```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
!
! Producto matriz banda comprimida por vector
!
! Construye un ejemplo simple: una matriz A de 5 por 5, con una
! sub-diagonal (m1 = 1) y con dos supra-diagonales (m2 = 2), pero
! es guardada en un formato comprimido  A_c de 5 filas y 3 columnas,
! una por cada diagonal.
!
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! INTEL:
!  ifort -vms -c banda_compr.f90
!  ifort -vms -o banda_compr.exe1 *.o
!
! GNU:
!  g95 -w -c banda_compr.f90 --free-form
!  g95 -w -o banda_compr.exe2 banda_compr.o
!
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
program  banda_compr
  implicit none
  !
  integer, parameter :: n  = 5 ! nro de filas
  integer, parameter :: m1 = 1 ! nro de sub-diagonales
  integer, parameter :: m2 = 2 ! nro de supra-diagonales
  !
  integer, dimension (:,:), allocatable :: a
  integer, dimension   (:), allocatable :: x, b
  integer :: i, j, k, m, ier
  !
  ! nro de columnas de la matriz "comprimida"
  m = m1 + 1 + m2 ! nro de: subdiagonales, ppal y supradiagonales
  !
  ! aloca
  allocate ( a (n,m), x (n), b (n), stat = ier)
  if (ier .ne. 0)  stop " aloca "
  !
  ! inicializa
  k = 0
  ii : do  i = 1, n
    jj : do  j = 1, m
    a (i,j) = k
    k = k + 1
    end do jj
  end do ii
  a (1,1) = 0
  k = 0
  do  i = n-1, n
   do  j = i-k, m ; a (i,j) = 0 ; end do
   k = k + 2
  end do
  forall (i=1:n)
    x (i) = n - i + 1
    b (i) = 0
  end forall
  !
  ! eco
  write (*,*)
  write (*,*) "nro de filas matriz comprimida ;  n = ", n
  write (*,*) "nro de colu. matriz comprimida ;  m = ", m
  write (*,*) "nro de   sub-diagonales        ; m1 = ", m1
  write (*,*) "nro de supra-diagonales        ; m2 = ", m2
  call eco2 (a, "matriz A en formato comprimido")
  call eco1 (x, "vector X a multiplicar")
  !
  ! tarea
  call cbanda1 (a, m1, m2, x, b)
  call cbanda2 (a, m1, m2, x, b)
  !
```



```
contains
!
! ----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! tarea matricial en una unica instruccion
subroutine cbanda1 (a, m1, m2, x, b)
 implicit none
 integer, dimension (:,:), intent (inout) :: a
 integer                 , intent (in)    :: m1, m2
 integer, dimension (:)  , intent (in)    :: x
 integer, dimension (:)  , intent (out)   :: b
 character (64) :: cad
 integer :: m, n
 !
 ! tamanios y control
 cad = "error (cbanda): m <> m1+m2+1 "
 n = size (a,1)
 m = siza2 ( size (a,2), m1+m2+1, cad)
 !
 ! tarea matricial
 b = sum (a * eoshift ( spread (x, dim = 2, ncopies = m), &
                        dim = 1,                          &
                        shift = prog_arit (-m1,m,1)       &
                      ), dim = 2)
 call  eco1 (b, "vector B con el producto matriz-vector")
 !
end subroutine
! ----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! tarea matricial en varias instrucciones para ver etapas
subroutine cbanda2 (a, m1, m2, x, b)
 implicit none
 integer, dimension (:,:), intent (inout) :: a
 integer                 , intent (in)    :: m1, m2
 integer, dimension (:)  , intent (in)    :: x
 integer, dimension (:)  , intent (out)   :: b
 integer, dimension (size(a,1),size(a,2)) :: y
 integer, dimension (size(a,2)) :: u
 character (64) :: cad
 integer :: m, n, i
 !
 ! tamanios y control
 cad = "error (cbanda): m <> m1+m2+1 "
 n = size (a,1)
 m = siza2 ( size (a,2), m1+m2+1, cad)
 !
 y = spread (x, dim = 2, ncopies = m)
 call  eco2 (y, "matriz Y con el vector X expandido")
 !
 u = prog_arit (-m1,m,1)
 call  eco1 (u, "vector U con la progresion aritmetica")
 !
 y = eoshift (y, dim = 1, shift = u)
 call  eco2 (y, "matriz Y con el vector X corrido segun U")
 !
 b = sum (a * y, dim = 2)
 call  eco1 (b, "vector B con el producto matriz-vector")
 !
end subroutine
! ----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! genera "ncuantos" elementos de la progresion aritmetica que
! empieza en "ini" y de incremento "delta"
! ----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
function prog_arit (ini, ncuantos, delta)
 integer      , intent (in) :: ini, ncuantos, delta
 integer, dimension (ncuantos) :: prog_arit
 integer :: k
 prog_arit = 0
 if (ncuantos > 0) prog_arit (1) = ini
 do  k = 2, ncuantos
   prog_arit (k) = prog_arit (k-1) + delta
```

```
   end do
  end function
  !-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
  ! control coherencia de las dimensiones de 2 arreglos matriciales
  integer function siza2 (n1, n2, cad)
   integer      , intent (in) :: n1, n2
   character (*), intent (in) :: cad
   if ( n1 .ne. n2 ) then
    write (*,*)
    write (*,'(a)') cad
    stop
   end if
   siza2 = n1
  end function
  !-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
  ! impresion
  subroutine eco1 (t, s)
   integer, dimension (:), intent (in) :: t
   character (*)         , intent (in) :: s
   write (*,*)
   write (*,*) s
   write (*,100) t
   read (*,*)
   100 format (20 (1x, i3) )
  end subroutine
  !-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
  ! impresion
  subroutine eco2 (t, s)
   integer, dimension (:,:), intent (in) :: t
   character (*)           , intent (in) :: s
   integer :: i, m, n
   m = size (t,1)
   n = size (t,2)
   write (*,*)
   write (*,*) s
   do  i = 1, m
   write (*,100) t (i,:)
   end do
   read (*,*)
   100 format (20 (1x, i3) )
  end subroutine
  !-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
end program
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! Problema del Circulo
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! INTEL:
!  ifort -vms -c m_ctes.f90
!  ifort -vms -c m_temps.f90
!  ifort -vms -c m_tools.f90
!  ifort -vms -c circulo.f90
!  ifort -vms -o circulo.exe1 *.o
!
! GNU:
!  g95 -c m_ctes.f90
!  g95 -c m_temps.f90
!  g95 -c m_tools.f90
!  g95 -c circulo.f90
!  g95 -o circulo.exe2 *.o
!
! ADAPTOR + MPICH:
!  adaptor -hpf -dm -free -keep -c m_ctes.f90
!  adaptor -hpf -dm -free -keep -c m_temps.f90
!  adaptor -hpf -dm -free -keep -c m_tools.f90
!  adaptor -hpf -dm -free -keep -c circulo.f90
!  adaptor -hpf -dm -free -o circulo.exe3 *.o
!
! Running with MPICH:
!  mpdboot -n 22 -f ~/mpd.hosts
!  mpdtrace
!  mpdringtest
!  mpiexec -machinefile ~/machi.dat -np 21 circulo.exe3
!  mpiexec -machinefile ~/machi.dat -1 -np 4 circulo.exe3
!  mpdallexit
!
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
program circulo
 use m_ctes
 use m_temps
 use m_tools
 implicit none
 integer (iin) :: h, i, j, k = 0, n
 integer (iin), dimension (:), allocatable :: v
 real    (idp), dimension (:), allocatable :: r
 !hpf$ distribute (block) :: v
 !hpf$ align r (i) with v (i)
 integer (iin), dimension (1) :: s
 !
 !carteles
 write (*,*)
 write (*,*) "problema del circulo"
 write (*,*) "exponente (0 <= h <= 20) h : ? " ; read (*,*) h
 !
 n = 2**h ! una potencia de 2
 write (*,*) "n = 2^h = ", n
 !
 !un control para verificar que "n" sea una potencia de 2
 if (iand(n,n-1).ne. 0) call errata ("n no es una potencia de 2")
 !
 !aloca: guarda con el orden "v" es la referencia de "r" en ALIGN
 ier = 0
 allocate (v (n), stat = ier (2) )
 allocate (r (n), stat = ier (1) )
 if ( any (ier .ne. 0) ) call errata (ier(1:2), "circulo: aloca")
 !
 !inicia cronometro;
 time = cero
 call system_clock (count = h1)   ! t1 = second ()
 !
 !genera una lista random
 call random_number (r)
 v = floor (100 * r)
```



```
 ! if (n < 100) write (*,*) "v", v
 !
 !tarea
 repetir: do
  ! este control es mas lento
  ! j = v (1)
  ! if ( all (v == j) ) exit  ! pues son todos iguales
  ! y este mas rapido
  s = sum (v)
  if (s (1) == 0) exit        ! pues son todos iguales
  v = abs (v - cshift (v,1) )
  ! if (mod (k,1000) == 0) write (*,*)"k = ",k
  k = k + 1
 end do repetir
 !
 !consulta cronometro
 call system_clock (count = h2)   ! t2 = second ()
 time (2) = h2 - h1
 !
 !salida
 call timer (n,h,"circulo")
 !
 !dloca
 ier = 0
 deallocate (r, stat = ier (1) )
 deallocate (v, stat = ier (2) )
 if ( any (ier .ne. 0) ) call errata (ier(1:2), "circulo: dloca")
 !
end program
! -------------------------------------------------------------------
```

```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! Aproximacion de numero pi
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! Compilation with INTEL:
!  ifort -vms -c m_ctes.f90
!  ifort -vms -c m_temps.f90
!  ifort -vms -c pi.f90
!  ifort -vms -o pi.exe1 *.o
!
! Compilation with GNU:
!  g95 -c m_ctes.f90
!  g95 -c m_temps.f90
!  g95 -c pi.f90
!  g95 -o pi.exe2 *.o
!
! Compilation with (ADAPTOR + MPICH):
!  adaptor -hpf -dm -free -c m_ctes.f90
!  adaptor -hpf -dm -free -c m_temps.f90
!  adaptor -hpf -dm -free -c pi.f90
!  adaptor -hpf -dm -free -o pi.exe3 *.o
!
! Running with MPICH:
!  mpdboot -n 4 -f ~/mpd.hosts
!  mpdtrace
!  mpdringtest
!  mpiexec -machinefile ~/machi.dat -np 2 pi.exe3
!  mpiexec -machinefile ~/machi.dat -1 -np 4 pi.exe3
!  mpdallexit
!
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
program  aproxima_pi
 use m_ctes
 use m_temps
 integer (iin), parameter :: p = 8, n = 10**p  ! nro de intervalos
 real    (idp), parameter :: h = 1.0d0/n        ! ancho
 real    (idp), dimension (n) :: area
 real    (idp) :: f, x, pi1
 integer (iin) :: m=1, i
 !hpf$ distribute (block) :: area
 !
 ! inicia cronometro;
 call system_clock (count = h1)   ! t1 = second ()
 !
 ! tarea
 !hpf$ independent, new (f,x)
 do i = 1, n
   x = h * (1.0*i - 0.5d0)
   f = 4.0d0 / (1.0d0 + x * x)
   area (i) = h * f               ! deberia ser f (x) pero no-anda
 end do
 pi1 = sum (area)
 !
 ! consulta cronometro
 call system_clock (count = h2)   ! t2 = second ()
 time (1) = h2 - h1
 !
 ! salida
 write (*,*)
 write (*,*) "p  = ", p
 write (*,*) "n  = ", n
 write (*,*) "pi = ", pi1
 call timer (n, m, "pi")
 !
 !contains
  !
  !pure function f (x)
  ! real (idp), intent (in) :: x
  ! real (idp) :: f
  ! f = 4.0d0 / (1.0d0 + x * x)
```



```
  !end function
end program
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! Ordena un arreglo de enteros por incrementos decrecientes
! en version matricial (cuasi-Shellsort)
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! INTEL:
!  ifort -vms -c shell_sort.f90
!  ifort -vms -o shell_sort.exe1 *.o
!
! GNU:
!  g95 -w -c shell_sort.f90 --free-form
!  g95 -w -o shell_sort.exe2 shell_sort.o
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
program shell_sort
 implicit none
 integer, parameter :: p = 14
 integer, parameter :: kaso = 1
 integer, dimension (:), allocatable :: a
 integer, dimension (p) :: b
 integer :: i, n, ier
 !
 !titulos
 write (*,*)
 write (*,*) "Ordena un arreglo por incrementos decrecientes  "
 write (*,*) "por el metodo de Shellsort, en version matricial"
 !
 !caso pre-definido
 select case (kaso)
 case (1)
   b = [ 14, 7, 12, 5, 10, 3, 8, 6, 13, 4, 11, 2, 9, 1]
   n = size (b)
   allocate ( a (n), stat = ier)
   if (ier .ne. 0) stop" aloca: a "
   a = b
 case default
   write (*,*)
   write (*,*) "tamanio del arreglo ; n : ? " ; read (*,*) n
   allocate ( a (n), stat = ier)
   if (ier .ne. 0)  stop" aloca: a "
   forall (i=1:n)  a (i) = n - i + 1
 end select
 write (*,*)
 write (*,*) "maximo entero representable ; huge = ", huge (1)
 write (*,*) "(usado como eventual relleno al ordenar)"
 write (*,*)
 write (*,*) "arreglo inicial  ; a = ", a
 write (*,*)
 !
 !tarea
 call  qshell (a)
 !
 !salida
 write (*,*)
 write (*,*) "arreglo ordenado ; a = ", a
 write (*,*)
 !
 contains
  subroutine  qshell (a)
   implicit none
   integer, dimension (:), intent (inout) :: a
   integer, dimension (:,:), allocatable :: t
   integer, dimension (2) :: molde
   integer, dimension (size(a)) :: g
   integer :: i, k, n, p
   n = size (a)
   g = huge (a) ! relleno eventual con "infinito" (si hace falta)
   i = n / 2
   i = 2 * i
   iterar: do while (i > 1)
     i = i / 2
```



```
     p = (n + i - 1) / i
     allocate ( t (i,p) )
     molde = (/ i , p /)
     t = reshape (source = a, shape = molde, pad = g)
     call  eco (t)
     ordenar: do while ( any (t (:,1:p-1  ) > t (:,2:p  ) ) )
       call  swap (  t (:,1:p-1:2) , t (:,2:p:2)    &
                   , t (:,1:p-1:2) > t (:,2:p:2) )
       call  eco (t)
       call  swap (  t (:,2:p-1:2) , t (:,3:p:2)    &
                   , t (:,2:p-1:2) > t (:,3:p:2) )
       call  eco (t)
     end do ordenar
     a = reshape (t, shape (a) )
     deallocate (t)
    end do iterar
   end subroutine
   ! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
   ! intercambia el contenido de los arreglos "a" y "b"
   subroutine swap (a,b,m)
    integer, dimension (:,:), intent (inout) :: a, b
    logical, dimension (:,:), intent (in)    :: m
    integer, dimension (size(a,1),size(a,2)) :: t
    where (m) ; t = a ; a = b ; b = t ; end where
   end subroutine
   ! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
   ! imprime eco actual por pantalla
   subroutine eco (t)
    integer, dimension (:,:), intent (in) :: t
    integer :: n, i, k
    i = size (t,1) ; k = size (t,2)
    write (*,*)
    imprimir: do  k = 1, i
     write (*,*)" t ", t (k,:)
    end do imprimir
    write (*,*)"pausa ... " ; read (*,*)
   end subroutine
end program
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```