OPTIMALE NACHLADESTRATEGIEN FÜR PUFFERSPEICHER UNTER BERÜCK - SICHTIGUNG DER PROGRAMMSTRUKTUR

von

Otto Spaniol

Inhalt

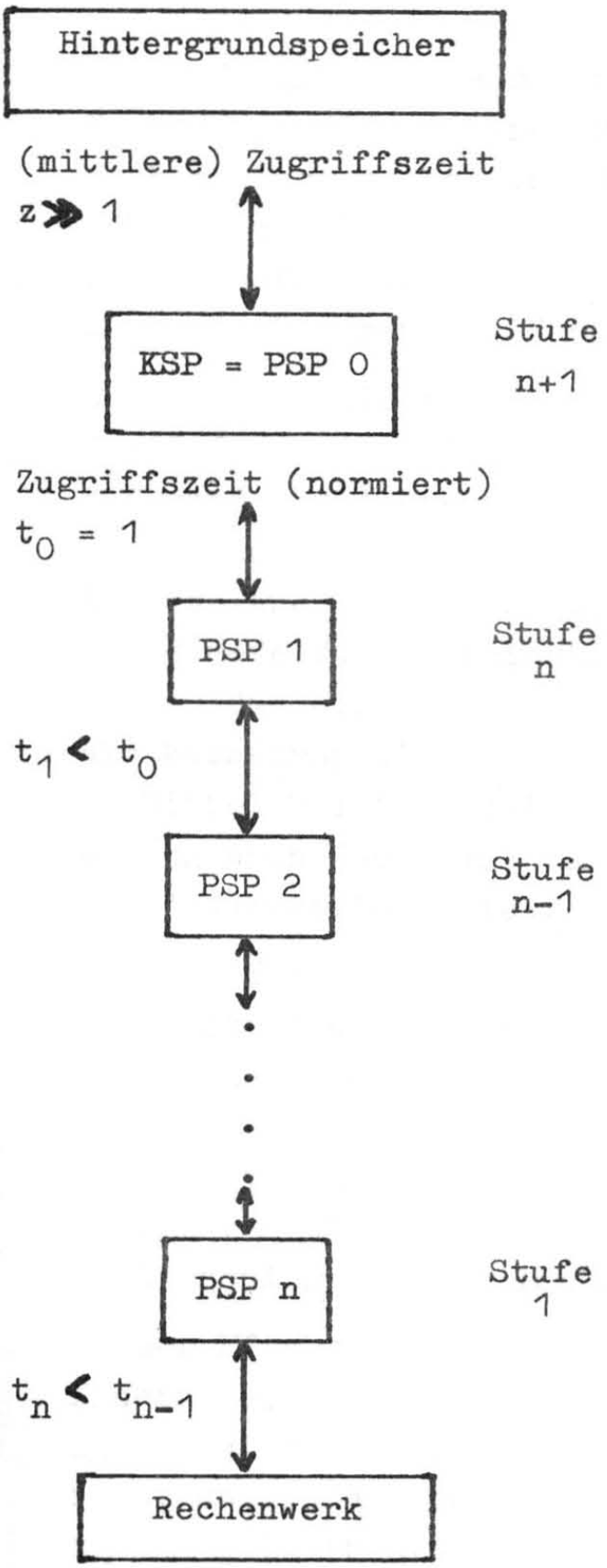
1. Einleitung und Überblick

Eines der Hauptprobleme, welche die Beschleunigung einer Rechenanlage erschweren, ist die gegenüber der Geschwindigkeit moderner Rechenwerke relativ lange Speicherzugriffszeit. Zwar gibt es Ansätze für neue Speichermedien, die mit den heutigen arithmetischen Einheiten mithalten könnten, doch sind solche Speicher zur Zeit noch unerschwinglich und werden auch auf lange Sicht wesentlich teurer bleiben als konventionelle Kernspeicher. Außerdem wird gleichzeitig mit der Entwicklung neuer Speichermedien die Beschleunigung der arithmetischen Einheit betrieben, so daß die Speicherzugriffszeit auf Dauer ein Engpaß bleiben wird, den man auch durch Verwendung von Look-ahead- und Streaming-Prinzipien höchstens abschwächen kann.

Aus Kostengründen ist es deshalb bei einer realen Rechenmaschine notwendig, die Kapazität des relativ teuren Kernspeichers klein gegenüber dem praktisch unbegrenzten Fassungsvermögen der billigeren Hintergrundspeichermedien (Platte, Band, Trommel) zu halten.

Die Motivation dafür ist neben technischen Restriktionen die Tatsache, daß Hintergrundinformationen im Vergleich zur Gesamtzahl der während des Programmlaufs angeforderten Daten relativ selten benötigt werden, wodurch bei geeigneter Programmierung der Zeitverlust durch die längere Zugriffszeit zum Hintergrund im Mittel nicht sehr stark zu Buche schlägt.

Eine Erweiterung dieses Prinzips bietet sich durch Ausdehnung dieser Speicherhierarchie mittels Einfügen von weiteren Stufen an. Man bezeichnet die neu hinzukommenden inneren Stufen der Hierarchie als Pufferspeicher (PSP). Je weiter eine Stufe vom eigentlichen Rechenwerk (Central Processing Unit, CPU) entfernt steht, desto größer muß ihr Fassungsvermögen sein, wenn das Prinzip sinnvoll angewandt werden soll; gleichzeitig wird die Zugriffszeit mit dem Abstand vom Rechenwerk anwachsen.



Das Prinzip arbeitet in der Weise, daß die als nächste benötigte Information (d.h. der Inhalt der Adresse eines Befehls oder eines Rechendatums) in der nächsthöheren Stufe gesucht wird, sofern sie in der niedrigeren nicht vorhanden ist.

Wird die Adresse, welche die gewünschte Information enthält, erst auf der Stufe $i > 1$ gefunden, so wird sie nicht nur in die CPU, sondern auch in die Stufen $i-1,\ldots,1$ übertragen; damit ist, wenn die Pufferspeicher mit aktuellen Daten gefüllt sind, jede Stufe in der nächsthöheren enthalten; (dies gilt noch nicht bei einem Kontextwechsel, d.h. etwa beim Beginn eines neuen Programms. In diesem Fall enthalten alle PSP wertlose Informationen, die zunächst durch aktuelle Daten zu ersetzen sind; dies wird als Einschwingvorgang bezeichnet.)

Der Transport von Information ist während des Einschwingvorgangs kein Problem, danach muß jedoch ein Teil der Information eines oder mehrerer PSP niedrigerer Stufe durch die neue verdrängt werden, genauer gesagt: mit der

neuen Information ausgetauscht werden. Von der Art dieses Nachladeprozesses hängt die mittlere Geschwindigkeit der Rechenanlage entscheidend ab, da große Fehlraten (Wahrscheinlichkeit dafür, daß die benötigte Adresse nicht im PSP nächsthöherer Stufe zu finden ist) zu einer großen Zahl von langsameren Nachladeoperationen führen.

Durch ungeschicktes Nachladen kann es vorkommen, daß gerade die Information wieder benötigt wird, die erst kurz zuvor aufgrund der Austauschkriterien der gewählten Nachladestrategie verdrängt wurde. Jedoch ist man gegen solche Effekte auch bei Verwendung der bestmöglichen Strategien nicht gefeit. Mit Leichtigkeit lassen sich Programme angeben, die eine beliebig vorgegebene Nachladestrategie als katastrophal erscheinen lassen. Die Frage ist, was man im Mittel bei "sinnvollen" Programmen erreichen kann, sofern sich überhaupt definieren läßt, was unter "sinnvoll" zu verstehen ist.

Der Grund für die Einführung einer Pufferspeicherhierarchie ist in erster Linie darin zu sehen, daß erfahrungsgemäß die meisten Programme aktive und weniger aktive Bereiche (sowohl der Rechendaten als auch der Befehle) haben; die aktiven Bereiche (Denning [2] spricht von "working set") ändern sich während des Programmlaufs, nach Denning relativ langsam.

Dieser Effekt ist auf die Schleifenstruktur der meisten Programme sowie auf andere Programmiergewohnheiten zurückzuführen. Man hofft deshalb auf eine relativ geringe Fehlrate, sobald die Pufferspeicher mit relevanten Informationen gefüllt sind.

Bereits hier sei bemerkt, daß man unter Verwendung aufwendiger Prinzipien die Fehlrate bei festem Nachladeprozeß weiter verringern kann, beispielsweise durch eine Reihe vorausschauender Zugriffe (Look-ahead-Prinzipien) oder durch gleichzeitiges Suchen in allen höheren Stufen der PSP-Hierarchie. Solche Varianten ändern jedoch an den qualitativen Aussagen nur wenig.

In der vorliegenden Arbeit lassen wir solche Varianten außer acht und beschäftigen uns nur mit der Struktur eines einzigen PSP; ferner wird praktisch unbegrenzte Kernspeicherkapazität vorausgesetzt, so daß die Probleme mit dem Nachladen von Hintergrundinformationen, wofür im Gegensatz zu KSP und PSP nur mittlere Zugriffszeiten angegeben werden können, entfallen.

Entscheidend für den Zeitgewinn, der sich durch die Verwendung eines Pufferspeichers erzielen läßt, sind in erster Linie zwei Punkte:
a. das Verhältnis der Zugriffszeiten von PSP und KSP
b. die (mittlere) Fehlrate P.

Die Geschwindigkeit des Pufferspeichers gegenüber dem Kernspeicher hängt neben technischen Restriktionen im wesentlichen von dem vertretbaren Kostenaufwand ab. Wir berücksichtigen diesen Faktor nicht, da sein Einfluß leicht zu überschauen ist.

Die Fehlrate hängt ihrerseits von verschiedenen Größen ab und zwar von:

1. der Größe des Pufferspeichers
2. der Größe der Nachladeeinheit
   (das Nachladen erfolgt blockweise, da der Zugriff zu einem Block praktisch ebensoschnell vollzogen werden kann wie zu einer einzelnen Adresse; ferner befinden sich wegen der vorausgesetzten Existenz von aktiven und inaktiven Bereichen mit hoher Wahrscheinlichkeit in der Nähe eines gerade benötigten Datums weitere Daten, auf die in naher Zukunft zugegriffen wird. Wir denken uns daher PSP und KSP als in gleichgroße Blöcke eingeteilt und tauschen nur ganze Blöcke aus).
3. der Art des Nachladeprozesses
4. der Struktur des durchschnittlichen Programms,
   d. h. von den Eigenschaften der zeitlichen Folge der Zugriffe zu Daten und Befehlen. Diese Zugriffsfolgen hängen wesentlich davon ab, ob es sich um Zugriffe auf

Daten (in den Akkumulator) oder Befehle (ins Befehlsregister) handelt. Die Trennung von Programm- und Rechenspeicher wie in [8] und entsprechende Aufteilung des Pufferspeichers ist deshalb sinnvoll.

Von besonderer Bedeutung ist der letzte Punkt, der alle anderen (insbesondere die Güte der Nachladestrategie) in starkem Maße beeinflußt. Trotzdem ist gerade dieser Punkt bei fast allen bisher veröffentlichten einschlägigen Arbeiten (vgl. z. B. [1] und das dort angegebene umfassende Literaturverzeichnis) entweder unberücksichtigt geblieben oder nur angedeutet worden.

Das Problem liegt darin, daß Aussagen über die Programmstruktur sehr schwierig sind und sich für spezielle Programme sofort durch Gegenbeispiele widerlegen lassen.

Unter Vernachlässigung dieses Punktes ist es gelungen, eine Reihe von mehr oder weniger einleuchtenden Strategien herzuleiten bzw. nachträglich zu motivieren. Beispielsweise wird häufig (etwa in Gelenbe [3]) angenommen, daß ein Programm in Datenblöcke I_1, I_2,... eingeteilt werden kann, auf die mit - während des Rechenlaufs konstanten - Wahrscheinlichkeiten $ß_1$, $ß_2$,... unabhängige Zugriffe ausgeführt werden. Der Existenz von aktiven und inaktiven Bereichen wird dadurch Rechnung getragen, daß die $ß_i$ verschieden sein können, doch bleibt durch die vorausgesetzte Konstanz dieser Werte jede zeitliche Änderung unberücksichtigt.

Unter diesen Voraussetzungen ist die Optimalität der Austauschstrategie A_o (alle Blöcke bis auf einen werden fest mit den Programmblöcken besetzt, denen die höchsten Wahrscheinlichkeiten $ß_i$ zugeordnet sind; alle Austauschprozesse beziehen sich auf einen einzigen Block des Pufferspeichers) intuitiv sofort einleuchtend, da das Verdrängen eines der m-1 fest besetzten PSP-Blöcke zu einer Erhöhung der mittleren Fehlrate im nächsten Schritt führt; vorausgesetzt ist hierbei o.B.d.A. $ß_1 \geq ß_2 \ldots \geq ß_{m-1} > ß_m \geq ß_{m+1} \geq \ldots$.

Auch das Working-Set-Model von Denning [2] und andere Arbeiten (z. B. Belady [4] , Aho und Ullman [5] , Kilburn [6]) sowie Strategien wie FIFO (first in, first out: Austausch des ältesten Blocks) und LRU (least recently used: Austausch des zeitlich am längsten nicht benutzen Pufferspeicherblocks) versuchen die unterschiedliche Aktivität von einzelnen Programmteilen zu berücksichtigen. Die Bedeutung dieser Strategien liegt vor allem in ihrer einfachen Realisierbarkeit, wodurch sie Überlegenheit über viele spezielle Austauschverfahren gewinnen, die zu einer geringeren Fehlrate führen, aber komplizierter und daher wesentlich schwerer zu implementieren sind.

Die Nichtberücksichtigung von Programmstrukturen und zeitlichen Änderungen der Zugriffswahrscheinlichkeiten zu den Programmadressen und -blöcken ist jedoch sicher eine zu starke Vereinfachung, wie zum Beispiel das Ergebnis von Gelenbe zeigt, wonach die Fehlraten für FIFO und RAND (random-Nachladen) unter den oben angegebenen Voraussetzungen gleich sind.

In der vorliegenden Arbeit versuchen wir, das Programmverhalten mehr als in den Vorgängerarbeiten zu berücksichtigen. Auf dieser Basis wird zunächst eine Formel für die Größe der Fehlrate hergeleitet. Durch Minimisieren dieses Wertes ergeben sich optimale Austauschkriterien. Dies geht natürlich nicht ohne einschränkende Voraussetzungen. Die wichtigste davon ist, daß die Verteilung der Größe der Datensprünge (Differenz der Adresse des aktuellen Datums und der Adresse des zugehörigen Vorgängerdatums, vgl. Abschnitt 2) sowohl für den Programm- als auch für den Rechenspeicher bestimmte charakteristische Eigenschaften hat, die im Laufe der Rechnung erhalten bleiben. Weiter müssen wir fordern, daß aufeinanderfolgende Datensprünge voneinander unabhängig sind und einer Art von Markow-Eigenschaft genügen. Wir legen also andere Voraussetzungen zugrunde als Denning, Gelenbe

und andere. Entsprechend unterschiedlich sind auch die resultierenden optimalen Strategien.

Im Gegensatz etwa zu der Strategie A_0 wird in dieser Arbeit gezeigt, daß unter relativ schwachen Voraussetzungen über die Gestalt der Datensprungverteilungen der folgende Austauschprozeß zu einer minimalen Fehlrate führt, also in unserem Sinne optimal ist:

> Ist ein Datum nicht im Pufferspeicher zu finden, so sind alle Blöcke aus der Nachbarschaft dieses Datums im Kernspeicher in den Pufferspeicher zu laden, sofern sie sich nicht bereits im Pufferspeicher befinden.

Abschnitt 3 der vorliegenden Arbeit enthält die exakte mathematische Formulierung dieser Nachladestrategie.

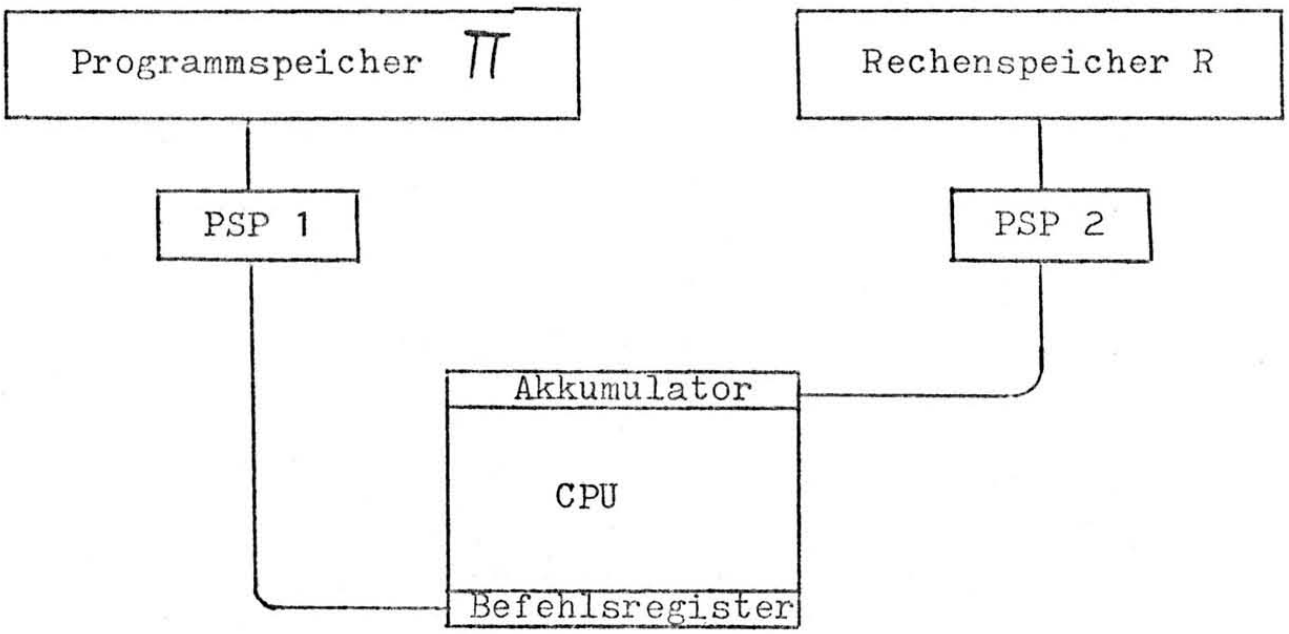
Ein wesentlicher Unterschied zwischen dieser Strategie und A_0 liegt bereits darin, daß der Austauschprozeß sich auf mehrere Blöcke gleichzeitig (im Extremfall auf den gesamten Pufferspeicher) erstrecken kann.

Intuitiv läßt sich unsere Optimalstrategie dadurch motivieren, daß durch das starke Überwiegen kleinerer Datensprünge mit hoher Wahrscheinlichkeit ein größerer Teil des Pufferspeichers inaktiv geworden ist, wenn ein Datum nicht im PSP zu finden ist.

2. Die Verteilung der Datensprünge in Programm- und Rechenspeicher

2.1. Grundlagen

Wie bereits in der Einleitung erwähnt, beschränken wir uns in der vorliegenden Arbeit auf eine PSP-Stufe zwischen Kernspeicher und Rechenwerk. Programm- und Rechenspeicher werden wegen der völlig unterschiedlichen Struktur von Befehls- und Rechendatenfolgen voneinander getrennt.



Viele Untersuchungen haben für beide PSP-Typen Gültigkeit. Wir werden deshalb allgemein voraussetzen, daß jeder der beiden Pufferspeicher aus n Blöcken der Größe k besteht und daß ein Programm jeweils $a > n$ Blöcke (von Π bzw. R) für Befehle bzw. Rechendaten benötigt. In speziellen Fällen wird der Speichertyp zur Kennzeichnung als Index verwandt.

Als erstes definieren wir den Begriff der Fehlrate:

Definition 1:

Sei $x_{-r},\ldots,x_{-1},x,\ldots$ die Folge der Adressen von Befehlen bzw. von Rechendaten, die ein Programm nacheinander benötigt. $B_{-r},\ldots,B_{-1},B,\ldots$ seien die zugehörigen Kernspeicherblöcke.

x bezeichne das Datum, auf das als nächstes zugegriffen werden soll (d.h. Zugriffe auf $x_{-r},\ldots,x_{-1}$ bereits ausgeführt). x heißt aktuelles Datum.

X $B_{-r},\ldots,B_{-1}$ sei der Inhalt des PSP beim Aufruf von x; wir setzen hier wie im folgenden voraus, daß der Pufferspeicher mit Daten des gerade laufenden Programms gefüllt ist.

$z = z(x) \in X$ heißt zugehöriges Vorgängerdatum von x, wenn gilt: $|z - x| \leq |u - x|$ für alle $u \in X$.
Gibt es mehrere Elemente mit dieser Eigenschaft, wird eines davon durch einen Zufallszahlengenerator ausgewählt.

$P := P(x \notin PSP \mid z \in PSP)$ heißt Fehlrate.
$Q := 1 - P$ bezeichnen wir als Vorhandenrate.

Bemerkung:

a. Wir nehmen generell an, daß der Speicherbereich für Befehle bzw. Daten eines Programms zusammenhängend ist.

b. Beim PSP des Programmspeichers setzen wir immer $z = x_{-1}$, wir verzichten hier also auf eine eventuell mögliche andere Wahl mittels eines Zufallszahlengenerators.

c. Die Differenzen x - z der Adressen von aktuellen Daten und zugehörigen Vorgängerdaten bezeichnen wir als Datensprünge.

Wir beschäftigen uns nun mit Modellen für die Verteilung der Datensprünge in Programm- bzw. Rechenspeicher und mit hieraus resultierenden Strategien, die zu möglichst großen Werten für die Vorhandenrate Q führen.

Definition 2:

y^{π} bzw. y^{R} bezeichne die Verteilung der Datensprünge von Programm- bzw. Rechenspeicher.
$y_i := P(x - z = i) \qquad i \in \mathbb{Z}$

In Abschnitt 2.2. wird die Gestalt von y^{π} bzw. y^{R} näher diskutiert. Wir kommen nun zu den für die folgenden Untersuchungen entscheidenden Voraussetzungen:

Voraussetzungen über die Gestalt der Datensprungverteilungen:

1. Die Datensprünge werden als unabhängige Zufallsvariable interpretiert, die aus einer diskreten Verteilung y stammen, welche die in Abschnitt 2.2. angegebenen charakteristischen Eigenschaften während des Programmlaufs beibehält; aus diesem Grund verzichten wir bei den Verteilungen y^{π} und y^{R} auf einen Zeitindex.

2. $Q := Q(k,z,x,y)$

   Dies ist eine Art Markow-Eigenschaft, die strenggenommen nicht erfüllt ist.

2.2. Diskussion der Verteilungen y^{π} und y^{R}

Wir versuchen im folgenden einige Charakteristika der Verteilung der Datensprünge in Programm- bzw. Rechenspeicher anzugeben, die erfahrungsgemäß bei der Mehrzahl der Programme in ähnlicher Form zu beobachten sind.

2.2.1. Rechenspeicherverteilung y^{R}

Es darf bei vielen Programmen angenommen werden, daß Vorwärtssprünge im Mittel etwa ebenso wahrscheinlich sind wie gleichgroße Rücksprünge. Für einfache Variablen ist diese Forderung weitgehend erfüllt; bei anderen Größen, z. B. bei Feldelementen, hängt die Gültigkeit dieser Annahme auch davon ab, ob es sich um ein übersetztes oder um ein handgeschneidertes Programm handelt. Compilierte Programme bearbeiten Schleifen öfter von unten nach oben (zahlreiche kleinere Vorwärtssprünge, anschließend großer Rücksprung), bei Maschinenprogrammen ist es oft umgekehrt wegen der in vielen Fällen einfacheren Endabfrage (durch das Programmieren einer Schleife in der Form, daß die Abfrage auf Null erfolgt, spart man eine Reihe von Maschinenbefehlen pro Zyklus; bei in problemorientierten Sprachen geschriebenen Programmen werden Schleifen meist nicht in dieser Weise programmiert, da man hier nur einen unwesentlichen Zeitgewinn erzielt, der zudem

noch in den meisten Fällen auf Kosten der Übersichtlichkeit des Programms geht).

Mit Abstand am häufigsten sind Sprünge der Längen $\pm$ 1 und 0; erstere entstehen zum Beispiel als Folge der linearen Fortschaltung in Aufwärts- bzw. Abwärtsschleifen. Letztere treten dann auf, wenn eine Adresse zweimal benötigt wird und der diese Adresse enthaltende Block des Pufferspeichers nicht zwischen dem ersten und dem zweiten Aufruf des Datums aufgrund der Kriterien der Nachladestrategie ausgetauscht wurde.

Weiter darf angenommen werden, daß die Wahrscheinlichkeit für einen Sprung, der über eine - programmabhängige - Größe hinausgeht, mit wachsender Sprunggröße abnimmt. Große Sprünge werden hauptsächlich durch die Schleifenstruktur des Programms und die Indexänderungen der darin vorkommenden Felder verursacht, und wir nehmen an, daß Schleifen mit wachsender Größe immer seltener werden abgesehen von den Unregelmäßigkeiten, die dadurch verursacht werden, daß als Laufgrenzen von Schleifen "runde" Zahlen bevorzugt werden.

Für sehr kleine Werte der Sprunggröße gilt diese Argumentation nicht mehr, da so kleine Schleifen selten vorkommen, doch wird diese Lücke durch die erfahrungsgemäß sehr häufige Verwendung von einfachen Variablen, die etwa als real- oder integer-Block zusammenhängend abgespeichert sind, weitgehend ausgeglichen.

Steht das zugehörige Vorgängerdatum z in der Nähe des Randes des für die Daten des Programms reservierten Speicherbereichs, so ist die Symmetriebeziehung ($y_i \approx y_{-i}$) mehr oder weniger stark verletzt. Im Extremfall (z steht ganz am Rand) sind nur Vorwärts- bzw. nur Rücksprünge möglich. Erhalten bleibt aber die Eigenschaft $y_i^R \geq y_{i+1}^R$ sowie $y_{-i}^R \geq y_{-i-1}^R$ für $i \geq 1$.

Bei vielen Untersuchungen ist nur die Verteilung des Sprungbetrags h von Bedeutung. ($h_i := P(\,|x - z| = i\,)$. Die Beziehung $h_i^R \geq h_{i+1}^R$ für $i \geq 1$ ist für den gesamten Bereich, den ein Programm im Rechenspeicher belegt, gut gesichert.

Wir setzen deshalb im folgenden voraus, daß die Werte h_i^R für $i \geq 1$ monoton fallen und daß die Verteilung y^R im wesentlichen symmetrisch ist, sofern das Vorgängerdatum z des aktuellen Datums x nicht zu nahe am Rand des für die Rechendaten des Programms reservierten Speicherbereichs steht.

2.2.2. Programmspeicherverteilung y^{π}

Unabhängig davon, ob es sich um ein übersetztes oder ein handgeschneidertes Maschinenprogramm handelt, sind hier die Sprünge der Länge +1 mit weitem Abstand dominierend.

Größere Vorwärtssprünge sind sehr selten; sie kommen praktisch nur zu Beginn und am Ende von bedingten Anweisungen in der zweiseitigen Form vor. Die Länge dieser Sprünge hängt vom Umfang der bedingten Anweisung ab; kürzere Sprünge sollten erfahrungsgemäß etwas häufiger sein als längere.

Rückwärtssprünge sind nur nach Beendigung eines Schleifendurchlaufs oder als Folge eines goto-Befehls zu erwarten; Rücksprünge unterhalb einer programmabhängigen Größe sind praktisch ausgeschlossen, da derart kleine Schleifen im allgemeinen sinnlos sind.

Sprünge der Länge Null sind selten; sie sind nur dann möglich, wenn ein Befehl des Programms mehrmals benötigt wird und der alte Block noch im PSP des Programmspeichers steht.

Wir setzen für die Untersuchungen des nächsten Abschnitts der Arbeit voraus, daß die oben skizzierten charakteristischen Eigenschaften der Verteilung y^{π} (Dominanz der Sprünge der Länge +1, alle anderen Sprünge demgegenüber praktisch vernachlässigbar) während des gesamten Programmlaufs in gleicher Form zu beobachten sind.

Bemerkung:
In einem einzigen Fall (das Vorgängerdatum z steht am rechten Rand des für die Befehle des Programms reservierten Speicherbereich) ergibt sich ein abweichendes Bild (nur Rücksprünge möglich). Wir lassen diesen Fall außer acht, verzichten also auf eine eventuell bessere Nachladestrategie, da er außerordentlich selten vorkommt.

3. Bestimmung von optimalen Nachladestrategien

Wir berechnen zunächst die Vorhandenrate Q, d.h. die Wahrscheinlichkeit dafür, daß sich das aktuelle Datum im Pufferspeicher befindet; wir können davon ausgehen, daß das zugehörige Vorgängerdatum noch im Pufferspeicher steht (vgl. Definition 1). Durch Maximieren des Ausdrucks für $Q = Q(k,z,x,y)$ erhalten wir optimale Nachladestrategien.

Die Urbilder von x und z im Kernspeicher seien $f^{-1}(x)$ bzw. $f^{-1}(z)$. Bezeichnen wir die Blöcke des Kernspeichers der Reihe nach mit $B_1,\ldots,B_a$ sowie den aktuellen Inhalt des Pufferspeichers mit $C = \{C_1,\ldots,C_n\}$ (es ist $C_i \in \{B_1,\ldots,B_a\}$ für alle i, da wir den Einschwingvorgang als beendet vorausgesetzt haben), so gilt:

$$Q = P(x \in C \mid z \in C)$$

$$= \sum_{i=1}^{a} P(f^{-1}(x) \in B_i \wedge B_i \in C \mid z \in C)$$

$$= \sum_{i=1}^{a} \sum_{m=1}^{a} P(f^{-1}(x) \in B_i \wedge B_i \in C \wedge f^{-1}(z) \in B_m \mid z \in C)$$

$$= \sum_{i=1}^{a} \sum_{m=1}^{a} P(B_i \in C \mid f^{-1}(x) \in B_i \wedge f^{-1}(z) \in B_m \wedge z \in C) \cdot$$
$$P(f^{-1}(x) \in B_i \mid f^{-1}(z) \in B_m \wedge z \in C) \cdot$$
$$P(f^{-1}(z) \in B_m \mid z \in C)$$

Zur Abkürzung setzen wir:

$$w_{m,i} := P(B_i \in C \mid f^{-1}(x) \in B_i \wedge f^{-1}(z) \in B_m \wedge z \in C)$$

$$Q_{i,j} := P(f^{-1}(x) \in B_{j+i} \mid f^{-1}(z) \in B_j \wedge z \in C)$$

$$r_m := P(f^{-1}(z) \in B_m \mid z \in C)$$

Die Größen $Q_{i,j}$ geben die Wahrscheinlichkeit dafür an, daß das aktuelle Datum im Kernspeicher i Blöcke vom zuge-

hörigen Vorgängerdatum (dieses steht in Block B_j) entfernt ist.

Unter etwas stärkeren Voraussetzungen über die Verteilung y lassen sich die Größen $Q_{i,j}$ in geschlossener Form angeben, doch ist eine solche Darstellung für die Herleitung einer optimalen Nachladestrategie nicht erforderlich. Insbesondere spielt die Blockgröße k hier keine wesentliche Rolle mehr; sie beeinflußt zwar die Zahlenwerte der $Q_{i,j}$, nicht aber die Größenordnungsrelationen, die zwischen den einzelnen $Q_{i,j}$ bestehen; die Größe von k hat daher höchstens geringen Einfluß auf die Wahl der Nachladestrategie.

Die Werte r_m geben den Belegungsgrad der einzelnen Kernspeicherblöcke an; wir setzen $r_m > 0$ für $m = 1,\ldots,a$ voraus (das Programm soll also keine überflüssigen Blöcke reservieren), doch sind die von uns angegebenen Strategien auch dann optimal, wenn einer oder mehrere der Werte r_m Null sind. Da wir verlangt haben, daß die Datensprungverteilungen ihren Typ während des Programmlaufs nicht ändern, sind Werte $r_m = 0$ höchstens an den Rändern des für das Programm reservierten Kernspeicherbereichs zu erwarten. Am einfachsten ist es, die r_m als gleichverteilt ($r_m = 1/a$ für alle m) anzunehmen; diese Bedingung ist für eine zwar stark eingeschränkte, aber jedenfalls nichtleere Klasse von Programmen erfüllt.

Mit den oben eingeführten Bezeichnungen erhalten wir für Q die Beziehung:

$$Q = \sum_{i=1}^{a} \sum_{m=1}^{a} w_{m,i} \cdot Q_{i-m,m} \cdot r_m$$

Wir werden zeigen, daß die optimale Strategie unabhängig von den Werten r_m ist ($r_m > 0$ vorausgesetzt). Die Bestimmung einer optimalen Nachladestrategie läuft im wesentlichen auf die Festlegung der Elemente der Matrix $W = (w_{m,i})$ hinaus. Diese Wahrscheinlichkeiten charakterisieren den Nachladeprozeß entscheidend.

Lemma 1:

Vor: $C_i \in \{B_1,\ldots,B_a\}$ für $i = 1,\ldots,n$
(d.h. Einschwingvorgang beendet)

Beh: $\sum_{i=1}^{a} w_{r,i} = n$ für $r = 1,\ldots,a$ (Summenbedingung)

Bew:

$$\sum_{i=1}^{a} w_{r,i}$$

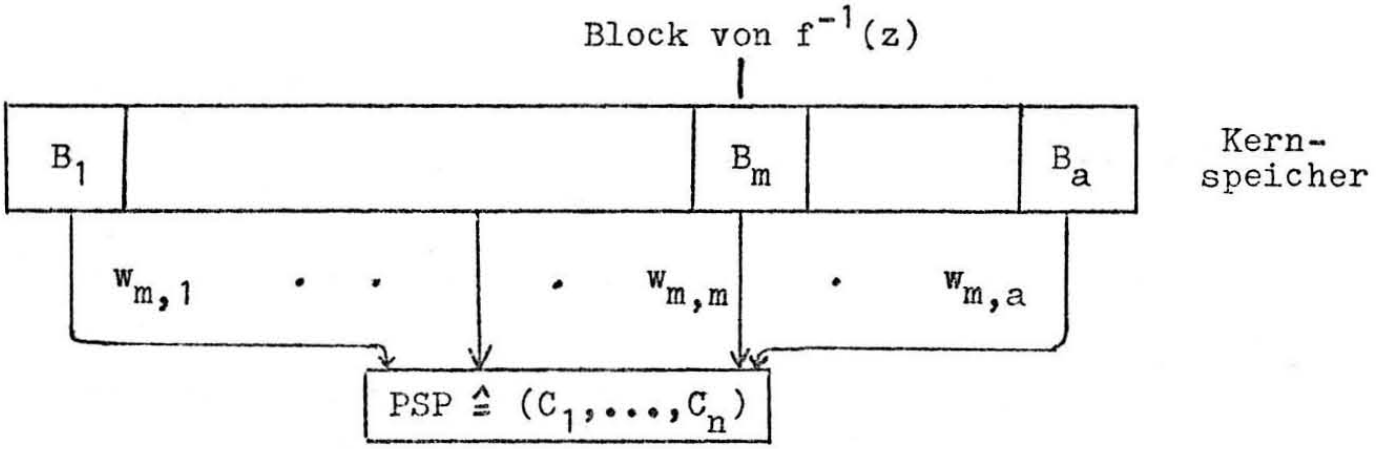
$$= \sum_{i=1}^{a} P(B_i \in C \mid f^{-1}(x) \in B_i \wedge f^{-1}(z) \in B_r \wedge z \in C)$$

$$= \sum_{i=1}^{a} \sum_{j=1}^{n} P(B_i = C_j \mid f^{-1}(x) \in B_i \wedge f^{-1}(z) \in B_r \wedge z \in C)$$

$$= \sum_{j=1}^{n} \sum_{i=1}^{a} P(C_j = B_i \mid f^{-1}(x) \in B_i \wedge f^{-1}(z) \in B_r \wedge z \in C)$$

$$= \sum_{j=1}^{n} 1 = n$$

Gesucht sind nun Strategien, die unter Berücksichtigung der Formel für Q, der Summenbedingung und der zusätzlichen Restriktionen $0 \leq w_{m,i}$, $r_m \leq 1$ für alle $i,m = 1,,..,a$ zu einer maximalen Vorhandenrate führen.



Aus der Formel für die Vorhandenrate Q wird ersichtlich, daß die Teilsummen

$$\left(\sum_{i=1}^{a} w_{l,i} \cdot Q_{i-l,l}\right) \cdot r_l \qquad \text{für } l = 1,\ldots,a$$

unabhängig voneinander maximiert werden können. Hieraus ergibt sich, daß die Größe der Koeffizienten $r_m \neq 0$ keinen Einfluß auf die optimale Strategie hat.

Die Wahl der Koeffizienten $w_{m,i}$ hängt von der Größe der Zahlen $Q_{i-m,m}$ in der Weise ab, daß größeren $Q_{i-m,m}$ auch größere Werte $w_{m,i}$ zuzuordnen sind (bei optimaler Strategie). Unter Berücksichtigung der Summenbedingung und der Restriktionen $0 \leq w_{m,i} \leq 1$ ergibt sich damit das folgende Ergebnis:

Satz 1:

a. Die Zeilen der Matrix $W = (w_{m,i})$ können unabhängig voneinander optimiert werden.

   Ist für Zeile m' die Belegungsrate $r_{m'} = 0$, so können die Werte $w_{m',1},\ldots,w_{m',a}$ beliebig gewählt werden, ohne daß die Vorhandenrate Q sich ändert.

b. Ist $(Q_{1',m},w_{m',k_1})$,....., $(Q_{a',m},w_{m',k_a})$ eine Umordnung der Folge $(Q_{1-m,m},w_{m,1})$,....., $(Q_{a-m,m},w_{m,a})$ mit der Eigenschaft $Q_{i',m} \geq Q_{i+1',m}$ für $i = 1,\ldots,a-1$, dann erhält man eine maximale Vorhandenrate $Q_{max} = Q(w_{m',i}, Q_{i',m}, r_m)$ durch die folgende Wahl der Elemente der Matrix W:

$$w_{m',k_i} = \begin{cases} 1 & \text{für } i = 1,\ldots,n \\ 0 & \text{" } i = n+1,\ldots,a \end{cases}$$

   Ist $Q_{n',m} > Q_{n+1',m}$ für $m = 1,\ldots,a$, dann ist die Matrix W bei optimaler Strategie eindeutig bestimmt.

   Im Falle $Q_{n',m} = Q_{n+1',m}$ für mindestens ein m gibt es mehrere Strategien, die zur gleichen optimalen Vorhandenrate Q führen.

Unter etwas stärkeren Voraussetzungen (deren Realität im Anschluß an Satz 3 diskutiert wird) ergibt sich die folgende Aussage:

Satz 2:

Vor: $Q_{i,r} \geqq Q_{j,r}$ und $Q_{-i,r} \geqq Q_{-j,r}$ für $0 \leqq i < j \leqq a$

$r = 1,\ldots,a$

Sei weiter $Q = Q(w_{m,i}, Q_{i-m,m}, r_m)$

$Q' = Q(w'_{m,i}, Q_{i-m,m}, r_m)$ mit:

$$w'_{m,i} = \begin{cases} w_{m,i} & \text{falls } m \neq m_0 \\ w_{m_0,i} & \text{" } m = m_0 \text{ , } i \neq b,c \\ w_{m_0,b} + d \\ w_{m_0,c} - d \end{cases} \Bigg\} \begin{array}{l} \text{falls } m_0 \leqq b < c \\ \text{oder } m_0 \geqq b > c \end{array} \quad \text{und } d > 0$$

Beh: $Q' \geqq Q$

Beweis:

Der Beweis dieser Aussage ist eigentlich bereits in Satz 1 enthalten. Trotzdem führen wir ihn hier durch:

$$Q' = \sum_{\substack{m=1 \\ m \neq m_0}}^{a} \sum_{i=1}^{a} w_{m,i} Q_{i-m,m} r_m + \sum_{\substack{i=1 \\ i \neq b,c}}^{a} w_{m_0,i} Q_{i-m_0,m_0} r_{m_0}$$

$$+ r_{m_0} \cdot (Q_{b-m_0,m_0} \cdot (w_{m_0,b} + d) + Q_{c-m_0,m_0} \cdot (w_{m_0,c} - d))$$

$$= Q + r_{m_0} \cdot (Q_{b-m_0,m_0} - Q_{c-m_0,m_0}) \cdot d$$

Wegen $d > 0$ ergibt sich damit:

$$Q' \geqq Q \iff Q_{b-m_0,m_0} \geqq Q_{c-m_0,m_0} \vee r_{m_0} = 0$$

Diese Bedingung ist aufgrund der Voraussetzungen gerade erfüllt.

Unter denselben Voraussetzungen wie in Satz 2 ist die Koeffizientenmatrix W bereits in weiten Grenzen festgelegt, wie der folgende Satz zeigt:

Satz 3:

Vor: $Q_{i,r} \geq Q_{j,r}$ und $Q_{-i,r} \geq Q_{-j,r}$ für $0 \leq i < j \leq a$

$r = 1,\ldots,a$

Beh:

Für die optimale Strategie $Q_{max} = Q(w_{m,i}, Q_{i-m,m}, r_m)$ gilt:

a. $w_{m,m} = 1$ für $m = 1,\ldots,a$

b. $w_{i,j} = w_{i,k} = 1 \quad (1 \leq j < k \leq a)$

$\Rightarrow \quad w_{i,l} = 1$ für $l = j+1,\ldots,k-1$

c. $w_{r,m} = 0 \Rightarrow$ $\begin{cases} w_{r,l} = 0 \text{ für } l = m+1,\ldots,a \text{ falls } r < m \\ w_{r,l} = 0 \text{ " } l = 1,\ldots,m-1 \text{ " } r > m \end{cases}$

d. Höchstens zwei Elemente der Matrix W sind pro Zeile von Null bzw. Eins verschieden, und zwar stehen diese, wenn vorhanden, zwischen den beiden Nullblöcken (von denen einer leer sein kann) und dem Einsblock.

e. Die Länge L_E des Einsblocks ist entweder n-1 oder n.

$L_E = n-1 \Leftrightarrow$ es gibt genau zwei von Null und Eins verschiedene Elemente in der betrachteten Zeile von W, die sich zu 1 addieren.

$L_E = n \Leftrightarrow$ Alle Elemente der betrachteten Matrixzeile sind entweder Eins oder Null.

f. Außerhalb eines Streifens der Breite 2n+1 um die Hauptdiagonale stehen in jedem Fall nur Nullen.

g. $w_{i,j} = 1$ für die Paare (i,j) mit:

$1 \leq i \leq j \leq n$ oder $a-n+1 \leq j \leq i \leq a$

Beweis:

g. ergibt sich aus c. und der Summenbedingung. Die übrigen Aussagen folgen sofort aus Satz 1 und Satz 2.

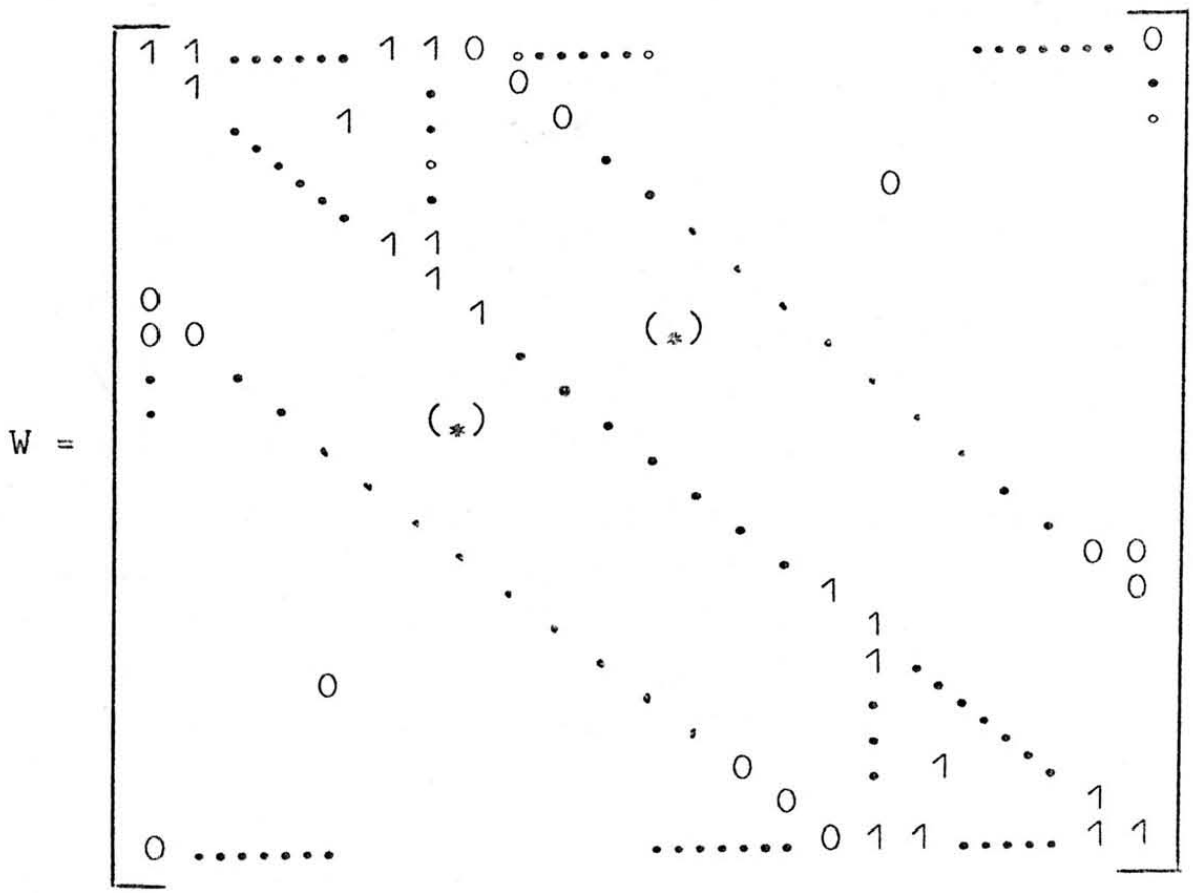
Bemerkung:

a. Es kann in Ausnahmefällen ($r_m = 0$ für mindestens ein m oder bei Gleichheit von mehreren $Q_{i,r}$) mehrere Matrizen

W geben, für die man dieselbe optimale Vorhandenrate Q_{max} wie für die Matrix W von Satz 3 erhält. Wir betrachten diese jedoch nicht, da die hieraus resultierenden Strategien wesentlich schwerer zu realisieren sind.

b. Es gibt in jedem Fall eine optimale Strategie, bei der alle Elemente von W Null oder Eins sind. Ist nämlich $L_E = n-1$ für eine Zeile von W, so kann man die beiden von Null und Eins verschiedenen Elemente a und b dieser Zeile (a+b = 1 nach Satz 3.e.) beliebig wählen (insbesondere also als Null bzw. Eins), ohne Q_{max} zu ändern.

c. Die Matrix W hat folgende schematische Gestalt:



Die Elemente $w_{m,i}$ in den Bereichen (*) lassen sich mit den bisher getroffenen Voraussetzungen noch nicht eindeutig festlegen, da ihre Wahl von der Größe der Werte $Q_{i,r}$ im Vergleich zu den Werten $Q_{-j,r}$ ($i,j \geq 0$) abhängt.

Bevor wir Voraussetzungen angeben, durch die alle Elemente von W festgelegt werden, diskutieren wir die Voraussetzungen der Sätze 2 und 3.

Diskussion der Voraussetzungen der Sätze 2 und 3:

Wir zeigen, daß eine geringfügige Verschärfung der von uns angenommenen Eigenschaften der Datensprungverteilungen y^{π} und y^{R} ausreicht, um die Bedingungen $Q_{i,s} \geq Q_{j,s}$ und $Q_{-i,s} \geq Q_{-j,s}$ für $0 \leq i < j$ und $s = 1,\ldots,a$ zu verifizieren.

Wir formen zunächst den Ausdruck für $Q_{i,s}$ etwas um:

Da bei unserer Strategie stets das aktuelle Datum in den Pufferspeicher geladen wird, wenn es nicht schon dort steht, ist $P(z \in C) = 1$. Damit wird:

$$Q_{i,s} = P(f^{-1}(x) \in B_{i+s} \mid f^{-1}(z) \in B_s)$$

$$= \sum_{r=1}^{k} P(f^{-1}(x) \in B_{i+s} \wedge f^{-1}(z) = b_r \in B_s) \;/\; P(f^{-1}(z) \in B_s)$$

$$= \sum_{r=1}^{k} P(f^{-1}(x) \in B_{i+s} \mid f^{-1}(z) = b_r \in B_s) \cdot p_r$$

$$= \sum_{r=1}^{k} P(\; ik-r+1 \leq y \leq (i+1)k-r \;) \cdot p_r$$

mit $p_i = P(f^{-1}(z) = b_i \in B_s) \;/\; P(f^{-1}(z) \in B_s)$

Es ist also $0 \leq p_i \leq 1$ und $p_1 + \ldots + p_k = 1$.

Vorausgesetzt wurde ferner $P(f^{-1}(z) \in B_s) \neq 0$ (sonst trivial); weiter haben wir der Übersichtlichkeit halber auf den an sich notwendigen Index s bei den Werten p_i verzichtet.

$$Q_{i,s} = \sum_{r=1}^{k} \sum_{j=ik-r+1}^{(i+1)k-r} y_j \cdot p_r$$

$$= y_{ik} + \sum_{j=1}^{k-1} \left[y_{ik-k+j} \cdot \sum_{r=k-j+1}^{k} p_r + y_{ik+k-j} \cdot \sum_{r=1}^{j} p_r \right]$$

Unter der Voraussetzungen, daß die Blöcke des Programms "symmetrisch" belegt sind und daß größere Sprünge nicht wahrscheinlicher sind als kleinere, kann man nun zeigen, daß die Voraussetzungen der Sätze 2 und 3 erfüllt sind.

Lemma 2:

a. Vor: $p_r = p_{k-r+1}$ für $r = 1,\ldots,k$

$y_i \geq y_{i+1}$ und $y_{-i} \geq y_{-i-1}$ für $i \geq 1$

Beh: $Q_{i,s} \geq Q_{i+1,s}$ und $Q_{-i,s} \geq Q_{-i-1,s}$ für $i > 0$

b. Vor: wie in a.

zusätzlich: $y_0 \geq y_k$ sowie

$y_{-j} \geq y_{k+j}$ für $j = 1,..,k-1$ $(s \neq 1)$

Beh: $Q_{0,s} \geq Q_{1,s}$ für $s = 2,\ldots,a$

c. Vor: wie in a.

zusätzlich: $y_0 \geq y_{-k}$ sowie

$y_j \geq y_{-k-j}$ für $j = 1,..,k-1$ $(s \neq a)$

Beh: $Q_{0,s} \geq Q_{-1,s}$ für $s = 1,\ldots,a-1$

d. Vor: wie in a.

zusätzlich : $y_0 \geq \sum_{i=k}^{k+\lfloor (k-1)/2 \rfloor} y_i$

Beh: $Q_{0,1} \geq Q_{1,1}$

e. Vor: wie in a.

zusätzlich : $y_0 \geq \sum_{i=k}^{k+\lfloor (k-1)/2 \rfloor} y_{-i}$

Beh: $Q_{0,a} \geq Q_{-1,a}$

Beweis:

Nach Voraussetzung gilt: $\sum_{r=k-j+1}^{k} p_r = \sum_{r=1}^{j} p_r \overset{Def}{=} T_j$

Damit wird:

$$Q_{i,s} - Q_{i+1,s}$$

$$= \sum_{j=1}^{k-1} (y_{(i-1)k+j} + y_{(i+1)k-j} - y_{ik+j} - y_{(i+2)k-j}) \cdot T_j + y_{ik} - y_{(i+1)k}$$

Teil a. des Hilfssatzes ist damit aufgrund der Voraussetzungen bewiesen.

$$Q_{0,s} - Q_{1,s}$$

$$= y_0 - y_k + \sum_{j=1}^{k-1} (y_{k-j} + y_{k-j} - y_j - y_{2k-j}) \cdot T_j$$

$$= \sum_{j=1}^{\lfloor (k-1)/2 \rfloor} (y_j - y_{k-j}) \cdot \sum_{i=j+1}^{k-j} p_i + \sum_{j=1}^{k-1} (y_{-j} - y_{k-j}) \cdot \sum_{i=j+1}^{k} p_i$$

$$+ y_0 - y_k$$

Sind die Voraussetzungen von b. erfüllt (der Fall s = 1 muß gesondert behandelt werden), so sind alle 3 Hauptterme dieser Differenz nichtnegativ. Dies beweist b. .

Für die Aussage c. läuft der Beweis analog.

Bei d. ist die für b. geforderte Voraussetzung nicht zu erfüllen, da mindestens einer der Werte y_{-j} den Wert Null hat. Entsprechendes ist bei e. der Fall. Da der erste der 3 Hauptterme in der obigen Summe auf jeden Fall nichtnegativ ist, ist die Beziehung $Q_{0,1} \geq Q_{1,1}$ aber erfüllt, wenn gilt:

$$y_0 - y_k \geq \sum_{j=1}^{k-1} y_{k-j} \cdot \sum_{i=j+1}^{k} p_i \ .$$

Diese Ungleichung ist insbesondere im Falle

$$y_0 = \sum_{i=k}^{k+\lfloor (k-1)/2 \rfloor} y_i \quad \text{erfüllt.}$$

Bei e. ist die Argumentation entsprechend.

Bemerkung:

a. Die in Lemma 2 getroffenen Voraussetzungen sind in den meisten Fällen sowohl für die Rechenspeicherverteilung y^R als auch für die Verteilung y^π der Datensprünge im Programmspeicher erfüllt. Bei y^π ist allerdings die für d. und e. zusätzlich vorausgesetzte Beziehung

$$y_0 \geq \sum_{i=k}^{k+\lfloor (k-1)/2 \rfloor} y_i \quad \text{bzw.} \quad y_0 \geq \sum_{i=k}^{k+\lfloor (k-1)/2 \rfloor} y_{-i} \quad \text{sehr fraglich.}$$

$Q_{0,1} \geq Q_{1,1}$ und $Q_{0,a} \geq Q_{-1,a}$ gilt trotzdem, da in den entsprechenden Differenzen jeweils der Summand $y_1(T_{k-1} - T_1)$ auftritt, der nach unseren Voraussetzungen über die Gestalt von y^π alle anderen Summanden erschlägt.

b. Da sich die meisten Unregelmäßigkeiten, die dadurch entstehen, daß die Voraussetzung über das monotone Abklingen

der Verteilungen y^{π} bzw. y^{R} nicht für alle y_i exakt erfüllt sind (vgl. dazu auch die Diskussion in Abschnitt 2.2.), durch die Summationen wieder aufheben, kann man einen großen Teil der Voraussetzungen im Spezialfall erheblich abschwächen oder sogar ganz weglassen. Dies gilt nicht zuletzt für die Voraussetzung $p_r = p_{k-r+1}$, die zwar intuitiv einleuchtend ist, aber nicht leicht überprüft werden kann. Man beachte auch, daß die Abschätzungen im Beweis von Lemma 2 bei vielen speziellen Verteilungen y sehr grob sind. In diesen Fällen ist es nicht gravierend, wenn die geforderten Voraussetzungen nicht in allen Einzelheiten erfüllt sind.

Wir untersuchen jetzt, unter welchen zusätzlichen Voraussetzungen an die Struktur der Datensprungverteilungen sich die durch Satz 3 noch nicht festgelegten Elemente der Matrix W in der Weise eindeutig bestimmen lassen, daß sich eine optimale Nachladestrategie ergibt:

Sind die Adressen des (Programm- bzw. Rechen-) Speichers fortlaufend mit $1,\ldots,a\cdot k$ durchnumeriert und ist $z = b\cdot k+r$ $b \in \{0,\ldots,a-1\}$, $0 \leq r \leq k-1$ (d.h. steht z in Block B_{b+1} des Kernspeichers), so ist die Beziehung $y_i \approx y_{-i}$ für die Werte $i \leq \min(a\cdot k-z\ ,\ z-1)$ im Falle der Rechenspeicherverteilung recht gut erfüllt. Auch für den Programmspeicher ist sie mit Einschränkungen vertretbar, abgesehen davon, daß hier $y_1 \gg y_{-1}$ gilt.

Legt man diese zusätzliche Voraussetzung zugrunde, so gilt $Q_{j,b+1} \approx Q_{-j,b+1}$ für $2 \leq j \leq \min(b,a-b)$.

Lemma 3:

Vor: $z = b\cdot k+r$; $y_i = y_{-i}$ für $i \leq \min(a\cdot k-z, z-1)$ wie eben;

$p_r = p_{k-r+1}$ für $r = 1,\ldots,k$ (ebenso wie in Lemma 2).

Beh: $Q_{j,b+1} = Q_{-j,b+1}$ für $1 \leq j \leq \min(b,a-b)$

Beweis:

$$Q_{-i,s} = y_{-ik} + \sum_{j=1}^{k-1}\left[y_{-ik-k+j}\cdot\sum_{r=k-j+1}^{k} p_r + y_{-ik+k-j}\cdot\sum_{r=1}^{j} p_r\right]$$

$$= y_{+ik} + \sum_{j=1}^{k-1} \left[y_{ik+k-j} \cdot \sum_{r=k-j+1}^{k} p_r + y_{ik-k+j} \cdot \sum_{r=1}^{j} p_r \right]$$

$$= y_{+ik} + \sum_{j=1}^{k-1} \left[y_{ik+k-j} \cdot \sum_{r=1}^{j} p_r + y_{ik-k+j} \cdot \sum_{r=k-j+1}^{k} p_r \right]$$

$$= Q_{i,s}$$

Diese Beziehung gilt, solange der Sprung nicht von Block $s = b+1$ auf einen Block erfolgt, der außerhalb des reservierten Speicherbereichs liegt, also für $1 \leq i \leq \min(b, a-b)$.

Die durch Verbindung der Aussagen von Lemma 2 und Lemma 3 zu gewinnende Beziehung

$$Q_{i,s} \geq Q_{j,s} \quad \text{für } 0 \leq |i| < |j| \leq \min (s-1 , a-s+1)$$

ist für die Verteilung y^R der Datensprünge des Rechenspeichers gut erfüllt, da y^R den Voraussetzungen der beiden Hilfssätze weitgehend genügt.

Für y^{π} gilt zwar $Q_{1,s} \neq Q_{-1,s}$ aufgrund der von uns vorausgesetzten Struktur dieser Verteilung, doch gilt auch hier wegen der Dominanz von y_1 im allgemeinen $Q_{1,s} \geq Q_{\pm 2,s}$ sowie $Q_{-1,s} \geq Q_{\pm 2,s}$.

Wir dürfen daher annehmen, daß die obige Beziehung für beide Speichertypen (eventuell mit relativ geringfügigen "Schmutzecken") erfüllt ist.

Unter dieser Voraussetzung sind wir nun in der Lage, alle Elemente der Matrix W in der Weise festzulegen, daß sich eine optimale Vorhandenrate $Q = Q_{max}$ ergibt. Darüber hinaus werden wir zeigen, daß man $w_{i,j} \in \{0;1\}$ für alle i und alle j wählen kann; dies ist von großer Bedeutung für die Implementierung der hieraus resultierenden Nachladestrategie.

Satz 4:

Vor: $Q_{i,s} \geq Q_{j,s}$ für $0 \leq |i| < |j| \leq \min(s-1, a-s+1)$
$s \in \{1,\ldots,a\}$

Beh: Die durch folgende Matrix W beschriebene Strategie ist optimal:

a. $n = 2t+1$ für $t \in \mathbb{N}_0$ (n = Blockzahl des PSP) :

$$w_{i,j} = \begin{cases} 1 & \text{für } j = i,\ i \circledast 1,\ \ldots,\ i \circledast t \ \circledast \in \{\oplus, \ominus\} \\ 0 & \text{sonst} \end{cases}$$

Dabei sind die Operationen $\oplus$ und $\ominus$ wie folgt definiert:

$$w_{i,i \oplus m} = \begin{cases} w_{i,i+m} & \text{falls } i+m \leq a \\ w_{i,i+m-n} & \text{" } \quad i+m > a \end{cases}$$

$$w_{i,i \ominus m} = \begin{cases} w_{i,i-m} & \text{falls } i-m \geq 1 \\ w_{i,i-m+n} & \text{" } \quad i-m < 1 \end{cases}$$

b. $n = 2t+2$ für $t \in \mathbb{N}_0$:

$$w_{i,j} = \begin{cases} 1 \text{ falls } j = i, i \circledast 1,\ \ldots,\ i \circledast t \ \circledast \in \{\oplus, \ominus\} \\ 1 \quad \text{" } \quad j = i+t+1 \text{ und } i-t-1 < 1 \\ 1 \quad \text{" } \quad j = i-t-1 \text{ und } i+t+1 > a \\ u_{i+t+1} \text{ falls } j = i+t+1 \text{ und } 1 \leq i-t-1 < i+t+1 \leq a \\ v_{i-t-1} \quad \text{" } \quad j = i-t-1 \text{ und } 1 \leq i-t-1 < i+t+1 \leq a \\ 0 \quad \text{sonst} \end{cases}$$

Die Werte u_{i+t+1} und v_{i-t-1} sind nicht frei wählbar, sondern an folgende Bedingungen gebunden:

$u_{i+t+1} = 0 \quad v_{i-t-1} = 1 \quad$ falls $\quad Q_{-t-1,i} > Q_{t+1,i}$

$u_{i+t+1} = 1 \quad v_{i-t-1} = 0 \quad$ " $\quad Q_{-t-1,i} < Q_{t+1,i}$

$u_{i+t+1} + v_{i-t-1} = 1 \quad$ " $\quad Q_{-t-1,i} = Q_{t+1,i}$

Es gibt also eine optimale Nachladestrategie mit der Eigenschaft $w_{i,j} \in \{0;1\}$ für $i,j = 1,\ldots,a$.

Der Beweis dieses Satzes folgt unmittelbar aus den Aussagen der Sätze 1 - 3.

Beispiele:

a. a = 16 , n = 7

$$W = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & & \\
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & \\
 & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & 0 & \\
 & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & \\
 & & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & \\
 & & & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \\
 & & & & & & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 & & & & & & & 1 & 1 & 1 & 1 & 1 & 1 \\
 & & & 0 & & & & & 1 & 1 & 1 & 1 & 1 \\
 & & & & & & & & 1 & 1 & 1 & 1 & 1 \\
 & & & & & & & & 1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

b. a = 16 , n = 8

$$W = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & & & & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & & & 0 & & \\
* & 1 & 1 & 1 & 1 & 1 & 1 & 1 & * & & & & \\
 & * & 1 & 1 & 1 & 1 & 1 & 1 & 1 & * & & & \\
 & & * & 1 & 1 & 1 & 1 & 1 & 1 & 1 & * & & \\
 & & & * & 1 & 1 & 1 & 1 & 1 & 1 & 1 & * & \\
 & & & & * & 1 & 1 & 1 & 1 & 1 & 1 & 1 & * \\
 & & & & & * & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 & & & & & & * & 1 & 1 & 1 & 1 & 1 & 1 \\
 & & & 0 & & & & 1 & 1 & 1 & 1 & 1 & 1 \\
 & & & & & & & 1 & 1 & 1 & 1 & 1 & 1 \\
 & & & & & & & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

Die durch * gekennzeichneten Matrixelemente sind nach Satz 4. b. in Abhängigkeit von den entsprechenden Wahrscheinlichkeiten $Q_{i,j}$ zu besetzen.

Literatur:

[1] Denning, P. J.:
Virtual Memory
Computer Surveys, Vol. 2 (1970), S. 153 - 189

[2] Denning, P. J.:
The Working Set Model for Program Behaviour
Comm. of the ACM 11 (1968), S. 323 - 333

[3] Gelenbe, E.:
A unified Approach to the Evaluation of a Class of Replacement Algorithms
1972. Erscheint in IEEE Transactions of Computers

[4] Belady, L. A.:
A Study of Replacement Algorithms for Virtual Storage Computers
IBM Syst. Journal 5 (1966), S. 78 - 101

[5] Aho, A. V., Denning, P. J. und Ullman, J. D.:
Principles of Optimal Page Replacement Algorithms
Journal of the ACM 18 (1971), S. 80 - 93

[6] Kilburn und andere:
One - Level Storage System
IEEE - EC 11 (1962), S. 223 - 335

[7] Lee, F. F.:
Study of "Look-aside" - Memory
IEEE - EC 18 (1969), S. 1062 - 1064

[8] Hotz, G.:
Informatik - Rechenanlagen
Teubner Studienbücher. Stuttgart 1972

In [1], [3] und [5] sind über 80 weitere Literaturstellen über diesen Themenkreis angegeben.