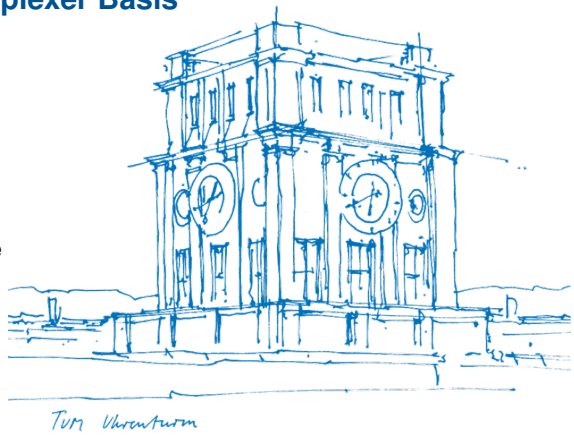
Grundlagenpraktikum: Rechnerarchitektur

Arithmetik in Zahlensystemen mit komplexer Basis



Chen Yang , Qichen Liu
Tanmay Amarendra Deshpande

Gruppe 134, Aufgabe A319
Lehrstuhl für Rechnerarchitektur und Parallele Systeme
Technical University of Munich

August 24th, 2023

Outline

Aufgabe
Komplexe Zahlensystemen

- g-adische Schreibweise einer Zahl :

$$A = \sum_{i=0}^{n} a_i \cdot g^i \tag{1}$$

- $a_i \in \{0, 1\}$, $g = -1 + i$ mit $i^2 = -1$
- $(101)_{-1+i} = 1 \cdot (-1+i)^0 + 0 \cdot (-1+i)^1 + 1 \cdot (-1+i)^2 = 1 + 0 + (-2i) = 1 - 2i$

Signaturen

Von Basis $-1+i$ zu Dezimalsystem

```
1 void to_carthesian(unsigned __int128 bm1pi, __int128*
    real, __int128* imag);
```

Von Dezimalsystem zu Basis $-1+i$

```
1 unsigned __int128 to_bm1pi(__int128 real, __int128 imag
    );
```

Anmerkung: nur die vier Grundrechenarten $(+, -, \times, \div)$ erlaubt

Outline

Lösungsansatz

1. Umrechnung vom Dezimalsystem in Basis $-1 + i$
2. to_bm1pi: Implementierungen und Optimierungen
3. Umrechnung von der Basis $-1 + i$ in Dezimalsystem
4. to_carthesian: Implementierungen und Optimierungen

Umrechnung zwischen Basen: Beispiel

Dezimalsystem in Basis 2: Recursive durch 2 teilen

Dividend	Divisor	Quotient	Rest
9	2	4	1
4	2	2	0
2	2	1	0
1	2	0	1

$$9_{10} = 1001_2$$

Umrechnung vom Dezimalsystem in Basis $-1+i$

Analog teilt man die zu konvertierende Zahl recursive durch $-1+i$:

$$\frac{a+bi}{-1+i} = \frac{b-a}{2} - \frac{a+b}{2} \cdot i \tag{2}$$

Fall 1: a und b beide gerade || beide ungerade:
$b-a$ und $a+b$ gerade
(2) hat einem Rest von 0.
Der Quotienten $\frac{b-a}{2} - \frac{a+b}{2} \cdot i$ muss weiter dividiert werden.

Umrechnung vom Dezimalsystem in Basis $-1+i$

Fall 2: a gerade und b ungerade || umgekehrt:
$b-a$ und $a+b$ ungerade. Umformulierung der Quotienten:

$$\frac{b-a}{2} - \frac{a+b}{2} \cdot i = \frac{b-a+1}{2} - \frac{a+b-1}{2} \cdot i - \frac{1+i}{2}$$

(2) hat einem Rest von 1.
Der Quotienten $\frac{b-a+1}{2} - \frac{a+b-1}{2} \cdot i$ muss weiter dividiert werden.

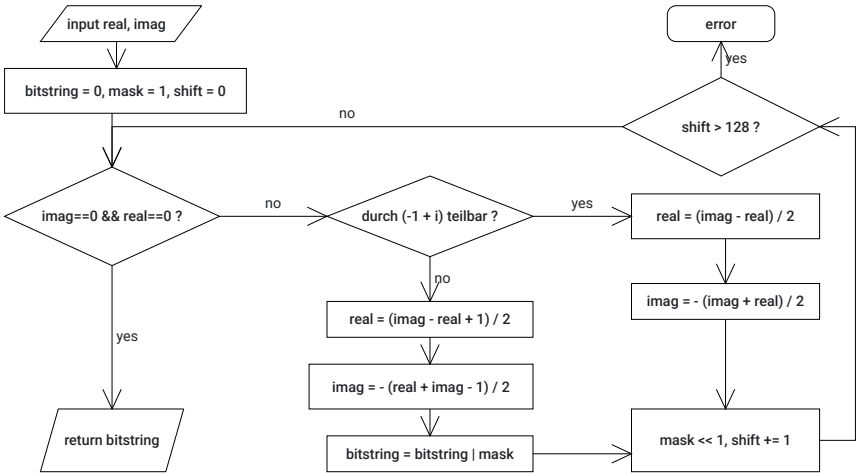
Beispiel

$3 - 2i$ in Basis $-1 + i$ umrechnen: Recursive durch $-1 + i$ teilen

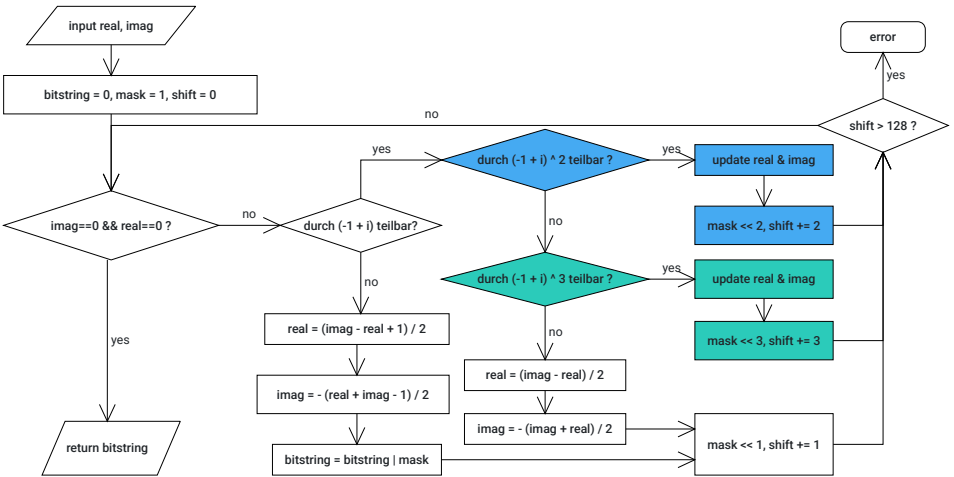
Dividend (Real)	Dividend(Imag)	Divisor	Quo. (Real)	Quo.(Imag)	Rest
3	-2	$-1 + i$	-2	0	1
-2	0	$-1 + i$	1	1	0
1	1	$-1 + i$	0	-1	0
-1	0	$-1 + i$	0	1	1
0	1	$-1 + i$	1	0	1
1	0	$-1 + i$	0	0	1

$$3 - 2i = (111001)_{-1+i}$$

to_bm1pi: Naive Implementierung



to_bm1pi: Optimierung

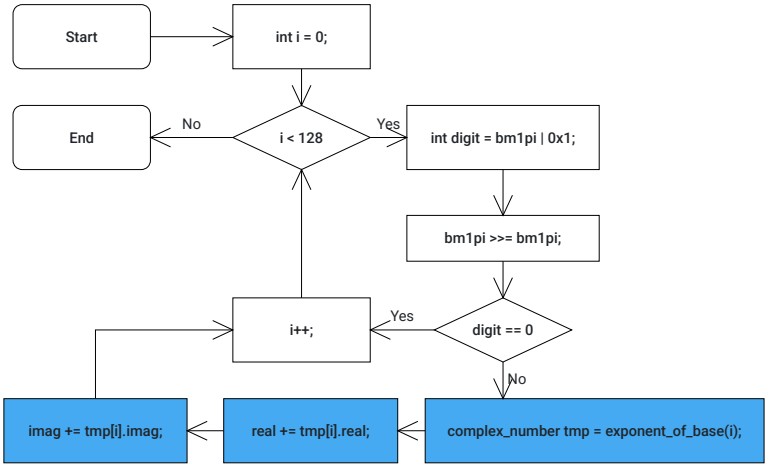


Umrechnung von der Basis $-1+i$ in Dezimalsystem

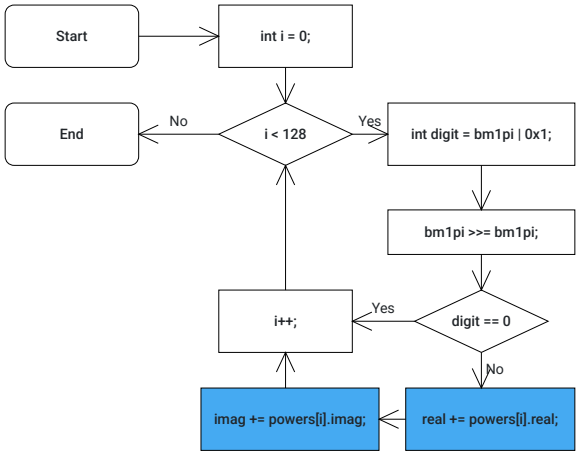
Beispiel:

$$(101)_{-1+i} = 1 \cdot (-1+i)^0 + 0 \cdot (-1+i)^1 + 1 \cdot (-1+i)^2 = 1 + 0 + (-2i) = 1 - 2i$$

to_carthesian: Naiver Ansatz



to_carthesian: Optimierung



to_carthesian: Optimierung (SIMD Alternative)

- Algorithmus schwer durch SIMD-parallelisierbar aufgrund Datentypgröße
- Ähnlich wie LUT-Optimierung
- Gleichzeitiges Laden von Real- und Imaginärteilen bei jeder Iteration mittels `_mm_loadu_si128()` im `__m128i`
- Gleichzeitiges Aufsummieren von Real- und Imaginärteil mittels `_mm_add_epi64()`
- Allerdings sehr geringe Beschleunigung : Konvertierungsoperationen und Berechnung des Offsets kosten Zeit

Outline

Tests

Vordefinierte Tests

- Randfälle/Grenzen (REAL_MAX, IMAG_MAX, 0, INT64_MAX, UINT128_MAX, usw.)
- Beliebig ausgewählte Zahlen (Bitlänge = 8, 16, ... ,128)

Zufällige Tests

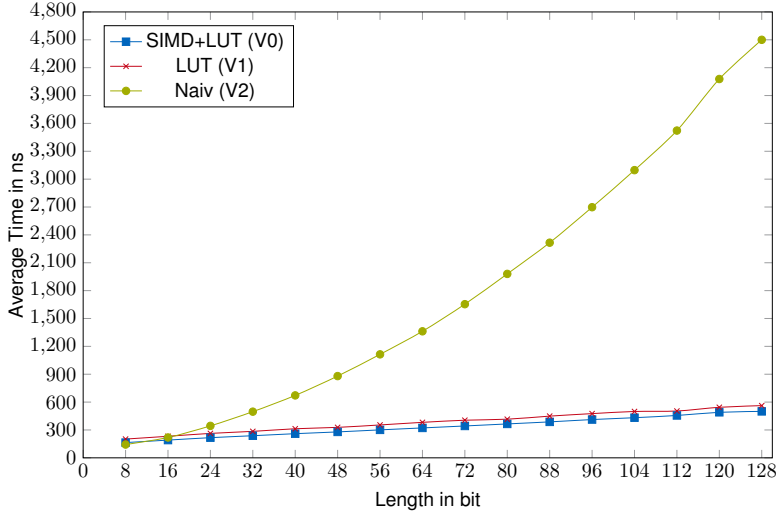
- Zufälliger Startwert start (Bitlänge zwischen 32 und 128) & zufällige Iterationenanzahl n
-
```
for(x = start; x < start + n; x++)
{to_bmpi(x, *real, *imag);
x == to_carthesian(real, imag);}
```

Outline

to_carthesian

- SIMD+LUT(V0):
  - $O(a_0 \cdot n + c \cdot 126) = O(a_0 \cdot n)$
  - Parallele Addition
  - Eine Load-Operation pro Iteration eingespart
- LUT(V1):
  - $O(a_1 \cdot n + c \cdot 126) = O(a_1 \cdot n) \;\; a_1 > a_0$
- Naive Implementierung: $O(128 \cdot c + \sum_{n \in I} n) = O(n^2)$
  - `exponent_of_base()` für jedes gesetzte Bit aufrufen

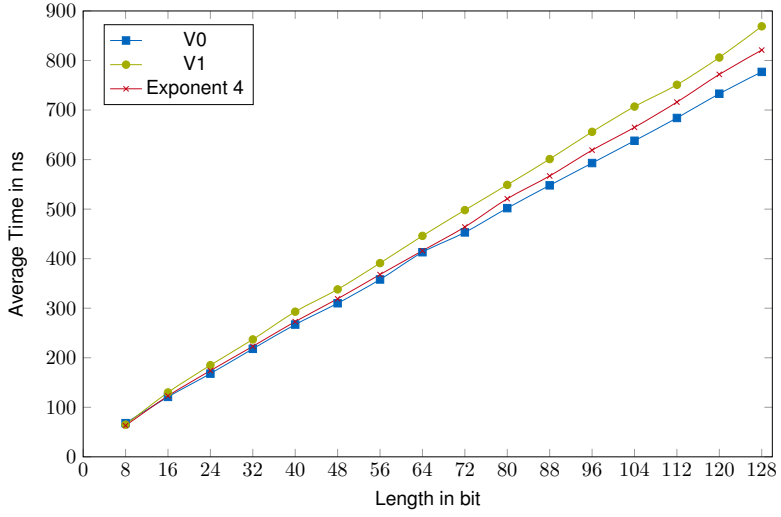
to_carthesian



to_bm1pi

- V0:
  - $O(a \cdot n)$
  - Implementiert naiven Ansatz
- V1:
  - $O(b \cdot n),\ b > a$ (Gilt nicht bei manchen Sonderfällen)
  - Naive Implementierung + 0en überspringen wo möglich

to_bm1pi


V0
V1
Exponent 4
Average Time in ns
Length in bit
0
100
200
300
400
500
600
700
800
900
0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
128

Randfälle

real	imag	time(V0)	time(V1)	bm1pi
4096	0	63ns	139ns	100000000000000000000000
-1638	-819	162ns	155ns	111111111111111111111111

Einfluss des Caches

Eingabe	Cache-Misses	Zeit
52045582664164951	6	135ns
zufälligen 56-bit Zahlen	340	303ns

Outline

Zusammenfassung und Ausblick

1. **Umrechnung von Dezimal in Basis** $(-1+i)$: wiederholte Division durch $(-1+i)$
   Beschleunigung : Division durch höhere Potenzen von Basis (wenn möglich)
2. **Umrechnung von Basis** $(-1+i)$ **in Dezimal**: Aufsummieren von nötigen Potenzen
   Beschleunigung: Potenzen in Lookup-Tabelle speichern anstatt immer neu zu berechnen: Größe Laufzeitverbesserung
   Alternative: SIMD Ansatz mit LUT : schwer parallelisierbar aufgrund der Datentypgröße
3. **Tests:** Decken größtmöglichen Eingabebeberiech ab und prüfen Randfälle
4. **Ergebnis**: Funktionierender Rechner zur Umwandlung zwischen Basis 10 und $(-1+i)$