

Paralleles Lesen und Schreiben von farbigen 2D-Barcodes mit MPI

**Seminar Wissenschaftliches Rechnen
ZHAW, Zürich**

Florian Lüthi*

14. Juni 2013

*`luethifl@students.zhaw.ch`

Inhaltsverzeichnis

1	Einleitung	3
2	Einführung in das Thema	4
2.1	Existierende Technologien	4
2.2	Vorgeschlagene Methode	5
3	Randbedingungen	8
4	Auswahl der Testdaten für die Arbeit	9
5	Algorithmische Beschreibung des Verfahrens	11
5.1	Codieren des Barcodes	11
5.2	Decodieren des Barcodes	11
5.3	Error Correction	11
5.4	Bewertung des Verfahrens	12
6	Analyse, Design und Implementierung des Verfahrens	14
6.1	Parallelisierung	14
7	Vergleich der Performancezeiten für ein, zwei oder mehrere CPU-Cores	15
8	Zusammenfassung	16
	Literaturverzeichnis	17

1 Einleitung

Diese Arbeit beschäftigt sich mit der von Md. Mashud Rana, M. E. Kawsar, M. E. Rabbani, S. M. M. Rashid und K. E. U. Ahmed vorgeschlagenen Methode des Lesens und Schreibens von farbigen zweidimensionalen Barcodes mit hoher Datenkapazität [2].

Sie beschreibt diese Methode, zeigt eine mögliche Umsetzung des Verfahrens in `C++` mit Parallelisierung mittels `MPI`, und macht eine quantitative Analyse der Implementation in Bezug auf die Performance. Zu guter Letzt zieht sie ein Fazit und überlegt sich den Sinn der angewandten Methode.

Zur Arbeit gehören dieses Dokument sowie die Beispielimplementation. Beides kann von <https://github.com/foyan/ColorizedBarCodec> bezogen werden.

Der Autor bedankt sich bei Dr. Alexander Herrigel für die Durchführung des Seminars.

2 Einführung in das Thema

2.1 Existierende Technologien

Seit den 1970er-Jahren [4] sind Barcodes aus Logistikbetrieben, Supermärkten etc. nicht mehr wegzudenken. Sie bilden das Rückgrat der automatisierten Warenbewirtschaftung. Das Scanning von eindimensionalen Barcodes wie in Abbildung 2.1 ist milliardenfach erprobt und in der Entwicklung eigentlich abgeschlossen.

Die Informationsdichte von eindimensionalen Barcodes ist aber sehr beschränkt. Beispielsweise erlaubt die eindimensionale Symbologie UPC-A maximal eine Billion eindeutiger Barcodes [8], was gemäss

$$\log_2 10^{12} = 39.863 \dots$$

einer maximalen Kapazität von weniger als 5 Bytes entspricht.

Aus diesem Grund wurden mit dem Aufkommen von billigen und sehr akkuraten Kameras in mobilen Devices zweidimensionale Symbologien entwickelt, welche es zumindest erlauben, ganze URLs von Webseiten oder ähnliches zu codieren. Google beispielsweise vermarktet den in Abbildung 2.2 ersichtlichen, ursprünglich von Toyota entwickelten *Quick Response* Code (QR) [7]. Dieser erlaubt es immerhin, maximal 2953 Bytes zu codieren [7].

Um die Kapazität weiter zu steigern, ist es nötig, eine dritte Dimension einzuführen. Weil aber Hologramme und 3D-Kameras noch nicht marktreif sind, liegt es nahe, die dritte Dimension als Farbe zu codieren. Microsoft geht in seiner Barcode-Technologie *High Capacity Color Barcode (HCCB)* seit 2007 [6] genau diesen Weg [3]. Ein Beispiel ist in Abbildung 2.3 abgebildet. Die maximale Kapazität von HCCB kann nicht genau genannt werden, weil die Barcodes keine definierte Grösse und Skalierung haben. Labortests scheinen aber gezeigt zu haben, dass es möglich ist, circa 2000 Bytes auf eine

Abbildung 2.1: Eindimensionaler Barcode (Quelle: Wikipedia)



Abbildung 2.2: QR Code (Quelle: Wikipedia)



Abbildung 2.3: High Capacity Color Barcode (Quelle: Wikipedia)



Fläche zu codieren, welche nicht viel grösser als eine Ein-Cent-Münze ist. Diese Fläche scheint sich mit einem 600-dpi-Laserdrucker herstellen zu lassen [3].

2.2 Vorgeschlagene Methode

Rana, Kawsar, Rabbani, Rashid und Achmed schlagen nun eine Methode vor, welche sich durch eine hohe Kapazität sowie ein sehr schnelles Leseverfahren auszeichnen soll [2]. Die grundlegenden Eckwerte sind die folgenden:

- Es werden 16 verschiedene Farben verwendet. Jede dieser Farben wird bijektiv einer Sequenz aus 4 Bits (Nibbles) zugeordnet (siehe Tabelle 2.1, [2]).
- Jedes Byte wird auf ein Zeichen abgebildet (siehe [2]).
- Die Fläche des Barcodes wird in ein 102×100 Zellen fassendes Raster aufgeteilt (im folgenden als Pixel bezeichnet). Bei einer angenommenen realen Grösse des Barcodes von 1.5×1.15 Zoll [2] ergibt sich daraus eine horizontale Auflösung von 68 DPI und eine vertikale Auflösung von 87 DPI.
- Jeweils 6 benachbarte Pixel in einem 3×2 -Feld codieren nun ein Byte, wobei ein ein vertikal gespiegeltes L für den ersten Nibble und ein horizontal gespiegeltes L für den zweiten Nibble entstehen (Abbildung 2.4). Über die Reihenfolge der

Tabelle 2.1: Zuordnung der Farben zu den Nibbles

Wert	Nibble (binär)	RGB-Wert
0	0000	rgb(255, 000, 000)
1	0001	rgb(000, 128, 000)
2	0010	rgb(000, 000, 025)
3	0011	rgb(165, 042, 042)
4	0100	rgb(128, 000, 000)
5	0101	rgb(100, 149, 237)
6	0110	rgb(255, 165, 000)
7	0111	rgb(255, 255, 000)
8	1000	rgb(238, 130, 238)
9	1001	rgb(128, 128, 128)
10	1010	rgb(000, 000, 128)
11	1011	rgb(000, 000, 000)
12	1100	rgb(173, 255, 047)
13	1101	rgb(138, 043, 226)
14	1110	rgb(128, 000, 128)
15	1111	rgb(095, 158, 160)

Wertigkeit der Nibbles wird keine Vorschrift gemacht – die abgebildete Rot/Grün-Kombination könnte also sowohl als 0000 0001 = 1 als auch als 0001 0000 = 16 interpretiert werden.

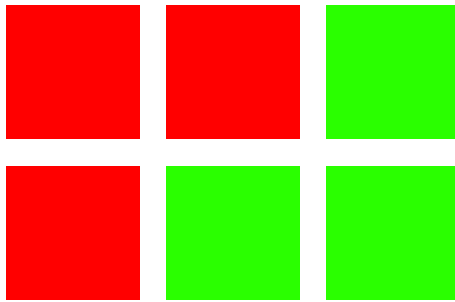
Weil je nach codierter Information die Länge der Codierung variiert, schlägt [2] vor, die ersten Nibbles als Header zu reservieren¹, welche als Information nur die Länge der nachfolgenden Informationen enthalten.

Die maximale Kapazität des Barcodes ist darum

$$C = \frac{102 \text{ px} \cdot 100 \text{ px}}{6 \text{ px}^2 / \text{byte}} - 2 \text{ bytes} = 1698 \text{ bytes}.$$

¹Bezüglich der Länge des Headers werden in [2] widersprüchliche Aussagen gemacht. Der Text reserviert die ersten 12 Pixel, was einer Länge von 2 Bytes entsprechen würde, wogegen die dazugehörige Abbildung eine Länge von 9 Pixeln kolportiert, was einer Länge von 1.5 Bytes entsprechen würde. Im Nachfolgenden wird darum eine Headerlänge von 2 Bytes angenommen.

Abbildung 2.4: Ein Byte in der vorgeschlagenen Methodik



3 Randbedingungen

Das beschriebene, untersuchte und implementierte Verfahren [2] impliziert in sich schon mehrere Randbedingungen beziehungsweise Einschränkungen bezüglich allgemeiner Brauchbarkeit, deren Diskussion in Kapitel 5.4 aufgenommen wird.

Vorauselend kann schon einmal gesagt werden, dass in Punkto Kapazitätserweiterung kein Fortschritt erreicht wird, weil der *QR*-Code intrinsisch schon eine höhere Kapazität aufweist, ohne allerdings den Vorteil der Farbdimension für sich reklamieren zu können.

Diese Arbeit hat darum explizit nicht das Ziel, eine wirkliche Alternative oder eine sinnvolle Entwicklungsrichtung in Sachen Barcodes aufzuzeigen. Sie ist eher als Schulbeispiel zur Umsetzung einer einfachen Algorithmik mittels Parallelisierung zu verstehen.

4 Auswahl der Testdaten für die Arbeit

Die Aufgabe des Algorithmus ist, Text in ein Bild umzuformen und wieder zurück. Dementsprechend werden drei Texte ausgewählt, welche zusammen mit ihren entsprechenden Barcodes die Testdaten bilden:

1. Um den minimalen Fall zu testen, wird der leere Text ausgewählt. Der korrespondierende Barcode ist in Abbildung 4.1 zu sehen.
2. Um einen durchschnittlichen Fall zu testen, wird die Definition der Aufklärung von Immanuel Kant aus dessen Essay *Beantwortung der Frage: Was ist Aufklärung?* von 1784 [1] verwendet:

Aufklärung ist der Ausgang des Menschen aus seiner selbstverschuldeten Unmündigkeit. Unmündigkeit ist das Unvermögen, sich seines Verstandes ohne Leitung eines anderen zu bedienen. Selbstverschuldet ist diese Unmündigkeit, wenn die Ursache derselben nicht am Mangel des Verstandes, sondern der Entschliessung und des Muthes liegt, sich seiner ohne Leitung eines anderen zu bedienen. Sapere aude! Habe Muth, dich deines eigenen Verstandes zu bedienen! ist also der Wahlspruch der Aufklärung. [5]

Die Länge dieses Textes beträgt (nach Ersetzung von ß durch ss sowie dem Einfügen einiger Zeilenumbrüche wegen der besseren Lesbarkeit) exakt 500 Bytes, was einer Ausnutzung der Kapazität von 29.4 Prozent entspricht. Der korrespondierende Barcode ist in Abbildung 4.2 zu sehen.

3. Um den maximalen Fall zu testen, wird ebendieser Text solange wiederholt, bis die maximale Kapazität erreicht ist. Der korrespondierende Barcode ist in Abbildung 4.3 zu sehen.

Abbildung 4.1: Der leere Barcode

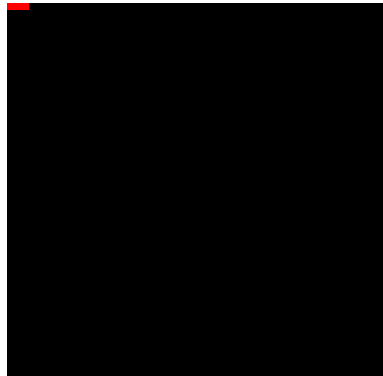


Abbildung 4.2: Die Antwort auf die Frage: Was ist Aufklärung? als Barcode

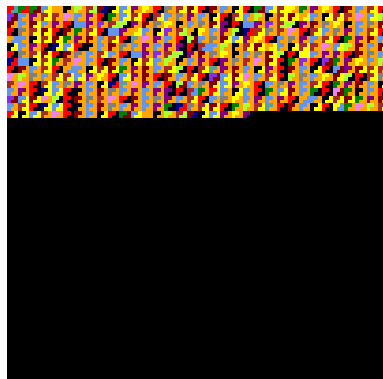
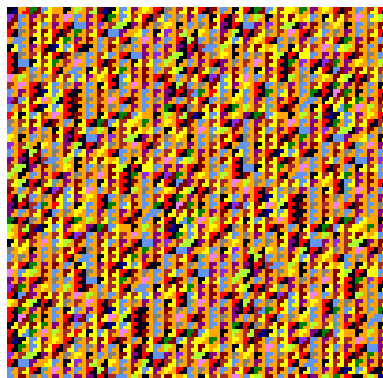


Abbildung 4.3: Möglichst viel Kant in einem Barcode



5 Algorithmische Beschreibung des Verfahrens

5.1 Codieren des Barcodes

Der folgende Algorithmus beschreibt den Codierungsvorgang eines Barcodes (seien `text` der zu codierende Text, `width` und `height` die Breite und Höhe des Barcodes, `color()` die Nibble-Farbe-Abbildung und `bc` der resultierende Barcode):

```
algo encode:
  for i in [0..|text|-1]:
    var x = (i*3) mod width
    var y = (i*3) / width * 2
    bc[x, y] = bc[x+1, y] = bc[x, y+1] = color(text[i].low)
    bc[x+1, y+1] = bc[x+2, y+1] = bc[x+2, y] = color(text[i].high)
```

5.2 Decodieren des Barcodes

Der folgende Algorithmus beschreibt den Decodierungsvorgang eines Barcodes (seien `bc` der zu decodierende Barcode, `width` und `height` die Breite und Höhe des Barcodes, `nibble()` die Farbe-Nibble-Abbildung, `detect_color()` der weiter unten beschriebene Error-Correction-Algorithmus und `text` der resultierende Text):

```
algo decode:
  for (y, x) in [0..height-1] x [0..width-1] step (2, 3):
    var c1 = detect_color(bc[x, y], bc[x+1, y], bc[x, y+1])
    var c2 = detect_color(bc[x+1, y+1], bc[x+2, y+1], bc[x+2, y])
    text.append(nibble(c2) nibble(c1))
```

5.3 Error Correction

Die in [2] vorgeschlagene probabilistische Error Detection und Correction-Methode funktioniert folgendermassen (seien `m`, `s1`, `s2` die Farben an den entsprechenden Position innerhalb des `Ls`):

```
algo detect_color:
  if m = s1:
    return m
```

```

if s1 = s2:
    return s1
return s2

```

5.4 Bewertung des Verfahrens

An der von [2] proklamierten Einfachheit und **computational efficiency** des Verfahrens gibt es gar nichts auszusetzen: Unter der gültigen Annahme, dass sowohl `color()` als auch `nibble` als reine Hashtable-Lookup-Funktionen eine asymptotische Laufzeitkomplexität von (sei x der Text oder der Barcode):

$$f(x) \in \mathcal{O}(1)$$

haben, beträgt die asymptotische Laufzeitkomplexität sowohl von `encode` wie auch von `decode`

$$f(x) \in \mathcal{O}(|x|).$$

Durch diese verfahrenstechnische Effizienz werden allerdings mehrere Mängel in Kauf genommen, die sich schnurstracks auf die Effektivität des Verfahrens auswirken:

1. Der Platz, den ein Byte an Information einnehmen kann, ist steht immer in einem festen Verhältnis zur gesamten Grösse des Barcodes (nämlich 6 Pixel). Dadurch hat ein resultierender Barcode immer automatisch die maximal mögliche Informationsdichte (was in den Abbildungen 4.1 und 4.2 sehr schön ersichtlich ist: da die zu codierenden Informationen weniger sind als die maximal mögliche Kapazität, bleibt ein geographisch zusammenhängender Teil des Barcodes ungenutzt, sprich schwarz).

Reifere Verfahren nutzen hingegen immer den ganzen zur Verfügung stehenden Platz, indem sie dynamisch die Informationsdichte ausdünnen. Dadurch vergrössern sie die Toleranz gegenüber der Ungenauigkeit während des Lesens (welche in der Realität bei Scans oder Fotografien natürlich immer da ist).

Das Verfahren könnte in dieser Hinsicht natürlich verbessert werden, weil die Länge der Information aus dem Header ersichtlich ist. Daraus liesse sich die physische Grösse eines Pixels berechnen. Allerdings ist der Header selber als mehrere Pixel codiert. Deshalb müsste trotzdem ein Verfahren implementiert werden, welches quasi die Grösse eines Pixels rekonstruiert – was sich natürlich saftig in einer höheren Gesamtkomplexität niederschlagen würde.

2. Das Verfahren beinhaltet keinerlei Paritätsprüfung der Daten. Abgesehen davon, dass die Farbwerte der gescannten Pixel exakt denjenigen in Tabelle 2.1 entsprechen müssen (was bei einem Scan in der Realität nur schon wegen den variierenden Lichtverhältnissen wegen sowieso utopisch ist), würde der als **Error Correction und Detection**-Algorithmus [2] kolportierte Algorithmus `detect_color()` bei drei

verschiedenen (und gemäss Tabelle 2.1 gültigen!) Farbwerten unreflektierterweise einfach den letzten der drei Werte als richtig erachten.

Das Verfahren könnte verbessert werden, indem es um eine typische Paritätsprüfungsmethode ergänzt werden würde – beispielsweise CRC. Die asymptotische Komplexität würde nicht darunter leiden; der Seiteneffekt wäre allerdings, dass nicht mehr zwei Nibbles ein ganzes Byte ergeben würden und darum das Mapping zwischen Bytes und Zeichen massiv komplizierter würde.

6 Analyse, Design und Implementierung des Verfahrens

6.1 Parallelisierung

Die Codierungs- und Decodierungs-Algorithmen selber sind einigermaßen simpel. Der parallelisierbare Teil der beiden Algorithmen besteht allerdings nur aus dem Mapping-Teil, da der zu codierende Text oder das zu decodierende Bild natürlich nur auf jeweils einem Node des Clusters vorliegen.

Da der Codierungs-Algorithmus mehr oder weniger die Umkehrfunktion des Decodierungs-Algorithmus ist, kann die Infrastruktur der beiden geteilt werden. Dadurch ist es möglich, die Parallelisierung vollständig aus den Algorithmen zu extrahieren: Sei `algo` einer der beiden besprochenen Algorithmen mit diesen Teilalgorithmen:

- `init`
- `get_sliced_input`
- `process_slice`
- `collect_slice`
- `finalize`,

dann sieht die parallelisierte Ausführung von `algo` auf dem Master-Node folgendermaßen aus:

```
algo.init()
for each slave node:
    var in = get_sliced_input(slave)
    invoke_slave(in)
var my_in = get_my_input()
algo.process_slice(my_input)
result += algo.collect_slice(my_input)
for each slave node:
    result += algo.collect_slice(slave)
algo.finalize()
```

Die parallelisierte Ausführung auf einem Slave-Node besteht im Wesentlichen aus der Ausführung von `algo.process_slice()`.

7 Vergleich der Performancezeiten für ein, zwei oder mehrere CPU-Cores

8 Zusammenfassung

Literaturverzeichnis

- [1] Immanuel Kant. Beantwortung der Frage: Was ist Aufklärung? *Berlinische Monatsschrift*, 4:481–494, 1784.
- [2] Md. Mashud Rana, M. E. Kawsar, M. E. Rabbani, S. M. M. Rashid und K. E. U. Ahmed. An Enhanced Two-Dimensional Color Barcode System. *Journal of Emerging Trends in Engineering and Applied Sciences (JETEAS)*, 2(1):126–131, 2011.
- [3] Microsoft Research. About High Capacity Color Barcode Technology. <http://research.microsoft.com/en-us/projects/hccb/about.aspx>, 2013. [Online; accessed 14-June-2013].
- [4] Wikipedia. Barcode — Wikipedia, The Free Encyclopedia. <http://en.wikipedia.org/w/index.php?title=Barcode&oldid=558925085>, 2013. [Online; accessed 14-June-2013].
- [5] Wikipedia. Beantwortung der Frage: Was ist Aufklärung? — Wikipedia, Die freie Enzyklopädie. http://de.wikipedia.org/w/index.php?title=Beantwortung_der_Frage:_Was_ist_Aufkl%C3%A4rung%3F&oldid=118761194, 2013. [Online; Stand 14. Juni 2013].
- [6] Wikipedia. High Capacity Color Barcode — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=High_Capacity_Color_Barcode&oldid=559150607, 2013. [Online; accessed 14-June-2013].
- [7] Wikipedia. QR code — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=QR_code&oldid=559415969, 2013. [Online; accessed 14-June-2013].
- [8] Wikipedia. Universal Product Code — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Universal_Product_Code&oldid=558763746, 2013. [Online; accessed 14-June-2013].

Abbildungsverzeichnis

2.1	Eindimensionaler Barcode (Quelle: Wikipedia)	4
2.2	QR Code (Quelle: Wikipedia)	5
2.3	High Capacity Color Barcode (Quelle: Wikipedia)	5
2.4	Ein Byte in der vorgeschlagenen Methodik	7
4.1	Der leere Barcode	10
4.2	Die Antwort auf die Frage: Was ist Aufklärung? als Barcode	10
4.3	Möglichst viel Kant in einem Barcode	10

Tabellenverzeichnis

2.1	Zuordnung der Farben zu den Nibbles	6
-----	---	---