technische universität
dortmund

Bachelorarbeit

Parallelisierung einer speichereffizienten Approximation der LZ77-Faktorisierung

Gajann Sivarajah

Gutachter:
Prof. Dr. Johannes Fischer
M.Sc. Patrick Dinklage

Technische Universität Dortmund
Fakultät für Informatik
LS-11
http://afe.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Hintergrund

Eine Referenz [1].

1.2 Aufbau der Arbeit

Kapitel 2

Grundlagen

Zunächst stellen wir die verwendete Terminologie und relevante Konzepte bzw. Phänomene dar.

2.1 Kompression

2.1.1 Verlustfreie Kompression

Der Prozess der Kompression überführt eine Repräsentation einer finiten Datenmenge in eine möglichst kompaktere Form. Eine verlustfreie Kompression ist gegeben, falls die Abbildung zwischen der ursprünglichen und komprimierten Repräsentation bijektiv ist. Die Korrektheit einer verlustfreien Kompression kann daher durch die Angabe einer Dekompressionsfunktion nachgewiesen werden. Ist diese Vorraussetzung nicht gegeben, so handelt es sich um eine verlustbehaftete Kompression, da eine Rekonstruktion der ursprünglichen Datenmenge nicht garantiert werden kann.

2.1.2 Eingabe

Unsere Eingabe sei durch eine n-elementige Zeichenfolge $S = e_1...e_n$ über dem numerischen Alphabet Σ mit $e_i \in \Sigma \; \forall i = 1, ..., n$ gegeben. Für jede beliebige Zeichenfolge S wird mit $|S|$ dessen Länge n bezeichnet. Der Ausdruck $S[i..j] \in \Sigma^{j-i+1}$ mit $1 \leq i \leq j \leq n$ beschreibt die Teilfolge $e_i...e_j$, wobei im Falle, dass $i = j$ ist, das einzelne Zeichen e_i referenziert wird. Alternativ kann ein einzelnes Zeichen e_i auch durch $S[i]$ referenziert werden. Eine Teilfolge der Form $S[1..k]$ mit $k \leq n$ wird als Präfix von S bezeichnet.

2.1.3 Faktorisierung

Ein charakteristisches Merkmal für die Klasse von Lempel-Ziv-Kompressionsverfahren ist die Repräsentation der Ausgabe in Form einer Faktorisierung. Für eine Eingabe $S = e_1...e_n$ wird eine Faktorisierung $F = f_1...f_z$ mit $z \leq n$ derart erzeugt, dass die Faktoren S in eine

equivalente Folge von nichtleeren Teilfolgen zerlegen. Hier ist jeder Faktor f_i mit $1 \leq i \leq z$ als Präfix von $S[|f_1...f_i - 1| + 1..n]$ definiert, der bereits in $S[1..|f_1...f_i|]$ vorkommt oder als einzelnes Zeichen ohne vorheriges Vorkommen. Die im Folgenden betrachteten Algorithmen können speziell der Klasse der LZ77-Kompressionsverfahren zugeordnet werden, dessen Faktoren im Schema des Lempel-Ziv-Storer-Szymanski repräsentiert werden sollen. Zur Darstellung von Referenzen wird das Tupel (len, pos) verwendet, wobei pos die Position des vorherigen Vorkommens und $len > 0$ die Länge des Faktors beschreibt. Einzelne Zeichen können wiederum durch das Tupel $(0, e)$ mit $e \in \Sigma$ dargestellt werden.

2.1.4 Dekompression

Die Dekompression beschreibt den Umkehrprozess der Kompression und erlaubt im Falle eine verlustfreien Kompression die Rekonstruktion der ursprünglichen Datenfolge. Im Falle von Verfahren der LZ77-Familie, kann die Dekompression durch die folgende Abbildung definiert werden,

$$DECOMP_{LZ77} : F(1..z) \rightarrow S(1..n). \tag{2.1}$$

Algorithmus 2.1 DECOMP$_{LZ77}$

Eingabe:$F = f_1...f_z$ *Ausgabe:*$S = e_1...e_n$

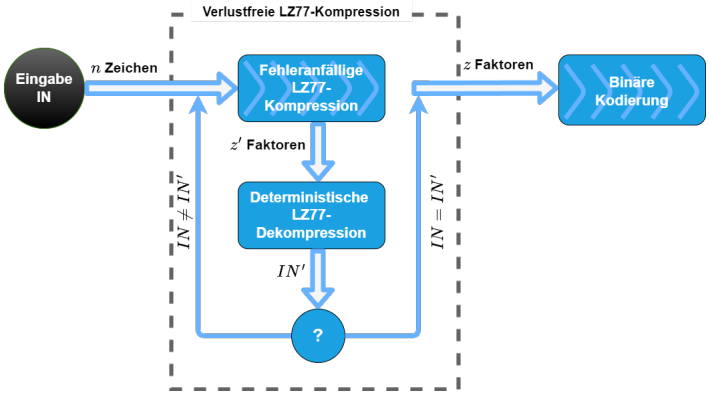
```
S ← ∅
for i = 1 to z do
  [len, ref] ← f_i
  if len = 0 then
    S ← S + ref
  else
    for j = 0 to len − 1 do
      S ← S + S[ref + j]
    end for
  end if
end for
return S
```

Der dargestellte Algorithmus 2.1 beschreibt eine mögliche Implementierung der Dekompression für eine Faktorisierung $F = f_1...f_z$ zu der Eingabe $S = e_1...e_n$. Der beschriebene Algorithmus iteriert durch alle Faktoren und fügt die referenzierten Zeichen einzeln in die Ausgabe S ein. Damit kann die Laufzeit des Algorithmus auf $O(n)$ geschätzt werden.

2.1.5 Verlustbehaftete Kompression

Im Rahmen dieser Arbeit werden einen Approximationsalgorithmus betrachten, der aufgrund der verwendeten String-Matching-Technik eine fehlerhafte Faktorisierung mit einer

Abbildung 2.1: Las-Vegas-Algorithmus



beschränkten Wahrscheinlichkeit erzeugen kann. Die Korrektheit der Dekompression kann intern und extern durch explizite Vergleiche der Zeichenfolgen erkannt werden. Da der Kompressionsprozess in diesem Fall mit anderen Parametern wiederholt werden kann, können wir einen verlustfreien Las-Vegas-Algorithmus konstruieren.

In Abbildung 2.1 wird die beschriebene Schleife illustriert. Der Algorithmus wird solange wiederholt, bis eine korrekte Faktorisierung erzeugt wurde. Dass die Anzahl der Wiederholungen beschränkt ist, werden wir in der Analyse des Approximationsalgorithmus und der praktischen Evaluation zeigen.

2.1.6 Binäre (De-)Kodierung

Die Abbildung $Bin_{IO} : \Sigma^* \to N$ gibt die Anzahl der Bits für die Kodierung einer beliebigen Zeichenfolge an. Im Rahmen dieser Arbeit gehen wir davon aus, dass die Eingabe S in binärer Form vorliegt und auf dem Alphabet $\Sigma = \{1, ..., 255\}$ erzeugt wurde. Jedes Zeichen wird durch 8 Bits, oder 1 Byte, dargestellt und erlaubt einen Offline-Zugriff. Die gesamte binäre Eingabegröße sei damit gegeben durch

$$N_{Bin} = \sum_{i=1}^{|S|} Bin_{IO}(S[i]) = 8 * |S|. \tag{2.2}$$

Die eingelesene Eingabefolge wird durch den Kompressionsalgorithmus in die Faktorfolge $S = f_1...f_z$ überführt. Jeder Faktor f_i, der Form (len, pos) oder $(0, e)$, werde durch $Bin_{IO}(f_i)$ Bits kodiert. Die binäre Ausgabegröße ergibt sich aus dem Ausdruck

$$Z_{Bin} = \sum_{i=1}^{z} Bin_{IO}(f_i). \tag{2.3}$$

, wobei die Anzahl der Bits für die Kodierung der Faktoren f_i durch den Kompressionsalgorithmus und der Kodierungsstrategie bestimmt wird.

2.1.7 Metriken

Die Qualität einer Kompression kann durch verschiedene Metriken quantifiziert werden. Zum Einen beschreibt die Kompressionsrate CR den Grad der Kompression und ist durch den Ausdruck,

$$CR = \frac{Z_{Bin}}{N_{Bin}} \tag{2.4}$$

, definiert. Da die Kodierung der Faktoren nicht eindeutig aus der Wahl des Kompressionsalgorithmus eingegrenzt wird, ist stattdessen die Anzahl der erzeugten Faktoren ein weiteres geeignetes Gütemaß. Für die Eingabe S der Länge n und der Ausgabe $f_1...f_z$ sei die Faktorrate durch

$$FR = \frac{z}{n} \tag{2.5}$$

gegeben. In beiden Fällen wird ein niedriger Wert bevorzugt, da dieser auf eine bessere Extraktion von Redundanzen hinweist.

2.2 Parallelität

Das Ziel dieser Arbeit ist die Entwicklung und Evaluation eines parallel Kompressionsalgorithmus. Im Folgenden definieren wir die Rahmenbedingungen und Konzepte der Parallelität.

2.2.1 Shared-Memory-Modell

Unser Algorithmus agiere auf einem Shared-Memory-Modell mit P Ausführungseinheiten, welches im Gegensatz zum Distributed-Memory-Modell allen beteiligten Ausführungseinheiten bzw. Prozessoren einen gemeinsamen Zugriff auf den Speicher ermöglicht. Im Rahmen der Arbeit und Kommunikation unter den Prozessoren wird man jedoch auf Konflikte bei gleichzeitigen Speicherzugriffen zustoßen. Ein parallel modellierter Algorithmus muss explizit hinsichtlich der Korrektheit und Effizienz Mechanismen zur Synchronisation implementieren.

2.2.2 Metriken

Das Ziel der Parallelisierung eines Algorithmus liegt hauptsächlich in einer Verbesserung der Laufzeit, insbesondere unter Berücksichtigung von Ressourcenkonflikten. Die zeitliche Beschleunigung der Laufzeit kann durch den Speedup SP bemessen werden. Für eine Eingabe S der Länge n brauche ein sequenzieller Durchlauf $T(n, p = 1)$ Zeit, während ein paralleler Algorithmus mit P Prozessoren $T(n, p = P)$ an Zeit benötigt. Der Speedup ist dabei definiert durch

$$SP(n, P) = \frac{T(n, 1)}{T(n, P)}. \tag{2.6}$$

Kapitel 3

Kompressionsalgorithmen

3.1 (exakte) LZ77-Kompression

Die resultierende Faktorisierung aus der exakten LZ77-Kompression wird im Folgenden als Referenz für die Evaluation der Ergebnisse anderer Algorithmen verwendet. Zusätzlich zu der bereits beschriebenen Struktur von Faktorisierung nach dem Lempel-Ziv-Schema erfüllen referierende Faktoren die Eigenschaft, dass sie stets die längste Zeichenfolge referenzieren, die ein vorheriges Vorkommen hat. Der Algorithmus hat hat damit einen Greedy-Charakter. Im Rahmen der Evaluation nutzen wir Implementierung des Algorithmus, welcher auf der Erzeugung eines Suffixarray basiert. Für eine Eingabe der Länge n kann die Laufzeit des Algorithmus auf $O(n)$ geschätzt werden. Der eingespannter Speicher kann auch auf $O(n)$ geschätzt werden, wobei in der Praxis ein großer konstanter Faktor zu erwarten ist.

Kapitel 4

Praktische Evaluation

4.1 Experimentelle Umgebung

Die folgenden Experimente wurden auf Rechner mit Ubuntu 24.04 und einem AMD EPYC 7763 CPU mit 16 nutzbaren Hardwarethreads durchgeführt. Die C++-Implementierung wurde mithilfe von GCC in der Version 13.2.0 kompiliert.

4.2 Implementierung

Die Implementierung der Algorithmen erfolgte im C++20-Standard.

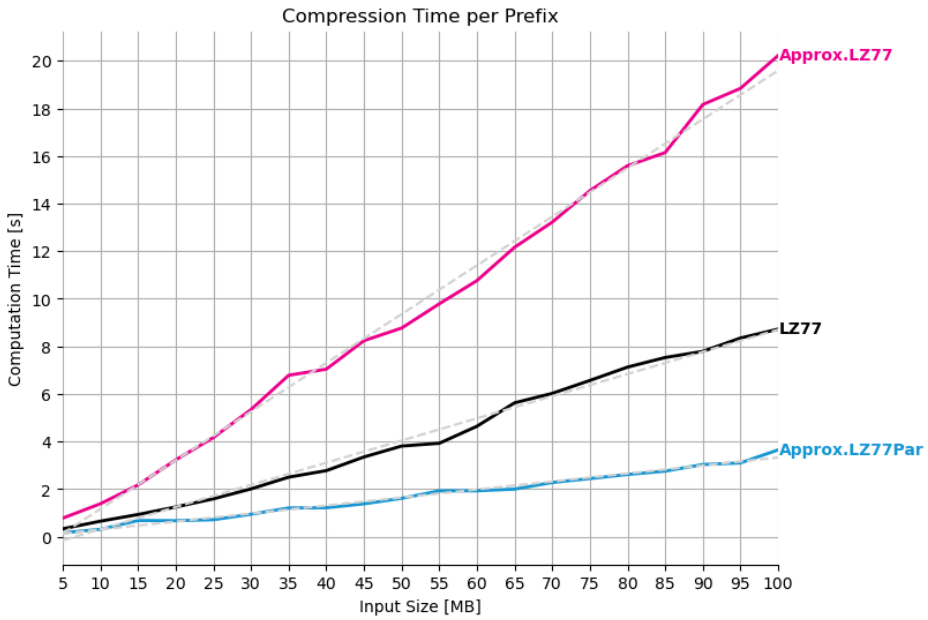
4.3 Daten

Die Experimente wurden auf verschiedenen Datenbausteinen vom Pizza & Chili-Corpus durchgeführt. Die verwendeten Datenbausteine sind in der folgenden Tabelle aufgelistet.

Abbildung 4.1: Auflistung der verwendeten Eingabedaten auf dem Pizza & Chili-Corpus. Neben der Bezeichnung der Datei ist die Dateigröße, die Größe des Alphabets, sowie eine kurze Beschreibung des Inhalts angegeben

Datei	Größe	Alphabet	Beschreibung
`dna`	100MB	4	DNA-Sequenzen
`english`	100MB	256	Englische Texte
`proteins`	100MB	20	Proteinsequenzen
`sources`	100MB	256	Quellcode
`xml`	100MB	256	XML-Dateien

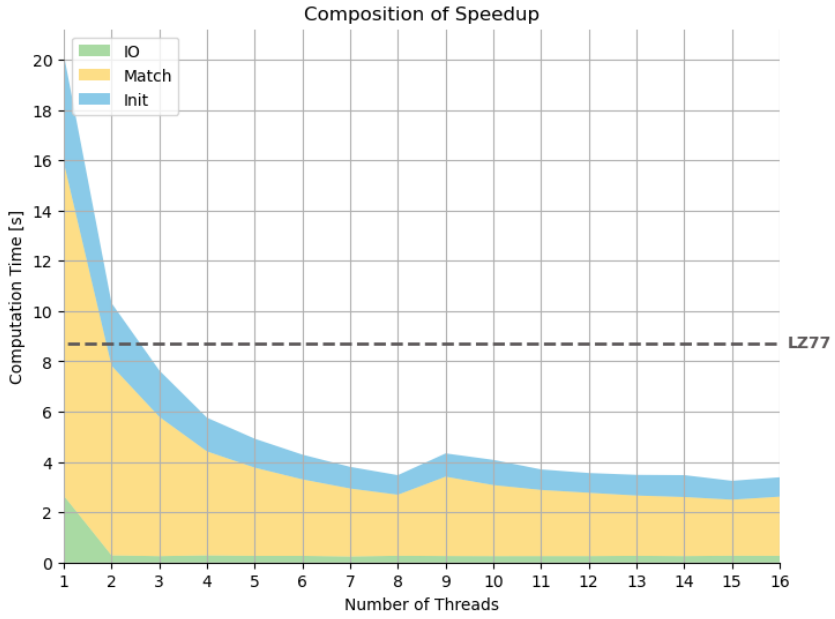
Abbildung 4.2: Laufzeit der Algorithmen auf Präfixen verschiedener Größen der Datei Proteins



4.4 Ergebnisse

Die Ergebnisse der Experimente sind in den folgenden Tabellen und Abbildungen dargestellt.

Abbildung 4.3: Laufzeit der parallelel Approximation von LZ77 mit verschiedenen Anzahlen von Threads



Anhang A

Weitere Informationen

Literaturverzeichnis

[1] Aggarwal, Alok und Jeffrey Scott Vitter: *The Input/Output Complexity of Sorting and Related Problems.* Communications of the ACM, 31(9):1116–1127, 1988.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 9. Juli 2024

Muster Mustermann