Parallel Programming for Solving Linear BVP's by Linear Superposition using CS_Tools

Bashir M.S. Khalaf
College of Education
University of Mosul, Mosul, Iraq

Firas M. Saeed
College of medicine
University of Mosul, Mosul, Iraq

Received on: 17/09/2012 **Accepted on: 30/01/2013**

ABSTRACT

The objective of this paper is to speed the solution of linear Boundary Value Problem (LPVPs) by parallel Programming by using cs_tools where the solution methods is the linear superposition Algorithm.

البرمجة المتوازية للحل لمسالة القيمة الحدودية الخطية باستخدام الطريقة فوق المركبة الخطية باستخدام CS-Tools

بشير محمد خلف
كلية التربية
جامعة الموصل، الموصل، العراق

فراس م سعيد
كلية الطب
جامعة الموصل، الموصل، العراق

تاريخ استلام البحث: 2012/09/17 **تاريخ قبول البحث: 2013/01/30**

الملخص

الهدف من هذا البحث هو زيادة سرعة حل مسائل القيم الحدودية بواسطة البرمجة المتوازية باستخدام cs_tools عندما تكون طريقة الحل هي الطريقة فوق المركبة.

1. Introduction

Theoretical techniques require practical Implementation to confirm their efficiency and validity: the latter is affected by the computer system available, especially when considering parallel algorithms. Practical problems of development or implementation e.g. the reliability and accessibility of the parallel systems can obviously offset the gain in execution time. Here, we have not attempted a comprehensive implementation of wide-ranging applications but a single straightforward problem for each of the three principal techniques presented. The Meiko equipment briefly is described in Appendix B [2].

In this paper, we discuss the programming and running of these parallel algorithms for the test of the problems on the Meiko Computing Surface by using CS-Tools, with C as the programming language. We have only considered the parallel solution of ODEs. This area of differential equations, is well known which consists of two main categories [1, 5 and 6]:

1) Initial value problems, and
2) Boundary value problems.

For the first category, there is no essential difference between the numerical solution of linear and non-linear cases, so that, we consider only the best of our parallel techniques for this category on the Meiko transputer system. We noted in reference [2] that the newly developed PBS technique offers several advantages over the other parallel techniques, such as:

1) It is completely parallel,

2) It is more effective in controlling the numerical stability of the numerical algorithms for initial value problems by virtue of integrating over small subintervals only.

This is, accordingly, the technique applied, and is considered in [2] .In contrast, there are various techniques available for solving either Linear or nonlinear boundary differential equations. Accordingly [2], we consider the parallel execution of the preferred parallel method for linear BVPs, namely, the parallel linear superposition algorithm, whereas in [2], we consider the parallel execution of the PS techniques for non-linear BVPs. The advantages of the PS techniques are:

1) They are completely parallel,

2) They are particularly effective in controlling the numerical instability of the solutions of the BVPs.

Before considering the parallel execution of these parallel techniques, it is essential to understand the problem of implementation to give a brief explanation of the Programmer's Guide to Sun-CS-Tools [5].

2. Introduction to Sun-CS-Tools [2, 3]

CS-Tools is a parallel program development system which runs on Sun workstations and supports parallel programming of In-Sun and Sun-hosted Computing Surface hardware, using standard C and FORTRAN only. It has four components:

1. A set of compilers,
2. A library of communications routines for C and FORTRAN,
3. A loader for distributed programs,
4. A runtime support environment.

2.1 Parallel Programming using CS-Tools [2, 3 and 5]

Here, we need to specify clearly the communicating processes model of parallel programming used in CS-Tools (further literature is available from meiko which illustrates, its use in a number of high computation application areas.

To make use of the communication processes approach, the programmer must first structure an application as a number of separate processes, each process being a conventional independent C program. The set of programs is constructed to work co-operatively on a single overall task. A key feature of the model is that processes communicate and synchronize only by means of message passing systems calls. The computing-surface hardware offers parallel processing in the form of a number of independent high performance microprocessors. A programmer makes use of this parallelism by arranging for different processes to execute on different processors. Inter-process communication and synchronization is provided entirely by system library calls. The programmer specifies which processes run on which processor by writing a simple text file which is read by mrun, the parallel network loader.

The Run Time Executive (RTE) provides operating system facilities to application programs running on any processor; where services cannot be provided locally they are referred to the Sun host machine.

The Computing Surface Network (CSN) provides a mechanism through which any application process is able to pass messages, apparently directly, to any other. It hides details of physical connectivity from the programmer: it supports the abstraction that messages are sent to and received from system-global message ports. The CSN handles the transfer of messages between these ports transparently to the user.

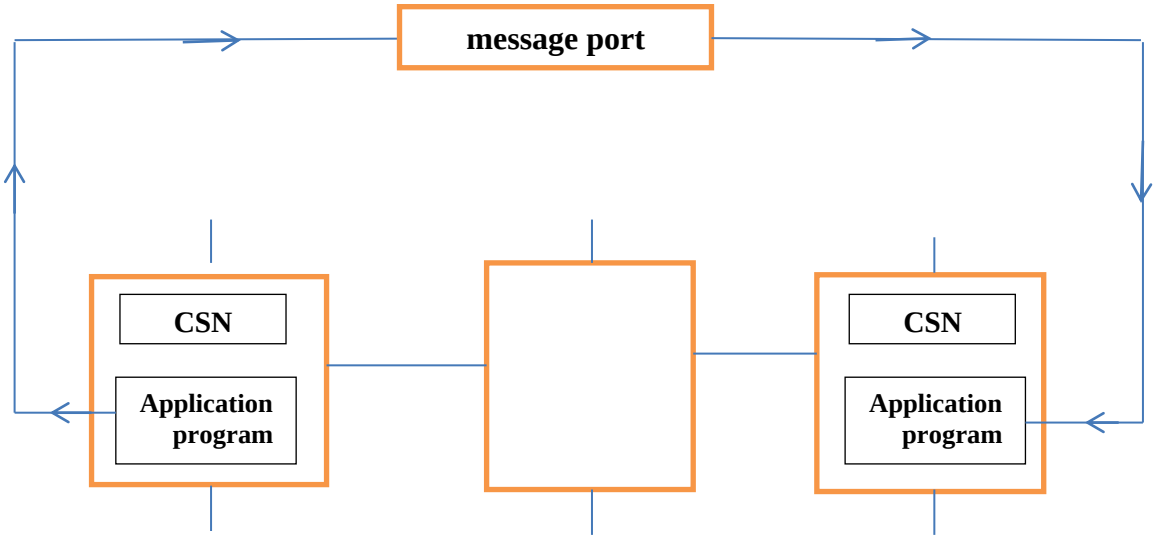

Figure (1). A model for sending messages through a port

A port may be referenced and used identically from any point in the network. Where it is necessary to transport messages between processors; this is done invisibly by the CSN. A port is created at run time by an application code fragment. On creation, each port is associated with a user defined name. Other code fragments may use this name to identify a destination address for messages. A port is a one way communication path only. Only one process creates each port from which it, then receives messages. Any number of processes may send message to it. Messages sent to ports can be automatically buffered and queued by the CSN. A block-until-received mode of operation is also supported.

A number of routines is provided to enable applications programmers to use ports from C programs. They include 'status reporting' and 'network control' as well as message-passing routines for various data types. Four of the basic message-passing routines, which exemplify the use of message ports are:

1) cs-createport ("name") creates a port and associates it with the user defined character string name. For example:

port1=cs_createport("portl");

creates the port portl.

2) cs_findport ("name".block) must be used by message senders before communication can commence with a port. For example:
portl-cs_findport("portl",1);
this can be used by the message sender to send messages to the port portl previously created with the name "portl". Block determines behaviour in the event that the port does not exist.
3) (void) cs_send(name, data, data-size, block) is used to send message data, held in a character buffer pointed to by data, to the port represented by the string name, data-size holds the number of bytes to be sent, block determines blocking behaviour (in a way similar to cs_findport).
4) (void) cs_recv(name,data,data-size) is used to receive a message from the port referenced by the string name. This message is copied into the buffer area pointed to by data, data-size is used to specify the maximum size of a message.

3. Running Parallel Linear Superposition on the Meiko Computing Surface using CS-Tools [2, 5 and 6]

The parallel linear superposition method requires at most (n+1) independent partial solutions of a system which consists of n first-order linear differential equations, followed by the solution of a linear system of at most n algebraic equations. The various cases of the distribution of the (n+1) partial solutions over the p independent processors of the computing system have been considered in detail in [2]. In this section, we consider the running of the linear parallel superposition algorithm on the School's particular transputer system, namely, the Meiko Computing Surface using CS-Tools; we discuss an example which is executed on the Computing Surface. The determination of the general solution of this particular example requires 5 partial solutions of the system in [1]. The general solution of the example is:

$$y(x)=y^*(x)+\sum_{i=1}^{4} A_i y_i\,(x),$$

Where y*(x)=PI (the particular integral).

Computing each of the partial solutions can be assigned to a single processor. A conventional C code is constructed for computing each partial solution by the Runge-Kutta-Merson algorithm. The respective codes which compute y1(x), y2(x), y3(x) and y4(x) are included in the files "superpyl.c", "superpy2.c", "superpy3.c" and "superpy4.c", and the code which computes y*(x) is included in the file "superppi. c". The last code also solves the linear algebraic system which determines A3 and A4 only, since the values of A1 and A2 can be estimated directly. The "superposition.par" file specifies the distribution of the executable files amongst the processors. The "superposition.par" file is composed of the following statements:

```
par
processor 1 superpyl
processor 2 superpy2
processor 3 superpy3
processor 4 superpy4
processor 5 superppi
endpar
```

3.1 The Mechanism of the Parallel Codes

The "superposition.par" file locates the code which computes, respectively, y1(x), y2(x), y3(x), y4(x) on the respective processor 1, 2, 3 and 4, and locates the code

which computes y*(x) and solves the linear algebraic equations on the processor 5. Accordingly the processors will operate as follows:

Processor 1:

Define ports for sending messages (data);

compute the values of y1(l), y' (l), y'' (l), y''' (l);

(i.e. y11(l), y21(l), y31(l), y41(1))

send the computed values to processor 5 through the corresponding ports;

Processor 2:

Define ports for sending messages (data);

compute the values of y2(1) ,y_2' (l), y_2'' (l),, y_2''' (l);

(i.e. y12(1),y22(1), y32(1), y42(1)

send the computed values to processor 5 through the corresponding ports; and similarly for Processor 3 and processor 4:

Processor 5:

Create ports for receiving the messages (data) from the processors 1, 2, 3.4;

compute the values of y*(l), $y^{*(1)}(1)$, $y^{*(2)}(1)$, $y^{*(3)}(1)$;

receive the sended values from the processors 1, 2, 3, 4; solve the linear algebraic system for A3 and A4;

The parallel codes of the parallel linear superposition algorithm are given in the appendix D.

REFERENCES

[1] Ibraheem K. and Kahalaf B. ”Shooting Neural Networks algorithm for Solving Boundary value problems in ODEs; Application and applied mathematics ,Vol.6 Issue 11 (June 2011), pp.1927-1941.

[2] Khalaf B. (1990), Parallel numerical algorithem for solving ODEs, Ph.D. thesis, School of computer studies, Leeds University.

[3] khalaf, B. and Al-nema, M. (2008). Generalize parallel Algorithms For BUPs in ODEs, The proceeding of the second conference on mathematics Sciences (CMS 2008) Jordan, pp.275-284.

[4] Meiko Scientific; "Meiko In-Sun Computing Surface Hardware"; Occam User Group Newsletter, No.12, January 1990, pp.98-99.

[5] Meiko LTD; "Hardware and Software Reference Manual"; Meiko LTD, 1987.

[6] Miklosko, J. and V.E. Kotov; "Algorithms, Software and Hardware of Parallel Computers"; Springer-Verlag, Berlin, 1984.

Appendix D
Parallel Codes For Parallel Linear Superposition

This Appendix lists the parallel codes of the parallel linear superposition for solving Example (5.5.1).

D.1.superpyl.c CODE (FILE)

This code determines the component of (Yl) of superposition method, that is yll,y21,y31 and y41.

```
#include<stdio.h>
# include ,<f/usr/include/cs .h"
#include<math.h>
main ()
{
Port portyll;
Port porty21;
Port porty31;
Port porty41;
void gl () ;
double exp(),t,tl,t2;
double rl,r2,r3,r4,yl[51
int i,sing,j;
portyll-cs_findport("porty11", 1);
porty21-cs_findport("porty2l",1);
porty31-c§_findport <"porty31", 1) ;
porty41-cs_findport("porty41",1);
tl-ticks();
gl(1.0,yl);
t2-ticks ();
(void) cs_send(portyll, (char *)fiyl[1] ,
sizeof(double),1);
(void) cs_send(porty21,(char *)iyl[2),sizeof(double),1);
(void) cs_send(porty3l,(char *)iyl[3],sizeof(double),1);
(void) cs_send(porty41,(char *)fiyl[4],sizeof(double),1);
h=0.0;
for(i=l;i<=6;++i)
{
gl(h,yl);
(void) cs_send(portyll, (char *)£yl[l],sizeof(double),1);
(void) cs_send(porty21,(char *)£yl[2],sizeof(double),1);
(void) cs_send(porty31,(char *)&yl[3],sizeof(double), 1);
(void)  cs_send(porty41,(char *)&yl[4],sizeof(double), 1);

h=h+0.2;
}
void gl(b,y) /^defines Yl*/
double b,y[5];
{
void rk();
rk(b,1.0,0.0,0.0,0.0,y) ;
}
```

```
void rk(b,al,a2,a3,a4,y)/*Runge-Kutta-Merson routine*/
double al,a2,a3,b,a4,y[5];
{
double kl[5],k2[5],k3[5],k4[5],k5[5],c[5];
double h=0.01,tt,  t,d,dl,cl,c3,q,exp(),f();
int i,n=4;
d=h/3.0; t=0.0;dl=9.0/8.0; cl=3.0/8.0,  q=0.5,c3=l.0/16.0;
y[l]=al;y[2]-a2;y[3]=a3;y(4]=a4;
while(t<b)
{
for(i=l;i<=n;++i)
kl[i]=d*f(i,t,y[1],y[2],y[3],y[4]);
for(i=1;i<=n;++i)
c[i]=y[i]+kl[i];
tt=t+d;
for(i=1;i<=n;++i)
k2[i]=d*f<i,tt,c[l],c[2],c[3],c[4]);
for(i=l;i<=n;++i)
c[i]=y[i]+q*(kl[i]+k2[i]);
for(i=l;i<=n;++i)
k3[i]=d*f(i,tt,c[l],c[2]
for(i=l;i<=n;++i)
c[i]=y[i]+dl*k3[i]+cl*kl[i];
tt=t+q*h;
for(i=l;i<=n;++i)
k4[i]=d*f(i,tt,c[l],c[2],c[3] ,c[4]) ;
for(i=l;i<=n;++i)
C[i]=y[i]+6.0*k4[i]-4.5*k3[i]+1.5*kl[i];
t=t+h;
for(i=1;i<=n;++i)
k5[i]=d*f <i,t,c[l],c[2],c[3],c[4]);
for(i=l;i<=n;++i)
y[i]=y[i]+0.5*(kl[i]+4.0*k4[i]+k5[i]);
 }
   }
double f (i,t,yl,y2,y3,y4)/*define the right hand side of
the   homogenous   differential   system   for   linear
superposition*/


int i;
double t,yl,y2,y3,y4;
double r,exp();
switch (i) {
case 1:
     r=y2;
break;
case 2:
     r=y3;
break;
```

```
case 3:
     r=y4;
breaks;
case  4 :
      r=0.0;
break;
}
return(r);
}
```

D.2.Superpy2.c CODE (FILE)

This code computes the component of (Y2) of superposition method, that is yl2, y22, y32 and y42.

```
#include<stdio.h>
#include "/usr/include/cs.h'
# include<math.h>
main()
Port  porty12.
Port porty22;
Port  porty32;
Port porty42,
void  g2 () ;
double rl,r2,r3,r4,y2[5],h;
int i.sing
portyl2=cs_findport("portyl2",l);
porty22=cs_findport("porty22",1);
porty32=cs_findport("porty32",1);
porty42=cs-findport("porty42", 1);
g2(1.0,y2);
(void) cs_send(portyl2, (char *)&y2[l],sizeof(double),1) ;
(void) cs_send(porty22,(char *)&y2[2],sizeof(double),1) ;
(void) cs_send(porty32,(char *)&y2[3],sizeof(double),1) ;
(void) cs_send(porty42,(char *)&y2[4],sizeof(double),1) ;
h=0.0;
for(i=l;i<=6;++i)
{
g2(h,y2);
(void) cs_send(portyl2,(char *)&y2[1],sizeof(double), 1);
(void) cs_send(porty22,(char *)&y2[2],sizeof(double), 1);
(void) cs_send(porty32,(char *)&y2[3],sizeof(double), 1);
(void) cs_send(porty42,(char *)&y2[4],sizeof(double), 1);
h=h+0.2;
      }
}

void g2(b,y)     /*defines Y2*/
double b,y[5];
{
void rk() ;
```

```
rk(b, 0.0,1.0,0.0,0.0,y) ;
}
void rk (b,al,a2,a3,a4,y)
/*same as in superpyl.c Code*/
double f<i,t,yl,y2,y3,y4)
/*same as in superpyl.c Code*/
```

D.3.superpy3.c CODE (FILE)

This computes the component of (Y3) of superposition method, that is y13, y23, y33 and y43.

```
#include<stdio.h>
#include "/usr/include/cs.h"
#include<math.h>
main ()
{
Port portyl3;
Port porty23;
Port porty33;
Port porty43;
void g3();
double rl,r2,r3,r4,y3[5],h;
int   i,sing;
portyl3=cs_f indport ("portyl3", 1);
porty23=cs_findport("porty23", 1) ;
porty33=cs_findport("porty33", 1) ;
porty43=cs_findport("porty43", 1) ;
g3 (1.0,y3);
(void) cs_send(portyl3,(char *)&y3[1],sizeof(double),1);
(void) cs_send(porty23,(char *)&y3[2],sizeof(double),1);
(void) cs_send(porty33,(char *)&y3[3],sizeof(double),1);
(void)  cs_send(porty43,(char *)&y3[4],sizeof(double),1);
h=0.0;
for (i=l;i<=6;++i)
{
g3(h,y3);
(void) cs_send(portyl3,(char *)&y3[1],sizeof(double),1);
(void) cs_send(porty23,(char *)&y3[2],sizeof(double),1);
(void) cs_send(porty33,(char *)&y3[3],sizeof(double),1);
(void) cs_send(porty43,(char *)&y3[4], sizeof(double),1);
h=h+0.2;
       }
    }
void g3(b,y)
double b,y[5);
{
void rk();
rk(b,0.0,0.0,1.0,0.0,y);
}
void rk(b,al,a2,a3,a4,y)
```

```
/*same as in superpyl.c*/
double f(i,t,yl,y2,y3,y4)
/*same as in superpyl.c*/
```

D.4.superpy4.c CODE (FILE)

This code computes the component of (Y4) superposition method. That is y14, y24, y34 and y44

```
#include<stdio.h>
#include "/usr/include/cs".
#include<math.h>
main ()
{
Port portyl4;
Port porty24;
Port porty34;
Port porty44;
void g4 ();
g4(1.0,y4);
(void) cs_send(portyl4,(char *)&y4[1],sizeof(double),1);
(void) cs_send(porty24,(char *)&y4[2], sizeof(double),1);
(void) cs_send(porty34,(char *)&y4[3], sizeof(double),1);
(void) cs_send(porty44,(char *)&y4[4],sizeof(double),1);
h=0.0;
for(i=l;i<=6;++i)
{
g4(h,y4);
(void) cs_send(portyl4,(char *)&y4[1],sizeof(double), 1);
(void) cs_send(porty24,(char *)&y4[2],sizeof(double),1);
(void) cs_send(porty34,(char *)&y4[3],sizeof(double),1);
(void) cs_send(porty44,(char *)&y4[4],sizeof(double),1);
h=h+0.2;
     }
 }
void g4 (b,y)      /*defines Y4*/
double b, y [5] ;
{
void rk () ;
rk(b,0.0,0.0,0.0,1.0,y);
}
void rk(b,al,a2,a3, a4,y)
/*same as in superpyl.c*/
double f(i,t,yl,y2,y3,y4)
/*same as in superpyl.c*/
```

D.5.superppi. c CODE (FILE)

This code computes the (PI) of superposition method, that is yl*, y2*,y3* and y4*

```
#include<stdio.h>
```

```
#include  "/usr/include/cs.h"
#include "/usr/usersb/pg/khalaf/gauselim.c"
#include<math.h>
main ()
{
Port portyll;
Port porty21;
Port porty31;
Port porty41;
Port portyl2;
Port porty32;
Port porty42;
Port portyl3;
Port porty23;
Port porty33;
Port porty43;
Port porty14;
Port porty24;
Port porty34;
Port porty44;
double a[14][14],b[14],x[14];
void g(),elimination(),backsubs();
double exp(),approxsol,t,tl,t2;
double h,y[5],yl[5],y2[5],y3[5],y4[5]
int i,sing,j;
portyll=cs_createport("portyll");
porty21=cs_createport("porty21");
porty31=cs_createport("porty31");
porty4l=cs_createport("porty41");
portyl2=cs_createport("porty12");
porty22=cs_createport("porty22");
porty32=cs_createport("porty32");
porty42=cs_createport("porty42");
portyl3=cs_createport("porty 13");
porty23=cs_createport("porty23");
porty33=cs__createport("porty33");
porty43=cs_createport("porty43");
portyl4=cs_createport("porty14");
porty24=cs_createport("porty24");
porty34=cs_createport ("porty34");
porty44=cs_createport(nporty44n);
tl=ticks();
g(1.0,y);
t2=ticks ();
(void) cs_recv(portyll,(char *)&yl[l],sizeof(double));
(void) cs_recv(porty21,(char *)&yl[2],sizeof(double));
(void) cs_recv(porty31,(char *)&yl[3],sizeof(double));
(void) cs_recv(porty41,(char *)&yl[4],sizeof(double));
(void) cs_recv(portyl2,(char *)&y2[1],sizeof(double));
```

```
(void) cs_recv(porty22,(char *)&y2[2],sizeof(double));
(void) cs_recv(porty32,(char *)&y2[3],sizeof(double));
(void) cs_recv(porty42,(char *)&y2[4],sizeof(double));
(void) cs_recv(porty13,(char *)&y3[1],sizeof(double));
(void) cs_recv(porty23,(char *)&y3[2],sizeof(double));
(void) cs_recv(porty33,(char *)&y3[3],sizeof(double));
(void) cs_recv(porty43,(char *)&y3[4],sizeof(double));
(void) cs_recv(portyl4,(char *)&y4[1],sizeof(double));
(void) cs_recv(porty24,(char *)&y4[2],sizeof(double));
(void) cs_recv(porty34,(char *)&y4[3],sizeof(double));
(void) cs_recv(porty44,(char *)&y4[4],sizeof(double));
t=ticks()-t2;
b[l]=1.0+exp(1.0)-yll]-ylll]-y2[l];
b[2]=4.0+exp(1.0)-y[2]-yl(2]-y2[2];
a[l][l]-y3[l];a[l][2]=y4[l];
a[2][l]-y3[2];a[2][2]=y4[2];
elimination(a,b,&sing, 2);
backsubs(a,b,x,2);
t2-ticks() ;
t=(t2-tl)*0.0000064;
h=0.0;
printf("   x      exact y    approx.y/n") ;
for(i=l;i<=6;++i)
{
g(h,y);
(void) cs_recv(portyll,(char *)&yl[1],sizeof (double));
(void) cs_recv(porty21,(char *)&yl[2],sizeof(double));
(void) cs_recv(porty31,(char *)&yl[3],sizeof(double));
(void) cs recv(porty41,(char *)&yl[4],sizeof(double));
(void) cs_recv(portyl2,(char *)&y2[1],sizeof(double));
(void) cs_recv(porty22,(char *)&y2[2],sizeof(double));
(void) cs_recv(porty32/(char *)&y2[3],sizeof(double));
(void) cs_recv(porty42,(char *)&y2[4],sizeof(double));
(void) cs_recv(portyl3,(char *)&y3[1],sizeof (double));
(void) cs_recv (porty23,(char *)&y3[2],sizeof(double));
(void) cs_recv (porty33,(char *)&y3[3],sizeof(double));
(void) cs_recv (porty43,(char *)&y3[41,sizeof(double));
(void) cs_recv (portyl4,(char *)&y4[1],sizeof(double));
(void) cs_recv (porty24,(char *)&y^[2],sizeof(double));
(void) cs_recv (porty34,(char *)&y4[3],sizeof(double));
(void) cs_ recv(porty44,(char *)&y4[4],sizeof(double));
approxsol=y[l]+yl[l]+y2[l]+x[l]*y3[l]+x[2]*y4[l];
printf("%10.10f   %10.13f
%10.13f\n",h,h*h*h*h+exp(h),approxol);
h=h+O.2;
  }
 }
void g(b,y)    /^defines the PI*/
double b,y[5];
(
```

```
void rk();
rk(b,0.0,0.0,0.0,0.0,y);
}
void rk(b,al,a2,a3,a4,y)
  /*same as in superpyl.c*/

double f(i,t,yl,y2,y3,y4)  /'defines the nonhomogenous
differential system*/
double t,yl,y2,y3,y4;
double r,exp();
switch(i)
 {
case 1:
    r=y2;
break;
case 2:
    r=y3;
break;
case 3:
   r=y4 ;
break;
case 4:
    r=24.0+exp(t);
break;
}
return(r);
}
```

D.6.superposition.par FILE

This file distributes the parallel codes of the linear superposition to the corresponding processors.

```
par
processor 11 superpyl
processor 12 superpy2
processor 13 superpy3
processor 14 superpy4
processor 15 superppi

endpar
```