Realisierung eines automatischen Normalmodus-Demandpagings im BS3

STARG-Tagung

Konstanz

7. Oktober 1976

Joachim Backes
Regionales Hochschulrechenzentrum
der Universität Kaiserslautern

Pfaffenbergstraße 95
D-6750 Kaiserslautern

Inhaltsverzeichnis

0 Anmerkung des Verfassers

Die in diesem Dokument in grafischer Form aufgezeigten Messergebnisse sind manuell anhand von in Papierform vorhandenen Ausdrucken mittels Office-Funktionen erstellt worden, da die Originaldatensätze nach über 30 Jahren leider nicht mehr in digitaler Form vorliegen. Insofern stellen die Grafiken möglicherweise nicht mehr den damaligen (1975) realen Sachverhalt dar.

1 Einführung

Dem TR 440-Benutzer stehen zwei Möglichkeiten zur Verfügung, Programme mit einem über dem benutzerspezifisch maximal zulässigen Hauptspeicher hinausgehenden KSB-Bedarf zu verarbeiten:

- Er kann Programmteile, deren Umfang den jeweiligen Montageobjekten entspricht, als transferierbar erklären und das Load-Unload-Management selbst übernehmen. Das heißt, dass der Benutzer dynamisch während des Programmlaufes durch explizite Anweisungen für das Laden und Verdrängen der einzelnen Programmteile sorgen muss.

- Zum zweiten kann durch Anbinden des Verwaltungsmoduls S&BEECALARM der Ladevorgang weitgehend automatisch ablaufen, wobei der Benutzer aber nur geringen Einfluss auf den Zeitpunkt des Verdrängens und die Auswahl der betroffenen Programmteile hat.

Beide Lösungen, im Prinzip ausreichend, weisen jedoch offensichtliche Nachteile auf:

- Grundsätzlich können nur ganze Objekte als transferierbar erklärt werden, wodurch zwangsläufig die Abbildung Montageobjekt ⇔ Gebiet einen erheblichen Haupt speicherverschnitt hervorruft.
- Haupt- bzw. Rahmenprogramme können in diese Philosophie offenbar nicht eingebettet werden.

Das nachstehend vorgestellte Lösungskonzept vermeidet diese Nachteile und ermöglicht darüber hinaus durch mehrere Parameter die Anpassung an den Zentralspeicherausbau unterschiedlich konfigurierter TR 440-Anlagen.

2 Messungen

So offensichtlich die Vorteile eines Demandpagings auch erscheinen mögen, so lässt sich eine Implementierung dennoch nur dann vertreten, wenn durch Messungen und anschließende Extrapolationen eine Durchsatzverbesserung des Gesamtsystems zu erwarten ist.

In die Messungen werden nur diejenigen Programme einbezogen, die per ◇STARTE initialisiert werden; dazu ist das ◇STARTE-Kommando als Prozedur definiert, wobei der eigentliche Objektlauf durch zwei Programme MESS1 und MESS2 umrahmt wird.

MESS1 besorgt sich aus der OKB des zu startenden Objektes die Aufteilung des Operators in die verschiedenen Gebiete, dessen Größe sowie die Job-Zeit und puffert diese Information in einer von dem -nachstehend beschriebenen- Sammeloperator MESS3 verwalteten Warteschlange.

MESS2 als Nachfolger des gestarteten Benutzerprogrammes übernimmt die von MESS1 generierte Zuordnung, errechnet daraus die Operator-Nettozeit des Userprogrammes und puf-

fert diese wiederum zusammen mit den übrigen Werten in jener Warteschlange. (Stichproben haben ergeben, dass Messfehler durch eingeschachtelte Operatorläufe gering sind).

MESS3 übernimmt die Sendungen des MESS2, um sie in einer Datei abzulegen, die in gewissen Zeitabständen ausgestanzt wird. Die Auswertung schließlich besteht aus mehreren Abschnitten:

- Bestimmen der Abhängigkeit zwischen einer festen Programmgröße und der Summe aller Rechenzeiten, die von sämtlichen Programmen dieser Länge aufgenommen wurden.
- Dieselbe Abhängigkeit bestimmen unter der Annahme, die Summe der Längen der Dauergebiete mit VK = KSP liege unterhalb einer festen Zahl.
- Bestimmen der entsprechenden Verteilungen.

Das Ergebnis der Messungen war folgendes: Unter Zugrundelegung eines Demandpagings, das die Gesamtgröße der residenten Dauergebiete auf 2 K reduziert, zeigt sich bei einer Gesamtzahl von 3770 Messungen, dass Programme von einer Größe ≤ 10K 62 % der Gesamtrechnerzeit aufnehmen, während es ohne Demandpaging erst 7 % waren; bei Programmen von einer Größe ≤ 30K ergab sich immerhin noch eine Differenz von 14 %. Nicht berücksichtigt wurde allerdings die Einplanung eines erhöhten Overheads.

3 Realisierung

Bereits in der Einführung wurde erwähnt, dass die Aufteilung eines Benutzerprogramms in Teile, die Montageobjekten entsprechen, für ein Demandpaging weniger geeignet ist. Damit nicht zwei verschiedene Speicherverwaltungen gleichzeitig ablaufen, muss man gewährleisten, dass ein Operator nicht zusätzlich zum Demandpaging noch irgendeiner anderen Form von Segmentierung unterworfen wurde.

Durch den Aufteilung eines jeden Operators aus Dauer- und Laufzeitgebiete ist der Weg zur Realisierung des Demandpagings quasi fest vorgegeben: Die wichtigsten Operator-Teile liegen in Dauer- bzw. Laufzeitgebieten der Verarbeitungsklasse Hauptspeicher. Diese Verarbeitungsklasse wird bereits beim Operatorstart realisiert, d. h. schon dann finden Ladevorgänge statt. Es ist einleuchtend, dass eine Abänderung der Verarbeitungsklassen nach der Montage, aber vor Programmstart in eine solche ≠ Hauptspeicher den Ladevorgang beim Programmstart verhindert. Zerteilt man zusätzlich die bezüglich ihrer Verarbeitungsklasse geänderten Gebiete in solche kleinerer Länge (wobei operatorrelative Adressen erhalten bleiben), so lässt sich beim nachfolgenden Programmlauf die Hauptspeicherbelegung verschnittfrei regulieren, und zwar einschließlich des Hauptprogrammes, vorausgesetzt, ein Verwaltungsmodul übernimmt diese Steuerung. Seine Aufgabe: BEEC-, BEIC- und Versorgungsalarme abfangen und je nach Inhalt des Adressenregisters das entsprechende Gebiet laden (unter Umständen auch zwei Gebiete bei BEIC-Alarmen, wenn die zu ladende Indexbasis über Seitengrenze hinausgeht). Die Verwaltungsmoduln selbst bleibt stets resident.

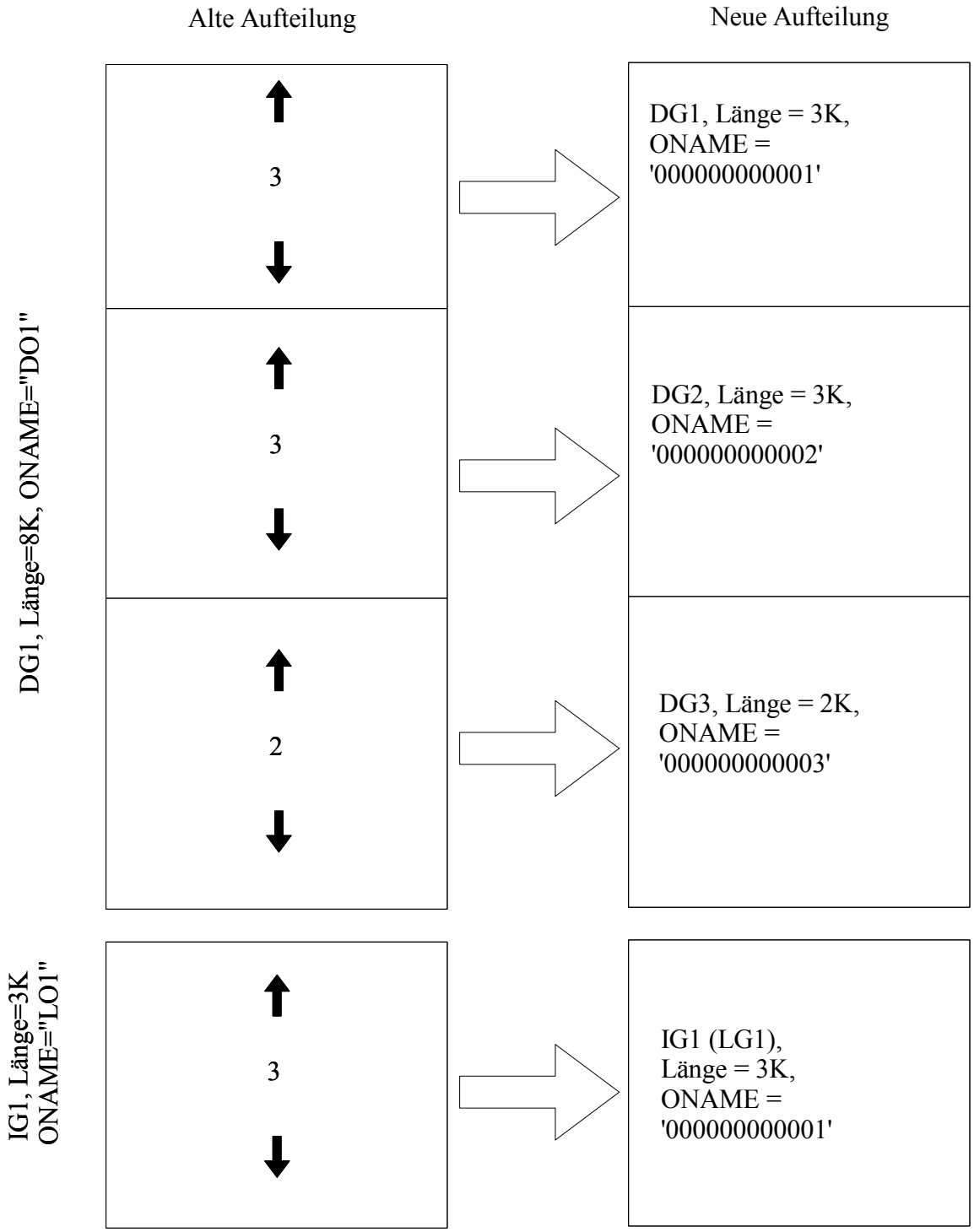
4 Modulbeschreibungen

Das Demandpaging-System besteht aus 2 Komponenten: Dem Aufteilungsoperator KL&DEMONTIER (nur Dauergebiete) bzw. KL&DEMOIG (auch Laufzeitgebiete), sowie dem Steuermodul KL&ALARM, der den Seitenwechsel durchführt. (Bei in höheren Sprachen abgefassten Programmen kann zusätzlich eine Modifikation von S&CC erforderlich sein).

4.1 KL&DEMONTIER (KL&DEMOIG)

KL&DEMONTIER (auf KL&DEMOIG wird nur bei Unterschieden hingewiesen) wird durch ◇OPAENDERE (◇OPIG) gestartet. Im wesentlichen bestehen die Spezifikationen aus Programmnamen (=PROGRAMM) sowie GRAD und RESIDENT.

Anhand der OKB werden die Dauergebiete (selbstverständlich auch die zugehörigen Initialgebiete) in Stücke mit der Länge SGRAD zerlegt; das jeweilige "alte" Gebiet wird nach erfolgter Zerlegung gelöscht. Lediglich Gebiete der Lange ≤GRAD werden bis auf den ONAMEN unverändert übernommen. Die entsprechenden Gebietsbeschreibungen werden einschließlich synthetischer ONAMEN in einer neuen OKB gerettet. Der Erläuterung möge nachstehendes Beispiel dienen:



DG1 wird aufgeteilt in 2 Gebiete der Länge 3K und eines der Länge 2K, IG1 (LG1) wird unverändert übernommen.

Damit im der Demontage folgenden Objektlauf wegen des Zugriffs des Moduls KL&ALARM auf die Gebietsbeschreibungen nicht mehrmals die OKB gelesen werden muss, legt KL&DEMONTIER relevante Information wie die Gebietsbeschreibungen, die Start- und Alarmadresse des Operators,die Angabe zu RESIDENT, Indexbasis sowie Unterprogrammordnungszähler in dem zum Modul KL&ALARM gehörenden Gebiet ab. Der so demontierte Operatorkörper erhält als Start- und Alarmadresse Eingänge in KL&ALARM.

4.2 KL&ALARM

Der Modul KL&ALARM sorgt bei Speicherschutzalarmen für Auswahl und Transfer der auszulagernden Programmseite(n) sowie für das Laden der Zielseite.

Wie bereits erwähnt, liegt die Startadresse des demontierten Operators in KL&ALARM. In der Anfangsbehandlung erweitert KL&ALARM die Gebietsbeschreibungen um die Gebietsnummern. Danach wird die Seite, in der die ursprüngliche Startadresse liegt (sowie die Indexbasis) geladen und der Operator dort fortgesetzt.

Der eigentliche Verwaltungsteil wird bei BEEC-, Versorgungs- (und BEIC-) Alarmen aufgerufen. Nachdem festgestellt ist, dass der Alarm nicht auf einem Programmierfehler beruht, wird zunächst durch Vergleich mit RESIDENT entschieden, ob die maximale KSP-Belegung bereits erreicht ist. Wenn nein, wird lediglich die alarmerzeugende Seite geladen. Wenn ja, wird zuvor unter Zuhilfenahme eines Algorithmus (s. u.) eine im KSP liegende Programmseite verdrängt. Mit den Registern aus dem Alarmkeller kann der Lauf fortgesetzt werden.

5 Verdrängungsalgorithmen

KL&ALARM enthält eine Liste (Länge = Maximalzahl geladener Gebiete), in der für jede geladene Seite S_i ($1 \leq i \leq$ RESIDENT) ein Verdrängungsgewicht $F(s_i)$ geführt wird mit

$$F(s_i) = \delta \cdot p(s_i) + (128\text{-}z) \cdot r(s_i),$$

wobei bei festem $\delta>0$ die Initialwerte $z = 0$ (Ladezähler) und $p(s) = r(s) = 0$ für alle Seiten s verwendet werden.

Aktionen bei Zugriffs-Alarmen auf Seite **s**	
z:=((z+1) ≡ 129)	
Hat das geänderte z den Wert 0?	
N	J
*p(**s**)++*	*p(x)=r(x)=0 für alle Seiten x*

Bei jedem relevanten BEEC-,Versorgungs- oder BEIC-Alarm wird F(s) für die geladenen Seiten neu berechnet; die Seite mit minimalem Verdrängungsgewicht wird verdrängt (Alarmauslösende Seite sowie Indexbasis erhalten unabhängig davon das Gewicht 'FFFFFF', damit jene nicht verdrängt werden).

F kann wie folgt interpretiert werden: Durch den ersten Summanden $\delta \cdot p(s)$ wird erreicht, dass relativ oft geladene Seiten nicht verdrängt werden. Der zweite Summand enthält ein "Gedächtnis" aus dem vorausgegangenen "128er-Zyklus". Sinnvollerweise muss dieser Einfluss mit der Zeit abnehmen, was man durch den Faktor 128-z erreicht. Um die Abhängigkeiten nicht über den gesamten Programmlauf hinweg wirksam werden zu lassen, werden die Algorithmusvariablen nach je 128 Verdrängungen normiert. Messungen haben ergeben, dass für die untersuchten Fälle bei $\delta=8$ ein Transferminimum erreicht wurde.

6 Einschränkungen

Zur Zeit ist KL&ALARM in den nachstehenden Situationen nicht arbeitsfähig:

(a) Sind im Versorgungsblock eines SSR-Befehles Verweise auf Adressen enthalten (die z.B. E/A-Puffern zugeordnet sein können), welche in einer nicht geladenen Seite liegen, so führt der Zugriff darauf im Abwicklermodus zwar zu einem BEEC-Alarm, der aber nicht an das Benutzerprogramm weitergegeben wird. Die Fortsetzung erfolgt auf der SSR-Fehleradresse mit einer nichtssagenden und demzufolge für einen Seitenwechsel nicht auswertbaren Information.

(b) Muss ein SSR-Befehl aufgrund einer fehlerhaften Versorgung über den Fehlerausgang verlassen werden, und liegt die Fehleradresse in einer nicht zugewiesenen Seite, so wird der Operatorlauf abgebrochen (SSR F zwecks Fortsetzung im Normalmodus wird mit Fehler verlassen).

(c) Es besteht durchaus die Möglichkeit, dass der Adressteil eines SSR-Befehls durch Modifizierung entstanden ist. Tritt in diesem Fall ein Versorgungsalarm auf, so kann KL&ALARM nicht sinnvoll fortsetzen, da wegen fehlender Information der Adressteil des SSR-Befehls nicht reproduzierbar ist.

(d) Es besteht keine Sperre gegen das unkontrollierte Ummelden der Alarmadresse durch das Benutzerprogramm.

Die mangelhafte Alarmbehandlung des Abwicklers, die dieses Fehlverhalten begründet, wurde bereits im Dezember 1974 der Fa. CGK mitgeteilt. Erst mit Einführung der MV18 (frühestens Oktober 1976) werden sich Verbesserungen ergeben: Fall (a) und (b) sind dadurch lösbar, dass per SSR 0 34 SSR-Fehler auf Ereignisalarme umgeleitet werden, d. h. auf einen Aufruf von KL&ALARM. Bei (a) kann KL&ALARM den fehlenden Bereich laden, bei (b) aus dem Alarmkeller oder dem Versorgungsblock (falls möglich) die SSR-Fehleradresse bestimmen. Fall (c) ist durch einen normierten Start mit Vorgabe des gesamten Alarmkellers lösbar, für Fall (d) existiert erst ab MV18 ein spezieller SSR-Befehl.

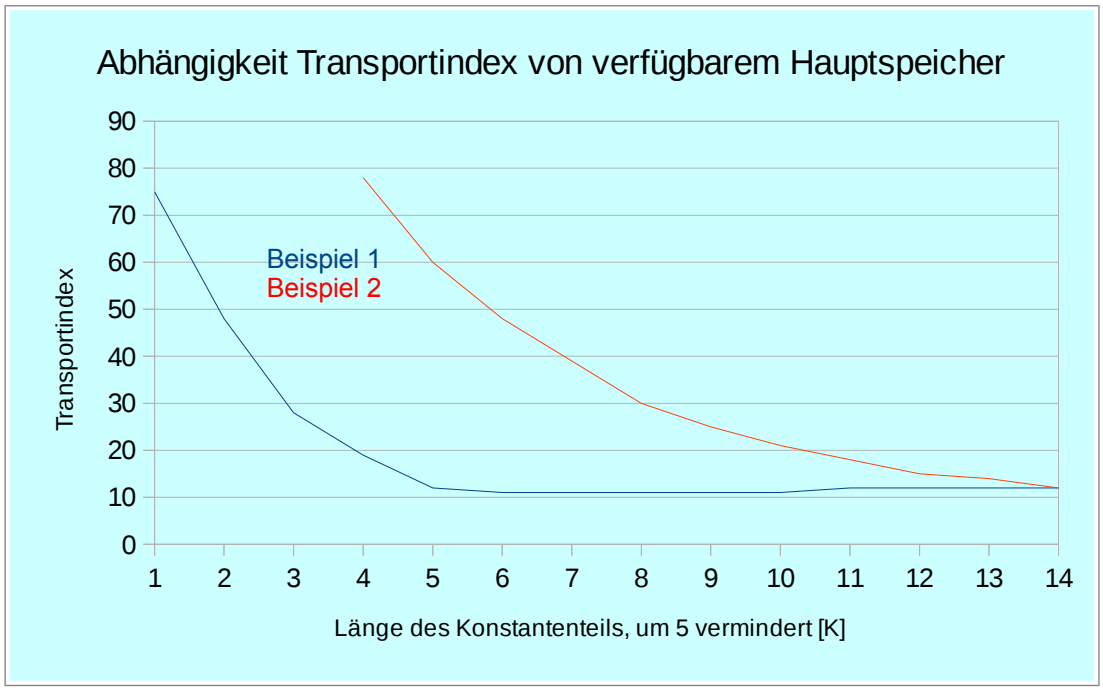
7 Ein Beispiel

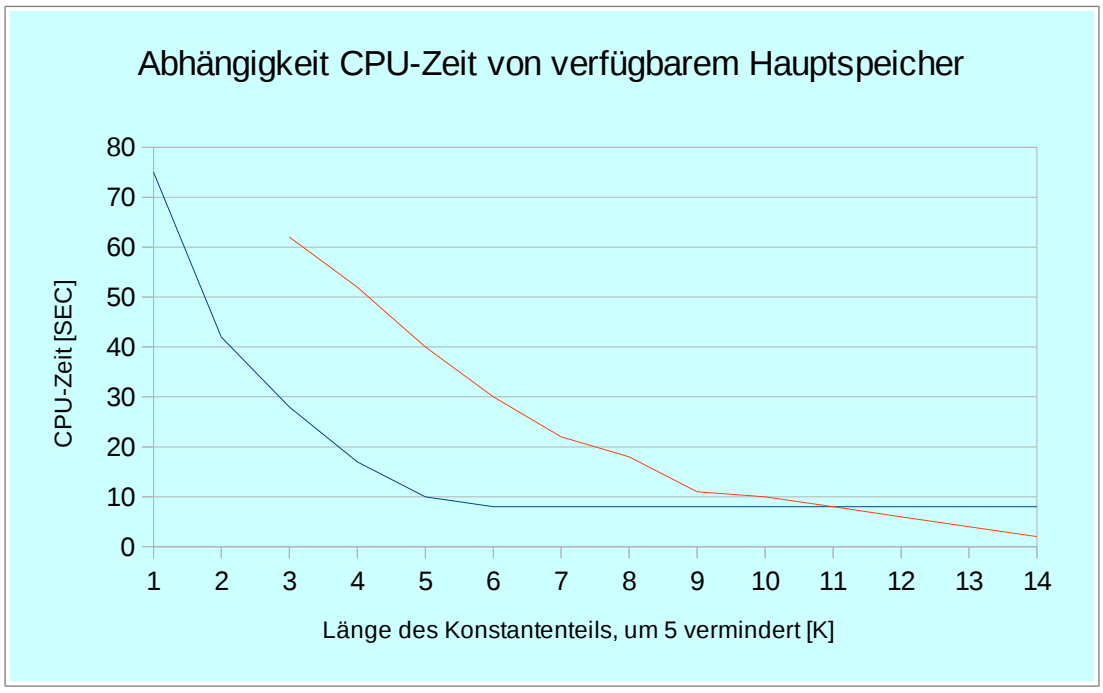
An den nachstehenden Beispielen soll erläutert werden, wie in Abhängigkeit von dem dem Programm zur Verfügung stehenden Hauptspeicher CPU-Belastung und Transport-index variieren.

Bei dem der Demontage unterzogenen Programm handelte es sich um den in Saar-brücken entwickelten PASCAL-Compiler. Im Testbeispiel 1 übersetzte der Compiler eine Quelle mit einem sehr großen Vereinbarungsteil, aber kleinem Ausführungsteil, d. h. der Compiler bewegte sich, vom Adressraum her gesehen, in einem relativ eng eingegrenzten Rahmen. Demnach blieb bei der KSP-Reduzierung in einem weiten Bereich die CPU- und E/A-Belastung konstant, erst als der verfügbare Hauptspeicher so gering geworden war, dass auch Teile des o.a. Rahmens geswappt wurden, stiegen CPU- und E/A-Last merklich an.

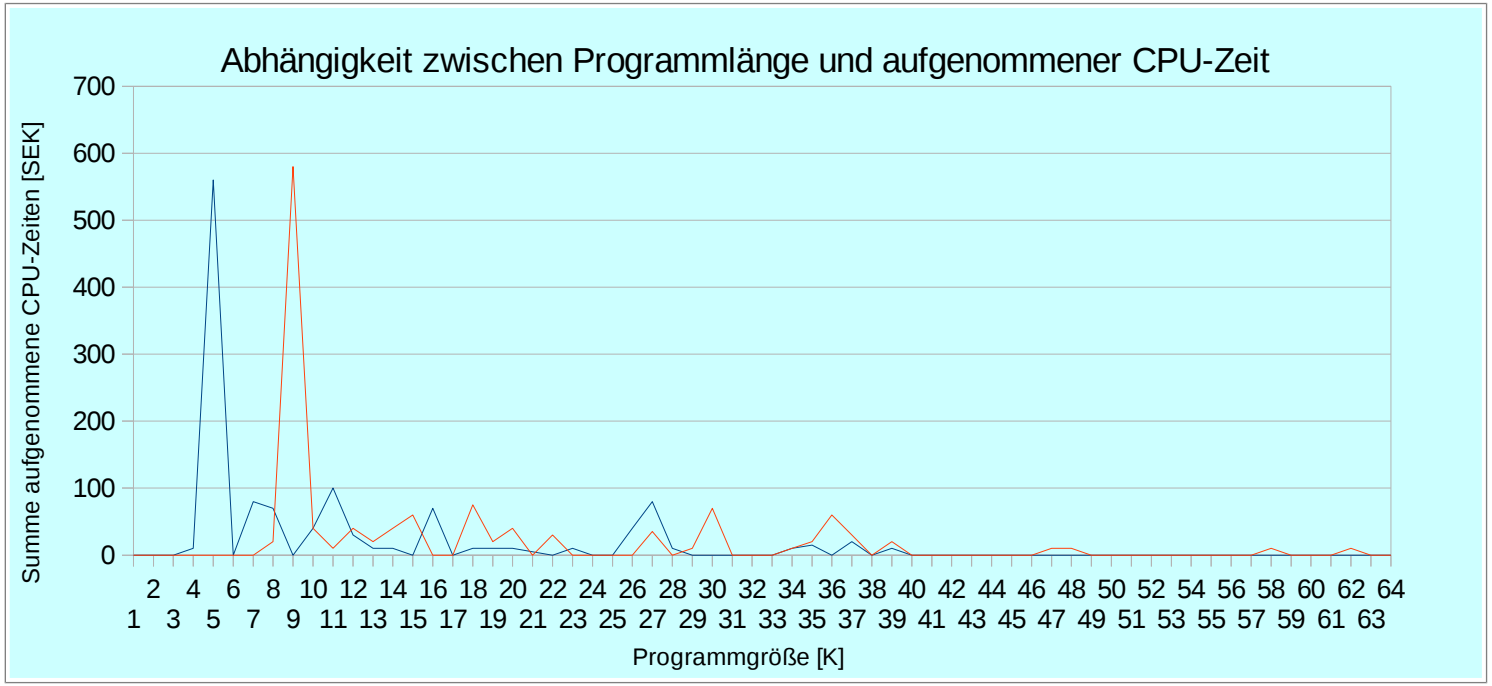
In Testbeispiel 2 dagegen war ein kleiner Vereinbarungsteil, aber größerer Ausführungsteil zu übersetzen, so dass alle Routinen des Compilers eine gleichmäßige Belastung erfuhren. Das aus dem Programm ersichtliche Verhalten lässt sich damit unmittelbar erklären.

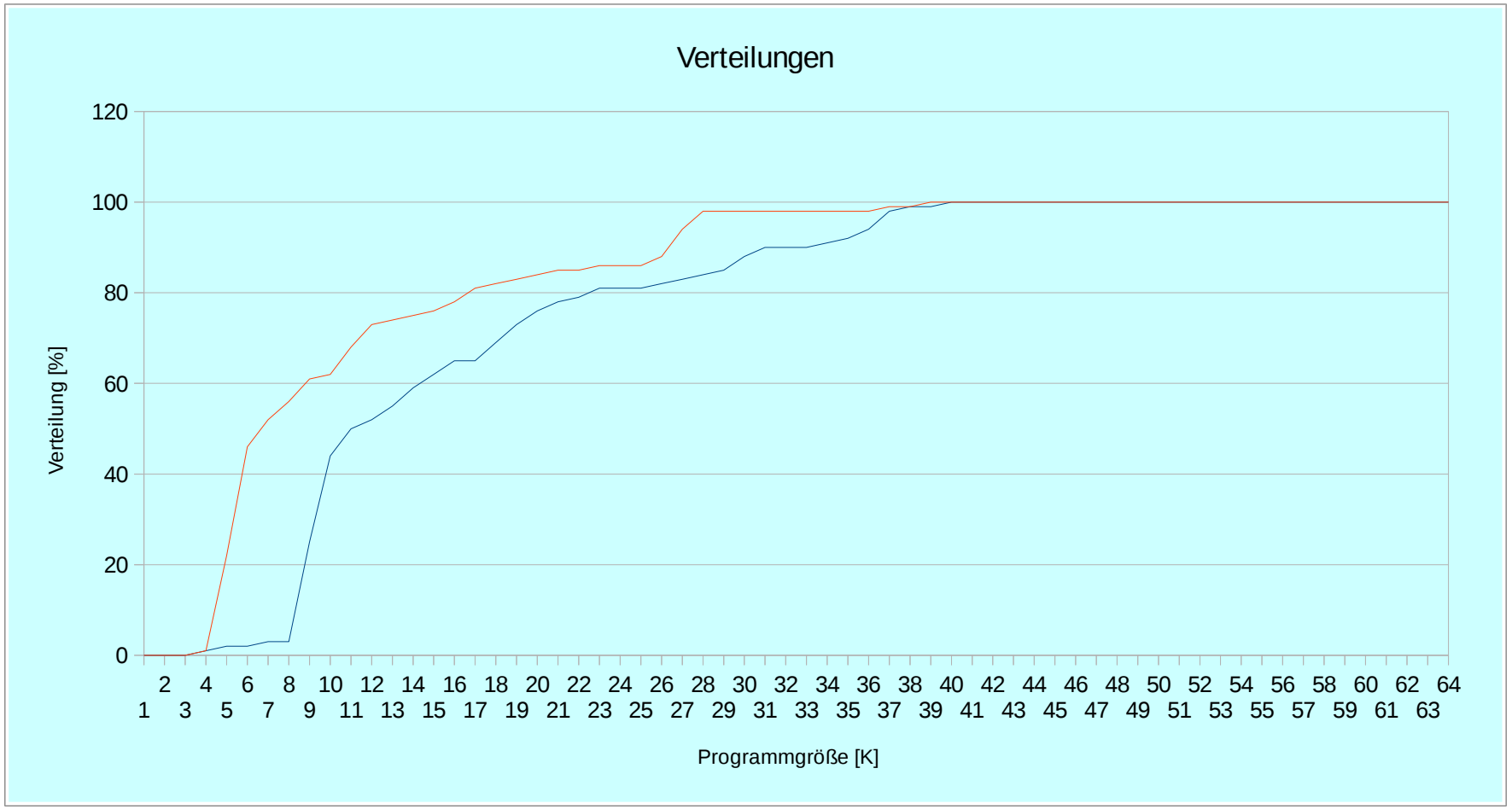
Wenn man allerdings berücksichtigt, dass eine Normalbehandlung durch KL&ALARM etwa 200 Befehle beansprucht, ist der hohe CPU-Zuwachs doch verwunderlich. Eine Erklärung waren relativ lange Ausführungszeiten für bestimmte SSR's, so z. B. SSR 3 8 (Gebiet zur Verarbeitung aufrufen) mit 24-25 msec, wie Messungen ergaben.

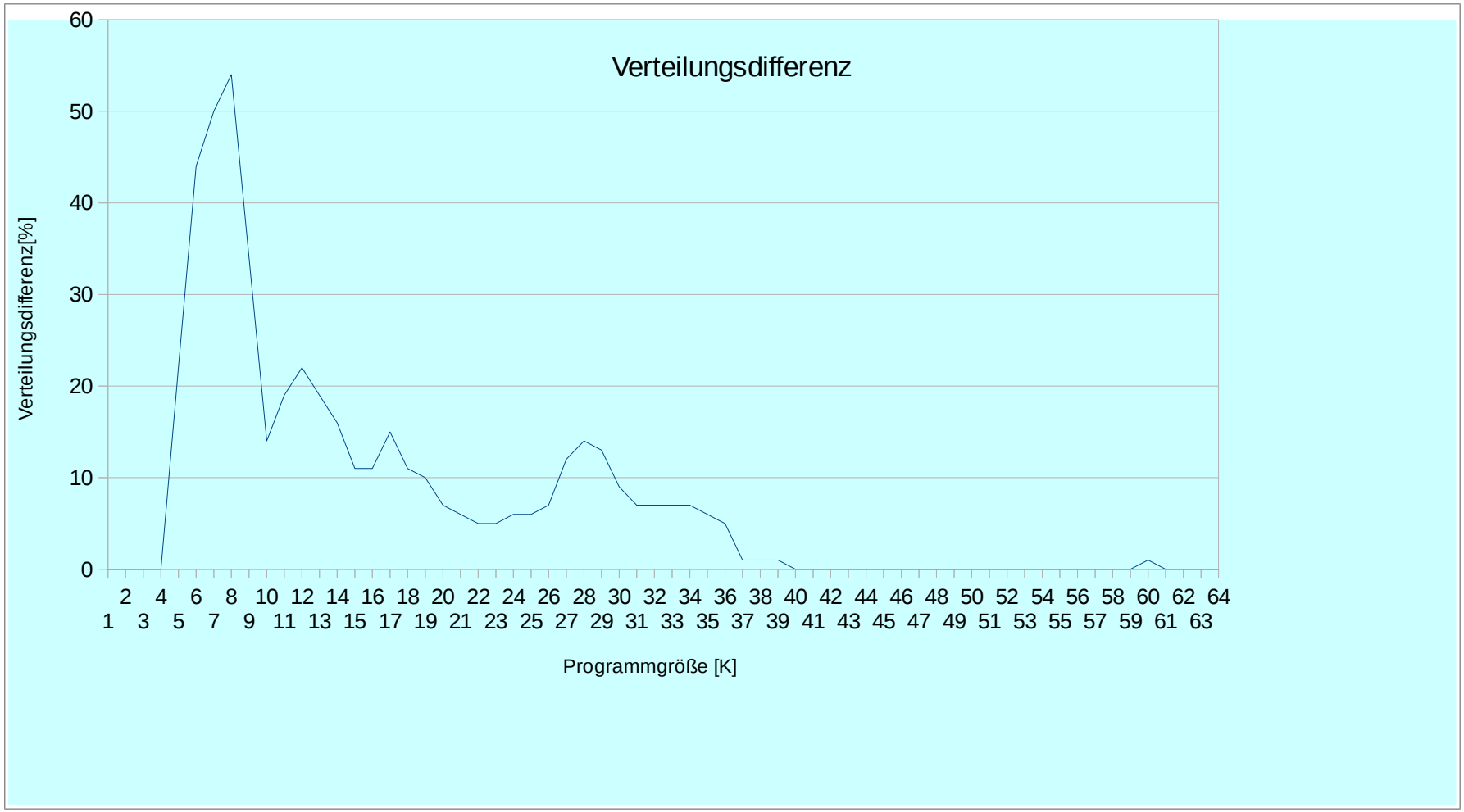

Abhängigkeit Transportindex von verfügbarem Hauptspeicher
Transportindex
90
80
70
60
50
40
30
20
10
0
Beispiel 1
Beispiel 2
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Länge des Konstantenteils, um 5 vermindert [K]


Abhängigkeit CPU-Zeit von verfügbarem Hauptspeicher
CPU-Zeit [SEC]
80
70
60
50
40
30
20
10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Länge des Konstantenteils, um 5 vermindert [K]

8 Anhang: Diagramme der Messungen aus 1.




Verteilungen
120
100
80
60
40
20
0
Verteilung [%]
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
Programmgröße [K]


Verteilungsdifferenz
Verteilungsdifferenz[%]
0
10
20
30
40
50
60
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
Programmgröße [K]