Programowanie współbieżne

LABORATORIUM - PROGRAMY, FORTRAN

Andrzej Baran
baran@kft.umcs.lublin.pl

Programy I

`circle.f90` - *Prosty program.*

```
! Circle

MODULE Circle_Operations
IMPLICIT NONE
CONTAINS
FUNCTION Area(radius)
REAL :: Area
REAL, INTENT(IN) :: Radius
Area = Radius**2 * 3.14159
END FUNCTION Area
FUNCTION Circumference(radius)
REAL :: Circumference
REAL, INTENT(IN) :: Radius
Circumference = Radius * 2 * 3.14159
END FUNCTION Circumference
END MODULE Circle_Operations

PROGRAM Circle
USE Circle_Operations
IMPLICIT NONE
REAL :: r=5.0
```

Programy II

```
    PRINT *,"Area=",Area(r)
    PRINT *,"Circumference=",Circumference(r)
    END PROGRAM Circle
```

Programy III

`oporniki.f90` - *Moduły. Interfejsy. Przykład dodawania oporów.*

```
!oporniki

MODULE oporniki
! dodawanie oporow
! a.rowno.b    - rownolegle
! a.szereg.b   - szeregowo
! Zadanie. DOPISAC czesc zwiazana z zamiana jednostek.

  TYPE opornik
     REAL wartosc
     CHARACTER*4 miano
  END TYPE opornik

  INTERFACE OPERATOR (.SZEREG.)
     MODULE PROCEDURE SZ
  END INTERFACE
  INTERFACE OPERATOR (.ROWNO.)
     MODULE PROCEDURE RO
  END INTERFACE
```

Programy IV

```
CONTAINS

  FUNCTION SZ(a, b) RESULT(res)
    TYPE(opornik), INTENT(IN) :: a, b
    TYPE(opornik) res
    res%wartosc=a%wartosc+b%wartosc
    IF (a%miano==b%miano) THEN
       res%miano=a%miano
    ELSE
       res%miano="?"
    END IF
  END FUNCTION SZ
  FUNCTION RO(a, b) RESULT(res)
    TYPE(opornik), INTENT(IN) :: a, b
    TYPE(opornik) res
    REAL r
    r=1.0/a%wartosc+1.0/b%wartosc
    res%wartosc=1.0/r
    IF (a%miano==b%miano) THEN
       res%miano=a%miano
    ELSE
       res%miano="?"
    END IF
  END FUNCTION RO
```

Programy V

```
END MODULE OPORNIKI

program d
  use oporniki
  TYPE(opornik) a, b, c

  a%wartosc=21.0; a%miano="om"
  b%wartosc=87.0; b%miano="om"

  c = a.ROWNO.b
  write(*,*) 'rowno : ', c%wartosc, c%miano

  c = a.SZEREG.b
  write(*,*) 'szereg: ', c%wartosc, c%miano

end program d
```

Programy VI

`quaternions.f90` - *Kwaterniony.*

```
module Quaternions

  type, public :: quaternion
     real :: a, b, c, d
  end type quaternion

  intrinsic :: conjg

  private quat_mul_real, real_mul_quat, quat_mul_int, &
       int_mul_quat, quat_mul, quat_sub, quat_div, &
       quat_div_real, quat_div_int, quat_conjg

  interface operator (+)
     module procedure quat_add
  end interface

  interface operator (*)
     module procedure quat_mul_real
     module procedure real_mul_quat
```

Programy VII

```
      module procedure quat_mul_int
      module procedure int_mul_quat
      module procedure quat_mul
    end interface

    interface operator (-)
      module procedure quat_sub
    end interface

    interface operator (/)
      module procedure quat_div
!     module procedure quat_div_real
!     module procedure quat_div_int
    end interface

    interface conjg
      module procedure quat_conjg
    end interface

  contains

    function quat_add(x,y) result (res)
      type(quaternion), intent(in) :: x, y
      type(quaternion) :: res
```

Programy VIII

```
  res % a = x % a + y % a
  res % b = x % b + y % b
  res % c = x % c + y % c
  res % d = x % d + y % d
end function quat_add

function quat_sub(x,y) result (res)
  type(quaternion), intent(in) :: x, y
  type(quaternion) :: res
  res % a = x % a - y % a
  res % b = x % b - y % b
  res % c = x % c - y % c
  res % d = x % d - y % d
end function quat_sub

function quat_conjg(x) result (res)
  type(quaternion), intent(in) :: x
  type(quaternion) :: res
  res % a = x % a
  res % b = -(x % b)
  res % c = -(x % c)
  res % d = -(x % d)
end function quat_conjg
```

Programy IX

```
function quat_mul_real(x,r) result (res)
  ! quat * real
  type(quaternion), intent(in) :: x
  real, intent(in) :: r
  type(quaternion) :: res
  res % a = x % a *r
  res % b = x % b *r
  res % c = x % c *r
  res % d = x % d *r
end function quat_mul_real

function real_mul_quat(r,x) result (res)
  ! real * quat
  type(quaternion), intent(in) :: x
  real, intent(in) :: r
  type(quaternion) :: res
  res = quat_mul_real(x,r)
end function real_mul_quat

function int_mul_quat(i,x) result (res)
  ! integer * quat
  type(quaternion), intent(in) :: x
  integer, intent(in) :: i
  type(quaternion) :: res
```

Programy X

```
  res = quat_mul_real(x,real(i))
end function int_mul_quat

function quat_mul_int(x,i) result (res)
  ! quat * integer
  type(quaternion), intent(in) :: x
  integer, intent(in) :: i
  type(quaternion) :: res
  res = int_mul_quat(i,x)
end function quat_mul_int

function quat_norm(q) result(res)
  type(quaternion), intent(in) :: q
  real res
  real :: ap, bp, cp, dp
  ap = q % a
  bp = q % b
  cp = q % c
  dp = q % d
  res = sqrt(ap*ap+bp*bp+cp*cp+dp*dp)
end function quat_norm

function quat_div(x,y) result (res)
  type(quaternion), intent(in) :: x, y
```

Programy XI

```
  type(quaternion) :: res
  real r
  r = 1.0/quat_norm(y)**2
  res = quat_mul_real(x,r)
end function quat_div

! MULTIPLICATION
function quat_mul(x,y) result (res)
  type(quaternion), intent(in) :: x, y
  type(quaternion) :: res
  real r
  res%a=0.; res%b=0.; res%c=0.; res%d=0.
end function quat_mul

subroutine quaternion_print(q)
  type(quaternion), intent(in) :: q
  real :: ap, bp, cp, dp
  ap = q % a
  bp = q % b
  cp = q % c
  dp = q % d
  print "(a1,4f12.6,a1)", "(",ap,bp,cp,dp,")"
end subroutine quaternion_print
```

Programy XII

```
end module Quaternions
!
!-----------------------------------------------
! TEST
Program Quater
  use Quaternions
  type(quaternion) :: u, v, w
  u=quaternion(1,2,2,1)
  v=quaternion(4,3,2,1)
  w=u+v
  call quaternion_print(w)
  call quaternion_print(conjg(w))
  call quaternion_print(w/w)
  call quaternion_print(2.*w)
  call quaternion_print(w*2.)
  u = quaternion(1,0,0,0)
  print *, quat_norm(u)
end program Quater
```

Programy XIII

`derivatives.f90` - *Pierwsze i drugie pochodne.*

```
!
! Derivatives 1, 2 in 1-dim
!
module derivatives

CONTAINS

  subroutine FirstDeriv_1d(npts, dx, u, u_x)
    implicit none
    integer, intent(in) :: npts
    real, intent(in) :: dx
    real, dimension(:), intent(in) :: u
    real, dimension(:), intent(out) :: u_x
    real two_invdx
    integer i
    two_invdx = 1d0/(2d0*dx)
    ! central...
    do i=2,npts-1
       u_x(i)=(u(i+1)-u(i-1))*two_invdx
    end do
```

Programy XIV

```
  ! forward...
  u_x(1)=(-3.*u(1)+4.*u(2)-u(3))*two_invdx
  ! backward
  u_x(npts)=(3.*u(npts)-4.*u(npts-1)+u(npts-2))*two_invdx
end subroutine FirstDeriv_1d

subroutine SecondDeriv_1d(npts, dx, u, u_xx)
  implicit none
  integer, intent(in) :: npts
  real, intent(in) :: dx
  real, dimension(:), intent(in) :: u
  real, dimension(:), intent(out) :: u_xx
  real inv_dx2
  integer i
  inv_dx2=1d0/(dx*dx)
  ! forward...
  u_xx(1)=(2.*u(1)-5.*u(2)+4.*u(3)-u(4))*inv_dx2
  ! central...
  do i=2,npts-1
     u_xx(i)=(u(i+1)-2.*u(i)+u(i-1))*inv_dx2
  end do
  ! backward
  u_xx(npts)=(2.*u(npts)-5.*u(npts-1)+4.*u(npts-2)-u(npts-3))*inv_dx2
end subroutine SecondDeriv_1d
```

Programy XV

```
end module derivatives
```

Programy XVI

`TestDerivs.f90` - *Pochodne. Test.*

```
!    Test of Derivatives routines

program TestDerivs
  use derivatives
  implicit none
  real func, func_x, func_xx
  integer, parameter :: levels=10   ! liczba poziomow do testowania
  real, parameter :: a=0d0          ! dolna granica przedzialu
  real, parameter :: b=1d0          ! gorna granica przedzialu
  real, dimension(:), allocatable :: u, u_x, u_xx
  integer i, j, npts, iall
  real dx, ux_error, uxx_error

  write(*,'(a)') "    npts     Error(1_Der)        Error(2_Der)"
  do i=2,levels+1,1
     npts = 2**i              ! liczba punktów rowna jest 2^level
     dx = 1d0/(npts-1)        ! ustaw dx na podstawie liczby punktow

     ! rezerwuj pamiec, dynamicznie
     allocate(u   (npts), stat=iall)
```

Programy XVII

```
allocate(u_x (npts), stat=iall)
allocate(u_xx(npts), stat=iall)

! function
do j=1,npts
   u(j) = func((j-1)*dx)
end do

call FirstDeriv_1d(npts, dx, u, u_x)
call SecondDeriv_1d(npts, dx, u, u_xx)

! error
ux_error  = 0d0
uxx_error = 0d0
do j=1,npts
   ux_error  = ux_error  + dx*(u_x (j)-func_x ((j-1)*dx))**2
   uxx_error = uxx_error + dx*(u_xx(j)-func_xx((j-1)*dx))**2
end do

write(*,'(i8, 2e20.9)') npts, sqrt(ux_error), sqrt(uxx_error)

! zwolnij pamiec
deallocate(u, u_x, u_xx)
```

Programy XVIII

```
  end do ! i

end program TestDerivs

real function func(x)
  func = x*x*x*x
end function func

real function func_x(x)
  func_x = 4d0*x*x*x
end function func_x

real function func_xx(x)
  func_xx = 12d0*x*x
end function func_xx

! wyniki obliczeń:
!
!    npts      Error(1_Der)          Error(2_Der)
!       4      0.448540793E+00       0.200411100E+01
!       8      0.696229028E-01       0.242946550E+00
!      16      0.130890098E-01       0.367216199E-01
!      32      0.276197408E-02       0.616467539E-02
!      64      0.627306127E-03       0.110691791E-02
```

Programy XIX

```
!      128      0.148910393E-03      0.211078793E-03
!      256      0.362347450E-04      0.428963548E-04
!      512      0.893428971E-05      0.929152239E-05
!     1024      0.221800198E-05      0.212429174E-05
!     2048      0.552552792E-06      0.504635577E-06
```

Programy XX

`hello.f90` - *Program demonstracyjny* `hello` *(Burkardt).*

```
program main

!*****************************************************************************8
!
!! MAIN is the main program for HELLO.
!
!  Discussion:
!
!    HELLO is a simple MPI test program.
!
!    Each process prints out a "Hello, world!" message.
!
!    The master process also prints out a short message.
!
!  Licensing:
!
!    This code is distributed under the GNU LGPL license.
!
!  Modified:
!
```

Programy XXI

```
!    30 October 2008
!
!  Author:
!
!    John Burkardt
!
!  Reference:
!
!    William Gropp, Ewing Lusk, Anthony Skjellum,
!    Using MPI: Portable Parallel Programming with the
!    Message-Passing Interface,
!    Second Edition,
!    MIT Press, 1999,
!    ISBN: 0262571323,
!    LC: QA76.642.G76.
!
   use mpi
!
! implicit none
!
  integer error
  integer id
  integer p
  real ( kind = 8 ) wtime
```

Programy XXII

```
!
!  Initialize MPI.
!
  call MPI_Init ( error )
!
!  Get the number of processes.
!
  call MPI_Comm_size ( MPI_COMM_WORLD, p, error )
!
!  Get the individual process ID.
!
  call MPI_Comm_rank ( MPI_COMM_WORLD, id, error )
!
!  Print a message.
!
  if ( id == 0 ) then

    wtime = MPI_Wtime ( )

    write ( *, '(a)' ) ' '
    write ( *, '(a)' ) 'HELLO_WORLD - Master process:'
    write ( *, '(a)' ) '  FORTRAN90 version'
    write ( *, '(a)' ) ' '
    write ( *, '(a)' ) '  An MPI test program.'
```

Programy XXIII

```
      write ( *, '(a)' ) ' '
      write ( *, '(a,i8)' ) '  The number of processes is ', p
      write ( *, '(a)' ) ' '

    end if

    write ( *, '(a)' ) ' '
    write ( *, '(a,i8,a)' ) '  Process ', id, ' says "Hello, world!"'

    if ( id == 0 ) then

      write ( *, '(a)' ) ' '
      write ( *, '(a)' ) 'HELLO_WORLD - Master process:'
      write ( *, '(a)' ) '  Normal end of execution: "Goodbye, world!".'

      wtime = MPI_Wtime ( ) - wtime
      write ( *, '(a)' ) ' '
      write ( *, '(a,g14.6,a)' ) '  Elapsed wall clock time = ', wtime, ' seconds

    end if
!
!  Shut down MPI.
!
    call MPI_Finalize ( error )
```

Programy XXIV

```
  stop
end
```

Programy XXV

`FirstDeriv1dp.f90` - *Pochodne. Wersja równoległa*

```
!
! Derivative 1, MPI
!
subroutine FirstDeriv1dp(npts, dx, u, u_x, mynode, totalnodes)
  implicit none
  integer, intent(in) :: npts, mynode, totalnodes
  real*8, intent(in) :: dx
  real*8, dimension(:), intent(in) :: u
  real*8, dimension(:), intent(out) :: u_x
  real*8 two_invdx
  MPI_STATUS status
  two_invdx = 1d0/(2d0*dx)

  if(mynode == 0) then
     ! forward...
     u_x(1) = (-3.*u(1)+4.*u(2)-u(3))*two_invdx
  end if

  if(mynode == totalnodes) then
     ! backward
```

Programy XXVI

```
   u_x(npts) = (3.*u(npts)-4.*u(npts-1)+u(npts-2))*two_invdx
end if

do i=2,npts-1
   ! central...
   u_x(i) = (u(i+1)-u(i-1))*two_invdx
end do

if (mynode == 0) then
   mpitemp = u(npts)
   call MPI_SEND(mpitemp, 1, MPI_DOUBLE, 1, 1, MPI_COM_WORLD)
   call MPI_RECV(mpitemp, 1, MPI_DOUBLE, 1, 1, MPI_COM_WORLD, status)
   u_x(npts) = (mpitemp-u(npts-1))*two_invdx
else if (mynode == totalnodes) then
   call MPI_RECV(mpitemp, 1, MPI_DOUBLE, mynode, 1, MPI_COM_WORLD, status)
   u_x(1) = (u(2)-mpitemp)*two_invdx
   mpitemp = u(1)
   call MPI_SEND(mpitemp, 1, MPI_DOUBLE, mynode, 1, MPI_COM_WORLD)
else
   call MPI_RECV(mpitemp, 1, MPI_DOUBLE, mynode, 1, MPI_COM_WORLD, status)
   u_x(1) = (u(2)-mpitemp)*two_invdx
   mpitemp = u(1)
   call MPI_SEND(mpitemp, 1, MPI_DOUBLE, mynode-1, 1, MPI_COM_WORLD)
   mpitemp = u(npts)
```

Programy XXVII

```
        call MPI_SEND(mpitemp, 1, MPI_DOUBLE, mynode+1, 1, MPI_COM_WORLD)
        call MPI_RECV(mpitemp, 1, MPI_DOUBLE, mynode+1, 1, MPI_COM_WORLD, status)
        u_x(npts) = (mpitemp-u(npts-1))*two_invdx
    end if
end subroutine FirstDeriv1dp
```

Programy XXVIII

FirstDeriv1dpTest.f90 - *Pochodne. Test.*

```
! przykład użycia procedur SENDRECV i SEND-NDRECV_REPLACE
!
integer mynode, totalnodes, mpierr
integer datasize ! liczba jednostek danych send/recv
integer process1, process2 ! rangi procesów wymieniających dane
integer tag1, tag2 ! znacznik komunikatu
real*8, dimension(:), allocatable :: buff, sendbuff, recvbuff
MPI_status status ! zmienna przechowujaca informacje o stanie

call MPI_INIT( ierr )
call MPI_COMM_SIZE(MPI_COM_WORLD, totalnodes, mpierr)
call MPI_COMM_RANK(MPI_COM_WORLD, mynode, mpierr)

! rezerwacja buforow danych, process1, process2
allocate(sendbuff(datasize))
allocate(recvbuff(datasize))
allocate(buff(datasize))

if (mynode == process1) then
   ! dane w sendbuff są wysylane do process2, odbierane od process2
```

Programy XXIX

```
   ! dane trafiają do bufora recvbuff
   call MPI_SENDRECV(sendbuff, datasize, MPI_DOUBLE, process2, &
        tag1, recvbuff, datasize, MPI_DOUBLE, process2, &
        tag2, MPI_COMM_WORLD, status)
   ! wywolanie powoduje wymiane (swap) zawartosci bufora buff
   ! z process2
   call MPI_SENDRECV_REPLACE(buff, datasize, MPI_DOUBLE, process2, &
        tag1, process2, tag2, MPI_COMM_WORLD, status)
end if

if (mynode == process2) then
   ! zawartosc sendbuff jest przekazana do procesu process1, a dane
   ! trzymane od procesu process1 sa umieszczane w buforze recvbuff
   call MPI_SENDRECV(sendbuff, datasize, MPI_DOUBLE, process1, &
        tag2, recvbuff, datasize, MPI_DOUBLE, process1, &
        tag1, MPI_COMM_WORLD, status)
   ! wymiana danych buff z procesem process1
   !
   call MPI_SENDRECV_REPLACE(buff, datasize, MPI_DOUBLE, process1, &
        tag2, process1, tag1, MPI_COMM_WORLD, status)
end if

! process1 ma w swoim buforze recvbuff zawartosc bufora sendbuff
! procesu process1; process2 w swoim buforze recvbuff ma zawartosc
```

Programy XXX

```
! buforu sendbuff procesu process1; zawartosc buforow buff obu
! procesów zostala zamieniona
```

Programy XXXI

`primes.f90` - *Poszukiwanie liczb pierwszych*

```
program primes
  !
  ! Program zlicza liczby pierwsze w przedziale
  ! (kmin, kmax); podaje ich ilosc i czas zliczania
  !
  implicit none
  integer, parameter :: kmin=1, kmax=1000
  integer numberOfPrimes, countPrimes
  real*4 startingTime, elapsedTime

  ! pomiar czasu
  startingTime=secnds(0.0)

  numberOfPrimes = countPrimes(kmin, kmax)
  print *, numberOfPrimes

  ! czas wykonania
  elapsedTime=secnds(startingTime)  ! elapsed time
  print *, "Time = ", elapsedTime
```

Programy XXXII

```
end program primes


function countPrimes(kmin, kmax) result(count)
  !
  ! procedura zlicza liczby pierwsze w przedziale
  ! (kmin, kmax) i zwraca ich liczbe
  !
  implicit none
  integer, intent(in) :: kmin, kmax
  integer number, count
  logical isPrime
  count = 0
  do number=kmin, kmax
     if (isPrime(number)) count = count + 1
  end do
end function countPrimes


function isPrime(number)
  !
  ! funkcja sprawdza, czy liczba jest pierwsza
  ! i zwraca .true. jesli jest lub .false.
  !
```

Programy XXXIII

```
    logical isPrime
    integer, intent(in) :: number
    integer k, limit

    limit = int(sqrt(float(number)))+1
    isPrime = .true.
    do k=2, limit
       if (mod(number,k) == 0) then
          isPrime = .false.
          return
       end if
    end do
  end function isPrime
```

Programy XXXIV

`matvec.f90` - *Mnożenie macierzy przez wektor*

```
! MPI Tutorial
! Dr. Andrew C. Pineda, HPCERC/AHPCC
! Dr. Brian Smith, HPCERC/AHPCC
! The University of New Mexico
! November 17, 1997
! Last Revised: September 18, 1998

program matvec2
  ! Perform matrix vector product -- Y = AX
  ! This is method two -- distribute A by block columns
  ! and X in blocks (of rows) and the partial vector sum of Y is on
  ! each processor.
  include 'mpif.h'
  integer, parameter :: dim1 = 80, dim2 = 10, dim3 = dim1*dim2
  integer ierr, rank, size, root, i, j
  integer sec_start, nano_start
  integer sec_curr, nano_curr
  integer sec_startup, nano_startup
  integer sec_comp, nano_comp
  integer sec_cleanup, nano_cleanup
```

Programy XXXV

```
real, dimension(dim1,dim1) :: a
real, dimension(dim1,dim2) :: apart
real, dimension(dim1) :: x, y, ypart
real, dimension(dim2) :: xpart
interface
   subroutine posix_timer(job_sec, job_nanosec)
     integer job_sec, job_nanosec
   end subroutine posix_timer
end interface
root = 0
call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, rank, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, size, ierr )
print *, 'START process on processor ', rank
if( rank == root ) then
   call posix_timer(sec_start, nano_start)
   do i = 1, dim1
      x(i) = 1.0
      do j = 1, dim1
         a(j,i) = i + j
      enddo
   enddo
endif
!  Distribute the 80x80 array A by columns as
```

Programy XXXVI

```
! 80x10 blocks stored in APART.
! Distribute the 80 dimensional
! array x in blocks of length 10
call MPI_SCATTER( a, dim3, MPI_REAL, apart, dim3, MPI_REAL, root,&
     & MPI_COMM_WORLD, ierr )
call MPI_SCATTER( x, dim2, MPI_REAL, xpart, dim2, MPI_REAL, root,&
     & MPI_COMM_WORLD, ierr )
if( rank == root ) then
   call posix_timer(sec_curr, nano_curr)
   sec_startup = sec_curr - sec_start
   nano_startup = nano_curr - nano_start
   sec_start = sec_curr
   nano_start = nano_curr
endif
do j = 1, dim1
   ypart(j) = 0.0
enddo
do i = 1, dim2
   do j = 1, dim1
      ypart(j) = ypart(j) + xpart(i)*apart(j,i)
   enddo
enddo
if( rank == root ) then
   call posix_timer(sec_curr, nano_curr)
```

Programy XXXVII

```
      sec_comp = sec_curr - sec_start
      nano_comp = nano_curr - nano_start
      sec_start = sec_curr
      nano_start = nano_curr
   endif
   call MPI_REDUCE( ypart, y, dim1, MPI_REAL, MPI_SUM, root, &
        MPI_COMM_WORLD, ierr )
   if( rank == root ) then
      call posix_timer(sec_curr, nano_curr)
      sec_cleanup = sec_curr - sec_start
      nano_cleanup = nano_curr - nano_start
   endif
   print *, 'Finish processor ', rank
   if( rank == root ) then
      print *, 'Matrix vector product, elements 10 and 60, are: ',&
           & y(10), y(60)
      print *, 'Startup execution times (sec, nano): ',&
           & sec_startup, nano_startup
      print *, 'Computation execution times (sec, nano): ',&
           & sec_comp, nano_comp
      print *, 'Cleanup execution times (sec, nano): ',&
           & sec_cleanup, nano_cleanup
   endif
   call MPI_FINALIZE( ierr )
```

Programy XXXVIII

```
end program matvec2
```

Programy XXXIX

gausselim.f90 - *Metoda Gaussa eliminacjii*

```
! MPI Tutorial
! Dr. Andrew C. Pineda, HPCERC/AHPCC
! Dr. Brian Smith, HPCERC/AHPCC
! The University of New Mexico
! November 17, 1997
! Last Revised: September 18, 1998

module GaussianSolver
  implicit none

  ! The default value for the smallest pivot that will be accepted
  ! using the GaussianSolver subroutines. Pivots smaller than this
  ! threshold will cause premature termination of the linear equation
  ! solver and return false as the return value of the function.
  real, parameter :: DEFAULT_SMALLEST_PIVOT = 1.0e-6

contains

  ! Use Gaussian elimination to calculate the solution to the linear
  ! system, A x = b. No partial pivoting is done. If the threshold
```

Programy XL

```
! argument is present, it is used as the smallest allowable pivot
! encountered in the computation; otherwise, DEFAULT_SMALLEST_PIVOT,
! defined in this module, is used as the default threshold. The status
! of the computation is a logical returned by the function indicating
! the existence of a unique solution (.true.), or the nonexistence of
! a unique solution or threshold passed (.false.).
! Note that this is an inappropriate method for some linear systems.
! In particular, the linear system, M x = b, where M = 10e-12 I, will
! cause this routine to fail due to the presence of small pivots.
! However, this system is perfectly conditioned, with solution x = b.
function gaussianElimination( A, b, x, threshold )
  implicit none
  logical gaussianElimination
  real, dimension( :, : ), intent( in ) :: A ! Assume the shape of A.
  real, dimension( : ), intent( in ) :: b ! Assume the shape of b.
  real, dimension( : ), intent( out ) :: x ! Assume the shape of x.
  ! The optional attribute specifies that the indicated argument 40
  ! is not required to be present in a call to the function. The
  ! presence of optional arguments, such as threshold, may be checked
  ! using the intrinsic logical function, present (see below).
  real, optional, intent( in ) :: threshold
  integer i, j ! Local index variables.
  integer N ! Order of the linear system.
  real m ! Multiplier.
```

Programy XLI

```
real :: smallestPivot = DEFAULT_SMALLEST_PIVOT
! Pointers to the appropriate rows of the matrix during the elmination.
real, dimension( : ), pointer :: pivotRow
real, dimension( : ), pointer :: currentRow
! Copies of the input arguments. These copies are modified during
! the computation. The target attribute is used to indicate that
! the specified variable may be the target of a pointer. Rows of
! ACopy are targets of pivotRow and currentRow, defined above.
real, dimension( size( A, 1 ), size( A, 2) ), target :: ACopy
real, dimension( size( b ) ) :: bCopy
!
! Status of the computation. The return value of the function.
!
logical successful

!
! Change the smallestPivot if the threshold argument was included.
!
if ( present( threshold ) ) smallestPivot = abs( threshold )
!
! Setup the order of the system by using the intrinsic function size.
! size returns the number of elements in the specified dimension of
! an array or the total number of elements if the dimension is not
! specified. Also assume that a unique solution exists initially.
```

Programy XLII

```
!
N = size( b )
ACopy = A
bCopy = b
successful = .true.
!
! Begin the Gaussian elimination algorithm. Note the use of array
! sections in the following loops. These eliminate the need for
! many do loops that are common in Fortran 77 code. Pointers are
! also used below and enhance the readability of the elimination
! process. Begin with the first row.
!
i = 1
! Reduce the system to upper triangular.
do while ( ( successful ) .and. ( i < N ) )
   !
   ! The following statement is called pointer assignment and uses
   ! the pointer assignment operator '=>'. This causes pivotRow
   ! to be an alias for the ith row of ACopy. Note that this does
   ! not cause any movement of data.
   ! Assign the pivot row.
   !
   pivotRow => ACopy( i, : )
   !
```

Programy XLIII

```
      ! Verify that the current pivot is not smaller than smallestPivot.
      !
      successful = abs( pivotRow( i ) ) >= smallestPivot
      if ( successful ) then
         !
         ! Eliminate the entries in the pivot column below the pivot row.
         !
         do j = i+1, N
            ! Assign the current row.
            currentRow => ACopy( j, : )
            ! Calculate the multiplier.
            m = currentRow( i ) / pivotRow( i )
            ! Perform the elimination step on currentRow and right
            ! hand side, bCopy.
            currentRow = currentRow - m * pivotRow
            bCopy( j ) = bCopy( j ) - m * bCopy( i )
         enddo
      endif
      ! Move to the next row.
      i = i + 1
   end do
   ! Check the last pivot.
   pivotRow => ACopy( N, : )
   if ( successful ) successful = abs( pivotRow( N ) ) >= smallestPivot
```

Programy XLIV

```
  if ( successful ) then
     do i = N, 2, -1 ! Backward substitution.
        ! Determine the ith unknown, x( i ).
        x( i ) = bCopy( i ) / ACopy( i, i )
        ! Substitute the now known value of x( i ), reducing the order of
        ! the system by 1.
        bCopy = bCopy - x( i ) * ACopy( :, i )
     enddo
  endif
  ! Determine the value of x( 1 ) as a special case.
  if ( successful ) x( 1 ) = bCopy( 1 ) / ACopy( 1, 1 )
  ! Prepare the return value of the function.
  gaussianElimination = successful
end function gaussianElimination

! Output A in Matlab format, using name in the Matlab assignment statement.
subroutine printMatrix( A, name )
  implicit none
  real, dimension( :, : ) :: A ! Assume the shape of A.
  character name ! Name for use in assignment, ie, name =
  ! ......
  integer n, m, i, j
  n = size( A, 1 )
  m = size( A, 2 )
```

Programy XLV

```
  write( *, fmt="(a1,a5)", advance = "no" ) name, ' = [ '
  ! Output the matrix, except for the last row, which needs no ';'.
  do i = 1, n-1
     ! Output current row.
     do j = 1, m-1
        write( *, fmt="(f10.6,a2)", advance = "no" ) A( i, j ), ', '
     enddo
     ! Output last element in row and end current row.
     write( *, fmt="(f10.6,a1)" ) A( i, m ), ';'
  enddo
  ! Output the last row.
  do j = 1, m-1
     write( *, fmt="(f10.6,a2)", advance = "no" ) A( i, j ), ', '
  enddo
  ! Output last element in row and end.
  write( *, fmt="(f10.6,a1)" ) A( i, m ), ']'
end subroutine printMatrix

! Output b in Matlab format, using name in the Matlab assignment statement.
subroutine printVector( b, name )
  implicit none
  real, dimension( : ) :: b ! Assume the shape of b.
  character name ! Name for use in assignment, ie, name = ......
  integer n, i
```

Programy XLVI

```
    n = size( b )
    write( *, fmt="(a1,a5)", advance = "no" ) name, ' = [ '
    do i = 1, n-1
       write( *, fmt = "(f10.6,a2)", advance = "no" ) b( i ), ', '
    enddo
    write( *, fmt = "(f10.6,a2)" ) b( n ), ']'
  end subroutine printVector
end module GaussianSolver
```

```
! A program to solve linear systems using the GaussianSolver module.
program SolveLinearSystem
  ! Include the module for the various linear solvers.
  use GaussianSolver
  implicit none
  integer, parameter :: N = 5 ! Order of the linear system.
  real, parameter :: TOO_SMALL = 1.0e-7 ! Threshold for pivots.
  ! Declare the necessary arrays and vectors to solve the linear system
  ! A x = b.
  real, dimension( N, N ) :: A ! Coefficient matrix.
  real, dimension( N ) :: x, b ! Vector of unknowns, and right hand side.
  real, dimension( N, N ) :: LU ! Matrix for LU factorization of A.
  logical successful ! Status of computations.
```

Programy XLVII

```
! The intrinsic subroutine, random_number, fills a real array or scalar,
! with uniformly distributed random variates in the interval [0,1).
call random_number( A ) ! Initialize the coefficient matrix.
call random_number( b ) ! Initialize the right-hand side.

! Output the matrix in Matlab format for ease of checking the solution.
call printMatrix( A, 'A' )
call printVector( b, 'b')

! Use Gaussian elmination to calcuate the solution of the linear system.
! The call below uses the default threshold specified in the
! GaussianSolver module by omitting the optional argument.
successful = gaussianElimination( A, b, x )
print *, '====================================='
print *, 'Gaussian Elimination:'
print *, '-------------------------------------'
if ( successful ) then
   call printVector( x, 'x' )
   print *, 'Infinity Norm of Difference = ', &
        maxval( abs ( matmul( A, x ) - b ) )
else
   print *, 'No unique solution or threshold passed.'
endif
```

Programy XLVIII

```
end program SolveLinearSystem
```

Programy XLIX

`m-w.f90` - *Model master-slave*

```
PROGRAM MASTER_WORKER
  USE MPI
  INTEGER istatus(MPI_STATUS_SIZE)
  PARAMETER (njobmx=124)
  !
  CALL MPI_INIT(ierr)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)

  !=======  MASTER
  !
  !  The master recives the masages from workers and if some worker
  !  is idle it sends to it a job number. the worker realizes this
  !  job and sends the massage again to the master. If all jobs are
  !  processed then if the worker has finished the master sends the
  !  term signal (job=-1)  to the worker and the worker exits.
  !
  IF (myrank == 0) THEN
```

Programy L

```
      itag=1
      DO njob=1,njobmx
         CALL MPI_RECV(iwk, 1, MPI_INTEGER, MPI_ANY_SOURCE,  &
              itag, MPI_COMM_WORLD, istatus, ierr)
         idest=istatus(MPI_SOURCE)
         CALL MPI_SEND(njob, 1, MPI_INTEGER, idest,  &
              itag, MPI_COMM_WORLD, ierr)
      END DO

      DO i=1,nprocs-1
         CALL MPI_RECV(iwk, 1, MPI_INTEGER, MPI_ANY_SOURCE,  &
              itag, MPI_COMM_WORLD, istatus, ierr)
         idest=istatus(MPI_SOURCE)
         CALL MPI_SEND(-1, 1, MPI_INTEGER, idest,  &
              itag, MPI_COMM_WORLD, ierr)
      END DO

   ELSE

      !=======  WORKER
      !
      !  The worker sends the signal to the master that it is idle.
      !  if there are jobs to process then the master sends to the
      !  worker the job number (njob) and the worker processes the job.
```

Programy LI

```
      !  If there are no other jobs the number njob sent to the worker
      !  is equal to -1 and after reciving it the worker exits.

      itag=1
      iwk=0

      DO
         CALL MPI_SEND(iwk, 1, MPI_INTEGER, 0,  &
              itag, MPI_COMM_WORLD, ierr)
         CALL MPI_RECV(njob, 1, MPI_INTEGER, 0,  &
              itag, MPI_COMM_WORLD, istatus, ierr)
         IF (njob == -1) EXIT

         CALL DOSOMETHING(njob)

      END DO
      !
      CALL MPI_FINALIZE(ierr)

    END IF

  END PROGRAM MASTER
```

Programy LII

```
SUBROUTINE DOSOMETHING(JOB)
  !
  ! MAIN JOB
  !
END SUBROUTINE DOSOMETHING
```

Programy LIII

`rwalk.f90` - *Random walk, Monte Carlo method; wersja sekwencyjna*

```
! RS6000 SP: Practical MPI programming
PROGRAM randomwalk
  PARAMETER (n = 10000)
  INTEGER itotal(0:9)
  REAL seed
  pi = 3.1415926
  DO i = 0, 9
     itotal(i) = 0
  END DO

  ! Uwaga: zalezy od kompilatora
  ! tutaj: gfortran
  iseed = 5
  CALL srand(iseed)
  DO i = 1, n
     x = 0.0
     y = 0.0
     DO istep = 1, 10
        angle = 2.0 * pi * rand()
        x = x + cos(angle)
```

Programy LIV

```
            y = y + sin(angle)
        END DO
        itemp = sqrt(x*x + y*y)
        itotal(itemp) = itotal(itemp) + 1
    END DO
PRINT *,"total =",itotal
END PROGRAM randomwalk
```

Programy LV

`rwalk-p.f90` - *Random walk, Monte Carlo method; w. równoległa*

```
! RS6000 SP: Practical MPI programming
PROGRAM randomwalk_p
  INCLUDE "mpif.h"

  PARAMETER (n = 100000)
  INTEGER itotal(0:9), iitotal(0:9)
  REAL seed

  CALL MPI_INIT(ierr)
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, nprocs, ierr)
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)

  CALL para_range(1, n, nprocs, myrank, ista, iend)
  pi = 3.1415926

  DO i = 0, 9
     itotal(i) = 0
  END DO

  ! Uwaga: zalezy od kompilatora
```

Programy LVI

```
! tutaj: gfortran
seed = 5 + myrank
CALL srand(seed)
DO i = ista, iend
   x = 0.0
   y = 0.0
   DO istep = 1, 10
      angle = 2.0 * pi * rand()
      x = x + cos(angle)
      y = y + sin(angle)
   END DO
   itemp = sqrt(x**2 + y**2)
   itotal(itemp) = itotal(itemp) + 1
END DO

CALL MPI_REDUCE(itotal, iitotal, 10, MPI_INTEGER, MPI_SUM, 0, &
     MPI_COMM_WORLD, ierr)

PRINT *,"total =",iitotal

CALL MPI_FINALIZE(ierr)
END
```

Programy LVII

```
SUBROUTINE para_range(n1, n2, nprocs, irank, ista, iend)
  iwork1 = (n2 - n1 + 1) / nprocs
  iwork2 = MOD(n2 - n1 + 1, nprocs)
  ista = irank * iwork1 + n1 + MIN(irank, iwork2)
  iend = ista + iwork1 - 1
  IF (iwork2 > irank) iend = iend + 1
END SUBROUTINE para_range
```

Programy LVIII

`BlasEx.f` - *Przykłady użycia biblioteki BLAS*

```
      Example
      implicit real*8 (a-h,o-z)
      parameter(maxn=200,m=80,k=maxn+1)
      parameter(zero=0.0d0,one=1.0d0)
      real*8 a(k,maxn),aa(k,maxn),x(k,m),b(k,m)
      integer ip(maxn)
C     ============================================================
C     Define the matrix
C     ============================================================
      n=maxn
      call inita(a,k,n)
      do i=1,n
         do j=1,n
            aa(j,i)=a(j,i)
         end do
      end do
C     ============================================================
C     LU decomposition
C     ============================================================
c     all dgetrf(n,n,a,k,ip,info)
```

Programy LIX

```
C     ============================================================
C     Defeine the vectors
C     ============================================================
      do jm=1,m
         do jn=1,n
            x(jn,jm)=jn+jm
         end do
      end do
      call dgemm('N','N',n,m,n,one,aa,k,x,k,zero,b,k)
C     ============================================================
C     Solution
C     ============================================================
      call dgetrs('N',n,m,a,k,ip,b,k,info)
      if(info.ne.0) then
         write(6,*) 'error in dgetrs info = ',info
         stop
      end if
C     ============================================================
C     Check result
C     ============================================================
      call check(a, b, k, n, m)

      end
```

Programy LX

wsad1 - ...

```
#!/bin/sh
# plik wsad1
# praca w srodowisku Xeon, gondor; plik PBS
# obliczenia sekwencyjne
#PBS -S /bin/sh
#PBS -N nazwa
#PBS -l walltime=10:00:00
#PBS -l mem=4GB
#PBS -l ncpus=1
#
export OMP_NUM_THREADS=1
cd /home/uzytkownik/katalog
./prog < input > output
```

Programy LXI

wsad2 - ...

```
#!/bin/sh
# plik wsad2
# OpenMP lub BLAS3 z MKL - xeony
#
#PBS -S /bin/sh
#PBS -N ichox
#PBS -l walltime=10:00:00
#PBS -l mem=4GB
#PBS -l ncpus=X
#PBS -l nodes=1:ppn=X
#
export OMP_NUM_THREADS=X
cd /home/uzytkownik/katalog
./prog < input > output
```

Programy LXII

wsad3 - ...

```
#!/bin/sh
# plik wsad3
# MPI - xeony
#
#PBS -S /bin/sh
#PBS -N ntrs
#PBS -l walltime=10:00:00
#PBS -l mem=10GB
#PBS -l nodes=2:ppn=8
#
cd /home/uzytkownik/katalog
source /opt/bin/intel64
export OMP_NUM_THREADS=1
mpirun -genv I_MPI_DEVICE ssm  -ppn 8 -n 16 ./ntrs < input > output
```

Programy LXIII

wsad4 - ...

```
#!/bin/sh
# plik wsad4
# wezel ROHAN, rohan
#
#PBS -S /bin/sh
#PBS -N testowe
#PBS -l mem=1GB
#PBS -l nodes=x:itanium:ppn=2
source /opt/bin/ia64
```

Programy LXIV

`Makefile.ex` - *Przykładowy plik Makefile*

```
PROG2   = ntrs
FILES2  = blt.f loca.f
LDLIBS  = -L/opt/intel/mkl/10.2.2.025/lib/em64t -lmkl_scalapack_lp64 \
        -lmkl_blacs_intelmpi_lp64 -lmkl_intel_thread  -lmkl_intel_lp64 \
        -lmkl_core -liomp5 -lpthread
OPTFLG  = -O3 -xT -openmp -parallel
all:
       mpiifort $(OPTFLG) -o $(PROG2) $(FILES2) $(LDLIBS)
clean:
       rm -f core *.o
```

Programy LXV