamische Anpassung der *Int*-Breite
  - evtl. mehr Parallelisierung durch *AVX-512*
  - Experimentieren mit anderen Datenstrukturen (*x[i]* nah im Speicher an *y[i]*)

Danke