Schnelle Exponentiation von Matrizen(A309)

Projektaufgabe – Aufgabenbereich Algorithmik

Qichen Liu Zhiyuan Ni Wenjie Zhu

Technische Universität München

Lehrstuhl für Rechnerarchitektur und Parallele Systeme

Grundlagenpraktikum: Rechnerarchitektur

31.08.2022

Inhaltsverzeichnis

Aufgabenstellung

Forschungsziel

- Exponentiation von Matrizen
- Berechnung der Fibonacci Folge

Funktionssignatur

```
void fib (uint64_t n, size_t len, uint8_t buf[len]);
```

Fibonacci Zahl

Definition

$$f_{n+1} = f_n + f_{n-1}$$

Matrix-Schreibweise

$$\begin{pmatrix} f_{n+1} & f_n \\ f_n & f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

Schnelle Exponentiation

Triviale Exponentiation

$$a^n = \underbrace{a \cdot a \cdot a \cdot \ldots \cdot a}_{n-1}$$

Binäre Repräsentation von n

$$n = \sum_{i=1}^{\lfloor \log_2 n \rfloor} (\mathcal{I} \cdot 2^i),\ \mathcal{I} \in [0, 1]$$

Schnelle Exponentiation

$$a^n = (\mathcal{I} \cdot a^1)(\mathcal{I} \cdot a^2)(\mathcal{I} \cdot a^4) \ldots (\mathcal{I} \cdot a^{\lfloor \log_2 n \rfloor})$$

Übertragung auf Matrix

$$f_5 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^5$$

$$5 = 1 \cdot 2^0 + 0 \cdot 2^1 + 1 \cdot 2^2$$

F^1	$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$
$F^2 = (F^1)^2$	$\begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix}$
$F^4 = (F^2)^2$	$\begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix}$

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^5 = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 5 & 3 \\ 3 & 2 \end{pmatrix} = \begin{pmatrix} 13 & \color{red}{8} \\ \color{red}{8} & 5 \end{pmatrix}$$

Länge Berechnung

allgemeine Formel der Fibonacci-Zahl

$$f_n = \left\lfloor \frac{\lambda^n}{\sqrt{5}} \right\rfloor \; mit \; \lambda = \frac{1+\sqrt{5}}{2}$$

Länge der Fibonacci-Zahl

$$\log_2 f_n = \left\lfloor \log_2 \frac{\lambda^n}{\sqrt{5}} \right\rfloor$$
$$= \lfloor n \; log_2 \alpha - \frac{1}{2} \log_2 5 \rfloor$$

Karazuba Multiplikation

Formel

$$
\begin{aligned}
x \cdot y &= (2^m x_1 + x_0)(2^m y_1 + y_0) \\
&= x_0 y_0 + 2^m (x_0 y_1 + x_1 y_0) + 2^{2m} x_1 y_1 \\
&= x_0 y_0 + 2^m (x_0 + x_1)(y_0 + y_1) - x_0 x_1 - y_0 y_1) + 2^{2m} x_1 y_1
\end{aligned}
$$

Pseudocode

```
function karatsuba(x, y)
if (x < 2 || y < 2)
    //fall back to traditional multiplication
    return x * y

l = max(len(x), len(y))
m = l / 2
r = m - r

//divide the number in the middle
x1, x0 = split_at(x, m)
y1, y0 = split_at(y, m)

//calculate each production with recursive calls
z0 = karatsuba(x0, y0)
z1 = karatsuba(x0 + x1, y0 + y1)
z2 = karatsuba(x1, y1)

return ((z2 * 2 ^ (r * 2)) + ((z1 - z2 - z0) * 2 ^ r) + z0)
```

Wahl der Datenstrukturen und Begründung

- Problem: Wie kann man große Zahlen im Speicher sinnvoll darstellen?
  - Integer Arrays
  - Binäre String-Darstellung ✓
- Vorteile
  - einfache Bestimmung der Länge
  - einfache bzw. eiffiziente Multiplikation der Bit-Operationen
- Nachteile
  - Speicherplatz aufwandig
  - Rekursion bis zum einzigen Byte durchführen
- Anderer Ansatz
  - Array variabler Länge

Aufbau der Implementierung

1. main.c: Parsen von Command Line Argument
2. fib_v0.c: Hauptimplementierung mit schneller Exponentiation
3. fib_v1.c: Vergleichsimplementierung mit normaler Exponentiation
4. mul.c: Implementierung von Karazuba-Multiplikation
5. cvt.c: Umwandlung binärer String-Repräsentation in dezimale/hexadezimale Repräsentation
6. test.c: Automatischer Test, wird nicht mitkompiliert
7. fib_numbers.h: Enthalten bis 1000-te Fibonacci-Zahl für Testzweck

Programmausgabe

```
40-th fibonacci number calculated after 0.001253 seconds
40-th fibonacci number calculated after 0.001026 seconds
fib(40) bin: 110000110010111111011001011
fib(40) dec: 102334155
fib(40) hex: 6197ECB
the benchmark was executed 2 time(s)
the total calculation last for 0.002279 second(s)
the average duration of each calculation is 0.001139 second(s)
```

Listing 1: Ausgabe mit Command ./fib -n40 -V0 -B2

Testverfahren

- Konvertierung zur String-Darstellung
- Testfunktionen
  - void FIB_N_CHECK (int n, bool v);
  - void CROSS_CHECK (int n);
  - void INDUCTION_CHECK (int start, int end, bool v);

Testausgabe

```
==================== Verifying 1-th fibonacci ====================
Expected: 1
Provided: 1
[SUCCESSFUL] Test passed!
......
(Zwischenausgaben weglässt wegen Platzmangel)
......
==================== Verifying 72-th fibonacci ====================
Expected: 1110001010101011101011110010100001001111101100000
Provided: 1110001010101011101011110010100001001111101100000
[SUCCESSFUL] Test passed!
[72/72] All tests passed!
```

Listing 2: Testausgabe von Induktion Check mit start = 1, end = 72, v = 0

Komplexität

- Erforderte Matrix Multiplikationen:
  - Naive Exponentiation: $\mathcal{O}(n)$
  - Schnelle Exponentiation: $\mathcal{O}(\log n)$
- Erforderte Additionen und Multiplikationen:
  - Fibonacci-Matrix: 4 Additionen zweier Big-Integer
  - Normale 2×2 Matrix: 4 Additionen und 8 Multiplikationen zweier Big-Integer
  - Naive Exponentiation: $\mathcal{O}(n)$ Additionen
  - Schnelle Exponentiation: $\mathcal{O}(\log n)$ Additionen und Multiplikationen
- Komplexität der Additionen und Multiplikationen:
  - Addition: $\mathcal{O}(n)$
  - Multiplikation: $\mathcal{O}(n^{\log_2 3})$

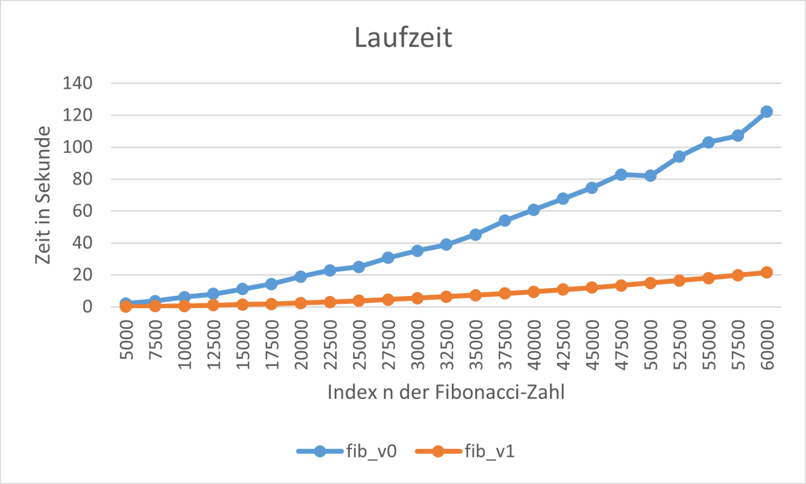
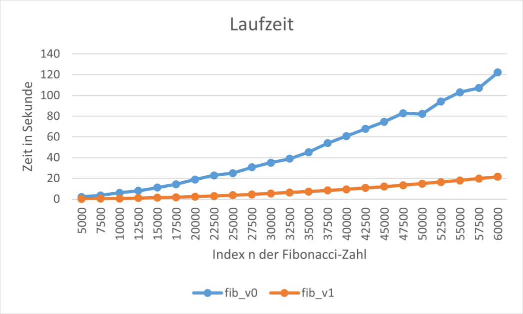


Figure: Laufzeit der fib_v0 und fib_v1

Analyse & Interpretation

- Wachstumsrate der Länge der Fibonacci-Zahl: $\mathcal{O}(n)$
- fib_v1: $\mathcal{O}(n^2)$
- Ähnliche Laufzeitkomplexität
- Keine wirkliche Performanzverbesserung



Quellenverzeichnis

- A.Karatsuba, Y.Ofman (1962)
  Multiplication of Many-Digital Numbers by Automatic Computers
  Doklady Akademii Nauk SSSR
- Prof. Dr. Otto Forster (2004)
  Die Fibonacci-Zahlen
  https://www.mathematik.uni-muenchen.de/~forster/v/zth/inzth_01.pdf
- Weisstein, Eric W. (2022)
  Karatsuba Multiplication
  https://mathworld.wolfram.com/KaratsubaMultiplication.html