RUHR - UNIVERSITÄT BOCHUM

Arbeitsbericht
des

Rechenzentrums

Direktor: Prof. Dr. H. Ehlich

Nr. 7404

Quellbezogene FORTRAN Optimierungen für den Compiler des TR 440

von

Richard Green und Karl-Heinz Mohn

Bochum, Juni 1974
Universitätsstraße 150, Gebäude NA

Inhaltsverzeichnis
==================

I. Einleitung

Seit der Entwicklung von Compilern für höhere Programmiersprachen wurden laufend Anstrengungen unternommen Methoden zu entwickeln aus einer vorliegenden Quelle, (z. B. in FORTRAN) in "optimaler" Zeit "optimalen" Code zu generieren. Hierbei lassen sich die Bemühungen in drei Richtungen unterscheiden:

1. Optimal in Hinblick auf die Übersetzungszeiten
2. Optimal in Hinblick auf den Speicherbedarf
3. Optimal in Hinblick auf die Laufzeiten des Objekt-Programms

Da ein Compiler eine Sprache in ihrer ganzen Allgemeinheit übersetzen muß, ist es für den Entwickler eines solchen Compilers außerordentlich schwierig, die weitreichendsten Optimierungen anzubringen. Eine weitere Schwierigkeit besteht darin, daß es im allgemeinen unbekannt, und meistens auch unterschiedlich ist, wie sich die einzelnen Programmierer in bestimmten Situationen verhalten werden. Es nutzt sehr wenig, wenn Statements oder Sequenzen optimiert werden, von denen der Entwickler glaubt, daß sie häufig benutzt werden, aber in Wirklichkeit selten oder fast nie von den Programmierern verwendet werden. Da es aber in diesem Sinne keinen "Standardprogrammierer" gibt, ist auch nicht zu erwarten, daß es einen "Standardcompiler" gibt, der genau wohldefinierte i.a. häufig vorkommende Sequenzen von Quelltexten optimiert, und somit jeder Benutzer sofort weiß, an welcher

Stelle und in welcher Situation er wie programmieren muß, um den effektivsten Code generiert zu bekommen. Im Gegenteil, der Grad der Optimierung und die Sprachelemente die optimiert werden, sind bei den verschiedenen Herstellern unterschiedlich. Häufig existieren für einen Rechner sogar mehrere Compiler-Versionen mit unterschiedlicher Optimierungsart und -umfang.

Ein wesentlicher Nachteil ist außerdem in den meisten Fällen, daß eine ausführliche und detaillierte, dem Benutzer leichtzugängliche Beschreibung des vorliegenden Compilers fehlt, so daß es für ihn extrem schwierig ist herauszufinden, welche Optimierungen vorgenommen werden und unter welchen Bedingungen. In den Fällen, in denen Beschreibungen existieren, sind sie entweder zu allgemein gehalten, d. h., es fehlen praktische Hinweise für den Benutzer in speziellen Situationen, oder aber sie sind mehr für den Implementierer geschrieben, so daß sie weit über das hinausgehen, womit sich selbst der Programmierer, der möglichst effektive Programme schreiben will, zu beschäftigen gewillt ist.

Ein weiterer wesentlicher Punkt ist der, daß selbst die denkbar besten Compiler nicht in der Lage sind, solche Programmteile zu erkennen, die später am häufigsten durchlaufen werden, um dort die Hauptarbeit der Optimierung zu leisten.

Aus diesen Gründen und angeregt durch die Arbeiten von D. Knuth [1], Ch. Larson [2] und P.B. Schneck und E. Angel [3], sollen die folgenden drei Aspekte untersucht werden:

1. Welche Optimierungen werden unter welchen Voraussetzungen von dem FORTRAN-Compiler der TR 440 vorgenommen, und durch welche Maßnahmen läßt sich der effektivste Code erhalten.

2. Herausfinden eventueller Programmprofile von auf der hiesigen Anlage gefahrenen FORTRAN-Programmen.

3. Entwicklung eines Precompilers, der aus einer beliebigen FORTRAN-Quelle eine solche macht, von der der vorliegende Compiler einen möglichst effektiven Code generiert.

In dem vorliegendem Arbeitsbericht wird zunächst ausführlich über empirische Untersuchungen zu Punkt 1) berichtet. Mit diesem Teil zu beginnen erschien aus zwei Gründen sinnvoll.

i) Auf Grund der durchgeführten Testläufe und Untersuchungen lassen sich bereits gewisse Tips und Rezepte formulieren, mit Hilfe derer der Programmierer durch Modifikationen der Quelle die Laufzeiten seiner Programme zum Teil wesentlich verkürzen kann.

ii) Mit Hilfe der hier gewonnenen Erkenntnisse lassen sich die Untersuchungen im zweiten Teil gezielter durchführen.

Es ging also im wesentlichen darum herauszufinden, welche Modifikationen des Quellprogramms in welcher Situation schnellere Objektprogramme zur Folge haben. Aus diesem Grunde wurde eine Vielzahl von FORTRAN-Programmen mit den verschiedensten Modifikationen geschrieben und Laufzeitvergleiche angestellt. Darüber hinaus wurde in den meisten Fällen der vom Compiler generierte Code genauer untersucht, um zusätzliche Informationen zu erhalten, warum in speziellen Fällen die erwartete Laufzeitersparnis nicht eintrat. Bei den Laufzeitvergleichen muß beachtet werden, daß sich ein gewisser Overhead nicht vermeiden läßt. Einmal dadurch, daß bei den Zeitangaben Rundefehler auftreten, und zum zweiten durch den Multiprogramming-Betrieb. Das zweite wurde dadurch im wesentlichen ausgemärzt, daß wir die Programme allein auf dem Rechner laufen ließen. Im übrigen ging es auch nicht so sehr darum, absolute Zeitvergleiche zu bekommen, als vielmehr die Trends deutlich zu machen. Bei allen Modifikationen wurde weiter nicht darauf geachtet, ob das laufzeitschnellere Objekt mehr Speicherplatz benötigt, oder ob die Quelle dadurch eventuell etwas unübersichtlicher wurde. Es wurde einzig und allein das Augenmerk darauf gerichtet, durch Maßnahmen die jeder Programmierer auf Quellenebene vornehmen kann, günstigere Laufzeiten zu erzielen. Bei der folgenden Zusammenstellung werden sicherlich auch Dinge erwähnt, die vielen Programmierern selbstverständlich erscheinen. Da diese jedoch dennoch häufig nicht beachtet werden, nahmen wir sie nicht nur der Vollständigkeit wegen auf.

II. Arithmetische Optimierungen:

1. Arithmetische Ausdrücke

i) Kommen in einem arithmetischen Ausdruck gleiche Unterausdrücke vor, so wird er automatisch vom Compiler optimiert. Es ist also nicht notwendig, diese in Klammern zu setzen, oder vorher einer Hilfsvariablen zuzuordnen. Kommen gleiche Unterausdrücke in verschiedenen Ausdrücken an verschiedenen Stellen im Programm vor, so sollte man auf jeden Fall vor dem ersten Auftreten eine eigene Anweisung einführen und bei jedem weiteren Auftreten die Variable verwenden, da der Compiler die Optimierungsmöglichkeit über mehrere Anweisungen nicht erkennt. Wird die Variable der linken Seite einer Anweisung in der unmittelbar folgenden Anweisung benutzt, so kann man überflüssige Speicher- und Bringe-Befehle sparen, wenn man diese Variable gleich als erste in diesem Ausdruck benutzt, z. B.

statt $\begin{matrix} A=.... \\ B= C*A \end{matrix}$ sollte $\begin{matrix} A=.... \\ B= A*C \end{matrix}$ geschrieben werden.

ii) Alle Ausdrücke in denen Konstanten vorkommen, sollten soweit wie möglich vereinfacht und vorberechnet und die Einführung redundanter Variablen vermieden werden.

Beispiel:

statt

```
PI = 3.14159265
A  = B + D / SIN (PI / 4)
Q  = B + A * SIN (PI * X)
```

schreibe

```
A = B + D * 1.4142136
Q = B + A * SIN (3.14159265 * X)
```

iii) Solche Ausdrücke und Teilausdrücke in einer DO-Schleife, die sich nicht verändern, sollten in jedem Fall vor die Schleife gesetzt werden, um bei jedem Durchlauf unnötige Arbeit zu sparen, z. B.

statt

```
      DO 10  I = 1, 100
      A(I) = B + C * P(I) /(Z + 1.)
10    CONTINUE
```

schreibe

```
      R = C / (Z + 1.)
      DO 10 I = 1, 100
      A(I) = B + P(I) * R
10    CONTINUE
```

oder

```
      DO 20 I = 1, 100
      R = A * D/Z + ARRAY(I) * P/Q
      P = P + 1
      :
      X = T + ARRAY(I + 1) * P/Q
20    CONTINUE
```

Hier läßt sich nicht nur P/Q als gemeinsamer und schleifenunabhängiger Unterausdruck herausziehen,

denn

```
      F = A * D/Z
      E = ARRAY(1) * P/Q
      DO 20 I = 1, 100
      R = F + E
      P = P + 1
      :
      E = ARRAY(I + 1)* P/Q
      X = T + E
   20 CONTINUE
```

liefert dasselbe mit wesentlich weniger Operationen.

2. Arithmetische Operationen

i) Wegen der wesentlich schnelleren Addition gegenüber der Multiplikation, sollte immer

A + A statt 2.*A geschrieben werden.

ii) Müssen in einem Programm oder Ausdruck mehrere Divisionen mit demselben Divisor durchgeführt werden, so bilde man einmal das Reziproke des Divisors und verwende dann nur noch Multiplikationen (siehe auch Anhag 2.3). Ferner schreibe man statt A/2.0 immer A*0.5 .

iii) Ganzzahlige Exponentiation mit Exponenten ≤ 4 werden auf wiederholte Multiplikation zurückgeführt, jedoch wird diese in optimierter Form durchgeführt, so daß es vorteilhafter ist, z. B. A**3 zu schreiben, statt der expliziten Form A*A*A. A**3.0 sollte hier niemals geschrieben werden. Bei Exponenten > 4 kann man durch Faktorisierung wesentliche Zeiteinsparungen gegenüber der Exponentiation erzielen z. B. A^6 = (A**3)**2 statt A**6. Darüber hinaus sollte A**0.5 immer als SQRT(A) geschrieben werden, da Exponentiation mit reellen Exponenten zu mehreren Unterprogrammaufrufen führt und somit SQRT ebenfalls wesentlich schneller ist.

3. Datenkonvertierung

i) Für jede Konvertierung von Variablen wird vom Telefunkencompiler jedesmal "in line" eine Code-Sequenz erzeugt. Die für eine Konvertierung z. B. von INTEGER nach REAL benötigte Zeit hängt von der Anzahl der Ziffern ab, ist jedoch wenigstens ungefähr zweimal so lang wie eine Multiplikation. Tritt also eine INTEGER-Variable mehr als einmal in einem oder mehreren gemischten Ausdrücken auf, so sollte auf jeden Fall eine Hilfsvariable eingeführt werden.

```
A = I*P -(I-2)/X + I * 3 + (I + 4)* Q
```

wird zu

```
C = I
A = C*(3.0 + P) + (C + 4.)* Q - (C - 2.)/ X
```

ii) In vielen Anwendungen wird der Schleifenparameter in gemischten arithmetischen Ausdrücken zur Berechnung benutzt, was eine große Anzahl von Konvertierungen zur Folge hat. In solchen Fällen läßt sich die folgende Technik verwenden, die auch für die Schleifenoptimierung vorteilhaft ist (siehe III. 3. und Anhang 1)

Beispiel: Initialisierung eines Feldes

```
      DO 10 I = 1, 100
   10 A(I) = I
```

stattdessen

```
      WERT = 0.0
      DO 10   I = 1, 100
      WERT = WERT + 1.0
   10 A(I) = WERT
```

iii) Bei dem durchgeführten Test haben sich die folgenden allgemeinen Regeln ergeben:

a) Während der schrittweisen Code Generierung für einen arithmetischen Ausdruck, wird beim Auftreten von unterschiedlichen Typen grundsätzlich von INTEGER nach REAL konvertiert. ("in line" Code erzeugt)

b) Konstanten werden zur Übersetzungszeit bereits konvertiert.

c) Die rechte Seite eines Ausdruckes wird dem Typ der Variablen der linken Seite angepaßt.

Beispiele:

1. ISUM = 6.2 oder ISUM = 6.0
   werden nach b) zur Übersetzungszeit bereits konvertiert. Im Code also kein Unterschied zu ISUM = 6

2. IF(SUM.GT.46) GOTO 2
   ergibt gleichen Code wie
   IF(SUM.GT.46.0) GOTO 2,
   also keine Konvertierung zur Laufzeit (nach b)).

3. Dagegen ergibt

   IF(ISUM.GT.6.0) GOTO 2

   den gleichen Code wie

   IF(FLOAT(ISUM).GT.6.0) GOTO 2,

   also eine Konvertierung zur Laufzeit (nach a)).

4. J = I * 6.0 erzeugt einen Code wie

   J = IFIX (FLOAT(I) * 6.0),

   d. h. zweifache Konvertierung (nach Regeln a) und c)).

III. Nichtarithmetische Optimierungen

1. Ein- Ausgabe

i) Da jede Ein-Ausgabe-Anweisung zu mehreren Unterprogrammaufrufen führt, sollte die Zahl dieser Anweisungen so gering wie möglich gehalten werden, indem man, wenn irgend möglich, die Listen mehrerer solcher Anweisungen zu einer verbindet.

ii) Bei Zwischenspeicherungen und Zurücklesen grundsätzlich "umformatet" Schreiben und Lesen benutzen.

iii) Nichtindizierte Feldnamen als E/A Listenelemente führen zu wesentlich schnelleren Programmen, als z. B. implizite DO-Schleifen.

iv) Die Methode, mehrere Listenelemente durch eine EQUIVALENCE-Anweisung zu einem eindimensionalen Feld zu machen, bringt keine Rechenzeitvorteile.

2. Bedingte Sprunganweisungen

i) Die logische IF-Anweisung ist, da es sich hier immer nur um eine zweifache Verzweigung handelt schneller als die arithmetische IF-Anweisung, die immer eine dreifache Verzweigung generiert. Wird also nur eine zweifache Verzweigung benötigt, so sollte man immer das logische IF benutzen.

zum Beispiel:

```
      IF(A-B) 2, 2, 4
    2 X = X + 1.0
      :
      :
    4 A = A * 2.0
```

wird zu

```
      IF(A.GT.B) GOTO 4
      X = X + 1.0
      :
      :
    4 A = A * 2.0
```

ebenfalls ist IF(A) 2, 2, 4 langsamer als IF(A.GT.0) GOTO 4 (über die Größe der Zeitersparnis siehe auch Programmbeispiel in Anhang 2.1).

ii) In den Fällen, in denen eine dreifache Verzweigung notwendig ist, sollte der Programmierer wenn möglich darauf achten, daß der arithmetische Ausdruck von der Form ist, daß er in den meisten Fällen negativ ist, denn dann werden in der Ausführungsphase die anderen häufig unnötigen Abfragen gespart.

Beispiel: Wenn man etwa erwartet, daß am häufigsten GAMMA > A(I) ist, so schreibe man

```
IF(A(I) - GAMMA) 1, 2, 3
```

und nicht etwa

```
IF(GAMMA - A(I)) 3, 2, 1
```

3. DØ - Schleifen:

i) DØ-Schleifen, in denen der Laufindex nicht zur Berechnung -außer als Feldindizes in ganz bestimmter Form

(Siehe Tabelle 1 in Abschnitt V.1.)- vorkommt, werden hochgradig optimiert und sollten daher immer einer Konstruktion mit der IF-Anweisung vorgezogen werden.

ii) Wenn möglich sollte man bei den Parametern (Anfangswert, Schnittweite und Endwert) statt Variable, Konstanten verwenden, da dies 1. zu schnelleren Laufzeiten führt (wegen Schleifenoptimierung) und außerdem ein schlechter Programmierstil ist.

Beispiel: statt

```
      N = 100
      :
      DØ  10 I = 1, N
      :
   10 CONTINUE
```

```
schreibe
      DØ  10 I = 1, 100
      :
   10 CONTINUE
```

iii) Wird dagegen der Laufindex als Variable zu Berechnungen in der Schleife benötigt, so kann man trotzdem die optimierte Form der DO-Schleife erhalten, wenn man die Technik von II. 3. verwendet, nämlich indem man eine zusätzliche Zählvariable einführt. (Siehe auch Anhang 1)

Beispiel:

```
      DØ  10 I = 1, 100
      :
      ISUM = I * K
      :
   10 CONTINUE
```

sollte ersetzt werden durch

```
      J     = 0
      DØ 10 I = 1, 100
      J     = J + 1
      :
      :
      ISUM  = J * K
      :
      :
10    CONTINUE
```

Diese Technik sollte jedoch nie verwandt werden, wenn der Laufindex nur als Feldindex benutzt wird.

iv) Im folgenden werden alle Situationen zusammengestellt, in denen eine Schleifenoptimierung durchgeführt wird und in denen diese unterbleibt.

1. Alle Schleifenparameter (Anfangswert, Schrittweite, Endwert) sind Konstanten.

2. Wenigstens ein Schleifenparameter ist eine Variable.

3. Laufvariable tritt nur als Index auf.

4. Laufvariable wird als Integer Variable benutzt.

5. Sprung aus der Schleife ohne, daß auf Laufvariable zugegriffen wird.

6. Sprung aus der Schleife mit anschließendem Zugriff auf die Laufvariable.

7. Keine geschachtelten DO-Schleifen.

Die optimierte Form der DO-Schleife wird vom Compiler generiert, wenn die Bedingungen 1 und 7 oder 1 und 3 und 7 oder 2 und 3 und 7 erfüllt sind. In allen anderen Fällen wird die unoptimierte Form generiert. Situation 4 läßt sich wie wir in II. 3. und III. 3. iii) beschrieben haben, in vielen Fällen umgehen.

IV. Unterprogramme

i) Da die Verwendung von Unterprogrammen immer einen gewissen Overhead verursacht, muß derjenige der jede mögliche Zeitersparnis erreichen will, sehr sparsam mit dem Gebrauch von Unterprogrammen sein.

ii) Eine arithmetische Anweisungsfunktion ist immer einem Funktionsunterprogramm vorzuziehen, wenn die Berechnung in einer Anweisung vorgenommen werden kann. Auch arithmetische Anweisungsfunktionen benötigen mehr Zeit, als wenn man die Anweisungen an den jeweiligen Stellen einfügen würde. Der Mehrverbrauch an Zeit ist abhängig von der Anzahl der Parameter und Komplexität des vorkommenden Ausdrucks. Er liegt etwa zwischen 10 % - 40 %.

iii) Der Gebrauch der CØMMØN-Anweisung zur Übergabe der Parameter zwischen rufenden und gerufenen Programmen führt ebenfalls zu Zeitersparnis.

V. Der Gebrauch von Feldern

1. Eindimensionale Felder

i) Obwohl alle Formen von arithmetischen Ausdrücken als Indizes erlaubt sind, sollte man nur die in Tabelle 1 aufgeführten verwenden, da sie den günstigsten Code erzeugen und wie in Abschnitt III. 3 i) beschrieben, unschädlich gegenüber Schleifenoptimierung sind

	Form	normalisierte Laufzeiten (wenn V_1 der Schleifenindex ist)
1	$C \pm V_1$	1.0
2	V_1	1.02
3	$V_1 \pm C$	1.10
4	$C \pm C*V_1$	1.14
5	$C * V_1$	1.14
6	$C * V_1 \pm C$	1.16
7	C	1.22
8	$V_1 + V_2$	1.60

Tabelle 1

Wie die Tabelle zeigt sind sowohl 1 und 3, als auch 4 und 6 für die Schleifenoptimierung unschädlich, dagegen nicht V * C oder der entsprechende Ausdruck für 4 und 6. Doch sollte, wie man sieht, aus Zeitgründen immer 1 und 4 verwendet werden.

ii) Wird in einer Schleife zu einem ganz bestimmten Feldelement häufig zugegriffen, so sollte diese durch die Einführung einer neuen Variablen ersetzt werden.

Beispiel:

```
      DO  10  I = 1, 100
          :
          IF(A(I) - A(1)  6, 7, 8
          :
          :
10    CØNTINUE
```

besser:

```
      A1 = A(1)
      DO  10  I = 1, 100
      :
      IF(A(I) - A1) 6, 7, 8
      :
      :
10    CØNTINUE
```

iii) Es ist immer besser die Indexausdrücke beizubehalten, als die Einführung einer neuen Variablen. Man schreibe

$$R = A(C_1 * V + C_2 * V + C_3) * B + X$$

und niemals

$$K = C_1 * V + C_2 * V + C_3$$

$$R = A(K) * B + X ,$$

da Indexausdrücke wesentlich schneller berechnet werden als arithmetische.

2. Mehrdimensionale Felder

i) Indexrechnungen für mehrdimensionale Felder sind schwierig zu optimieren, deshalb sollte hier das

von verschiedenen Autoren aufgestellte Gebot gelten: "Man benutze nie ein N-dimensionales Feld, wenn bereits ein (N-1)-dimensionales ausreicht.

ii) Wenn mehrdimensionale notwendig sind, so können verschiedene Berechnungen dadurch verkürzt werden, daß man das N-dimensionale Feld mittels einer EQUIVALENCE-Anweisung mit einem 1-dimensionalen Feld gleichsetzt. Zum Beispiel die Initialisierung eines 3-dimensionalen Feldes

```
      DIMENSIØN  A(1O, 1O, 1O)
      DØ 1O I  = 1, 1O
      DØ 1O K  = 1, 1O
      DØ 1O L  = 1, 1O
   1O A(I,K,L) = C
```

läßt sich effektiver als

```
      DIMENSIØN  A(1O, 1O, 1O), AA(1OOO)
      EQUIVALENCE (A(1,1,1), AA(1))
      DO 1O I = 1, 1OOO
   1O AA(I) = C
```

programmieren.

Andere Operationen wie z. B. Sortieren, Bestimmung des größten bzw. kleinsten Elementes oder der Elemente die Null sind, u.s.w. können auf die gleiche Weise vorgenommen werden.

VI. Einige ergänzende Hinweise

i) Schreibt man überall dort wo eigentlich ein STOP stehen soll ein GOTO zum Ende des Programms, und dort auch statt

```
n STOP
  END        nur   n END,
```

so spart man erstens Übersetzungszeit, und zweitens wird das Programm schneller beendet.

ii) Zur Initialisierung von Variablen sollte immer die DATA-Anweisung an Stelle von Zuweisungen verwendet werden, da die Data-Anweisungen die Zuordnung bereits zur Übersetzungszeit vornimmt, während die andere Art Ausführungszeit benötigt und dies bei jedem Lauf!

VII. Anhang

1. Ergänzungen zur DO-Schleifen-Optimierung

Wie bereits in III. 3 erwähnt, werden DO-Schleifen hochgradig optimiert, wenn der Laufindex im Wiederholungsbereich nicht zur Berechnung herangezogen wird. Um in speziellen Fällen diese Optimierung weiterhin beizubehalten, wurde dort die Einführung einer Zählvariablen vorgeschlagen. Wir geben hier die resultierende Codes der drei verschiedenen Formen der DO-Schleife an.

a) optimierte Form

```
      DO 10 J = 1,100
     [
     [
10   CONTINUE
        ↓
      B   (FESTK.)   1
      SB  (FESTK.) 100
      TRX
     [
     [
     [
     [
     [SZX
```

b) unoptimierte Form

```
      DO 10 J = 1,100
     [K = J * M
     [
10   CONTINUE
        ↓
      B  (FESTK.) 1
      C
     [B   J
     [AA  1
     [C   J
     [ :
     [K = J * M
     [ :
     [SBA 100
     [SKG 0
```

c) verbesserte Form

```
      I = 0
      DO 10 J = 1,100
     [I = I + 1
     [K = I * M
     [ :
10   CONTINUE
        ↓
      BA O
      C
      B   (FESTK.)   1
      SB  (FESTK.) 100
      TRX
     [B   I
     [AA  1
     [C   I
     [ :
     [K = I * M
     [ :
     [SZX
```

[deutet den Wiederholungsbereich an.

Wir sehen, daß bei der verbesserten Form im Wiederholungsbereich die Befehle SBA und SKG durch SZX ersetzt werden, dafür kamen bei der Initialisierung 3 Befehle (aber nur einmal) hinzu. Eine spürbare Verbesserung gegenüber der unoptimierten Form tritt etwa ab 3 - 4 Iterationen ein.

2. Um die Wirksamkeit der beschriebenen Maßnahmen an verschiedenen Programmen deutlich zu machen, nahmen wir uns zum Schluß ein Übungsprogramm eines Studenten, ein Programm der Programmbibliothek (SCALE) und zwei Ausschnitte aus einem Benutzerprogramm.

   2.1. Abb.1 zeigt das Übungsprogramm eines Studenten zur Berechnung von Wochentagen bei Eingabe des Datums. Die folgenden Änderungen wurden vorgenommen:

   a) Die erste READ-Anweisung wurde durch eine DATA-Anweisung ersetzt.

   b) Alle arithmetischen IF-Anweisungen mit nur zwei Ausgängen wurden durch logische IF-Anweisungen ersetzt. Dies bewirkt zusätzlich zu der Laufzeitersparnis eine Verminderung der Übersetzungszeit, da einige Anweisungsnummern fortfallen.

Durch diese sehr einfachen Änderungen wurden die folgenden Verbesserungen erzielt:

	Original	geändertes Programm
Übersetzerzeit	1.0	0.9
Montage	1.0	0.98
Laufzeit	1.0	0.74

2.2. Abb.2 zeigt die Auflistung des Bibliotheksunterprogramms SCALE in den beiden Versionen.

Es wurden die Anweisungen zu Beginn in einer DATA-Anweisung zusammengefaßt, einige arithmetische IF-Anweisungen durch logische IF-Anweisungen ersetzt und einige überflüssige Anweisungen entfernt; dies ergab wie die folgende Tabelle zeigt, eine Verbesserung von ca. 20 %.

	Original	geänderte Version
Anzahl der Maschinen-Befehle	191	140
Laufzeit für ein Feld der Dimension 100	2.4 msec.	2.0 msec.

2.3. Abb. 3 zeigt die Subroutine KURVE in beiden Versionen. Gemeinsame Unterausdrücke über mehrere Anweisungen wurden eleminiert, redundante Anweisungen

(D1, D2, D3, D) entfernt und die wiederholte Division durch D(I) durch Multiplikation mit 1/D(I) ersetzt.

	Original	geänderte Version
Übersetzungszeit	0.83 sec.	0.62 sec.
Laufzeit	4.83 msec.	3.1 msec.

2.4. Abb. 4 zeigt einen Ausschnitt aus dem zugehörigen Hauptprogramm. Hier wurden die Divisionen durch Multiplikation mit dem Reziproken ersetzt, darüber hinaus sind diese Faktoren relative Konstanten bezüglich der DO-Schleifen und können aus diesem Grunde vor die Schleife gezogen werden. Außerdem wurden die Konvertierungen auf eine reduziert. Durch diese Maßnahmen wurde die Laufzeit dieses Programmausschnittes von 8.83 msec. auf 4.02 msec., d.h. auf weniger als die Hälfte reduziert.

3. Zusammenstellung einiger Ausführungsseiten

		µsec.
Multiplikation	(Gleitkomma = GK)	3.375
Multiplikation	(Festkomma = FK)	3.44
Division	(GK)	13.3
Division	(FK)	13.75
Add./Sub.	(GK)	1.75
Add./Sub.	(FK)	0.5
Konvertierung	(FK→GK)	5.9
X ** N	(N ≤ 4)	11.2
FPOWR	(Festkomma Exponentiationen)	72.3
FPOWR+2	(Gleitkomma Exponentiationen)	208.7
SQRT		122.9

```
1500  C  PROGRAMM ZUR BERECHNUNG DER WOCHENTAGE
1510        DIMENSION L(12),M(4),IT(12)
1520        READ(5,100) (L(IB),IB=1,12),(M(IR),IR=1,4),(IT(IB),IB=1,12)
1530     100 FORMAT(16I1,12I2)
1540       1 READ(5,101) IA,IB,IC,ID
1550     101 FORMAT(4I2)
1560        IF(IA) 60,60,2
1570  C    UNTERSUCHUNG AUF EXISTENZ
1580       2 IF(IT(IB) - IA) 80,20,20
1590       4 IF(IB.EQ.1) GOTO 40
1600         IF(IA - 29) 40,80,80
1610  C    UNTERSUCHUNG, OB JANUAR ODER FEBRUAR EINES SCHALTJAHRES
1620      20 MC = MOD(IC,4)
1630         IF(IB.GT.2) GOTO 40
1640         IF(MOD(ID,4)) 21,21,4
1650      21 IF(IC.LT.17) GOTO 33
1660         IF(ID) 22,22,33
1670      22 IF(MC) 33,33,4
1680  C    BERECHNUNG DER JAHRZAHL N
1690      40 N = ID + ID/4
1700         GOTO 10
1710      33 N = ID + ID/4 - 1
1720  C    BERECHNUNG DER JAHRHUNDERTZAHL MJ
1730      10 IF(IC - 16) 11,11,12
1740      11 MJ = 18 - IC
1750         GOTO 44
1760      12 MJ = M(MC+1)
1770  C    BERECHNUNG DER WOCHENTAGE
1780      44 I = IA + L(IB) + MJ + N
1790         J = MOD(I,7) + 1
1800  C  NULLE-ERGÄNZUNG BEIM AUSDRUCK
1810      5 IF(ID-10) 56,51,51
1820      51 WRITE(6,102) IA,IB,IC,ID
1830     102 FORMAT( / 7H   DER ,I2,1H.,I2,1H.,2I2,8H IST EIN)
1840         GOTO (93,94,95,96,97,98,99),J
1850      56 WRITE(6,1022)IA,IB,IC,ID
1860    1022 FORMAT( / 7H   DER ,I2,1H.,I2,1H.,I2,1HO,I1,8H IST EIN)
1870         GOTO (93,94,95,96,97,98,99),J
1880      80 IF(ID - 10)  86,81,81
1890      81 WRITE(6,180) IA,IB,IC,ID
1900     180 FORMAT( / 7H   DEN ,I2,1H.,I2, 1H.,2I2,15H GIBT ES NICHT!)
1910         GOTO 1
1920      86 WRITE(6,1801) IA,IB,IC,ID
1930    1801 FORMAT( / 7H   DEN ,I2,1H.,I2, 1H.,I2,1HO,I1,19H GIBT ES NICHT!)
1940         GOTO 1
1950      93 WRITE(6,103)
1960     103 FORMAT(1H+,25X,8H SONNTAG)
1970         GOTO 1
1980      94 WRITE(6,104)
1990     104 FORMAT(1H+,25X,7H MONTAG  )
2000         GOTO 1
2010      95 WRITE(6,105)
2020     105 FORMAT(1H+,25X,9H DIENSTAG)
2030         GOTO 1
2040      96 WRITE(6,106)
2050     106 FORMAT(1H+,25X,9H MITTWOCH)
2060         GOTO 1
2070      97 WRITE(6,107)
2080     107 FORMAT(1H+,25X,11H DONNERSTAG)
2090         GOTO 1
2100      98 WRITE(6,108)
2110     108 FORMAT(1H+,25X,8H FREITAG)
2120         GOTO 1
2130      99 WRITE(6,109)
2140     109 FORMAT(1H+,25X,8H SAMSTAG)
2150         GOTO 1
2160      60 STOP
2170         END
```

Abb. 1

```
002200  C  PROGRAMM ZUR BERECHNUNG DER WOCHENTAGE
002210        DIMENSION L(12),M(4),IT(12)
002220        DATA L/0,3,3,6,1,4,6,2,5,0,3,5/,M/6,4,2,0/
002230        DATA IT/31,29,31,30,31,30,31,31,30,31,30,31/
002240      1 READ(5,101) IA,IB,IC,ID
002250    101 FORMAT(4I2)
002260        IF(IA.LE.0)GO TO 60
002270        IF(IT(IB).LT.IA)GO TO 80
002280        MC=MOD(IC,4)
002290        IF(IB.GT.2)GO TO 40
002300        IF(MOD(ID,4).GT.0)GO TO 4
002310        IF(IC.LT.17)GO TO 33
002320        IF(ID.GT.0)GO TO 33
002330        IF(MC.LE.0)GO TO 33
002340      4 IF(IB.EQ.1)GO TO 40
002350        IF(IF.GE.29)GO TO 80
002360     40 I=ID+ID/4
002370     10 IF(IC.LE.16)GO TO 11
002380        MJ=M(MC+1)
002390        GO TO 44
002400     33 I=ID+ID/4-1
002410        GO TO 10
002420     11 MJ=18-IC
002430     44 I=IA+L(IB)+MJ+I
002440        J=MOD(I,7)+1
002450     50 IF(ID.LT.10)GO TO 56
002460        WRITE(6,102)IA,IB,IC,ID
002470    102 FORMAT( / 7H   DER ,I2,1H.,I2,1H.,2I2,8H IST EIN)
002480        GOTO (93,94,95,96,97,98,99),J
002490     56 WRITE(6,1022)IA,IB,IC,ID
002500   1022 FORMAT( / 7H   DER ,I2,1H.,I2,1H.,I2,1HO,I1,8H IST EIN)
002510        GOTO (93,94,95,96,97,98,99),J
002520     80 IF(ID.LT.10)GO TO 86
002530        WRITE(6,180)IA,IB,IC,ID
002540    180 FORMAT( / 7H   DEN ,I2,1H.,I2, 1H.,2I2,15H GIBT ES NICHT!)
002550        GOTO 1
002560     86 WRITE(6,1801) IA,IB,IC,ID
002570   1801 FORMAT( / 7H   DEN ,I2,1H.,I2, 1H.,I2,1HO,I1,15H GIBT ES NICHT!
002580        GOTO 1
002590     93 WRITE(6,103)
002600    103 FORMAT(1H+,25X,8H SONNTAG)
002610        GOTO 1
002620     94 WRITE(6,104)
002630    104 FORMAT(1H+,25X,7H MONTAG  )
002640        GOTO 1
002650     95 WRITE(6,105)
002660    105 FORMAT(1H+,25X,9H DIENSTAG)
002670        GOTO 1
002680     96 WRITE(6,106)
002690    106 FORMAT(1H+,25X,9H MITTWOCH)
002700        GOTO 1
002710     97 WRITE(6,107)
002720    107 FORMAT(1H+,25X,11H DONNERSTAG)
002730        GOTO 1
002740     98 WRITE(6,108)
002750    108 FORMAT(1H+,25X,8H FREITAG)
002760        GOTO 1
002770     99 WRITE(6,109)
002780    109 FORMAT(1H+,25X,8H SAMSTAG)
002790        GOTO 1
002800     60 CONTINUE
002810        END
```

Abb. 1 Forts.

31

```
)0130          SUBROUTINE  SCALE   (ARRAY,AXLEN,NPTS,INC)
)0140          DIMENSION  ARRAY(1),SAVE(7)
)0150          SAVE(1)=1.0
)0160          SAVE(2)=2.0
)0170          SAVE(3)=4.0
)0180          SAVE(4)=5.0
)0190          SAVE(5)=8.0
)0200          SAVE(6)=10.0
)0210          SAVE(7)=20.
)0220          FAD=0.01
)0230          K=IABS(INC)
)0240          N=NPTS*K
)0250          YO=ARRAY(1)
)0260          YN=YO
)0270          DO  25  I=1,N,K
)0280          YS=ARRAY(I)
)0290          IF  (YO-YS)  22,22,21
)0300    21    YO=YS
)0310          GO  TO  25
)0320    22    IF  (YS-YN)  25,25,24
)0330    24    YN=YS
)0340    25    CONTINUE
)0350          FIRSTV=YO
)0360          IF  (YO)  34,35,35
)0370    34    FAD=FAD-1.0
)0380    35    DELTAV=(YN-FIRSTV)/AXLEN
)0390          IF (DELTAV) 70,70,40
)0400    40    I=ALOG10(DELTAV)+1000.0
)0410          P=10.0**(I-1000)
)0420          DELTAV=DELTAV/P-0.01
)0430          DO  45  I=1,6
)0440          IS=I
)0450          IF  (SAVE(I)-DELTAV)  45,50,50
)0460    45    CONTINUE
)0470    50    DELTAV=SAVE(IS)*P
)0480          FIRSTV=DELTAV*AINT(YO/DELTAV+FAD)
)0490          T=FIRSTV+(AXLEN+0.01)*DELTAV
)0500          IF (T-YN)  55,57,57
)0510    55    IS=IS+1
)0520          GO  TO  50
)0530    57    FIRSTV=FIRSTV-AINT((AXLEN+(FIRSTV-YN)/DELTAV)/2.0)*DELTAV
)0540          IF (YO*FIRSTV) 58,58,59
)0550    58    FIRSTV=0.0
)0560    59    IF  (INC) 61,61,65
)0570    61    FIRSTV=FIRSTV+AXLEN*DELTAV
)0580          DELTAV=-DELTAV
)0590    65    N=N+1
)0600          ARRAY(N)=FIRSTV
)0610          N=N+K
)0620          ARRAY(N)=DELTAV
)0630    67    RETURN
)0640    70    DELTAV=1.0
)0650          FIRSTV=FIRSTV-0.5
)0660          GO TO 65
)0670          END
```

Abb. 2

32

```
002930          SUBROUTINE SCALE1 (ARRAY,AXLEN,NPTS,INC)
002940          DIMENSION ARRAY(1),SAVE(7)
002950          DATA SAVE/1.0,2.0,4.0,5.0,8.0,10.0,20.0/,FAD/0.01/
002960
002970
002980
002990
003000
003010
003020
003030          K=IABS(INC)
003040          N=K*NPTS
003050          YU=ARRAY(1)
003060          YN=YU
003070          DO 25 I=1,N,K
003080          YS=ARRAY(I)
003090          IF(YS.GE.YU)GO TO 22
003100          YU=YS
003110          GO TO 25
003120       22 IF(YS.GT.YN) YN=YS
003130
003140       25 CONTINUE
003150          FIRSTV=YU
003160          IF(YU.GE.0.0)GO TO 35
003170          FAD=FAD-1.0
003180       35 DELTAV=(YN-FIRSTV)/AXLEN
003190          IF(DELTAV.LE.0.0)GO TO 70
003200
003210          P=10.0**IFIX(ALOG10(DELTAV))
003220          DELTAV=DELTAV/P+0.01
003230          DO 45 I=1,6
003240          IS=I
003250          IF(DELTAV.LE.SAVE(I))GO TO 50
003260       45 CONTINUE
003270       50 DELTAV=SAVE(IS)*P
003280          FIRSTV=DELTAV*AINT(YU/DELTAV*FAD)
003290
003300          IF(FIRSTV+(AXLEN+0.01)*DELTAV.GE.YN)GO TO 57
003310          IS=IS+1
003320          GO TO 50
003330       57 FIRSTV=FIRSTV-AINT((AXLEN+(FIRSTV-YN)/DELTAV)/2.0)*DELTAV
003340          IF(FIRSTV*YU.GT.0.0)GO TO 59
003350          FIRSTV=0.0
003360       59 IF(INC.GT.0)GO TO 65
003370          FIRSTV=FIRSTV+AXLEN*DELTAV
003380          DELTAV=-DELTAV
003390       65 I=I+1
003400          ARRAY(I)=FIRSTV
003410          I=I+K
003420          ARRAY(I)=DELTAV
003430       67 RETURN
003440       70 DELTAV=1.0
003450          FIRSTV=FIRSTV-0.5
003460          GO TO 65
003470          END
```

Abb. 2 Forts.

```
      SUBROUTINE  KURVE
      DIMENSION  Y1(10),Y2(10),Y3(10),W1(10),W2(10),AK(10),BK(10),CK(10)
     1,II(11)
      DIMENSION  D(10),D1(10),D2(10),D3(10)
      COMMON  /CB/ NRI,Y1,Y2,Y3,W1,W2,AK,BK,CK,NKK,DX,II,NZ
C     DARSTELLUNG DER GESCHWINDIGKEITSVERTEILUNG MIT HILFE EINES POLYNOMS
C     3. ORDNUNG
      DO 10  I=1,NRI
      D(I)=Y1(I)**3*(Y2(I)**2*Y3(I)-Y2(I)*Y3(I)**2)-Y2(I)**3*(Y1(I)**2*Y
     13(I)-Y1(I)*Y3(I)**2)+Y3(I)**3*(Y1(I)**2*Y2(I)-Y1(I)*Y2(I)**2)
      D1(I)=W1(I)*(Y2(I)**2*Y3(I)-Y2(I)*Y3(I)**2)-W2(I)*(Y1(I)**2*Y3(I)-
     1Y1(I)*Y3(I)**2)
      D2(I)=-W1(I)*(Y2(I)**3*Y3(I)-Y2(I)*Y3(I)**3)+W2(I)*(Y1(I)**3*Y3(I)
     1-Y1(I)*Y3(I)**3)
      D3(I)=W1(I)*(Y2(I)**3*Y3(I)**2-Y2(I)**2*Y3(I)**3)-W2(I)*(Y1(I)**3*
     1Y3(I)**2-Y1(I)**2*Y3(I)**3)
      AK(I)=D1(I)/D(I)
      BK(I)=D2(I)/D(I)
      CK(I)=D3(I)/D(I)
   10 CONTINUE
      RETURN
      END

      SUBROUTINE  KURVE
      DIMENSION  Y1(10),Y2(10),Y3(10),W1(10),W2(10),AK(10),BK(10),CK(10)
     1,II(11)
      COMMON  /CB/ NRI,Y1,Y2,Y3,W1,W2,AK,BK,CK,NKK,DX,II,NZ
C     DARSTELLUNG DER GESCHWINDIGKEITSVERTEILUNG MIT HILFE EINES POLYNOMS
C     3. ORDNUNG
      DO 10  I=1,NRI
      Y11=Y1(I)
      Y12=Y11**2
      Y13=Y12*Y11
      Y21=Y2(I)
      Y22=Y21**2
      Y23=Y22*Y21
      Y31=Y3(I)
      Y32=Y31**2
      Y33=Y32*Y31
      TL=Y22*Y31-Y21*Y32
      TM=Y12*Y31-Y11*Y32
      RD=1./(TL*Y13-TM*Y23+Y33*(Y12*Y21-Y11*Y22))
      AK(I)=(W1(I)*TL-W2(I)*TM)*RD
      BK(I)=-(W1(I)*(Y23*Y31-Y21*Y33)+W2(I)*(Y13*Y31-Y11*Y33))*RD
      CK(I)=(W1(I)*(Y23*Y32-Y22*Y33)-W2(I)*(Y13*Y32-Y12*Y33))*RD
   10 CONTINUE
      RETURN
      END
```

Abb. 3

```
00400C          T(J)=UT*(NW(LT1)+NW(LT2))+T(J)
00401C          LM1=55+NP6
00402C          LM2=LM1+55
00403C          MG(J)=UM*(NW(LM1)+NW(LM2))+MG(J)
00404C      120 CONTINUE
00406C          DO 330 J=1,NPJ
00407C          PDYE(J) = PDYE(J)/NMR
00408C          PSTE(J) = PSTE(J)/NMR
00409C          PDYA(J) = PDYA(J)/NMR
00410C          PSTA(J) = PSTA(J)/NMR
00411C          T(J) = T(J)/(2*NMR)
00412C          MG(J) = MG(J)/(2*NMR)
00413C          DO 330 K=1,NPK
00414C          DO 330 I=1,NPI
00415C          PDY(I,K,J) = PDY(I,K,J)/NMR
00416C          PST(I,K,J) = PST(I,K,J)/NMR
00417C      330 CONTINUE
00418C
00419C  C       BERECHNUNG DES GESCHWINDIGKEITSFELDES
00420C          CALL  BERECH (RUE,RHO,NM)
00421C          RE = RHO*UR/VISE
00422C          P15=2.0/RHO
```

```
00428C          T(J)=UT*(NW(LT1)+NW(LT2))+T(J)
00429C          LM1=55+NP6
00430C          LM2=LM1+55
00431C          MG(J)=UM*(NW(LM1)+NW(LM2))+MG(J)
00432C      120 CONTINUE
00433C
00434C          RHO=1.0/ NMR
00435C          RHO2=RHO*0.5
00436C          DO 330 J=1,NPJ
00437C          PDYE(J)=PDYE(J)*RHO
00438C          PSTE(J)=PSTE(J)*RHO
00439C          PDYA(J)=PDYA(J)*RHO
00440C          PSTA(J)=PSTA(J)*RHO
00441C          T(J)=T(J)*RHO2
00442C          MG(J)=MG(J)*LM2
00443C          DO 330 K=1,NPK
00444C          DO 330 I=1,NPI
00445C          PDY(I,K,J)=PDY(I,K,J)*RHO
00446C          PST(I,K,J)=PST(I,K,J)*RHO
00447C      330 CONTINUE
00448C
00449C  C       BERECHNUNG DES GESCHWINDIGKEITSFELDES
00450C          CALL  BERECH ( RUE,RHO,NM)
00451C          RE = RHO*UR/VISE
00452C          P15=2.0/RHO
```

Abb. 4

Literatur:

[1] D. Knuth

An Empirical Study of FORTRAN Programs
Stanford University, Computer Science Depertment
Report CS - 186

[2] Chr. Larson

The Efficient Use of FORTRAN
Datamation August 1, 1971

[3] P. B. Schneck and E. Angel

A FORTRAN to FORTRAN Optimising Compiler
The Computer Journal Vol 16 Number 4

Bisher erschienene Arbeitsberichte des Rechenzentrums

der Ruhr-Universität Bochum

Nr. 7101: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA; eine Dialogsprache für den TR 440 (vergriffen)

Nr. 7102: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, ein Dialogsystem und seine Implementierung in ALGOL (vergriffen)

Nr. 7103: K.-H. Mohn, M. Rosendahl, H. Zoller
AIDA, Manual für den Benutzer (vergriffen)

Nr. 7104: 4. Jahresbericht des Rechenzentrums (Juni 1970 bis Juni 1971)

Nr. 7105: H. Wupper
WR MB02 - Ein einfaches Band-Betriebssystem für einen mittleren Rechner

Nr. 7201: H. Windauer
Existenzsätze zur (0,1,..,R-2,R) - Interpolation

Nr. 7202: W. Schelongowski
DIATRACE, Ein System zur interaktiven Assemblerprogrammierung

Nr. 7203: M.Jäger, M. Rosendahl, R. Staake
Einführung in die Listenverarbeitung anhand der Dialogsprache AIDA

Nr. 7204: R. Mannshardt, P. Pottinger
Einführung in die Benutzung des Teilnehmer-Rechensystems TR 440 in der RUB (vergriffen)

Nr. 7205: 5. Jahresbericht des Rechenzentrums (1.7.1971 bis 30.6.1972)

Nr. 7206: M. Rosendahl
BOGOL-IAS, ein Weg zur systemnahen Programmierung in ALGOL am TR 440

Nr. 7207: W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungswechsels von Verbrennungskraftmaschinen (Modulbeschreibung und Eingabekonventionen)

Nr. 7208: W. Stark
ILW, Programmsystem zur Berechnung des Instationären Ladungswechsels von Verbrennungskraftmaschinen (Regelmechanismus und Berechnung der Rohrströmung)

Nr. 7209: H. Ehlich
Anregung und Kritik zum Betriebs- und Programmiersystem der TR 440

Nr. 7210: M. Rosendahl
BOGOL-STRING, eine flexible Zeichenkettenverarbeitung in ALGOL 60

Nr. 7211: H.Camici, H. Claus, H. Ehlich, D. Kipp
Arbeitsbericht über ein Programm zur Haushaltsführung

Nr. 7301: R. Mannshardt, K.-H. Mohn, H.J. Münch, P. Pottinger
Einführung in die Benutzung des Teilnehmer Rechensystems TR 440
2. geänderte Auflage (vergriffen)

Nr. 7302: K.-H. Mohn
Über einige Anwendungen des Computers in der Medizin

Nr. 7303: R. Buchmann
BODAT,ein schnelles und platzsparendes System zur Datenmanipulation und -speicherung in ALGOL 60 und FORTRAN

Nr. 7304: M. Hauenschild
Ansätze zur komplexen Kreisarithmetik

Nr. 7305: R. Buchmann
RB&QUELLHALT, ein TR440-Datenbanksystem zur platzsparenden Quellhaltung auf Datenträgern mit direktem Zugriff (LFD,WSP)

Nr. 7306: 6. Jahresbericht des Rechenzentrums (1.7.1972 bis 31.12.1973)

Nr. 7401: R. Buchmann
Der Systemoperator BO&BS3QP
Messungen und Steuerungen des Betriebssystems auf Operatorebene

Nr. 7402: R. Mannshardt
Herleitung und Prüfung spezieller Runge-Kutta-Verfahren mit impliziten Rechenschritt

Nr. 7403: R.Buchmann, H. Wupper
Unzulänglichkeiten des TR 440 Programmiersystems und ihre Umgehung

Nr. 7404: R.Green, K.-H.Mohn
Quellbezogene FORTRAN Optimierungen für den Compiler des TR 440