Rolf-Dieter Klein, Tobias Thiel

PC PAR 68000, ein Parallelrechner für PC

Teil 2: Die Software

Im ersten Teil wurde die Hardware zum PC PAR 68000 besprochen, der in jeden kompatiblen PC eingesetzt werden kann. Hier sollen nun mit ein paar Beispielen die praktischen Einsatzmöglichkeiten von PC PAR gezeigt werden. Es könnte die geeignete Basis für einen Superrechner sein.

Schnelle Programme werden am effizientesten in Assemblercode erstellt. Dazu gibt es einen Cross Assembler (ASM68K) für den Parallelrechner. Dieser Assembler besitzt einige Besonderheiten. Es ist möglich, Include-Dateien für höhere Programmiersprachen zu erzeugen. Darin müssen die gemeinsam genutzten Variablen verzeichnet sein. Will man nämlich vom PC aus auf Variable im Parallelrechner zugreifen, so muß man deren Adresse kennen. Diese Adressen weist der Assembler normalerweise automatisch zu, falls man nicht von Hand im Assemblerprogramm mit ORG-Befehlen Speicherorte festlegt. Der PC PAR-Assembler kann diese Adressen an die höhere Programmiersprache weitermelden, so daß man dort mit symbolischen Namen arbeiten kann und nicht auf absolute Adressen angewiesen ist.

PC PAR und Pascal

Bild 1 zeigt den Ablauf beim Programmieren in Pascal. Das Parallelrechnerprogramm hat den Namen „name.a68“ und das MS-DOS-Programm, das in Pascal geschrieben ist, den Namen „name.pas“. Mit dem Assembler AMS68K wird zunächst das Assemblerprogramm übersetzt. Der Schalter -L gibt an, daß zusätzlich eine Listing-Datei erzeugt wird, was aber für den weiteren Ablauf nicht wichtig ist. Der Schalter -p gibt an, daß eine Include-Datei für Pascal erzeugt werden soll. Die Datei „name.p68“, die den Objekt-Code enthält, wird ebenfalls erzeugt. Nun kann man das steuernde Pascal-Programm übersetzen, zum Beispiel von der integrierten Turbo-Pascal-Umgebung aus, bei der man auch im Einzelschritt testen kann. Wenn man die integrierte Umgebung verwendet, muß man allerdings erst das Parallelrechnerprogramm in die Parallelrechner laden (man kann das Programm auch direkt von Turbo-Pascal aus laden). Ein besonderes Programm „LOADER“ ist dafür Voraussetzung. Das Datenformat der Objekt-Datei ist dazu besonders aufgebaut. Die ersten beiden Byte in der Datei stellen eine Kennung dar, die zur Überprüfung vom Steuerprogramm aus dient. Dann folgen einzelne Records, die wie folgt aufgebaut sind:
Type.Byte: Type = 1 kündigt Daten an und es folgen die Ladeadresse und die Anzahl der einzulesenden Byte sowie die Byte selbst: Adresse.Longint Anzahl.Word (Daten....).

Type = 0 bedeutet Dateiende.

Beispiel:

89 14	Header
01	Type = 1
78 56 34 12	Adresse = $12345678
05 00	Anzahl = 5
0A 0B 0C 0D 0E	Daten
01	Type = 1
00 80 00 00	Adresse = $8000
02 00	Anzahl = 2
FF 55	Daten
00	Type = 0, Ende erreicht.

Die Werte sind binär in der Datei gespeichert. Nach der Endekennung folgen Vermerke über Assemblerversion und Copyright. LOADER kann auch die Header von CPM/68K oder dem Aztec-Cross-C-Compiler verarbeiten.

PC PAR und C

Bild 2 zeigt den Ablauf. Diesmal wird eine Datei „name.h“ erzeugt, die man per #include-Befehl im C-Programm verwenden kann. Es gibt einen Make-Befehl, der nach den Angaben einer entsprechenden Make-Datei ein lauffähiges Programm erarbeitet. Eine Make-Datei sähe dann zum Beispiel wie folgt aus:

```
#Makedatei

name.exe: name.pas name.inc
        tpc name.pas /m
name.inc: name.a68
asm68k name.a68 -p
```

Die Make-Datei besagt: Das Programm „name.exe“ ist abhängig von „name.pas“, also der Pascal-Quelle und „name.inc“, der Include-Datei. Um „name.exe“ zu erhalten wird der Befehl „tpc ...“ ausgeführt. „name.inc“ ist wiederum abhängig von „name.a68“. Um „name.inc“ zu erhalten wird „asm68k ...“ ausgeführt. Der Assembler erzeugt auch „name.p68“, diese Datei kann dann mit dem Befehl „loader name.p68“ in die Parallelrechner geladen werden.

Bild 3 zeigt ein Assemblerprogramm für PC PAR. Das Programm zählt die Speicherzelle alpha fortlaufend um 1 hoch. Die Zelle soll in der LSB...MSB-Form vom PC aus direkt lesbar sein. Außerdem soll das Ausgabeport abwechselnd mit den Werten $55 und $AA geladen werden.

Am Anfang des Programms stehen die Befehle SHARED ... Damit werden Variable oder Konstante als gemeinsam für den PC und Parallelrechner deklariert. Diese Variablen werden vom Assembler in die Include-Datei eingebaut. Danach folgen die Befehle dc.l stack und dc.l start. Nach einem Reset-Befehl holt sich die CPU 68000 zuerst aus den ersten beiden Langworten den Stackpointer und die Startadresse. Daher muß diese Information hier abgelegt sein. Der Zähler von alpha wird in dem Register D0.L gehalten und zunächst auf 0 gesetzt. Mit den Befehlen ror, swap und ror wird in Register D1.L der gedrehte LSB..MSB-Wert vorbereitet und mit dem move.l-Befehl in

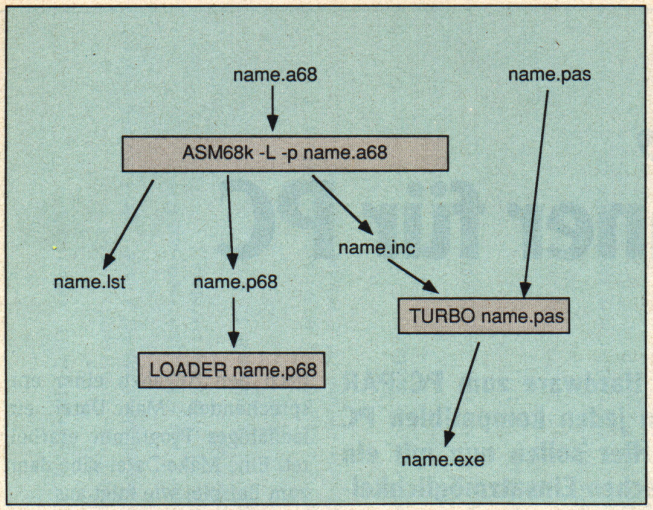

Bild 1. Ablauf zur Erstellung der Parallelrechnerprogramme in Pascal

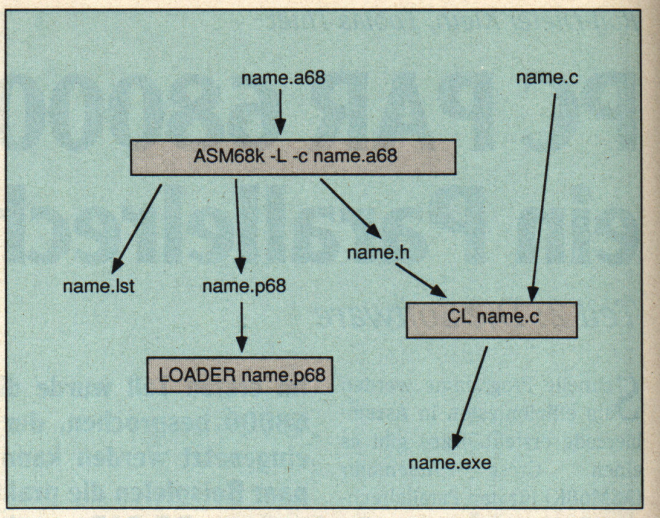

Bild 2. Ablauf zur Erstellung der Parallelrechnerprogramme in C

alpha abgelegt. Die Adresse $FFFFFFFF entspricht dem IO-Port des Parallelrechners, in dem nacheinander die Werte $55 und $AA abgelegt werden. Danach folgt der Sprung zurück zur Marke „schleife".

Die Variable alpha ist mit „alpha: dc.l 0" deklariert. Danach folgt noch eine sogenannte Signatur-Variable. Dort steht der Wert $AA14. Eine solche Signatur kann der MS-DOS-Rechner später abfragen und feststellen, ob sich das richige Programm im richtigen Parallelrechner befindet und wo überhaupt Parallelrechner etwas tun.

Bild 4 zeigt die erzeugte Include-Datei. Dort sind die Adresse von „alpha", hier $2C und „signatur", hier $30 eingetragen. In Bild 5 ist das Pascal-Programm abgedruckt. Mit (*$I treff.inc*) wird die Include-Datei geladen. Die Include-Datei heißt hier Treff.inc, da das Parallelrechnerprogramm „treff.a68" hieß. Der Speicher des Parallelrechners wird auf Adresse $D000:0000 bis $D000:$FFFF eingeblendet. Turbo-Pascal erlaubt es auf elegante Weise, Variablen auf absolute Speicheradressen zu legen. So wird die Variable „alphavar" auf die Speicheradresse $D000:alpha gelegt. Der Wert von „alpha" stammt aus der Include-Datei. Anstelle von

```
PCPAR68000 Assembler V 1.3 , (C) 1989 Rolf-Dieter Klein, Tobias Thiel
PASS 1
PASS 2
000000:                 ; Kleines Testprogramm fuer Parallelrechner PCPAR68000
000000:                 ; Rolf-Dieter Klein, Tobias Thiel
000000:                  shared alpha    ; fuer Pascal Hauptprogramm gemeinsame Variable
000000:                  shared signatur ; SignaturZelle fuer Test, Rechner da und geladen.
000000: 00000fd2         dc.l stack      ; StackPointer          RESET VEKTOR
000004: 00000008         dc.l start      ; Start des Programms   FUER CPUSTART
000008:                 start:
000008: 7000             moveq.l #0,d0          ; Variable loeschen
00000a:                 schleife:
00000a: 2200             move.l d0,d1           ; in Hilfsregister d1 laden.
00000c: e059             ror.w #8,d1            ; 68000: msb ... lsb
00000e: 4841             swap d1                ; 8086er: lsb .. msb
000010: e059             ror.w #8,d1            ; daher umdrehen noetig
000012: 23c1 0000002c    move.l d1,alpha        ; dann Variable an PC-shared alpha
000018: 5280             addq.l #1,d0           ; Variable hochzaehlen
00001a: 13fc 0055        move.b #$55,$ffffffff ; Statusport mit alternierender
00001e: ffffffff
000022: 13fc 00aa        move.b #$aa,$ffffffff ; Sequenz fuer Test belegen.
000026: ffffffff
00002a: 60de             bra.s schleife                ; Endlos wiederholen
00002c:                 alpha:                 ; gemeinsame Variable
00002c: 00000000         dc.l 0                        ; Variable
000030:                 signatur:              ; gemeinsame Variable
000030: aa14             dc.w $AA14             ; Testwert in PC: 14AA in 68000 AA14.
000032:                                         ; ggf Stackbereich
000032:                  ds 2000                ; wird hier nicht benoetigt.
000fd2: 00000000        stack: dc.l 0           ; Stack beginnt hier, und geht nach unten
000fd6:                  end
------ Symbol Table -------
Symbol:           alpha Wert: 0000002c Art: 374
Symbol:        schleife Wert: 0000000a Art: 374
Symbol:        signatur Wert: 00000030 Art: 374
Symbol:           stack Wert: 00000fd2 Art: 374
Symbol:           start Wert: 00000008 Art: 374
------ End of Compilation
```

Bild 3. Assemblerprogramm für einfachen Test

```
(* Header File for CONST - section in PASCAL *)
 alpha = $2c;
 signatur = $30;
(* END Header File PASCAL *)
```

Bild 4. Erzeugte Include-Datei

```
program testpcpar;

uses
 CRT;

const
 segment = $D000;     (* Fenster fuer Parallelrechner      *)
 (*$I treff.inc*)     (* Includefile mit shared Variablen *)

var
 alphavar : LongInt absolute segment:alpha;       (* shared Variable *)
 signaturvar : Integer absolute segment:signatur; (* shared Variable *)
 i : Integer;  (* fuer SChleifenzaehler *)

begin
 clrscr;
 for i:=0 to 31 do Port[$300+i] := 0; (* ausblenden und ggf. starten *)
 repeat
  gotoxy(1,1);                (* Bildschirm links oben *)
  for i:=0 to 31 do begin     (* Alle Parallelrechner ausgeben *)
   Port[$300+i] := 2;         (* einblenden des Speicherbereichs *)
   if signaturvar = $14AA then    (* Nur wenn Rechner verfuegbar *)
    writeln('Alpha=',alphavar:15,' Port=',Port[$300+i],'    ');
                              (* direkter Zugriff, Wert ausgeben *)
   Port[$300+i] := 0;         (* weiterrechnen lassen und ausblenden *)
  end;
 until keypressed;
 for i:=0 to 31 do Port[$300+i] := 1; (* Rechner Stopp, ausblenden *)
end.
```

Bild 5. Pascal-Programm für Beispiel auf dem PC

```
(* ****************************************** *)
(* Mandelbrotmenge mit 8086  Parallelrechner *)
(* Rolf-Dieter Klein, Tobias Thiel           *)
(* V 1.0 890424  in Turbo-Pascal             *)
(* ****************************************** *)

program  mandel;

uses Crt,Graph;

const
 MAXX = 320;
 MAXY = 200;
 MAXTIEFE = 32;

 YMIN = -1200;
 YMAX = 1200;
 XMIN = -1700;
 XMAX = 650;
 NORM = 1000; (* Normierungsfaktor *)

var
 GraphMode, GraphDriver : Integer;
 GraphError : Integer;
 nx,ny : integer;

function iterate(nx,ny : integer) : integer;

var tiefe : integer;
 x,y,xp,yp,xalt : longint;

begin
 xp := ((XMAX - XMIN) * longint(nx)) DIV (MAXX - 1) + XMIN;
 yp := ((YMAX - YMIN) * longint(ny)) DIV (MAXY - 1) + YMIN;
 tiefe := 0;
 x := 0;
 y := 0;
 while (tiefe < MAXTIEFE) and ((x * x + y * y) < 40000000) do begin
  xalt := x;
  x := (x * x - y * y) DIV NORM + xp;
  y := (2 * xalt * y) DIV NORM + yp;
  tiefe := tiefe + 1;
 end;
 iterate := tiefe;
end;


begin
 GraphDriver := EGA;
 GraphMode := EGAHi;
 InitGraph(GraphDriver,GraphMode,'\TURBO');
 If GraphResult <> 0 then begin
  writeln('EGA Mode nicht gefunden');
  Halt;
 end;
 nx := 0;
 repeat
  for ny := 1 to MAXY-1 do begin
   PutPixel(nx,ny,iterate(nx,ny) mod 16);
  end;
  nx := nx + 1;
 until keypressed or (nx >= MAXX);
 repeat
 until keypressed;
 CloseGraph;
end.
```

Bild 6. Mandelbrot, PC-Referenzprogramm

$D000 steht im Listing die Konstante „segment", damit wird das Programm übersichtlicher. Das Pascal-Programm blendet zunächst alle Parallelrechner aus und startet diese. Dabei wird hier nicht gefragt, ob auf der betreffenden Adresse ein Parallelrechner vorhanden ist oder nicht. Der Bereich $300..$31F ist hier einmal für die Rechner vorgesehen. In der Hauptschleife wird nun je ein Rechner eingeblendet. Dies geschieht durch die Zuweisung Port [$300+i] := 2. Der Parallelrechner arbeitet dabei weiter. Ist ein Parallelrechner auf der betreffenden Adresse vorhanden, so hat die Signaturvariable den Wert $14AA. Nur dann ist das Parallelrechnerprogramm geladen und der Parallelrechner auf der abgefragten Port-Adresse auch vorhanden. Der aktuelle Wert der Variablen alphavar (also der Variablen „alpha" im Parallelrechner) wird dann mit einer writeln-Anweisung direkt ausgegeben. Bei der Programmierung dieses Testprogrammes wurde auf Semaphoren verzichtet. So kann es sein, daß der PC, der alpha in zwei Schritten liest, einen Wert ermittelt, der sich aus zwei verschiedenen Zählerständen ergibt, weil der 68000 zwischen den PC-Zugriffen den Zähler verändert hat.

Die Mandelbrotmenge

Die Mandelbrotmenge ist zur Demonstration von Rechenkraft immer wieder sehr beliebt. *Bild 6* zeigt eine Integer-Version, die auf dem PC allein abläuft. Sie dient nur als Referenz für Zeitmessungen. Um hohe Genauigkeit zu erhalten, wird im Programm teilweise auch mit Longint gearbeitet. Die Zahlen sind auf den Wert 1000 normiert (das heißt, daß zum Beispiel 1.2 als Integerzahl 1200 dargestellt wird). Es wird hier der EGA-Modus verlangt, aber nur eine Fläche von 320x200 Punkten ausgegeben. Sie können das Programm somit leicht auf CGA-Auflösung umstellen. *Bild 7* zeigt das in 68000-Assembler übersetzte Programm. Hier wurden zur Abwechslung einmal keine Shared-Variablen verwendet. Die Grenzwerte sind als Konstanten direkt eingetragen, man hätte sie auch über Shared Variablen vom PC-Programm aus übertragen können. Das Programm rechnet immer eine Spalte der Mandelbrotmenge aus. Dazu wird in der Variablen „spalte", die Nummer der Spalte, also die x-Koordinate eingegeben. Der Parallelrechner wird dann gestartet. Wenn er fertig ist, erscheint am Port des Parallelrechners der Wert $ff. Das Hauptprogramm im PC kann daraufhin den PC PAR ins Speicherfenster einblenden und ab der Variablen „ergebnis" die Werteliste für die Helligkeitsstufen auslesen. Danach kann der Parallelrechner mit einer neuen Spalte gestartet werden. *Bild 8* zeigt ein GW-Basic-Programm dazu, das Parallelrechner verwendet. Im Parallelrechnerprogramm wurde mit der ORG-Anweisung dafür gesorgt, daß die gemeinsamen Variablen ab Adresse $8000, also im PC ab $D800:0000 beginnen. Daher wird dort die Variable SEG auf $D800 gesetzt. Man kann dann mit den Befehlen PEEK und POKE zugreifen. Die Ausgabe der Bildpunkte (Zeilen 130-150) geschieht langsam. Die Mandelbrotrechnung ist im Verhältnis dazu sehr schnell durchgeführt. *Bild 9* zeigt ein Programm in Turbo-Pascal, das mehrere Parallelrechner verwendet. Das Programm erkennt automatisch, wieviele Parallelrechner vorhanden sind. Der Bildschirmbereich wird in etwa gleich große Streifen eingeteilt und jeder Parallelrechner bekommt eine Spalte am Anfang dieses Bereichs zugeteilt. In dem Feld „arra" ist verzeichnet, ob eine betreffende Spalte gerade berechnet wird, oder noch nicht, oder schon berechnet ist, damit Parallelrechner, die ihren Bereich durchgerechnet haben, anderen Kollegen, die noch nicht so weit sind, helfen können. Einem solchen Parallelrechner wird nämlich einfach immer die nächste freie Spalte zugewiesen. Wenn ein Rechner mit seiner Spalte fertig ist, so liefert er den Wert $FF an seinem Port. Es wird dann die Spalteninformation in das Feld „linbuf" eingelesen, und der Rechner bekommt dann sofort die nächste Spalte zugewiesen. Erst dann wird mit der Ausgabe der Spalte begonnen. Diese Vorgehensweise ist wichtig, um einen möglichst überlappenden Betrieb zu erzielen. Die Parallelrechner sollen möglichst immer beschäftigt sein. Dennoch kann es passieren, daß der nächste Parallelrechner auf die Ausgabe des Hauptrechners

```
PCPAR68000 Assembler V 1.3 , (C) 1989 Rolf-Dieter Klein, Tobias Thiel
PASS 1
PASS 2
000000:                 ; ASM68 Mandelbrotmenge Programm fuer PCPAR 68000
000000:                  org 0
000000: 000007d0         dc.l 2000
000004: 00000008         dc.l start
000008:  = 00000140     MAXX equ 320
000008:  = 000000c8     MAXY equ 200
000008:  = 00000020     tiefe equ 32
000008:  = fffffb50     ymin equ -1200
000008:  = 000004b0     ymax equ 1200
000008:  = fffff95c     xmin equ -1700
000008:  = 0000028a     xmax equ 650
000008:                 start:
000008: 13fc 0000        move.b #0,$ffffffff
00000c: ffffffff
000010: 2239 00008000    move.l spalte,d1
000016: 41f9 00008004    lea ergebnis,a0
00001c:                 lp2:
00001c: 243c 00000000    move.l #0,d2
000022:                 lp1:
000022: 48e7 6000        movem.l d1-d2,-(a7)
000026: 203c 000004b0    move.l #ymax,d0
00002c: 90bc fffffb50    sub.l #ymin,d0
000032: c1c2             muls d2,d0
000034: 81fc 00c7        divs #MAXY-1,d0
000038: 48c0             ext.l d0
00003a: d0bc fffffb50    add.l #ymin,d0
000040: 2400             move.l d0,d2
000042: 203c 0000028a    move.l #xmax,d0
000048: 90bc fffff95c    sub.l #xmin,d0
00004e: c1c1             muls d1,d0
000050: 81fc 013f        divs #MAXX-1,d0
000054: 48c0             ext.l d0
000056: d0bc fffff95c    add.l #xmin,d0
00005c: 2200             move.l d0,d1
00005e: 6100 0024        bsr mandel
000062: 4cdf 0006        movem.l (a7)+,d1-d2
000066: 20c0             move.l d0,(a0)+
000068:                 skip:
000068: d4bc 00000001    add.l #1,d2
00006e: b4bc 000000c7    cmp.l #MAXY-1,d2
000074: 6600 ffac        bne lp1
000078: 13fc 00ff        move.b #$ff,$ffffffff
00007c: ffffffff
000080:                 halt:
000080: 6000 fffe        bra halt
000084:                 mandel:
000084: 4283             clr.l d3
000086: 4284             clr.l d4
000088: 4285             clr.l d5
00008a: 4286             clr.l d6
00008c: 4287             clr.l d7
00008e:                 loop:
00008e: bebc 00000020    cmp.l #tiefe,d7
000094: 6400 003c        bcc eman
000098: 2005             move.l d5,d0
00009a: d086             add.l d6,d0
00009c: b0bc 003d0900    cmp.l #4000000,d0
0000a2: 6400 002e        bcc eman
0000a6: 2005             move.l d5,d0
0000a8: 9086             sub.l d6,d0
0000aa: 81fc 03e8        divs #1000,d0
0000ae: 48c0             ext.l d0
0000b0: d081             add.l d1,d0
0000b2: c9c3             muls d3,d4
0000b4: 2c02             move.l d2,d6
0000b6: cdfc 03e8        muls #1000,d6
0000ba: d884             add.l d4,d4
0000bc: d886             add.l d6,d4
0000be: 89fc 03e8        divs #1000,d4
0000c2: 2600             move.l d0,d3
0000c4: 2a03             move.l d3,d5
0000c6: cbc3             muls d3,d5
0000c8: 2c04             move.l d4,d6
0000ca: cdc4             muls d4,d6
0000cc: 5287             addq.l #1,d7
0000ce: 6000 ffbe        bra loop
0000d2:                 eman:
0000d2: 2007             move.l d7,d0
0000d4: 4e75             rts
008000:                  org $8000
008000:                 spalte:
008000: 00000140         dc.l MAXX
008004:                 ergebnis:
008004: 00000000         dc.l 0,0,0,0,0,0,0,0,0,0,0
008008: 00000000
00800c: 00000000
008010: 00000000
008014: 00000000
008018: 00000000
00801c: 00000000
008020: 00000000
008024: 00000000
008028: 00000000
00802c: 00000000
008030:                          end
------ Symbol Table -------
Symbol:              eman Wert: 000000d2 Art: 374
Symbol:          ergebnis Wert: 00008004 Art: 374
Symbol:              halt Wert: 00000080 Art: 374
Symbol:              loop Wert: 0000008e Art: 374
Symbol:               lp1 Wert: 00000022 Art: 374
Symbol:               lp2 Wert: 0000001c Art: 374
Symbol:            mandel Wert: 00000084 Art: 374
Symbol:              maxx Wert: 00000140 Art: 288
Symbol:              maxy Wert: 000000c8 Art: 288
Symbol:              skip Wert: 00000068 Art: 374
Symbol:            spalte Wert: 00008000 Art: 374
Symbol:             start Wert: 00000008 Art: 374
Symbol:             tiefe Wert: 00000020 Art: 288
Symbol:              xmax Wert: 0000028a Art: 288
Symbol:              xmin Wert: fffff95c Art: 288
Symbol:              ymax Wert: 000004b0 Art: 288
Symbol:              ymin Wert: fffffb50 Art: 288
------ End of Compilation
```

Bild 7. Assemblerprogramm für Mandelbrotberechnung

```
10 DEF SEG=&HD800
20 SCREEN 9
30 CLS
40 X = 320
50 KEY OFF
60 REM start
70 POKE(3),X AND 255: POKE (2),INT(X / 256)
80 OUT &H300,2
90 WHILE INP(&H300) <> 255
100 WEND
110 OUT &H300,3
120 IF X < 0 OR X > 639 THEN PRINT "x=";X:STOP
130 FOR Y=1 TO 199
140 PSET (X,Y),PEEK(Y * 4 + 3) MOD 16
150 NEXT Y
160 X = X - 1
170 IF X > 0 THEN 60
180 WHILE INKEY$<>" ": WEND
190 SCREEN 0
```

Bild 8. Beispiel eines Basic-Programms zur Parallelrechner-ansteuerung

warten muß. Dies könnte man verbessern, indem er zum Beispiel zwei Spalten berechnet, die Erledigung der ersten Spalte schon meldet und dann mit der nächsten anfängt. Diese Technik ergibt volle Überlappung.

Bild 10 zeigt einige Meßergebnisse. Das Programm „MAND86", schneidet dabei natürlich schlecht ab (für einen besseren Vergleich müßte man die zeitkritschen Routinen in Assembler programmieren). Allerdings erzeugt Turbo-Pascal schnellen Code, der schon zur Größenabschätzung dienen kann. Ein Parallelrechner benötigt auf dem AT386 48.8 Sekunden, zwei Parallelrechner 22.0 Sekunden und vier Parallelrechner 12.2 Sekunden. Also mit vier Parallelrechnern ist man auch vier mal so schnell, die Sache wird linear beschleunigt. Hier sieht man auch, daß es kaum einen Unterschied zwischen den verschiedenen PCs gibt (V20, 286 oder 386). Dies kommt daher, daß der Rechenaufwand bei der Mandelbrotberechnung groß ist gegenüber der Ausgabezeit der einzelnen Spalten. Solange gilt:
(Summe Rechenzeit auf Parallelrechner) dividiert durch (Anzahl Parallelrechner) größer Ausgabezeit, ist eine lineare Beschleunigung möglich. Ein Beispiel bei dem der PC mit seiner langsamen Zeichengeschwindigkeit den Flaschenhals bildet, zeigt der nächste Abschnitt.

```
 (* ****************************************** *)
 (* Mandelbrotmenge mit 68000 Parallelrechner *)
 (* Rolf-Dieter Klein, Tobias Thiel           *)
 (* V 1.0 890424  in Turbo-Pascal             *)
 (* ****************************************** *)

program  mandel;

uses Crt,Graph;

const
 MAXX = 320;
 MAXY = 200;

var
 buff: array[0..32767] of byte absolute $D000:$0000;
 buff1: array[0..32767] of byte absolute $D000:$8000;
 oldx,anzi,a,x,y,i,j,idx : integer;
 v1,v2 : byte;
 GraphMode, GraphDriver : Integer;
 GraphError : Integer;
 anzpar : integer;
 arra : array [0..MAXX-1] of (none,busy,ready);  (* merker fuer erledigte x-Koordinaten *)
 linbuf : array [0..MAXY-1] of byte;
 finished : boolean;
 adrbuf : array [0..31] of integer;

begin
 for i:=0 to 31 do Port[$300+i] := 1; (* ausblenden aller Speicher *)
 anzpar := 0;
 for i:=0 to 31 do begin
  adrbuf[i] := 0; (* Loeschen Adressfeld *)
  Port[$300 + i] := 3; (* Einblenden *)
  v1 := buff[0];
  v2 := buff[1];
  buff[0] := not(v1);
  buff[1] := not(v2);
  if (not(v1) = buff[0]) and (not(v2) = buff[1]) then begin
   buff[0] := v1;
   buff[1] := v2;
   if (v1 = buff[0]) and (v2 = buff[1]) then begin
    adrbuf[i] := $300+i;
    anzpar := anzpar + 1;
   end;
  end;
  Port[$300+i] := 1; (* wieder ausblenden *)
 end;
 if anzpar < 1 then begin
  Writeln('Fehler: Kein Parallelrechner gefunden');
  Halt;
 end else begin
  writeln('Anzahl Parallelrechner: ',anzpar);
 end;
 GraphDriver := EGA;
 GraphMode := EGAHi;
 InitGraph(GraphDriver,GraphMode,'\TURBO');
 If GraphResult <> 0 then begin
  writeln('EGA Mode nicht gefunden');
  Halt;
 end;
 for i:=0 to MAXX-1 do arra[i] := none;
 anzi := 0;
 for i:=0 to 31 do begin (* alle Rechner starten *)
  if adrbuf[i] <> 0 then begin
   Port[adrbuf[i]] := 3; (* Einblenden, x ablegen *)
   x := (MAXX DIV anzpar) * anzi;
   arra[x] := busy; (* dort merken *)
   anzi := anzi + 1;
   buff1[3] := x mod 256; (* Startwerte *)
   buff1[2] := x div 256;
   Port[adrbuf[i]] := 0; (* Ausblenden und starten *)
   repeat
   until Port[adrbuf[i]] = 0; (* warten bis laeuft *)
  end;
 end;
 finished := false;
 repeat
  for i:=0 to 31 do begin
   if adrbuf[i] <> 0 then begin
    if Port[adrbuf[i]] = 255 then begin (* Rechner fertig *)
     Port[adrbuf[i]] := 3; (* einblenden und CPU stoppen *)
     x := buff1[3] + buff1[2] * 256;
     for y:=1 to MAXY-1 do begin
      linbuf[y] := buff1[y * 4 + 3];
     end;
     arra[x] := ready;
     oldx := x;
     x := x + 1;
     if arra[x] <> none then begin (* suchen *)
      finished := FALSE;
      j := 0;
      repeat
       if (arra[(j + x) mod MAXX] = none) then begin
        idx := (j + x) mod MAXX;
        buff1[3] := idx mod 256; (* Startwerte *)
        buff1[2] := idx div 256;
        Port[adrbuf[i]] := 0; (* starten und ausblenden *)
        arra[idx] := busy;
        repeat
        until Port[adrbuf[i]] = 0; (* warten bis laeuft *)
        finished := TRUE;
       end;
       j := j + 1;
      until (j = MAXX+1 ) OR finished;
      if j=MAXX+1 then Port[adrbuf[i]] := 1;
     end else begin (* ok frei x+1 *)
      buff1[3] := x mod 256; (* Startwerte *)
      buff1[2] := x div 256;
      arra[x] := busy;
      Port[adrbuf[i]] := 0; (* starten und ausblenden *)
      repeat
      until Port[adrbuf[i]] = 0; (* warten bis laeuft *)
     end;
     for y:=1 to MAXY-1 do begin
      PutPixel(oldx,y,(linbuf[y]) mod 16);
     end;
    end;
   end;
  end;
 until keypressed;
 for i:=0 to 31 do Port[$300+i] := 1; (* ausblenden aller Speicher und Stop *)
 CloseGraph;
end.
```

Bild 9. Pascalprogramm für Mandelbrotmenge mit Parallelrechner

	V20-8MHz	286-10Mhz	386-16Mhz
Mand86	1360.0 sec	513.0 sec	264.0 sec
Mandel 1 Par	50.0 sec	48.8 sec	48.8 sec
Mandel 2 Par	27.0 sec	23.0 sec	22.6 sec
Mandel 3 Par	*	16.3 sec	15.6 sec
Mandel 4 Par	*	12.7 sec	12.2 sec

	V20-8MHz	286-10Mhz	386-16Mhz
Mand86	0.036	0.095	0.18
Mandel 1 Par	1.0	1.0	1.0
Mandel 2 Par	1.89	2.12	2.16
Mandel 3 Par	*	2.99	3.12
Mandel 4 Par	*	3.84	4.00

Bild 10. Rechenzeiten (links) und Beschleunigungsfaktoren (rechts) bei der Mandelbrotmenge

```
-129.0000 -124.0000 172.0000 216.0000 0 0.0000 96.0000
20 0 0 0 0 0
16.000000
0
1
2.000000
3
1.000000
0.000000
0.000000
2.540000
1
0
16.000000
1
0
0
1
0
0
2
0
0.000000 0.000000 0.000000
1
0
2000
200000.00
```

Bild 11. Koordinatenliste von Design-CAD-3D

```
224.000000
226.500000
0.8660254
0.5000000
0.9396926
-0.3420201
*
21 1 0 0 0 0
29 2 0 0 0 1
-116 -37 0
-116 92 0
29 2 0 0 0 1
-116 92 0
43 92 0
29 2 0 0 0 1
43 92 0
43 -37 0
29 2 0 0 0 1
43 -37 0
-116 -37 0
30 4 0 0 0 1
1
2
3
4
29 2 0 0 0 1
-116 92 0
-116 92 96
```

```
29 2 0 0 0 1
-116 92 96
-116 -37 96
29 2 0 0 0 1
-116 -37 96
-116 -37 0
30 4 0 0 0 1
1
6
7
8
29 2 0 0 0 1
-116 -37 96
43 -37 96
29 2 0 0 0 1
43 -37 96
43 -37 0
30 4 0 0 0 1
-8
10
11
4
29 2 0 0 0 1
-116 92 96
43 92 96
29 2 0 0 0 1
43 92 96
43 -37 96
30 4 0 0 0 1
-7
13
14
-10
29 2 0 0 0 1
43 92 0
43 92 96
30 4 0 0 0 1
-3
16
14
11
30 4 0 0 0 1
6
13
-16
-2
1 2 0 0 0 1
-83 -5 0
12 67 80
1 2 0 0 0 1
-88 -4 32
16 74 16
1 3 0 0 0 1
-69 72 80
-97 50 80
-57 45 80
3 3 0 0 0 1
-129 -124 80
-45 -115 80
-43 -96 80
Test
```

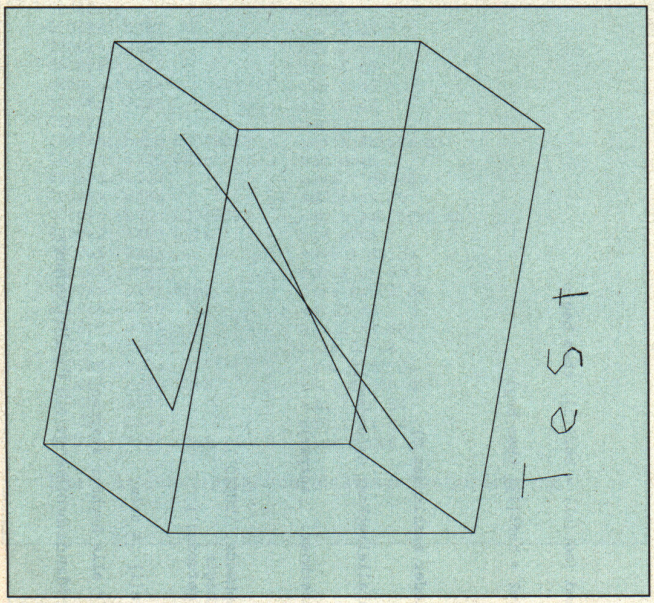

Bild 12. Durch die Datenliste dargestelltes 3D-Objekt

3D-CAD

Die schnelle Ausgabe von 3D-Modellen ist eine wichtige Anwendung von Parallelrechnern. Relativ neu und preiswert ist das Programm Design-CAD-3D. Das Programm erlaubt, 3D-Modelle interaktiv mit Maus und Tastatur zu erstellen. Glücklicherweise gab es zu diesem Programm eine Beschreibung der verwendeten Objektdateien. Das folgende Programm ist in der Lage 3D-Dateien auf dem Bildschirm darzustellen. Dabei kann es sowohl ohne Parallelrechner als auch mit arbeiten. Zunächst zur Objektdatei. Alle Werte sind als lesbare ASCII-Zeichen in der Datei abgelegt, was insbesondere die Weiterberarbeitung stark erleichtert.

Zunächst folgt ein Header mit sieben Werten:
minimum X-Koordinate
minimum Y-Koordinate
Breite der Zeichnung
Höhe der Zeichnung
reserviert
Minimum Z-Koordinate
Tiefe der Zeichnung

Danach können Kommandos folgen, die wie folgt aussehen:
Kommando-Code
Anzahl
reserviert
reserviert
reserviert
Farbe

Als Kommandos gibt es:
0 = Handle hier nicht ausgewertet, jedoch Punkte folgen (x, y, z)
1 = Linien, Punkteliste folgt (x, y, z)
3 = Text, Punkteliste folgt, dann Text als ASCII in einer Zeile
15 = Attribute, Anzahl Zeilen folgt
18 = Dimension – Linear, Anzahl Zeilen folgt
19 = Dimension – Angular, Anzahl Zeilen folgt
20 = System-Parameter, bis in Zeile das Zeichen „*“ erscheint
21 = Neuer Layer, Anzahl gibt die Layerzahl an, nichts folgt
29 = Kantenliste, Punkte (x, y, z) folgen
30 = Flächenliste, Index folgt, wenn <0 dann negativer Durchlauf

Bild 11 zeigt ein Beispiel, so wie es mit Design-CAD-3D erzeugt wurde und *Bild 12* zeigt das dadurch dargestellte Objekt. Das Programm ist in Turbo-Pascal geschrieben und *Bild 13* zeigt das komplette Listing.

Zunächst wird die Datenliste eingelesen und in ein internes Integer-Format umgerechnet. Dazu wird das Bildschirmformat als Normierungsfaktor herangezogen. Das Unterprogramm „bildaus“ berechnet das Bild mit Hilfe des PCs. Dabei kann man zwei Parameter, nämlich den Betrachtungswinkeln phix und phiy übergeben. Die für

```
(* ************************************** *)
(* Design CAD 3D Display Program          *)
(* fuer PCPAR68000 zeigt 3D-CAD Datei     *)
(* auf dem Bildschirm an unter Benutzung  *)
(* des Parallelrechners PCPAR68000        *)
(* Rolf-Dieter Klein, Tobias Thiel        *)
(* V 1.0  890526 in Turbo-Pascal 5.0      *)
(* ************************************** *)

program  dc3disp;

uses Crt,Graph;

const
 MAXWIDTH = 640;   (* bei EGA Aufloesung *)
 MAXHEIGHT = 350;  (* 640 x 350 Bildpunkte, alles normieren *)
 MAXH2 = MAXWIDTH DIV 2;       (* 1/2 Width  entsprechend eintragen *)
 MAXV2 = MAXHEIGHT DIV 2;      (* 1/2 Height *)
 MAXEDGE = 7000;
 MAXARA = 3000;
 VIEWDIST = 1000;  (* Distanz *)

 (*$I disp.inc*)   (* Laden der Include-Datei von ASM68K erzeugt *)

 SEG = $d000;      (* Parallelrechner Speichersegment *)

type
     nextedge = ^edgestr;
     edgestr = record
                index: integer;
                nexte : nextedge;
               end;
     nextpoint = ^pointstr;
     pointstr = record
              x,y,z : Integer;
              nextp : nextpoint;
             end;
     nextprec = record
                 ptr : nextpoint;
                 art : (none,line,edge,index);
                 col : Integer;
                end;
var
 parsinx : integer absolute SEG:shsinx;
 parsiny : integer absolute SEG:shsiny;
 parcosx : integer absolute SEG:shcosx;
 parcosy : integer absolute SEG:shcosy;
 parscale : integer absolute SEG:shscale;
 parmaxwidth : integer absolute SEG:shmaxwidth;
 parmaxheight : integer absolute SEG:shmaxheight;
 parkanten : array[0..32000] of integer absolute SEG:shkanten;
 parsignatur : Longint absolute SEG:shsignatur;

 edgelist : array[1..MAXEDGE] of nextprec;
 aralist  : array[1..MAXARA] of nextedge;

 lineedgecnt,linecnt,edgecnt,aracnt : Integer;  (* Counters *)
 pptr,pptr1 : nextpoint;   (* Pointer als Hilfspointer *)
 eptr,eptr1 : nextedge;

 oldx,anzi,a,idx : integer;
 v1,v2 : byte;
 GraphMode, GraphDriver : Integer;
 GraphError : Integer;
 anzpar : integer;
 finished : boolean;
 adrbuf : array [0..31] of integer;
 name: string;
 datei: text;   (* Eingabe Datei *)
 zeile: string;
minx,miny,deltax,deltay,dummy1,minz,deltaz : Real;
delta : Real;
cmd,anz,dum1,dum2,dum3,color : Integer;
 choice,i,j   : Integer;
 x,y,z : Real;
 xp,yp,zp : Integer; (* Projektion *)
 layer : Integer; (* Aktueller Layer *)
 ch : char;       (* Zeichen fuer Hauptprogrammbefehle *)
 phix,phiy : Integer; (* Hauptprog *)
 scale : Integer;     (* Scalierung in % *)

procedure bildaus(phix,phiy : integer); (* Bild Ausgeben nur mit PC *)
var
 tmp : Longint;
 sinx,siny,cosx,cosy : Longint;
 x1,y1,z1 : Longint; (* Temp *)
 x2,y2,z2 : Longint;

begin
 (* Muss nur Am Anfang berechnet werden, daher zeitunkritisch *)
 ClearViewport; (* Bildschirm loeschen zuerst *)
 sinx := trunc(256 * sin(phix*PI/260));
 siny := trunc(256 * sin(phiy*PI/260));
 cosx := trunc(256 * cos(phix*PI/260));
 cosy := trunc(256 * cos(phiy*PI/260));
 (* Linienliste ausgeben *)
 if edgecnt > 0 then begin (* Linien Vorhanden *)
  for i:=1 to edgecnt do begin
   if edgelist[i].art = line then begin
    pptr := edgelist[i].ptr;   (* Start LinienListe *)
    if pptr <> NIL then begin
     x1 := pptr^.x ;
     y1 := pptr^.y ;
     z1 := pptr^.z ;
     x2 := (cosy * x1 + siny * z1) div 256;
     y2 := y1;
     z2 := (-siny * x1 + cosy * z1) div 256;
     xp := trunc(x2);
     yp := trunc((cosx * y2 + sinx * z2) div 256);
     zp := trunc((-sinx * y2 + cosx * z2) div 256);
     tmp := (longint(xp) * longint(VIEWDIST))
              div (longint(VIEWDIST) + longint(zp));
     tmp := (tmp * scale) DIV 100;
     xp := trunc(tmp)+MAXH2;
     tmp := (longint(yp) * longint(VIEWDIST))
              div (longint(VIEWDIST) + longint(zp));
     tmp := (tmp *  scale) DIV 100;
     yp := MAXHEIGHT-trunc(tmp)-MAXV2;
     moveto(xp,yp);
     setcolor(edgelist[i].col);
     pptr := pptr^.nextp;
    end;
    while pptr <> NIL do begin
     x1 := pptr^.x ;
     y1 := pptr^.y ;
     z1 := pptr^.z ;
     x2 := (cosy * x1 + siny * z1) div 256;
     y2 := y1;
     z2 := (-siny * x1 + cosy * z1) div 256;
     xp := trunc(x2);
     yp := trunc((cosx * y2 + sinx * z2) div 256);
     zp := trunc((-sinx * y2 + cosx * z2) div 256);
     tmp := (longint(xp) * longint(VIEWDIST))
              div (longint(VIEWDIST) + longint(zp));
     tmp := (tmp *  scale) DIV 100;
     xp := trunc(tmp)+MAXH2;
     tmp := (longint(yp) * longint(VIEWDIST))
              div (longint(VIEWDIST) + longint(zp));
     tmp := (tmp *  scale) DIV 100;
     yp := MAXHEIGHT-trunc(tmp)-MAXV2;
     lineto(xp,yp);
     pptr := pptr^.nextp;
    end;
   end;
  end;
 end;
```

Bild 13. 3D-Ausgabeprogramm für PC und Parallelrechnerbetrieb

```
(* Flaechenliste ausgeben mit Transformation *)
if aracnt > 0 then begin (* Flaechen Vorhanden *)
 for i:=1 to aracnt do begin (* alle Flaechen *)
  eptr := aralist[i];   (* Start FlaechenListe *)
  while eptr <> NIL do begin
   j := eptr^.index;    (* Kantenindex holen *)
   if j < 0 then j:=-j; (* negativer Durchlaufsinn, Ignorieren *)
   if j > edgecnt then begin
    Closegraph;
    writeln('Fehler in Indexliste ',j);
    Halt;
   end;
   pptr := edgelist[j].ptr; (* und Pointer *)
   setcolor(edgelist[j].col);
   if pptr <> NIL then begin
    x1 := pptr^.x ;
    y1 := pptr^.y ;
    z1 := pptr^.z ;
    x2 := (cosy * x1 + siny * z1) div 256;
    y2 := y1;
    z2 := (-siny * x1 + cosy * z1) div 256;
    xp := trunc(x2);
    yp := trunc((cosx * y2 + sinx * z2) div 256);
    zp := trunc((-sinx * y2 + cosx * z2) div 256);
    tmp := (longint(xp) * longint(VIEWDIST))
           div (longint(VIEWDIST) + longint(zp));
    tmp := (tmp *  scale) DIV 100;
    xp := trunc(tmp)+MAXH2;
    tmp := (longint(yp) * longint(VIEWDIST))
           div (longint(VIEWDIST) + longint(zp));
    tmp := (tmp *  scale) DIV 100;
    yp := MAXHEIGHT-trunc(tmp)-MAXV2;
    moveto(xp,yp);

    pptr := pptr^.nextp;
   end;
   while pptr <> NIL do begin
    x1 := pptr^.x ;
    y1 := pptr^.y ;
    z1 := pptr^.z ;
    x2 := (cosy * x1 + siny * z1) div 256;
    y2 := y1;
    z2 := (-siny * x1 + cosy * z1) div 256;
    xp := trunc(x2);
    yp := trunc((cosx * y2 + sinx * z2) div 256);
    zp := trunc((-sinx * y2 + cosx * z2) div 256);
    tmp := (longint(xp) * longint(VIEWDIST))
           div (longint(VIEWDIST) + longint(zp));
    tmp := (tmp * scale) DIV 100;
    xp := trunc(tmp)+MAXH2;
    tmp := (longint(yp) * longint(VIEWDIST))
           div (longint(VIEWDIST) + longint(zp));
    tmp := (tmp *  scale) DIV 100;
    yp := MAXHEIGHT-trunc(tmp)-MAXV2;
    lineto(xp,yp);
    pptr := pptr^.nextp;
   end;
   eptr := eptr^.nexte; (* weitere Kanten *)
  end; (* while eptr *)
 end;
end; (* if *)
end;

procedure bildpar(phix,phiy:integer); (* Bild ausgeben mit Parallelrechner *)
var idx,xp,yp,col :integer;
 beendet : Integer;
 maxh,maxv : Integer;
begin
anzi := 0;
for i:=0 to 31 do begin (* alle Rechner starten *)
 if adrbuf[i] <> 0 then begin
  Port[adrbuf[i]] := 3; (* Einblenden, phix,phiy ablegen *)
  maxh := MAXWIDTH;
  maxv := MAXHEIGHT;
  parmaxwidth := swap(maxh);
  parmaxheight := swap(maxv);
  parsinx := swap(trunc(256 * sin(phix*PI/260)));
  parsiny := swap(trunc(256 * sin(phiy*PI/260)));
  parcosx := swap(trunc(256 * cos(phix*PI/260)));
  parcosy := swap(trunc(256 * cos(phiy*PI/260)));
  parscale := swap(scale);
  Port[adrbuf[i]] := 0; (* Ausblenden und Start *)
 end;
end;
ClearViewport; (* Erst hier Bildschirm loeschen, da ueberlappend *)
beendet := 0; (* Anzahl Rechner *)
repeat
 i := 0;
 repeat
  if adrbuf[i] <> 0 then begin (* Rechner fertig *)
   if Port[adrbuf[i]] = 1 then begin (* Berechnete Werte Da *)
    (* Abfrage kritisch, denn es ist denkbar, dass man einen *)
    (* Rechner mehr als einmal als Fertig einstuft *)
    Port[adrbuf[i]] := 2; (* Einblenden *)
    beendet := beendet + 1; (* in diesem Fall *)
    idx := 0;
    while parkanten[idx] <> 0 do begin (* Kantenliste 1,2 = bef, 0=end *)
     col := parkanten[idx] and 255; (* wegen Swap *)
     idx := idx + 1;
     if parkanten[idx] <> -129 then begin (* -129 = $FF7F swapped *)
      Setcolor(col);
      moveto(parkanten[idx+3],parkanten[idx+4]);
      idx := idx + 5; (* auf Koordinate *)
      while parkanten[idx] <> -129 do begin
       lineto(parkanten[idx+3],parkanten[idx+4]);
       idx := idx + 5; (* nur noch Koordinaten *)
      end; (* while *)
     end; (* Koor *)
     idx := idx + 1;
    end;
    Port[adrbuf[i]] := 1; (* Ausblenden und Stopp *)
   end;
  end;
  i := i + 1;
 until (i >= 32) or (beendet >= anzpar);
until beendet = anzpar;
for i:=0 to 31 do begin (* alle Rechner ausblenden *)
 if adrbuf[i] <> 0 then begin
  Port[adrbuf[i]] := 1; (* STOP und ausblenden *)
 end;
end;
end;

procedure initpars;  (* Initialisierung der Parallelrechner *)
 var
  idx,reidx,idxmax : Integer;
  breakzahl,breakcnt : Integer;
  parcnt : Integer;
begin
 for i:=0 to 32 do Port[$300+i] := 1; (* ausblenden aller Speicher *)
 anzpar := 0;
 for i:=0 to 31 do begin
  adrbuf[i] := 0; (* Loeschen Adressfeld *)
  Port[$300 + i] := 3; (* Einblenden *)
  if parsignatur = $12AB0514 then begin (* Rechner gefunden *)
   adrbuf[i] := $300+i;
   anzpar := anzpar + 1;
  end;
  Port[$300 + i] := 1; (* wieder ausblenden *)
 end;
 if anzpar < 1 then begin
  Writeln('Fehler: Kein Parallelrechner mit DISP.P68 gefunden');
  Writeln('        LOADER DISP.P68  zuerst ausfuehren');
  Writeln('        oder keine Parallelrechner vorhanden.');
  Halt;
 end;
```

```
(* Init der Datenfelder Datenliste einlesen *)
(* Linienliste ausgeben *)
(* Verteilen der Kantenlisten auf die Parallelrechner etwa gleichmaessig *)
(* hier nur adhoc Methode verwendet, sonst Programmliste zu lang *)
reidx := 0;    (* ersten Rechner freischalten *)
parcnt := 1;   (* mind. einen Rechner verwendet *)
idxmax := 0;
repeat
 if adrbuf[reidx] <> 0 then Port[adrbuf[reidx]] := 3;
 reidx := reidx + 1;
until (adrbuf[reidx-1] <> 0) or (reidx = 32);
breakzahl := (linecnt + lineedgecnt) div anzpar; (* Dannach naechsten Rechner *)
writeln(breakzahl,' Kanten pro ',anzpar,' Parallelrechner');
breakcnt := 0; (* Nur die Linien und Flaechenkanten aufteilen *)
idx := 0;
if edgecnt > 0 then begin (* Linien Vorhanden *)
 for i:=1 to edgecnt do begin
  if edgelist[i].art = line then begin (* Linien zuerst ausgeben *)
   pptr := edgelist[i].ptr;    (* Start LinienListe *)
   parkanten[idx] := swap(1 + edgelist[i].col * 256); (* Linien Kennung + Farbe *)
   idx := idx + 1;
   if pptr <> NIL then begin
    parkanten[idx] := swap(pptr^.x) ;
    parkanten[idx+1] := swap(pptr^.y) ;
    parkanten[idx+2] := swap(pptr^.z) ;
    idx := idx + 5;  (* 5 wegen leerer xp,yp Felder *)
    pptr := pptr^.nextp;
   end;
   while pptr <> NIL do begin
    parkanten[idx] := swap(pptr^.x) ;
    parkanten[idx+1] := swap(pptr^.y) ;
    parkanten[idx+2] := swap(pptr^.z) ;
    idx := idx + 5;
    pptr := pptr^.nextp;
   end;
   parkanten[idx] := -129; (* Ende Kennung $7fff non swapped *)
   idx := idx + 1;
   if idx > 32000 then begin
    writeln('Fehler Index Ueberlauf Linienliste, mehr als 64K Byte');
    writeln('pro Parallelrechner, mehr Parallelrechner verwenden,');
    writeln('oder kleinere Datei.');
    Halt;
   end;
   breakcnt := breakcnt + 1;
   if (breakcnt > breakzahl) and (parcnt < anzpar ) then begin  (* check next Rechner *)
    parcnt := parcnt + 1;
    if idxmax < idx then idxmax := idx;
    parkanten[idx] := 0;    (* neuer Befehl = Ende Liste *)
    idx := 0;               (* Starten von Vorne *)
    Port[adrbuf[reidx-1]] := 1; (* ausblenden *)
    repeat
     if adrbuf[reidx] <> 0 then Port[adrbuf[reidx]] := 3;
     reidx := reidx + 1;
    until (adrbuf[reidx-1] <> 0) or (reidx = 32);
    breakcnt := 0; (* wieder von vorne *)
   end; (* check breakzahl *)
  end;
 end;
end;
(* Flaechenliste ausgeben mit Transformation *)
if aracnt > 0 then begin (* Flaechen Vorhanden *)
 for i:=1 to aracnt do begin (* alle Flaechen *)
  eptr := aralist[i];   (* Start FlaechenListe *)
  while eptr <> NIL do begin
   j := eptr^.index;    (* Kantenindex holen *)
   if j < 0 then j:=-j; (* negativer Durchlaufsinn, Ignorieren *)
   if j > edgecnt then begin
    Closegraph;
    writeln('Fehler in Indexliste ',j);
    Halt;
   end;
   pptr := edgelist[j].ptr; (* und Pointer *)
   parkanten[idx] := swap(1 + edgelist[j].col * 256); (* Linien Kennung + Farbe *)
   idx := idx + 1;
   if pptr <> NIL then begin
    parkanten[idx] := swap(pptr^.x) ;
    parkanten[idx+1] := swap(pptr^.y) ;
    parkanten[idx+2] := swap(pptr^.z) ;
    idx := idx + 5;
    pptr := pptr^.nextp;
   end;
   while pptr <> NIL do begin
    parkanten[idx] := swap(pptr^.x) ;
    parkanten[idx+1] := swap(pptr^.y) ;
    parkanten[idx+2] := swap(pptr^.z) ;
    idx := idx + 5;
    pptr := pptr^.nextp;
   end;
   parkanten[idx] := -129; (* Ende Kennung $7fff non swapped *)
   idx := idx + 1;
   if idx > 32000 then begin
    writeln('Fehler Index Ueberlauf Feldliste, mehr als 64K Byte');
    writeln('pro Parallelrechner, mehr Parallelrechner verwenden,');
    writeln('oder kleinere Datei.');
    Halt;
   end;
   breakcnt := breakcnt + 1;  (* Check lineedgecnt *)
   if (breakcnt > breakzahl) and (parcnt < anzpar ) then begin  (* check next Rechner *)
    parcnt := parcnt + 1;
    if idxmax < idx then idxmax := idx;
    parkanten[idx] := 0;    (* neuer Befehl = Ende Liste *)
    idx := 0;               (* Starten von Vorne *)
    Port[adrbuf[reidx-1]] := 1; (* ausblenden *)
    repeat
     if adrbuf[reidx] <> 0 then Port[adrbuf[reidx]] := 3;
     reidx := reidx + 1;
    until (adrbuf[reidx-1] <> 0) or (reidx = 32);
    breakcnt := 0; (* wieder von vorne *)
   end; (* check breakzahl *)
   eptr := eptr^.nexte; (* weitere Kanten *)
  end; (* while eptr *)
  (* hier nur Flaechen aufteilen zwischen den Rechnern *)
 end; (* for *)
end; (* if *)
parkanten[idx] := 0;  (* Ende Kennung *)
if idx > 32000 then begin
 writeln('Fehler Index Ueberlauf Feldliste, mehr als 64K Byte');
 writeln('pro Parallelrechner, mehr Parallelrechner verwenden,');
 writeln('oder kleinere Datei.');
 Halt;
end;
for i:=0 to 31 do begin
 if adrbuf[i] <> 0 then Port[adrbuf[i]] := 1; (* ausblenden, STOP *)
end;
if idxmax < idx then idxmax := idx;
writeln('Verwendete Maxindexzahl in Speicher-Worten:',idxmax);
writeln('Start Display = CR eingeben:');
readln;
end;

begin
 writeln;
 writeln('Design-CAD-3D Display Programm, Rolf-Dieter Klein, Tobias Thiel');
 writeln('fuer PCPAR68000, (C) 1989 Munich, V 1.0');
 write('Design-CAD-3D Dateiname (incl. extension):');
 readln(name);
 Assign(datei,name);
 Reset(datei);
 layer := 0;
 linecnt := 0;      (* nur fuer Linienaufteilung *)
 lineedgecnt := 0; (* Kanten in Flaechen zaehlen *)
 edgecnt := 0;
 aracnt := 0;
 readln(datei,minx,miny,deltax,deltay,dummy1,minz,deltaz);
 delta := deltax;
 if deltay > delta then delta := deltay;
```

```
if deltaz > delta then delta := deltaz;  (* MAX suchen *)
while not EOF(datei) do begin
 Readln(datei,cmd,anz,dum1,dum2,dum3,color);
 if cmd = 20 then begin (* Spezialfall System Parameter, ueberlesen *)
  repeat
   readln(datei,zeile);
  until zeile[1] = '*';  (* Ende Kennung ist immer erstes Zeichen *)
 end else begin  (* Befehl ausfuehren *)
  case cmd of
   0: begin
       for i:=1 to anz do readln(datei,x,y,z);
      end; (* Handles ignorieren *)
   1: begin
       if anz >= 1 then begin
        edgecnt := edgecnt + 1; (* Eintragszahl erhoehen *)
        linecnt := linecnt + 1; (* fuer Rechneraufteilung *)
        new(pptr);
        edgelist[edgecnt].ptr := pptr;
        edgelist[edgecnt].art := line;
        edgelist[edgecnt].col := color;
        pptr^.nextp := NIL;
        readln(datei,x,y,z);    (* mind ein Eintrag *)
        pptr^.x := trunc((x-minx-deltax/2)/delta * MAXHEIGHT);
        pptr^.y := trunc((y-miny-deltay/2)/delta * MAXHEIGHT);
        pptr^.z := trunc((z-minz-deltaz/2)/delta * MAXHEIGHT);
        for i:=2 to anz do begin
         new(pptr1);
         pptr^.nextp := pptr1;
         pptr := pptr^.nextp;
         readln(datei,x,y,z);
         pptr^.nextp := NIL;
         pptr^.x := trunc((x-minx-deltax/2)/delta * MAXHEIGHT);
         pptr^.y := trunc((y-miny-deltay/2)/delta * MAXHEIGHT);
         pptr^.z := trunc((z-minz-deltaz/2)/delta * MAXHEIGHT);
        end;
       end; (* if *)
      end; (* Lines *)
   3: begin
       for i:=1 to anz do readln(datei,x,y,z);
       readln(datei,zeile); (* Text Info Folgt *)
      end; (* text *)
   15: begin
        for i:=1 to anz do readln(datei,zeile);
       end;  (* Attribute ignorieren *)
   18: begin
        for i:=1 to anz do readln(datei,zeile);
       end; (* Dimension linear *)
   19: begin
        for i:=1 to anz do readln(datei,zeile);
       end; (* Dimension angular *)
   21: begin
        layer := anz; (* steht dort *)
       end; (* neuer Layer *)
   29: begin
        if anz >= 1 then begin
         edgecnt := edgecnt + 1; (* Eintragszahl erhoehen *)
         new(pptr);
         edgelist[edgecnt].ptr := pptr;
         edgelist[edgecnt].art := edge;
         edgelist[edgecnt].col := color;
         pptr^.nextp := NIL;
         readln(datei,x,y,z);    (* mind ein Eintrag *)
         pptr^.x := trunc((x-minx-deltax/2)/delta * MAXHEIGHT);
         pptr^.y := trunc((y-miny-deltay/2)/delta * MAXHEIGHT);
         pptr^.z := trunc((z-minz-deltaz/2)/delta * MAXHEIGHT);
         for i:=2 to anz do begin
          new(pptr1);
          pptr^.nextp := pptr1;
          pptr := pptr^.nextp;
          readln(datei,x,y,z);
          pptr^.nextp := NIL;
          pptr^.x := trunc((x-minx-deltax/2)/delta * MAXHEIGHT);
          pptr^.y := trunc((y-miny-deltay/2)/delta * MAXHEIGHT);
          pptr^.z := trunc((z-minz-deltaz/2)/delta * MAXHEIGHT);
         end;
        end; (* if *)
       end; (* Kantenliste *)
   30: begin
        edgecnt := edgecnt + 1; (* belegt auch eine Kante (dummy) *)
        edgelist[edgecnt].ptr := NIL; (* hier nicht eintragen *)
        edgelist[edgecnt].art := none; (* Area nicht hier eingetragen *)
        if anz >= 1 then begin
         aracnt := aracnt + 1; (* Eintragszahl erhoehen *)
         new(eptr);
         aralist[aracnt] := eptr;
         eptr^.nexte := NIL;
         readln(datei,j);  (* mind ein Eintrag *)
         eptr^.index := j; (* < 0 dann Richtungssinn anders *)
         lineedgecnt := lineedgecnt + 1;
         for i:=2 to anz do begin
          new(eptr1);
          eptr^.nexte := eptr1;
          eptr := eptr^.nexte;
          readln(datei,j);
          eptr^.nexte := NIL;
          eptr^.index := j;
          lineedgecnt := lineedgecnt + 1;
         end;
        end; (* if *)
       end; (* Flaechenliste *)
    end;
   end;
  end;
  close(datei);
  (* Daten Eingelesen *)
  (* Bild ausgeben *)
  writeln('Linienelemente    = ',edgecnt);
  writeln('Flaechen          = ',aracnt);
  if (edgecnt > MAXEDGE) or (aracnt > MAXARA) then begin
   writeln('Fehler Speicherueberlauf, zuviele Elemente');
   Halt;
  end;
  writeln;
  writeln('1=auf dem PC rechnen');
  writeln('2=auf dem Parallelrechner rechnen');
  readln(choice);
  writeln;
  if choice = 2 then
   initpars;
  GraphDriver := EGA;
  GraphMode := EGAHi;
  InitGraph(GraphDriver,GraphMode,'\TURBO');
  If GraphResult <> 0 then begin
   writeln('EGA Mode nicht gefunden oder EGAVGA.BGI-Treiber fehlt.');
   Halt;
  end;
  phix := 20;  (* in Grad *)
  phiy := 30;
  scale := 80; (* Skalierung in % *)
  repeat
   if choice = 2 then
    bildpar(phix,phiy) (* erst nach Start der CPUs loeschen in UPRG *)
   else
    bildaus(phix,phiy); (* mit 8086, Bildschirm loeschen in UPRG *)
   ch := readkey;
   case ch of
   '5': begin phix := 20; phiy := 30; scale := 80; end; (* Norm *)
   '8': begin phix := phix + 10; end;
   '2': begin phix := phix - 10; end;
   '6': begin phiy := phiy + 10; end;
   '4': begin phiy := phiy - 10; end;
   '+': begin if scale < 500 then scale := scale + 10; end;
   '-': begin if scale > 10 then scale := scale - 10; end;
  end;
 until (ch = 'q') or (ch = 'Q');
 CloseGraph;
end.
```

```
; 3D - Transformationsprogramm fuer PCPAR68000
; Rolf-Dieter Klein, Tobias Thiel
; 890526

 dc.l stack      ; Stack Definition
 dc.l start      ; Hauptprogramm Startadresse nach Reset.

PORT EQU $FFFFFFFF

                ; nach Start muessen die Daten bereits gueltig sein.
start:          ; Start Berechnen der neuen Koordinaten
 move.b #0,PORT ; Port loeschen, Befehl bereit
 move.w shmaxwidth.w,d0
 asr.w #1,d0
 move.w d0,mh2
 move.w shmaxheight.w,d0
 asr.w #1,d0
 move.w d0,mv2   ; temp Variable
 lea shkanten,a0         ; Index
schleife:                ; Alle Eintraege abarbeiten
 move.w (a0)+,d0         ; Art Kennung holen
 beq ende                ; Ende der Liste erreicht
schleife1:
 move.w (a0)+,d1         ; d1=x   (d4=x1)
 cmp.w #$7fff,d1         ; Ende Erkennung fuer Punkteliste
 beq.s schleife          ; Weiter in aeusserer Schleife
 move.w (a0)+,d2         ; d2=y   (d5=y1)
 move.w (a0)+,d3         ; d3=z   (d6=z1)
; Transformation 1, Rotation um die y-Achse
 move shcosy.w,d0
 muls d1,d0              ; x1 := (cosy * x + siny * z) div 256
 move shsiny.w,d4        ; Ergebnis dort
 muls d3,d4
 add.l d0,d4             ; Langwort
 asr.l #8,d4             ; / 256  zur Normierung.
 move d2,d5              ; bleibt unveraendert
 move shsiny.w,d0
 neg.w d0                ; z1 := (-siny * x + cosy * z) div 256
 muls d1,d0
 move shcosy.w,d6
 muls d3,d6
 add.l d0,d6             ; Langwort gueltig
 asr.l #8,d6             ; / 256 zur Normierung
; Transformation 2, Rotation um die x-Achse
 move d4,d1              ; x2 := x1;
 move shcosx.w,d0        ; y2 := (cosx * y1 + sinx * z1)
 muls d5,d0
 move shsinx.w,d2
 muls d6,d2
 add.l d0,d2             ; Langwort
 asr.l #8,d2             ; / 256  zur Normierung
 move shsinx.w,d0              ; z2 := (-sinx * y1 + cosx * z1);
 neg.w d0
 muls d5,d0
 move.w shcosx.w,d3
 muls d6,d3
 add.l d0,d3
 asr.l #8,d3             ; / 256 zur Normierung,
; Ergebnis in d1,d2,d3 (x,y,z)
; Projektion folgt anschliessend
 move.w shview.w,d0
 add.w d3,d0             ; /(view + len) vorbereiten
 beq skip                ; =0, dann Fehler bei Division, ueberspringen.
 muls shview.w,d1
 divs d0,d1              ; (x * view) / (view + len)
 muls shview.w,d2        ; (y * view) / (view + len)
 divs d0,d2              ; x,y vorbereitet, dann scalierung durchfuehren
 ;
```

Bild 14. 3D-Parallelrechnerprogramm in Assembler

```
skip:
 move.w shscale,d0       ; xneu := (x * scale) / 100
 muls d0,d1
 divs #100,d1            ; yneu := (y * scale) / 100
 muls d0,d2
 divs #100,d2            ; x,y Ent
;
 add.w mh2,d1
 add.w mv2,d2            ; Zentrieren
 neg.w d2                ; neg. Koordinatensystem
 add.w shmaxheight,d2    ; h-y bilden.
 rol #8,d1
 rol #8,d2               ; LSB MSB fuer PC vorbereitet.
 move.w d1,(a0)+         ; Ergebnis in Liste ablegen ueberschreiben.
 move.w d2,(a0)+         ;
 bra schleife1
ende:
 move.b #1,PORT  ; Ende Berechnung beendet
warte:
 nop
 bra.s warte     ; Fertig mit Berechnung

mh2: dc.w 320    ; Temp generiert
mv2: dc.w 175

 shared shsignatur
 shared shsinx,shsiny,shcosx,shcosy,shview,shscale,shmaxwidth,shmaxheight
 shared shkanten

shsignatur: dc.l $1405AB12  ; Signatur fuer Ladeerkennung
shsinx:         dc.w 0      ; uebergebener Sin-Wert in 256 * sin(phi) MSB LSB
shsiny:         dc.w 0      ;
shcosx:         dc.w 256    ; entsprechend fuer Cos-Wert
shcosy:         dc.w 256
shview:         dc.w 1000   ; Distanz
shscale:        dc.w 80     ; Skalierung in %  (Default Werte werden von PC ueberschrieben)
shmaxwidth:     dc.w 640    ; Breite Bildschirm
shmaxheight:    dc.w 350    ; Hoehe Bildschirm

                ; ALLES MSB LSB PC muss drehen.
shkanten:       ; Kantenliste dynamisch art,x,y,z,xp,yp,x,y,z.... $7fff
                ; Art: 0 = Endeliste 1 = Linie, 2 = Kante
                ; Ergebnisse werden aber in LSB, MSB ablegt nach xp,yp
                ; damit fuer PC schneller zugreifbar.
        dc.w 1          ; Beispielliste wird ueberschrieben
        dc.w 0,0,0,0,0
        dc.w 100,100,100,0,0,$7fff
        dc.w 2
        dc.w 0,0,100,0,0
        dc.w 0,100,0,0,0
        dc.w 200,0,0,0,0
        dc.w 0,200,0,0,0
        dc.w 0,50,0,0,0,$7fff
        dc.w 0

 org $fff0              ; Stack von oben nach unten
stack: dc.w 0           ; startet hier

 end
```

Bild 15. Include-Datei vom ASM68K-Programm erzeugt

```
(* Header File for CONST - section in PASCAL *)
 shsignatur = $e0;
 shsinx = $e4;
 shsiny = $e6;
 shcosx = $e8;
 shcosy = $ea;
 shview = $ec;
 shscale = $ee;
 shmaxwidth = $f0;
 shmaxheight = $f2;
 shkanten = $f4;
(* END Header File PASCAL *)
```

die Transformation notwendigen Werte müssen nur einmal vor der Schleife berechnet werden. Die gesamte übrige Berechnung erfolgt mit Integer-Arithmetik – wie auch in dem korrespondierenden Parallelrechnerprogramm.
Das Unterprogramm „bildpar" zeigt das Bild mit Hilfe der Parallelrechner. Jeder der Parallelrechner hat dazu einen Teil der Koordinatenliste bekommen. Hier wird nur der neue Werte für phix und phiy als Sinus-Wert, sowie die Skalierung an das Parallelrechnerprogramm über Shared-Variablen übergeben. Das Programm ist nicht zeitkri-

```
# makefile fuer Turbopascal DCDISP
#

dc3disp.exe: dc3disp.pas disp.inc
 tpc dc3disp.pas /b /Ec:\turbo
 copy c:\turbo\dc3disp.exe

disp.inc: disp.a68
 asm68k disp.a68 -p
 loader disp.p68
```

Bild 16. Make-Datei zum Erzeugen des 3D-Anzeigeprogramms

tisch, es wird daher mit einer Schleife gearbeitet. Das Feld „adrbuf" enthält die durch das Initialisierungsprogramm festgestellten Parallelrechneradressen, so daß man nicht jedesmal danach suchen muß. Danach wird der Bildschirm gelöscht, die Parallelrechner berechnen aber schon das Bild. In der Ausgabeschleife wird nun gewartet bis einer der Parallelrechner fertig ist. Dann werden die Daten ausgelesen und gleich auf dem Bildschirm ausgegeben. Hier zeigte sich später, daß die Ausgabe der Linien auf PCs recht lange dauert, ggf. länger als die Berechnung der Parallelrechner.

Die Schleife wird solange durchlaufen, bis alle Parallelrechner ihr Ergebnis abgeliefert haben. Wenn die Ausgabezeit klein gegenüber der Berechnungszeit wird, dann müßte man darauf achten, daß man nicht versehentlich einen Parallelrechner zweimal abfragt, denn die Fertigmeldung wird nicht gelöscht. Um dem abzuhelfen, könnte man zum Beispiel in einer Liste verzeichnen, welcher Rechner schon ein Ergebnis geliefert hat. Hier ist es aber so, da die Rechner etwa gleich viel zu berechnen haben, daß spätestens nach der Ausgabe der Liste des ersten Rechners alle anderen auch fertig sind und damit jeder nur einmal in der Schleife abgefragt wird.

Bei der Initialisierung der Parallelrechner wird wieder anhand der Signatur entschieden, ob ein Parallelrechner vorhanden und geladen ist. Es wird dann die Anzahl der Kanten durch die Anzahl der Parallelrechner geteilt und jedem Parallelrechner etwa die gleiche Zahl von Kanten zugewiesen. Besser wäre noch eine Aufteilung in Punkte, da eine Kante aus vielen Punkten bestehen kann. Doch Optimierungen seien den Lesern überlassen.

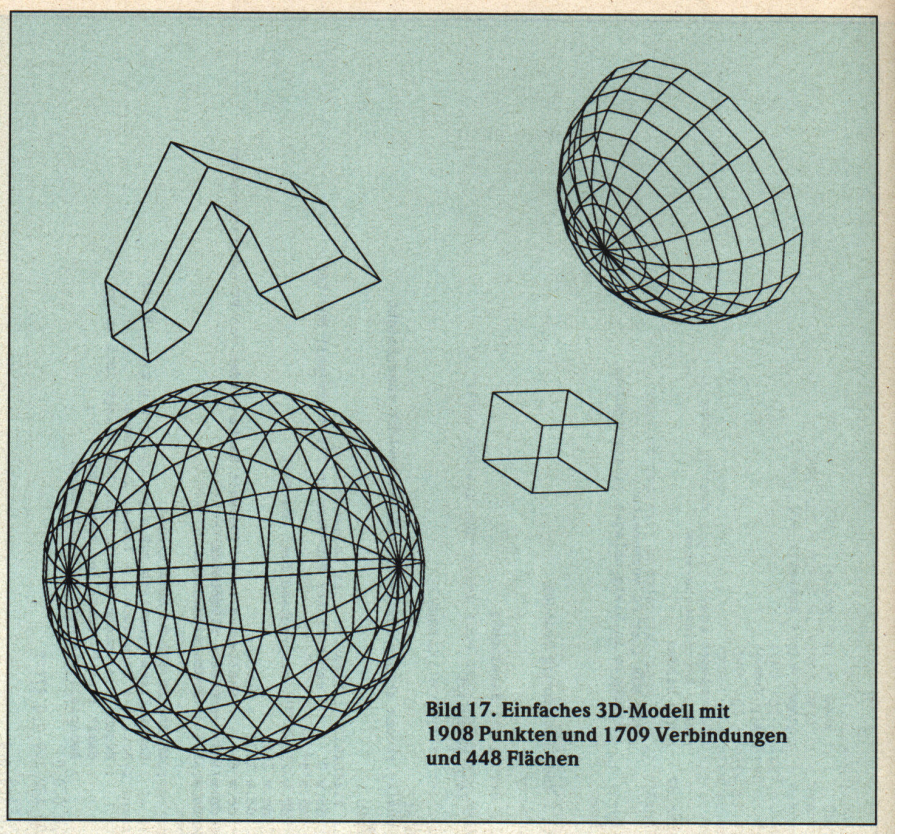

Bild 17. Einfaches 3D-Modell mit 1908 Punkten und 1709 Verbindungen und 448 Flächen

Das Ausgabeprogramm „bildpar" oder „bildaus" wird vom Hauptprogramm aus aufgerufen. Dabei kann man mit den Tasten 2,8,4,6 je eine Drehung in 10 Grad-Schritten um die y- oder x-Achse bestimmen. Mit den Tasten „+" und „–" kann der Skalierungsfaktor eingestellt werden. Dadurch ist Zoomen möglich.

Im Hauptprogramm wird die Datenliste eingelesen und in eine dynamische Datenstruktur umgewandelt, so daß die 64-KByte-Segmentgrenze nicht in Erscheinung tritt. Die Pointer werden allerdings der Einfachheit halber in festen Feldern gehalten und mit Konstanten, die am Anfang des Programms stehen, dimensioniert. Bild 14 zeigt das Parallelrechnerprogramm. Es hat die Aufgabe, die 3D-Transformationen und Projektion mit einer Kantenliste auszuführen. Die Ergebnisse werden als x,y – Paare wieder in diese Liste eingetragen. Dabei liegen die Koordinaten in x,y,z-Form mit MSB-LSB im Speicher vor und werden dazu vom Turbopascal-Programm mit dem Befehl SWAP entsprechend gedreht,was zeitunkritisch ist, da es nur einmal bei der Initialisierung geschehen muß. Die Ergebnisse werden in LSB-MSB Form in der Liste abgelegt und dazu von den Parallelrechnern gedreht. Rechenoperationen sollte man möglichst immer parallel ablaufen lassen, nur dadurch kann man sie beschleunigen. Hier wird wieder mit Shared-Variablen gearbeitet, da dies zu leichter modifizierbaren Programmen führt. Sie beanspruchen auch noch weniger Speicherplatz, da ansonsten eine ORG-Anweisung aus Sicherheitsgründen immer Lücken lassen müßte. Bild 15 zeigt die Include-Datei, wie sie der Assembler erzeugt hat. Bild 16 zeigt das Beispiel einer Make-Datei, wie man sie zur Erzeugung des Beispielprogramms verwenden kann.

Die Meßergebnisse

Bild 17 zeigt ein 3D-Modell, ungefähr mit 1900 Punkten, 1400 Verbindungen und 440 Flächen, und Bild 18 zeigt ein 3D-Modell mit 7133 Punkten und 1709 Verbindungen ohne Flächenelementen (die zur Mehrfachausgabe von Verbindungen führen können). *Bild 19* zeigt die Meßergebnisse. Die wesentliche Beschleunigung tritt schon bei einem einzigen Parallelrechner auf. Interessant ist hier, daß der Einsatz eines Co-Prozessors beim Original Design-CAD-Programm kaum eine Verbesserung in der Ausführungszeit ergibt. Das große Bei-

spiel kann nur mit einer 1MByte-EMS-Erweiterung auf dem PC mit Design-CAD-3D berechnet werden, sonst tritt Speicherüberlauf ein. Aus dem gleichen Grund kann es auch auf einem Parallelrechner allein nicht berechnet werden, obwohl dort die Speicherbelegung der Koordinaten durch Integerwerte wesentlich kompakter ist. Aus den Tabellen ergibt sich aber auch, daß der begrenzende Faktor die Bildausgabe ist. Weitere Parallelrechner helfen hier nicht, denn die Ausgabezeit ist für die Gesamtzahl der Punkte konstant. Will man die Parallelrechner weiter ausnützen, so muß man einen Graphik-Coprozessor einsetzen, der die Bildschirmausgabe beschleunigt. Solche Prozessoren gibt es, zum Beispiel den ACRTC von Hitachi, der auf einigen Bildschirmkarten für den professionellen Bereich eingesetzt wird. Der reine Datentransport, also die Kommunikation, ist noch nicht das Problem, denn man kann ja beim PCPAR direkt in den Speicher greifen und die Werte so schnell auslesen, als wären sie in einem normalen RAM gespeichert. Aber die Berechnung der Linien ist zeitaufwendig, denn das muß Punkt für Punkt durchgeführt werden, und dann müssen die Linien punktweise auf dem Bildschirm ausgegeben werden.

Tips zur Parallelisierung

Das Programm sollte zunächst auf dem PC selbständig ablaufen. Suchen Sie dann nach zeitkritischen Stellen. Der Code für diesen Teil muß dann in Assembler übersetzt werden. Überlegen Sie sich auch, wie man die Kommunikation mit dem Parallelrechner durchführen kann. Wenn die Parallelrechner Daten untereinander austauschen müssen, so wie es zum Beispiel bei der Finite-Elementmethode mit Multigrid nötig ist, muß der Hauptrechner den Datentransport übernehmen. Hier ist es aber immer möglich, mehrere Parallelrechner bei einem Schreibzugriff auf die gleiche Adresse einzublenden und so mit einem Zugriff mehrere Rechner zu erreichen.

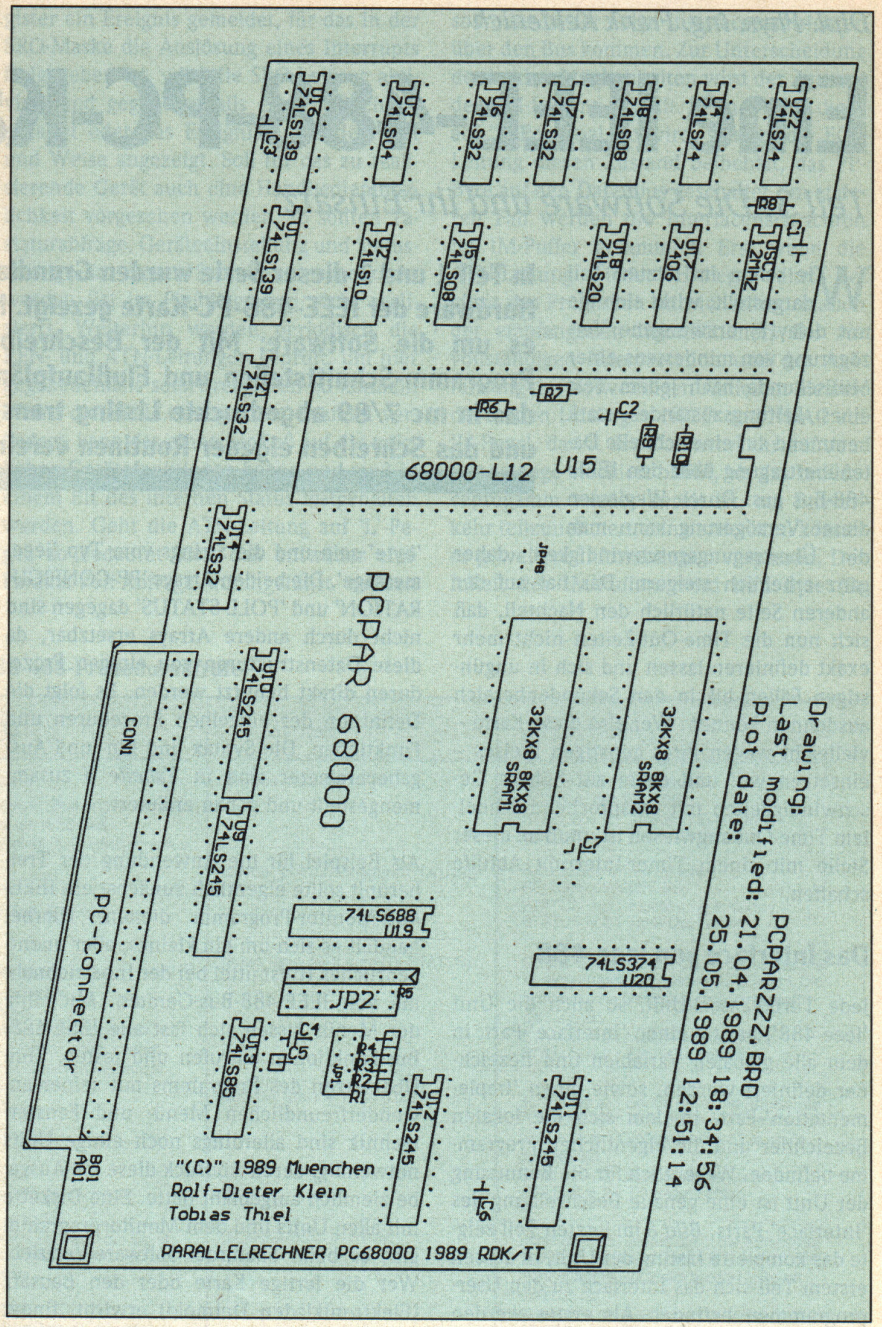

Bild 18. Komplexes Modell mit 7133 Punkten und 1709 Verbindungen ohne Flächenelemente

	7133 Punkte 1709 Verbindungen 0 Flächen		1908 Punkte 1402 Verbindungen 448 Flächen	
	286-10	386-16	286-10	386-16
Design-CAD-3D	-	75.10 sec	-	15.05 sec
Design-CAD-3D mit 80287	-	60.53 sec	-	12.42 sec
DC3-DISP ohne Par	14.8 sec	8.84 sec	7.41 sec	4.45 sec
mit 1 Parallelrechner	-	-	1.38 sec	1.05 sec
mit 2 Parallelrechner	2.25 sec	1.70 sec	1.15 sec	0.83 sec
mit 3 Parallelrechner	2.09 sec	1.53 sec	1.15 sec	0.77 sec
mit 4 Parallelrechner	2.03 sec	1.43 sec	1.15 sec	0.77 sec

Bild 19. Rechenzeiten für 3D-Anzeigeprogramm DC3DISP und Design-CAD-3D

Literatur

[1] The Massively Parallel Processor, edited by J. L. Potter, 1985. The MIT Press, Cambridge Massachusetts, London, England.

[2] Parallel MIMID Computation: The HEP Supercomputer and its Applications, edited by J. S. Kowalik, ISBN 0-262-16100-1. The MIT Press, Cambridge Massachusetts, London, England. ISBN 0-262-11101-2.

[3] 1988 International Conference of SUPERCOMPUTING, Conference Proceedings, ACM Press. ISBN 0-89791-272-1.