PVM und Express im Überblick

Peter Junglas

26. Januar 1994

Inhaltsverzeichnis

1 Einführung

Zielsetzung:

- Verknüpfung von Graphik-Workstations, Vektorrechnern und Parallelrechnern zu einem großen Parallelrechner, um unterschiedliche Resourcen für ein Programm zusammenzubringen,
- Zusammenschalten mehrerer (möglicherweise verschiedener) Workstations als "billigem Parallelrechner",
- Portabilität von parallelen Programmen.

1.1 Beispiele für Anwendungen

- Struktur-Analyse mit FEM
- Strömungsmechanik, Flugzeug-Design
- Molekül-Dynamik
- neuronale Netzwerke
- Simulation von Kernkraftwerken
- Computer-Tomographie
- Magneto-Hydrodynamik
- Hochenergiephysik
- Klimaforschung
- Implementierung verteilter Verfahren aus der linearen Algebra
- u.v.a.

1.2 Verschiedene Systeme

- PVM3:

  am weitesten verbreitetes Message-Passing-System
  für fast alle Workstations, Vektorrechner und einige
  Parallelrechner verfügbar
  einige auf PVM basierende Tools (Hence, Xab)

- Express:

  ähnliche Struktur wie PVM
  auch "höhere" Befehle
  bessere Tools (Debugger, div. Profiler)
  kommerziell (ParaSoft, Vertrieb: Genias)

- P4:

  unterstützt Message Passing und Shared Memory
  Weiterentwicklung von PARMACS-Makros von Argonne
  Abspaltung: PARMACS kommerziell, GMD

- TCGMSG:

  ähnlich zu P4
  benutzt Shared Memory für lokale Kommunikation
  hauptsächlich in der Theoretischen Chemie eingesetzt

- Linda:

  von David Gelernter et. al. (Yale) entwickelt
  anderes Konzept: Tupelspace (kein Message Passing)
  Compiler-Erweiterung (Macros) für C, Eiffel, PVM
  einige kommerzielle Anbieter

- Standardisierungsbemühung:
  MPI = Message Passing Interface
  Draft verfügbar

1.3 System-Überblick

- Programm-Modell:

  - Menge von wechselwirkenden Komponenten ("Tasks"), gleichartig oder verschieden, jede kann direkt mit jeder anderen kommunizieren
  - Verteilung von Tasks auf Prozessoren: transparent oder programmgesteuert

- Struktur:

- Kontroll-Dämon (pvmd3 bzw.exnetserv)

  einer bzw. vier auf jeder beteiligten Maschine
  Erzeugen und Vernichten von Tasks
  Weiterleiten von Messages

- Bibliothek (libpvm3.a bzw. EXPRESS-Bibliotheken)

  Task Management
  Kommunikation
  Information
  Synchronisation
  höhere Funktionen in EXPRESS (s.u.)

2 Ein einfaches Beispielprogramm

2.1 Master-Slave-Modell

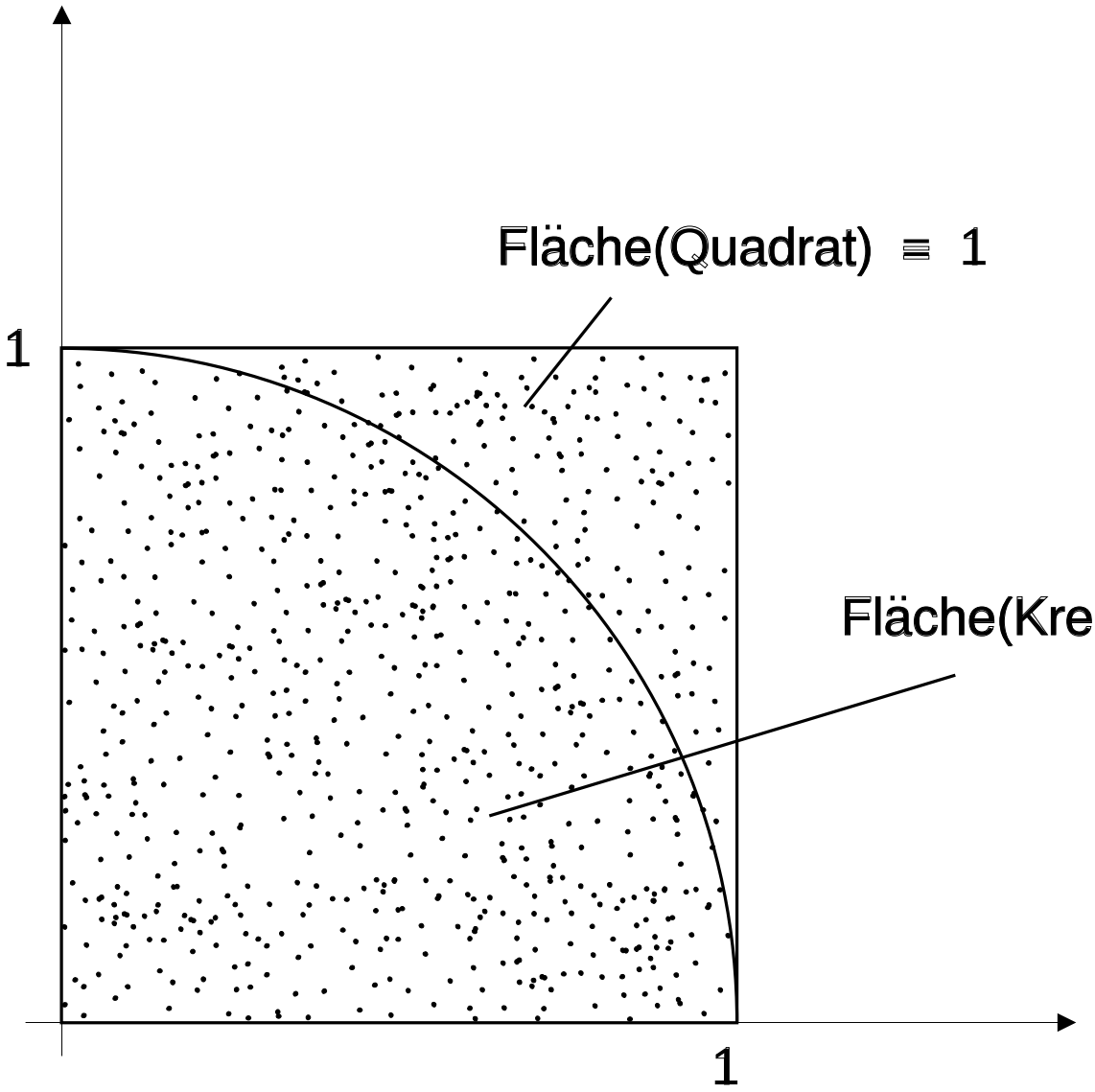
- Master:

  beim PVM-System anmelden
  Slaves starten
  Arbeit an die Slaves verteilen
  Teil-Ergebnisse der Slaves einsammeln
  Gesamt-Ergebnis berechnen und ausgeben

- Slave:

  beim PVM-System anmelden
  Arbeit vom Master empfangen
  Teil-Ergebnis berechnen
  Resultat an Master schicken
  beim PVM-System abmelden

2.2 Berechnung von Pi mit der "Montecarlo-Methode"



- Teilergebnis berechnen:

  bestimme N gleichverteilte Zufallspunkte im Quadrat
  [0, 1] x [0, 1]
  NT = Anzahl von Zufallspunkten im Viertelkreis
  $\pi \approx 4 * NT/N$

3 Arbeiten mit PVM

- PVM-Name der Maschinen-Architektur feststellen (z.B. SGI, HPPA, CNVX, RS6K, SUN4, ALPHA)

- Vorbereitungen beim ersten Mal: s. Script starter

- einfaches Hostfile = Liste der Rechnernamen:

```
indi1.cip3s
sg02.cip3s
convex.rz
anton.rz
```

  zusätzliche Optionen

- Konsolprogramm startet eine neue virtuelle Maschine oder verbindet sich mit der laufenden. Aufruf:

```
pvm [hostfile]
```

- wichtige Kommandos in pvm

help	Liste aller oder Erklärung eines Kommandos
add	Hinzufügen von Maschinen
delete	Entfernen von Maschinen
conf	Anzeige der Konfiguration der aktuellen Maschine
ps	Anzeige von Benutzer-Prozessen
reset	Beenden aller PVM-User-Prozesse
spawn	Starten von PVM-Programm (Ausgabe sammeln)
quit	Verlassen von pvm (virtuelle Maschine läuft weiter)
halt	Beenden der virtuellen Maschine

4 Erweiterte Möglichkeiten in EXPRESS

- I/O-Modell: single, multi und async mode

- Routinen zur automatischen Gebiets-Zerlegung
  ("domain decomposition")

- Routinen zur automatischen Verteilung von Daten

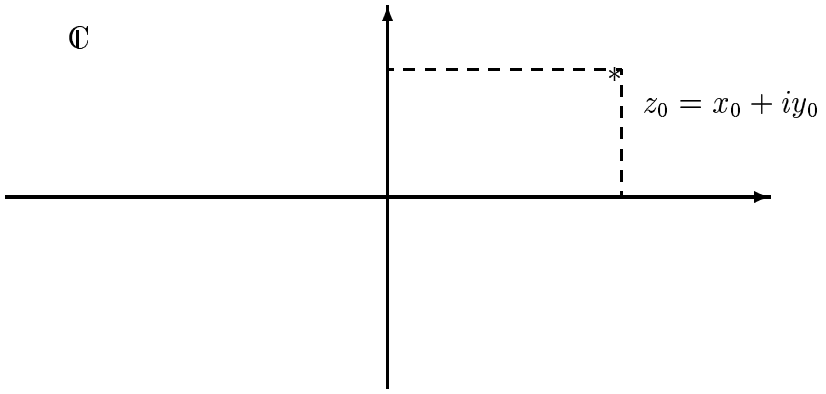
- Bibliothek für verteilte Graphik

- Tools (Debugger ndb, Profiler etool, ctool, xtool)

5 Arbeiten mit EXPRESS

- Beim ersten Mal: exsetup
- Maschine konfigurieren mit domtool
- Dämonen starten mit exinit
- Programm starten (eigenes oder mit cubix)
- Dämonen anhalten mit exclean

6 Paralleles Apfelmännchen mit EXPRESS



- Ausgehend von $z_0 \in \mathbb{C}$ bilde die Folge

$$z_n = z_{n-1}^2 + z_0$$

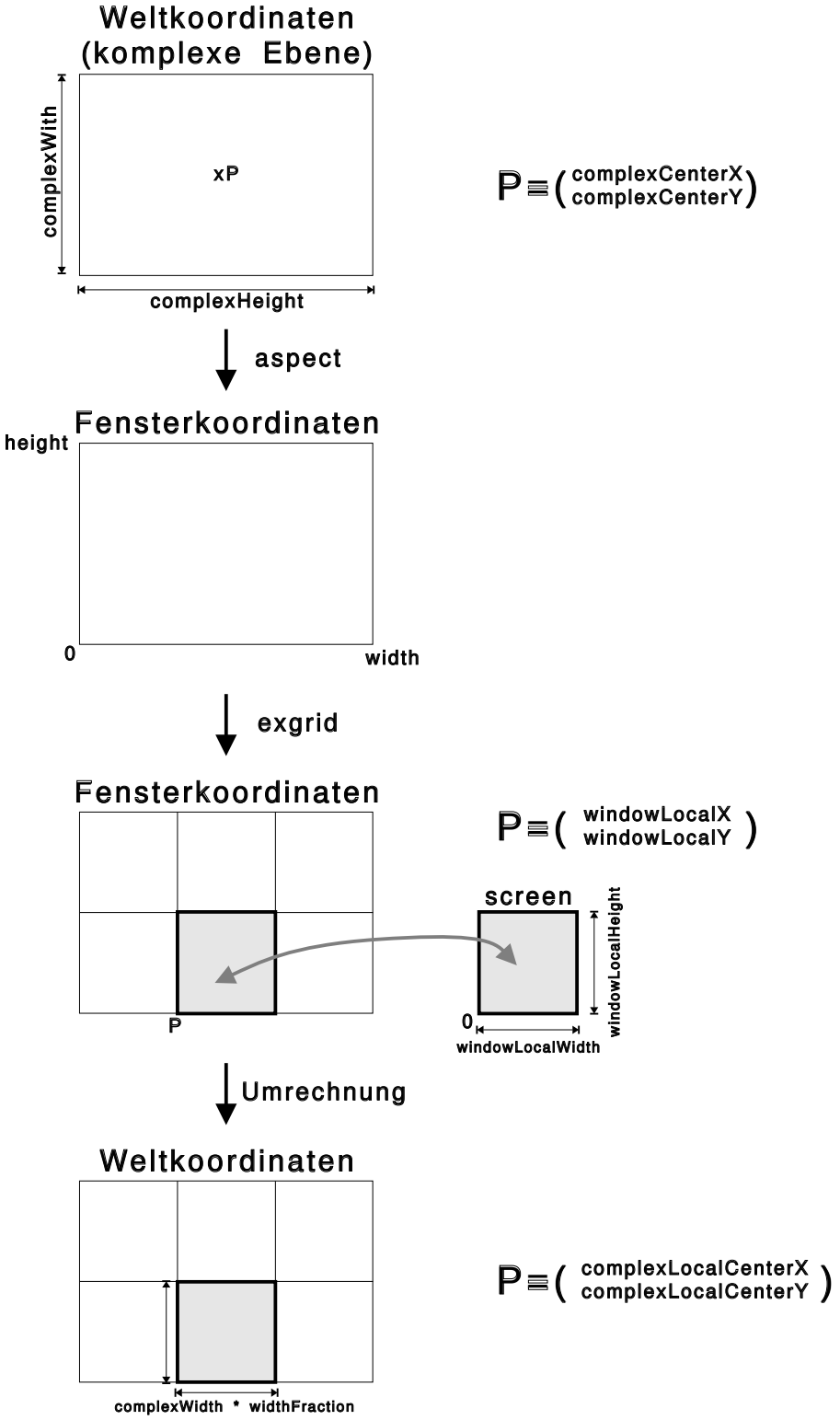
  Frage:
  Für welche z_0 bleibt $|z_n|$ beschränkt?
  (Mandelbrot-Menge, "Apfelmännchen")

- Satz:
  Ist $|z_n| > 2$ für ein $n \in \mathbb{N}$, dann ist die Folge (z_n) unbeschränkt.

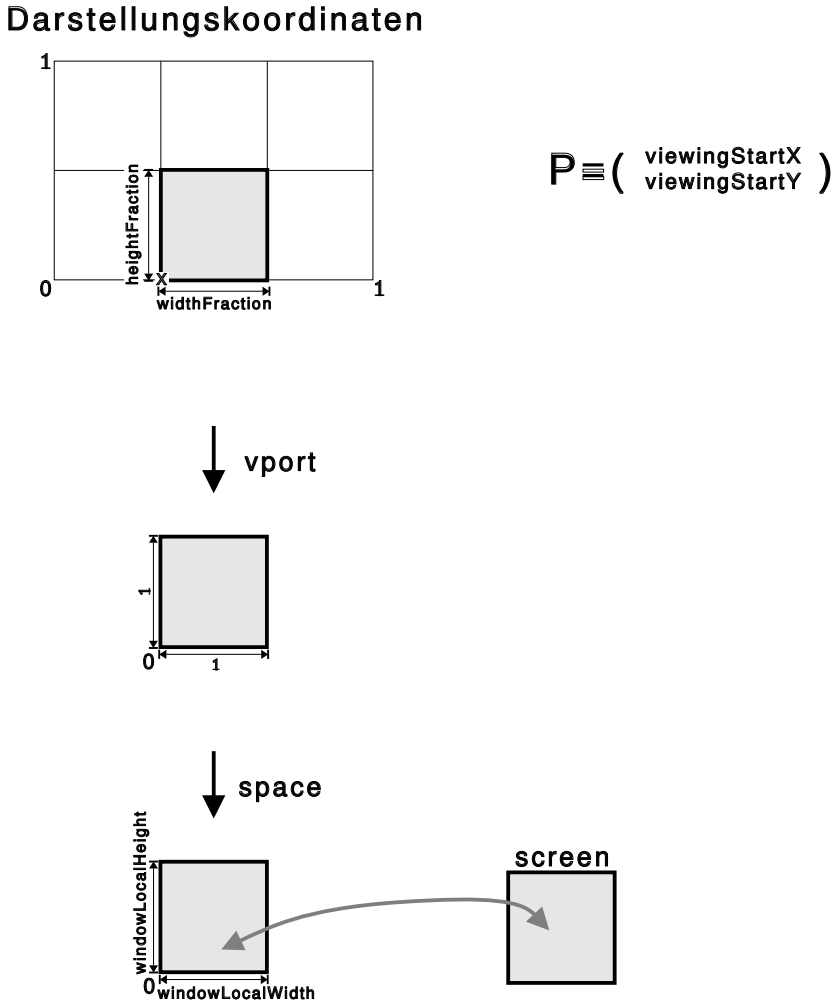
- Algorithmus:
  Bilde für z_0 die Folge $(z_n), n = 1, 2, 3, \ldots$ solange, bis $|z_n| > 2$ oder $n = NMAX$. Punkte z_0, für die NMAX erreicht wird, sind Kandidaten für die Mandelbrot-Menge.

- graphische Darstellung:
  Punkte z_0 mit Abbruch bei $n_0 < NMAX$ werden mit einer Farbe n_0 dargestellt.

6.1 Erläuterungen zum Programm mandel 1



6.2 Erläuterungen zum Programm mandel 2



6.3 Vorführung

- PVM:
  - heterogene Maschine starten
  - Programm pi
  - Programm lbpi
  - Programm xep
- EXPRESS:
  - homogene Maschine starten
  - Programm mandel
  - etool, ctool
  - Programm laplace

A Sourcen der Beispielprogramme

A.1 pi_m.c

```
/*
 *      pi_m.c
 *
 *      Einfaches Beispiel fuer PVM
 *
 *      Berechnung von PI durch ein "Montecarlo"-Verfahren:
 *      Jeder Rechen-Prozess wuerfelt eine gewisse Anzahl von Werten aus,
 *      der Master-Prozess sammelt die Ergebnisse ein.
 *
 *      Aufruf:
 *              pi [Anzahl]
 *
 *       wobei Anzahl die Gesamtzahl der auszuwuerfelnden Punkte angibt
 */

#include "pi.h"

void main(int argc, char *argv[]) {
  /*
   * Master-Hauptprogramm:
   * Starten der Rechen-Prozesse
   * Verteilen der Aufgaben und Einsammeln der Ergebnisse
   */

  long   anzahl;                        /* Gesamtzahl der Punkte */
  long   p_anz;                         /* Zahl der Punkte pro Prozessor */
  long   treffer_einzel;                /* Zahl der Treffer eines Prozesses */
  long   treffer_gesamt = 0;            /* Gesamtzahl der Treffer */
  double pi;                            /* Ergebnis */
  int    nproc;                         /* Gesamtzahl der Rechen-Prozesse */
  int    *tids;                         /* Array der TIDs */
  int    i;

  pvm_mytid();                          /* Beim pvmd anmelden */
  pvm_config(&nproc, NULL, NULL);       /* Zahl der Prozessoren erfragen */

  /* TID-Array allozieren */
```

```
  tids = (int *) malloc(nproc * sizeof(int));

  /* soviele Node-Programme wie Prozessoren starten */
  pvm_spawn(SLAVE, NULL, PvmTaskDefault, NULL, nproc, tids);

  /* Anzahl der zu wuerfelnden Punkte pro Prozessor */
  if (argc != 2) {
    anzahl = DEFAULT_ANZAHL;
  } else {
    anzahl = 1000 * atol(argv[1]);
  }
  p_anz = (long) ceil(anzahl/nproc);
  anzahl = nproc * p_anz;              /* falls es nicht aufgeht */

  /* Arbeit an alle Rechen-Prozesse verteilen */
  pvm_initsend(PvmDataDefault);
  pvm_pklong(&p_anz, 1, 1);
  pvm_mcast(tids, nproc, MSGANZAHL);

  /* Einsammeln und Zusammensetzen der Ergebnisse */
  for (i = 0; i < nproc; i++) {
    pvm_recv(-1, MSGRESULT);
    pvm_upklong(&treffer_einzel, 1, 1);
    treffer_gesamt += treffer_einzel;
  }

  /* Bestimmen und Ausgeben des Ergebnisses */
  pi = (double)treffer_gesamt/(double)anzahl*4;
  printf("\nPI = %lf\n", pi);

  /* beim pvmd abmelden */
  pvm_exit();
}
```

A.2 pi_s.c

```
/*
 *      pi_s.c
 *
 *   Berechnung von PI durch "Montecarlo-Methode":
```

```
 *   Jeder Rechen-Prozess wuerfelt eine Anzahl von Zahlen im Einheitsquadrat
 *   und gibt die Zahl der Treffer im Viertelkreis zurueck.
 *   PI ist dann Trefferzahl/Gesamtzahl * 4
 */

#include "pi.h"

void main(int argc, char *argv[]) {
  /*
   * Hauptprogramm des Rechen-Prozesses
   * erledigt die Kommunikation mit dem Master
   */

  int  mastertid;                          /* TID des Masters */
  long anzahl;                             /* Gesamtzahl der Punkte */
  long treffer;                            /* Zahl der Treffer */

  long calc(long anzahl);                  /* eigentliche Rechnung */

  /* beim pvmd anmelden */
  pvm_mytid();

  /* TID des Masters holen */
  mastertid = pvm_parent();

  /* Anzahl der Punkte vom Master empfangen */
  pvm_recv(mastertid, MSGANZAHL);
  pvm_upklong(&anzahl, 1, 1);

  /* Ergebnis ausrechen */
  treffer = calc(anzahl);

  /* Ergebnis an Master schicken */
  pvm_initsend(PvmDataDefault);
  pvm_pklong(&treffer, 1, 1);
  pvm_send(mastertid, MSGRESULT);

  pvm_exit();
}

#include <sys/types.h>
```

```
long calc(long anzahl) {
  /*
   * eigentliche Berechnung:
   * anzahl Zufallspunkte im Einheitsquadrat werden ausgewuerfelt
   * und die Zahl der Treffer im Viertelkreis (x*x + y*y < 1)
   * zurueckgegeben.
   */

  double  x, y;                         /* Zufallspunkt im Einheitsquadrat */
  long    treffer = 0;                  /* Anzahl der Treffer */
  int     i;

  /* Zufallszahlen-Generator initialisieren */
  srand(getpid());

  for(i=0; i<anzahl; i++) {
    x = ((double) rand())/RAND_MAX;
    y = ((double) rand())/RAND_MAX;

    if ( x*x + y*y <= 1.0 ) {
      treffer++;
    }
  }

  return(treffer);
}
```

A.3 pi.h

```
/*
 *   pi.h   - Header fuer die einfache Version des PI-Programms
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <pvm3.h>

#define SLAVE   "pi_s"              /* Name der Rechen-Programme */
#define MSGANZAHL  1
```

```
#define MSGRESULT  2

#define DEFAULT_ANZAHL 10000       /* Defaultwert fuer Anzahl */
```

A.4 mandel.c

```
/*
 *  mandel.c
 *
 *  computes mandelbrot set using EXPRESS
 *    adapted from Parasoft example program
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "express.h"

void main(void) {
  /* world coordinates */
  double complexCenterX;       /* total computing area        */
  double complexCenterY;
  double complexWidth;
  double complexHeight;
  double complexLocalCenterX;  /* local part of computing area  */
  double complexLocalCenterY;

  /* window coordinates */
  int height;                   /* total size of the screen window */
  int width;
  double heightFloat;           /* same as floats */
  double widthFloat;
  int windowLocalX;             /* local part of screen area */
  int windowLocalY;
  int windowLocalHeight;
  int windowLocalWidth;

  /* viewing coordinates */
  double viewingStartX;         /* local part of viewing area  */
  double viewingStartY;         /* (= fractions of total screen) */
  double heightFraction;
```

```
double widthFraction;

int *screen;                    /* Array with results */
struct nodenv nenv;             /* data structures for EXPRESS routines */
int global[2];
int local_start[2];
int local_sz[2];
int nprocs[2];


void mbrot_fill(int screen[], int windowWidth, int windowHeight,
                double complexCenterX, double complexCenterY,
                double complexWidth, double complexHeight);
void mbrot_display(int screen[], int wd, int ht);


printf("Real and imag. components of the center of the screen: ");
scanf("%lf %lf", &complexCenterX, &complexCenterY);
printf("Horizontal dimension of the screen in the complex plane: ");
scanf("%lf", &complexWidth);

/*
 * Open the graphical device and determine its
 * resolution
 */

openpl(DONTCARE, NULL);
aspect(&widthFloat, &heightFloat);
complexHeight = complexWidth * heightFloat/widthFloat;
height        = (int)heightFloat;
width         = (int)widthFloat;

/*
 * Call exparam, exgridinit and exgridsize to
 * determine what portion of the screen is to be
 * calculated in this processor.
 */

exparam(&nenv);
exgridsplit(nenv.nprocs, 2, nprocs);
exgridinit(2, nprocs);
```

```
global[0] = height;
global[1] = width;
exgridsize(nenv.procnum, global, local_sz, local_start);

/*
 * Allocate the array representing the screen
 */

windowLocalHeight = local_sz[0];
windowLocalWidth = local_sz[1];
screen = (int *)calloc(windowLocalHeight*windowLocalWidth, sizeof(int));

/*
 * Determine the coordinates of the center
 * controlled by this processor.
 */

windowLocalY   = local_start[0];
windowLocalX   = local_start[1];
heightFraction = (double)windowLocalHeight/heightFloat;
widthFraction  = (double)windowLocalWidth/widthFloat;
viewingStartX  = (double)windowLocalX/widthFloat;
viewingStartY  = (double)windowLocalY/heightFloat;
complexLocalCenterX = complexCenterX +
  complexWidth*(viewingStartX + (widthFraction - 1.)/2.);
complexLocalCenterY = complexCenterY +
  complexHeight*(viewingStartY + (heightFraction - 1.)/2.);

/*
 * Fill in the screen
 */

mbrot_fill(screen, windowLocalWidth, windowLocalHeight,
           complexLocalCenterX, complexLocalCenterY,
           complexWidth * widthFraction, complexHeight * heightFraction);

/*
 * Set up a viewport on the device so this
 * processor maps into the right place
 */
```

```
  vport(viewingStartX, viewingStartY,
        viewingStartX+widthFraction, viewingStartY+heightFraction);

  /*
   * Set up the graphics coordinate system to
   * correspond with pixels
   */

  space(0.0, 0.0, (double)windowLocalWidth,(double)windowLocalHeight);

  /*
   * Finally, render the image on the device.
   */

  mbrot_display(screen, windowLocalWidth, windowLocalHeight);

  getchar();
  closepl();
  exit(0);
}

/*
 * The following routine does the actual computation.
 */

void mbrot_fill(int screen[], int windowWidth, int windowHeight,
                double complexCenterX, double complexCenterY,
                double complexWidth, double complexHeight) {
  int i, j, N;
  double ptx, pty, zx, zy, mag;
  double orgx, orgy, incx, incy;

  /*
   * Figure out the lower left corner and screen
   * increments in the complex plane.
   */

  orgx = complexCenterX-0.5*complexWidth;
  orgy = complexCenterY-0.5*complexHeight;
  incx = complexWidth/(double)windowWidth;
  incy = complexHeight/(double)windowHeight;
```

```
  /*
   * Loop over positions in the display.
   */

  for(j=0; j<windowHeight; j++) {
    pty = orgy + incy*(double)j;
    for(i=0; i<windowWidth; i++) {
      ptx = orgx + incx*(double)i;
      N = 1;
      zx = ptx;
      zy = pty;
      do {
        mag = zx*zx + zy*zy;
        zx = zx*zx-zy*zy+ptx;
        zy = 2*zy*zx+pty;
        N++;
      } while(N<1000 && mag<4.0);
      *screen++ = N;
    }
  }

  return;
}

/*
 * This routine puts the pixels on the screen.
 * The caller has already setup the viewport
 * for the range (0,0) -> (wd,ht).
 */

void mbrot_display(int screen[], int wd, int ht) {
  int i, j, icol;

  for(j=0; j<ht; j++) {
    for(i=0; i<wd; i++) {
      icol = ((*screen) % 256);
      color(icol);
      marker(0, 0.5+(double)i,
             0.5+(double)j, 1.0);
      screen++;
```

```
    }
    asendplot();
  }
  return;
}
```