Aspekte der Systemnahen Programmierung 18/19

Final Presentation

4. März 2019

CRC32 (A401)

team120
Liudongnan Yang, Mehmet Dereli, Alexandru Balotescu

Problemstellung und Spezifikation

- Informationstheorie
  - wichtiger Teilbereich der Informatik
  - beschäftigt sich mit der korrekte Kodierung und Dekodierung von Nachrichten
- CRC32

Problemstellung

- Algorithmen und das Internet sind nicht ohne Fehler
- CRC32
  - fehlererkennendes Verfahren
  - sollte ein Fehler identifiziert werden, dann muss die Nachricht neu vom Sender beantragt werden
- ISO/OSI-Modell
  - 1. Schicht (Physikalische Schicht): hier können Fehler auftauchen
  - 2. Schicht (Sicherungsschicht): hier wird das CRC32 Verfahren angewendet

Spezifikation

- Zyklische Redundanzprüfung → Polynomdivision → Prüfsummen
- die zu übertragende Nachricht *N* der Länge *l* als Polynom:

$$N = x_1 x_2 x_3 ... x_n = \sum_{i=0}^{l-1} a_i \cdot x^i \text{ mit } a_i \in \{0, 1\} = N(x)$$

- Generatorpolynom *G(x)* bei CRC32:

$$G(X) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + x^0$$

Verfahren

Schritt 1

- Multiplizieren der Nachricht *N(x)* mit dem Grad des Generatorpolynoms *G(x)*
  - *N(x)* werden *grad(G(x))* Nullen angehängt → *N'(x)*

Verfahren

Schritt 2

- korrekte Prüfsumme für *N(x) berechnen*
  - *N'(x)* durch *G(x)* polynomdividieren
  - den Rest der Polynomdivision an *N(x)* angehängen → *N"(x)*

Verfahren

Schritt 3

- Korrektheit überprüfen
  - *N”(x)* durch Generatorpolynom *G(x)* polynomdividieren
  - Rest der Division = 0 → kein Übertragungsfehler (mit hoher Wahrscheinlichkeit)
  - Rest der Division ≠ 0 → Übertragungsfehler, erneute Sendung der Nachricht ist erforderlich

Beispiel

- Ablauf des Verfahrens unabhängig vom Grad des Generatorpolynoms
- Beispiel anhand von CRC5:

$$G(X) = x^5 + x^4 + x^2 + x^0 = 110101$$

$$N(X) = 10010011$$

Beispiel

$$G(X) = x^5 + x^4 + x^2 + x^0$$
$$G(X) = 110101$$
$$N(X) = 10010011$$

Schritt 1

- *grad(G(x))* = 5 → *N(x)* werden 5 Nullen angehängt
  - $N'(x) = 10010011\,00000$

Beispiel

$$G(X) = x^5 + x^4 + x^2 + x^0$$
$$G(X) = 110101$$
$$N(X) = 10010011$$
$$N'(x) = 1001001100000$$

Schritt 2

- Prüfsumme berechnen
  - Polynomdivision zwischen *N'(x)* und *G(x)*
  - *Rest = 01111* → $N''(x) = 1001001101111$

Beispiel

$$G(X) = x^5 + x^4 + x^2 + x^0$$
$$G(X) = 110101$$
$$N(X) = 10010011$$
$$N'(x) = 1001001100000$$
$$N''(x) = 1001001101111$$

Schritt 3

- Auf Korrektheit überprüfen
  - Polynomdivision zwischen *N”(x)* und *G(x)*
  - *Rest = 00000* → es ist mit hoher Wahrscheinlichkeit kein Fehler aufgetreten

Sicherheit

- CRC32 ist unter bestimmten Umständen fehleranfällig:
  - Fehler, die länger als den Grad des Generatorpolynoms sind
  - Fehler, die aus mehreren Bursts bestehen
  - Alle Fehler, die ein Vielfaches des Generatorpolynoms sind
    - Beispiel

Beispiel - Fehler

- Alle Fehler, die ein Vielfaches des Generatorpolynoms sind, können nicht erkannt werden
- Beispiel:
  - die Nachricht $N(x) = 00101100$
  - das Generatorpolynom $G(x) = x^3 + x^1 + x^0$
  - Polynomdivision gleich Null

Beispiel - Fehler

$$N(x) = 00101100$$
$$G(x) = x^3 + x^1 + x^0$$

- wir fügen nun der Nachricht N(x) ein Fehler F(x) hinzu, der ein Vielfaches des Generatorpolynoms G(x) ist:

$$\begin{aligned} N(x) &= 00101100 \\ \oplus F(x) &= 10110000 \\ \hline M(x) &= 10011100 \end{aligned}$$

Beispiel - Fehler

$$N(x) = 00101100$$
$$G(x) = x^3 + x^1 + x^0$$
$$F(x) = 10110000$$
$$M(x) = 10011100$$

- der Rest der Polynomdivision von M(x) durch G(x) ist aber ebenfalls gleich Null
- mehrere Nachrichten bilden auf den selben Rest ab
- Fehler werden nicht erkannt

Lösungsfindung

- Doppelte Implementierung
  - C
  - Assembler
- In 3 Stufen aufgeteilt
  - CRC4 in C
  - CRC32 in C
  - Lookup Table
    - Optimierter CRC32
    - CRC32 in Assembler

Implementation – CRC4 in C

- analog zu CRC32

ABER

  - bietet eine leichtere Einführung in das Thema an
  - einfacher zu testen und nachzurechnen
  - erleichtert das Erkennen von Gedankenfehler

Implementation – CRC32 in C

- CRC4 als Basis
- Optimierung der Struktur durch Auslagerung einzelner Teilen in Methoden

Implementation – Optimierter CRC32

- Neue Methode zur Berechnung der Polynomdivision: **Lookup Table**
  - Ergebnisse der Polynomdivisionen werden direkt nachgeschaut und müssen somit nicht vom Programm berechnet werden
- *unsigned int Table_CRC32[256] = (0, 4c11db7, 9823b6e, d4326d9, 130476dc, 17c56b6b, 1a864db2, 1e475005, 2608edb8, 22c9f00f, […])*

Implementation – CRC32 in Assembler

- erfolgt analog zum optimierten Algorithmus in C
- Implementierung erfolgt mit Hinsicht auf den Aufrufkonventionen

ABER

- das Programm unterstützt keine SIMD-Befehle
- die nächsten 4 Byte-Folgen werden immer durch das aktuelle Byte verändert und können somit nicht parallel bearbeitet werden

Entwickler-Dokumentation in C

- lässt sich in zwei Lösungswege teilen
  - intuitiver Weg (iterativ)
  - optimierter Weg (Lookup Table)

Intuitive Version

- `unsigned int CRC32_C_helper2(char* data)`
  - das Rahmenprogramm für die Berechnung der Prüfsumme
- `int char_counter(char* data)`
  - ermittelt die Länger der Nachricht
- `char* padding(int data_length, char* data)`
  - “padded” die Nachricht

Intuitive Version

```
unsigned int CRC32_C_helper2(char* data)
{
    char* padded_data = padding2(data);

    unsigned int reg = 0 ;
    unsigned int pos = 0 ;

    //main loop
    //stop the loop when there's no char in the stack
    while (pos != strlen(padded_data)+4) {
        for(int cur_bit = 7; cur_bit >= 0 ; cur_bit--) {
                int high = (reg >> 31) & 0x01;
                //lsl
                reg <<= 1;
                //load data from buffer & push up the next bit from the buffer into reg
                reg |= ((padded_data[pos]>>cur_bit)&0x00000001);

                if(high == 1) {
                    reg = reg ^ POLY;
                }
        }
        pos++;
    }
    return reg;
}
```

Lookup Table Version

- *`unsigned int CRC32_C_table(char* data)`*
  - das Rahmenprogramm für die Berechnung der Prüfsumme
- *`generateCRC32_Table()`*
  - berechnet die 256 unterschiedlichen Ergebnisse die eine Polynomdivision zwischen einer 8 Bit Folge und dem Generatorpolynom haben kann

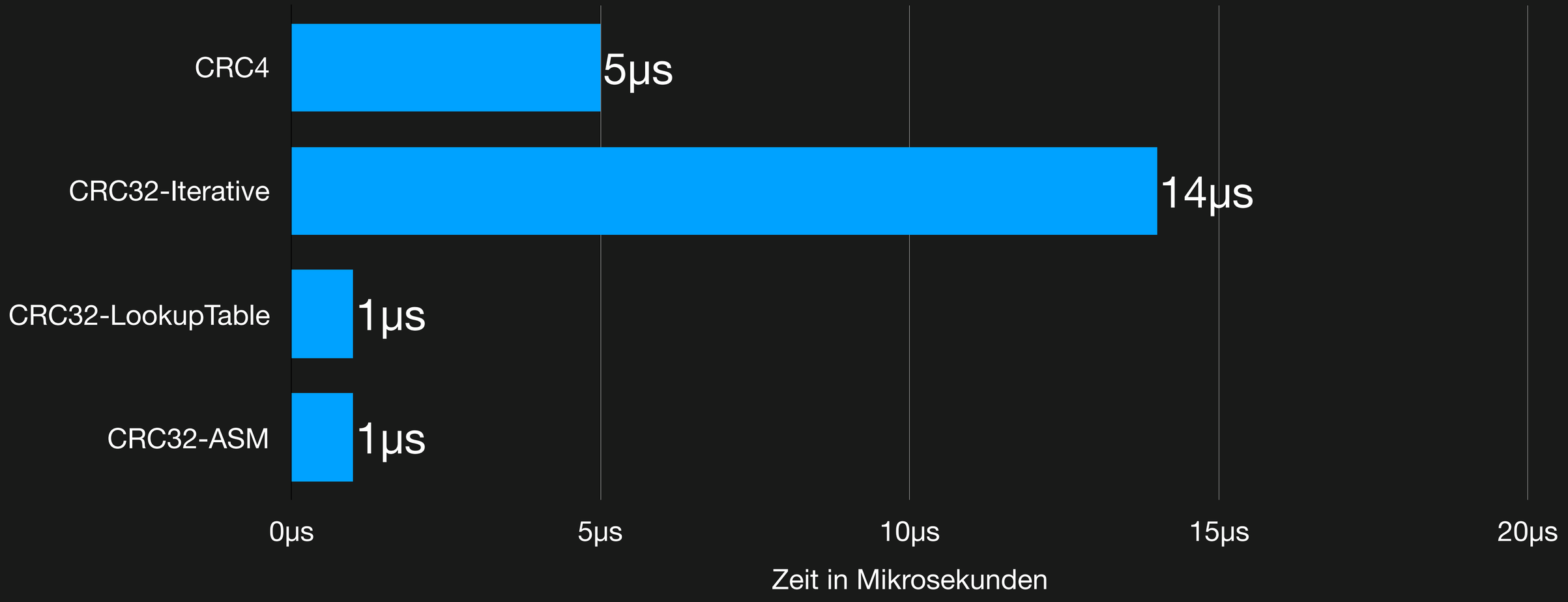
Entwickler-Dokumentation in Assembler

- wird von der C-Mainfunktion aus aufgerufen
  - gepaddete Datei wird übergeben
- Assemblerfunktion generiert Tabelle und speichert sie in x19
- die gepaddete Nachricht wird aufgeteilt in:
  - die ursprüngliche Nachricht
  - der gepaddete Anteil

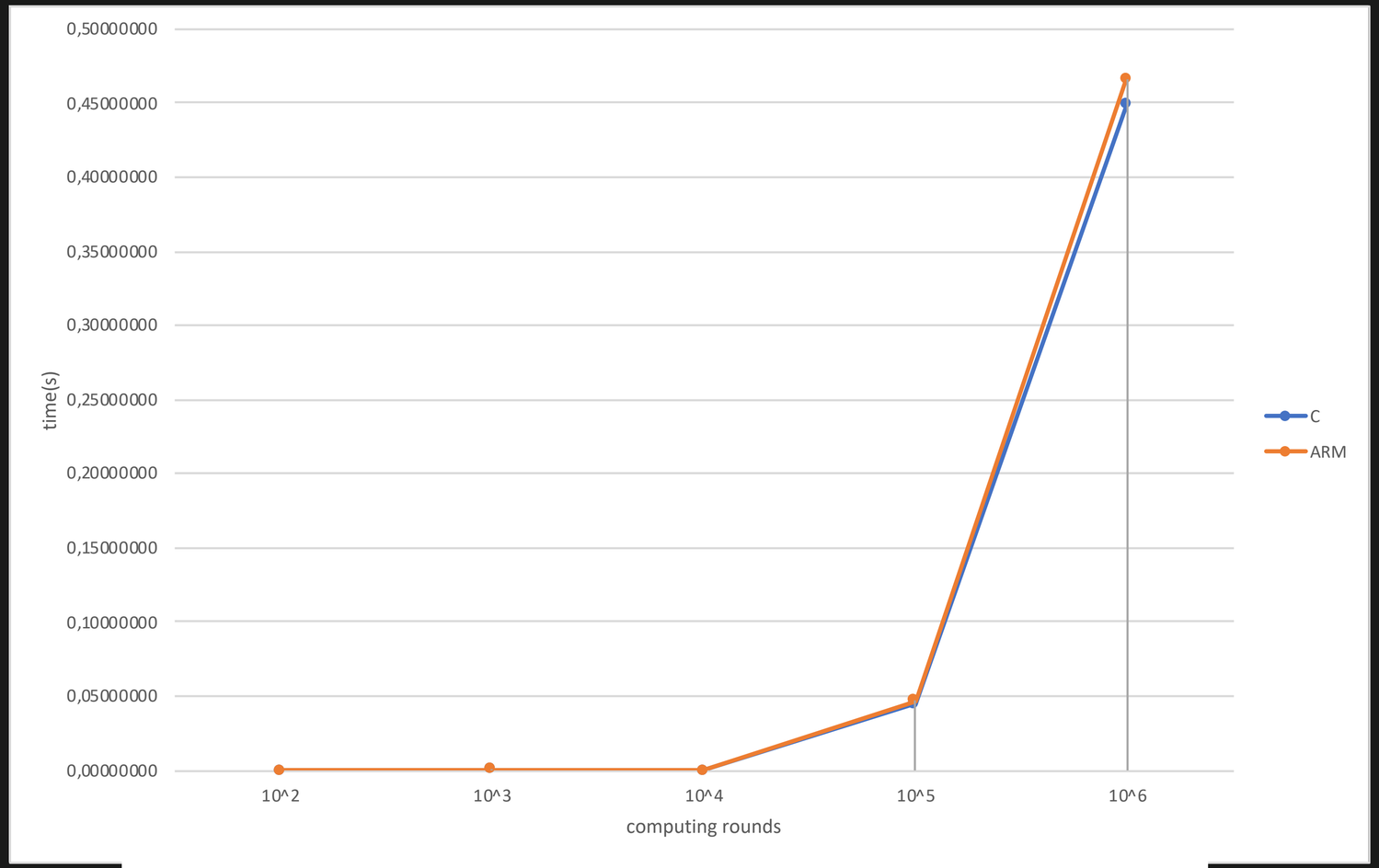
Ergebnisse

- Um die Effizienz der Implementationen zu vergleichen, wird die Prüfsumme einer Nachricht von den verschiedenen Implementationen berechnet.
- benutzte Nachricht: “SayHellotoMyLittleFriend”

Ergebnisse


CRC4
5µs
CRC32-Iterative
14µs
CRC32-LookupTable
1µs
CRC32-ASM
1µs
0µs
5µs
10µs
15µs
20µs
Zeit in Mikrosekunden

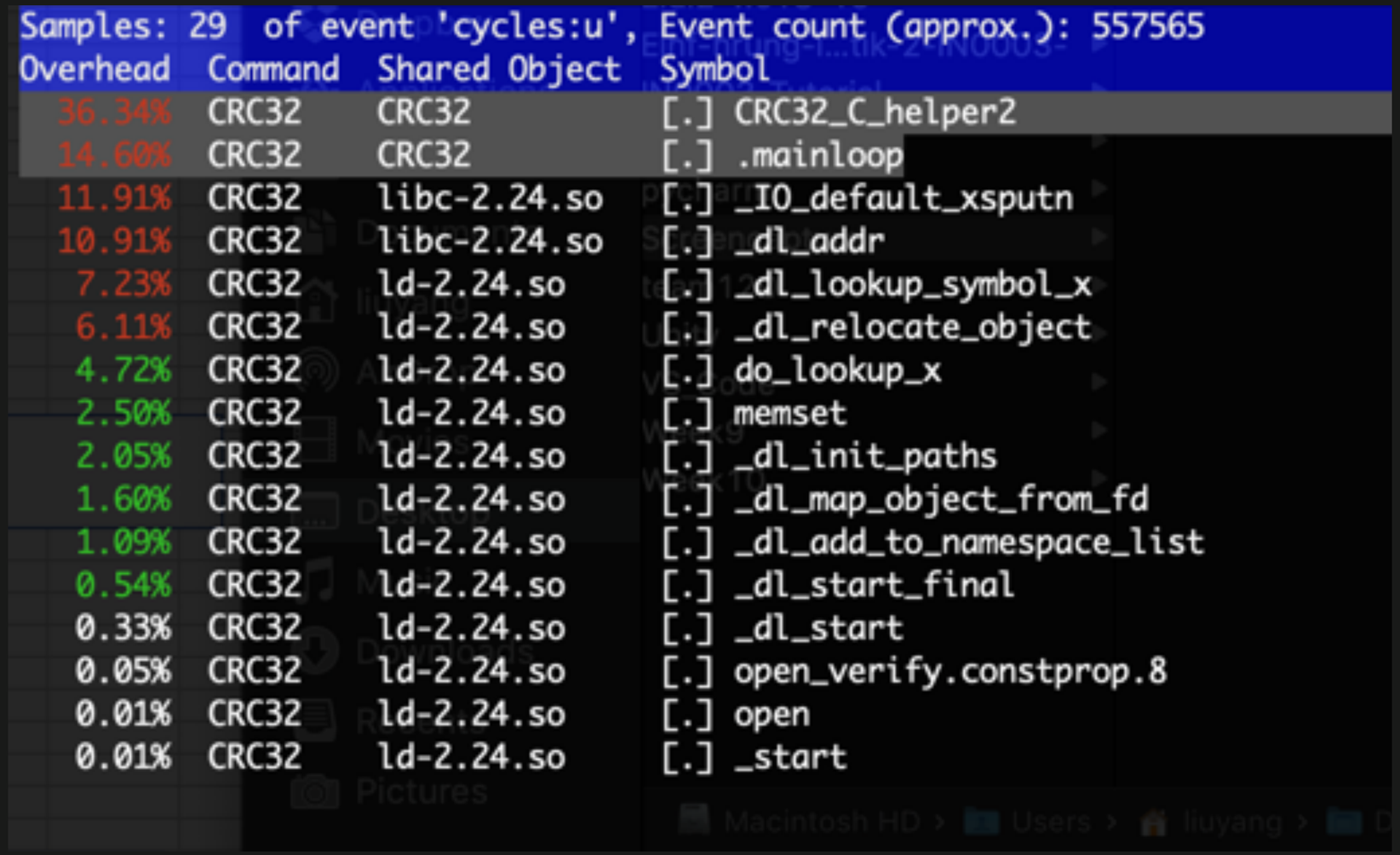
Ergebnisse



C	0,000044	0,000433	0,004337	0,044882	0,448642
ARM	0,00005	0,00045	0,004506	0,046646	0,466613

- Ergebnisse sehr ähnlich
- Unterschied anfangs vernachlässigbar klein
- mit der Anzahl der Durchläufe wächst auch der Unterschied und C wird tatsächlich schneller

Ergebnisse



```
Samples: 29  of event 'cycles:u', Event count (approx.): 557565
Overhead  Command  Shared Object  Symbol
  36.34%  CRC32    CRC32          [.] CRC32_C_helper2
  14.60%  CRC32    CRC32          [.] .mainloop
  11.91%  CRC32    libc-2.24.so   [.] _IO_default_xsputn
  10.91%  CRC32    libc-2.24.so   [.] _dl_addr
   7.23%  CRC32    ld-2.24.so     [.] _dl_lookup_symbol_x
   6.11%  CRC32    ld-2.24.so     [.] _dl_relocate_object
   4.72%  CRC32    ld-2.24.so     [.] do_lookup_x
   2.50%  CRC32    ld-2.24.so     [.] memset
   2.05%  CRC32    ld-2.24.so     [.] _dl_init_paths
   1.60%  CRC32    ld-2.24.so     [.] _dl_map_object_from_fd
   1.09%  CRC32    ld-2.24.so     [.] _dl_add_to_namespace_list
   0.54%  CRC32    ld-2.24.so     [.] _dl_start_final
   0.33%  CRC32    ld-2.24.so     [.] _dl_start
   0.05%  CRC32    ld-2.24.so     [.] open_verify.constprop.8
   0.01%  CRC32    ld-2.24.so     [.] open
   0.01%  CRC32    ld-2.24.so     [.] _start
```

Perf records

Ergebnisse

```
0000000000001148 <CRC32_C_table>:
    1148:       a9be7bfd        stp     x29, x30, [sp, #-32]!
    114c:       910003fd        mov     x29, sp
    1150:       f9000bf3        str     x19, [sp, #16]
    1154:       aa0003f3        mov     x19, x0
    1158:       97fffe02        bl      960 <strlen@plt>
    115c:       91001005        add     x5, x0, #0x4
    1160:       90000084        adrp    x4, 11000 <__FRAME_END__+0xfa98>
    1164:       52800001        mov     w1, #0x0                        // #0
    1168:       52800000        mov     w0, #0x0                        // #0
    116c:       f947e884        ldr     x4, [x4, #4048]
    1170:       14000007        b       118c <CRC32_C_table+0x44>
    1174:       d503201f        nop
    1178:       38614a63        ldrb    w3, [x19, w1, uxtw]
    117c:       11000421        add     w1, w1, #0x1
    1180:       b8627882        ldr     w2, [x4, x2, lsl #2]
    1184:       2a002060        orr     w0, w3, w0, lsl #8
    1188:       4a000040        eor     w0, w2, w0
    118c:       eb2140bf        cmp     x5, w1, uxtw
    1190:       d3587c02        ubfx    x2, x0, #24, #8
    1194:       54ffff21        b.ne    1178 <CRC32_C_table+0x30>  // b.any
    1198:       f9400bf3        ldr     x19, [sp, #16]
    119c:       a8c27bfd        ldp     x29, x30, [sp], #32
    11a0:       d65f03c0        ret
    11a4:       d503201f        nop
```

Disassembly

Genauigkeit

- Wann benutzt man CRC32 und wann CRC4?
  - mögliche Anzahl der Resten ≤ Länge der Nachricht
- Version muss also der Nachricht angepasst werden, damit es einen möglichst effizienten Tradeoff zwischen Schnelligkeit und Genauigkeit gibt.

Zusammenfassung

- Assembler vs. höhere Programmiersprachen
  - kaum Unterschiede (siehe Folie 28)
- CRC-Algorithmus
  - Lookup Table definitiv die bessere Implementierung

Quellen

infocenter.arm.com/help/topic/com.arm.doc.ihi0042f/IHI0042F_aapcs.pdf

E. Stein, Taschenbuch Rechnernetze und Internet, Chapter Fehlererkennung durch CRC, pages 86–87.

https://blog.csdn.net/xx326664162/article/details/51718857

https://www.rapidtables.com/convert/number/ascii-hex-bin-dec-converter.html

http://www.ghsi.de/pages/subpages/Online%20CRC%20Calculation/index.php?Polynom=10011&Message=53+61+79+48+65+6C+6C+6F+74+6F+4D+79+4C+69+74+74+6C+65+46+72+69+65+6E+64+

https://web.archive.org/web/20050408104838/http://www.4d.com/docs/CMU/CMU79909.HTM

http://www.keil.com/support/docs/2320.htm