```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
!
! Compilation with (adaptor + mpich):
!  adaptor -hpf -dm -v -free -o prime_basic.exe prime_basic.f90
!
! Running with mpich:
!  mpdboot -n 21 -f ~/mpd.hosts
!  mpdtrace
!  mpiexec -machinefile ~/machi.dat -np 2 prime_basic.exe
!  mpdallexit
!
! Common output:
!    threads =            1
!    cblocks =            1
!    there are      5761455  primes until    100000000
!
! Specific output:
!
!  mpiexec -machinefile ~/machi.dat -np 1 prime_basic.exe
!    nodes   =            1
!    time needed :    7.388600
!
!  mpiexec -machinefile ~/machi.dat -np 2 prime_basic.exe
!    nodes   =            2
!    time needed :    3.975300
!
!  mpiexec -machinefile ~/machi.dat -np 4 prime_basic.exe
!    nodes   =            4
!    time needed :    2.112800
!
!  mpiexec -machinefile ~/machi.dat -np 8 prime_basic.exe
!    nodes   =            8
!    time needed :    1.189800
!
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! data parallel source program for sieving primes
program prime_basic
  implicit none
  integer, parameter :: n1 = 10000
  integer, parameter :: n  = n1 * n1
  logical, dimension (:), allocatable :: a
  !hpf$ distribute (block) :: a
  integer :: k, s
  integer :: h1, h2, h3, ier = 0
  real    :: time
  !
  print *
  print *, "counting primes in range 2 to n, normal version: "
  print *, "with communication to find out the number to be sieved"
  print *, "nodes   = ", number_of_nodes   ()
  print *, "threads = ", number_of_threads ()
  print *, "cblocks = ", number_of_cblocks ()
  !
  allocate ( a (n), stat = ier)
  if (ier .ne. 0) stop " error aloca: a "
  !
  call system_clock (count=h1)
  !
  a = .true.                 ! presuponemos que todos son primos
  a (1) = .false.            ! pero 1 no es un numero primo
  k = 2                      ! pero 2 lo es y desde ahi empezamos
  do while (k*k .le. n)      ! cota inferior teorica para buscar
    a (k*k:n:k) = .false.    ! los multiplos de "k" no son primos
    k = k + 1                ! pasamos al sgte numero
    do while (.not. a (k))   ! busca el sgte numero no marcado
      k = k + 1
    end do
  end do
  s = count (a)              ! cuenta solo los que tienen TRUE
```



```
  !
  call system_clock (count=h2, count_rate=h3)
  h2 = h2 - h1
  time = float (h2) / float (h3)
  !
  print *, "there are ", s, " primes until ", n
  print *, "time needed : ", time
  !
  deallocate (a, stat = ier)
  if (ier .ne. 0) stop " error dloca: a "
  !
end program
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! compilation with (adaptor + mpich):
!  adaptor -hpf -dm -v -free -o prime_better1.exe prime_better.f90
!  adaptor -hpf -dm -v -cb -free -o prime_better2.exe prime_better.f90
!
! running with mpich:
!  mpdboot -n 21 -f ~/mpd.hosts
!  mpdtrace
!  mpiexec -machinefile ~/machi.dat -np 2 prime_better1.exe
!  mpiexec -machinefile ~/machi.dat -np 8 prime_better2.exe -nb 400
!  mpdallexit
!
! Common output:
!    threads =            1
!    cblocks =            1
!    there are      5761455  primes until    100000000
!
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! Specific outputs without -cb:
!   mpiexec -machinefile ~/machi.dat -np 1 prime_better1.exe
!     nodes   =            1
!     time needed :    5.089800
!   mpiexec -machinefile ~/machi.dat -np 2 prime_better1.exe
!     nodes   =            2
!     time needed :    2.707800
!   mpiexec -machinefile ~/machi.dat -np 4 prime_better1.exe
!     nodes   =            4
!     time needed :    1.377400
!   mpiexec -machinefile ~/machi.dat -np 8 prime_better1.exe
!     nodes   =            8
!     time needed :   0.6808000
!
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! Specific outputs with -cb:
!   mpiexec -machinefile ~/machi.dat -np 1 prime_better2.exe -nb 400
!     nodes   =            1
!     time needed :    2.201400
!   mpiexec -machinefile ~/machi.dat -np 2 prime_better2.exe -nb 400
!     nodes   =            2
!     time needed :    1.208800
!   mpiexec -machinefile ~/machi.dat -np 4 prime_better2.exe -nb 400
!     nodes   =            4
!     time needed :   0.7175000
!   mpiexec -machinefile ~/machi.dat -np 8 prime_better2.exe -nb 400
!     nodes   =            8
!     time needed :   0.4746000
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! data parallel source program for sieving primes
program prime_better
  implicit none
  integer, parameter :: n1 = 10000
  integer, parameter :: n  = n1 * n1
  logical, dimension (:), allocatable :: a
  logical, dimension (n1) :: a1    ! replicated
  !hpf$ distribute a (block)
  integer :: k, i, s
  integer :: h1, h2, h3, ier = 0
  real    :: time
  !
  print *
  print *, "counting primes in range 2 to n, improved version: "
  print *, "replicated computation of primes until sqrt (n)    "
  print *, "nodes   = ", number_of_nodes   ()
  print *, "threads = ", number_of_threads ()
  print *, "cblocks = ", number_of_cblocks ()
  !
  allocate ( a (n), stat = ier)
  if (ier .ne. 0) stop " error aloca: a "
  !
```



```
  call system_clock (count = h1)
  !
  !print *, "replicated sieving up to ", n1
  a1 = .true.
  a1 (1) = .false.
  k = 2
  do while (k*k .le. n1)
     a1 (k*k:n1:k) = .false.
     k = k + 1
     do while (.not. a1 (k))
       k = k + 1
     end do
  end do
  !
  !print *, "distributed sieving up to ", n
  s = 0
  a (1) = .false.
  !hpf$ parallel, reduction (s) begin
  a (2:n) = .true.
  do  k = 2, n1
    if (a1 (k)) then
      ! a (k*k:n:k) = .false.
      !hpf$ independent
      do i = k*k, n, k
        if ( a (i) )  a (i) = .false.
      end do
    end if
  end do
  !hpf$ independent
  do  i = 1, n
   if ( a (i) ) s = s + 1
  end do
  !hpf$ end parallel
  !
  call system_clock (count=h2, count_rate=h3)
  h2 = h2 - h1
  time = float (h2) / float (h3)
  !
  print *, "there are ", s, " primes until ", n
  print *, "time needed : ", time
  !
  deallocate (a, stat = ier)
  if (ier .ne. 0) stop " error dloca: a "
  !
end program
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
!
! compilation with (adaptor + mpich):
!  adaptor -hpf -dm -v -free -o prime_local1.exe prime_local.f90
!  adaptor -hpf -dm -v -cb -free -o prime_local2.exe prime_local.f90
!
! running with mpich:
!  mpdboot -n 21 -f ~/mpd.hosts
!  mpdtrace
!  mpiexec -machinefile ~/machi.dat -np 2 prime_local1.exe
!  mpiexec -machinefile ~/machi.dat -np 10 prime_local2.exe -nb 400
!  mpdallexit
!
! Common output:
!    threads =            1
!    cblocks =            1
!    there are      5761455  primes until    100000000
!
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! Specific outputs without -cb:
!   mpiexec -machinefile ~/machi.dat -np 1 prime_local1.exe
!     nodes   =            1
!     time needed :    5.141800
!   mpiexec -machinefile ~/machi.dat -np 2 prime_local1.exe
!     nodes   =            2
!     time needed :    2.718000
!   mpiexec -machinefile ~/machi.dat -np 4 prime_local1.exe
!     nodes   =            4
!     time needed :    1.381100
!   mpiexec -machinefile ~/machi.dat -np 8 prime_local1.exe
!     nodes   =            8
!     time needed :   0.6817000
!
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! Specific outputs with -cb:
!   mpiexec -machinefile ~/machi.dat -np 1 prime_local2.exe -nb 400
!     nodes   =            1
!     time needed :    1.818100
!   mpiexec -machinefile ~/machi.dat -np 2 prime_local2.exe -nb 400
!     nodes   =            2
!     time needed :   0.8939000
!   mpiexec -machinefile ~/machi.dat -np 4 prime_local2.exe -nb 400
!     nodes   =            4
!     time needed :   0.4150000
!   mpiexec -machinefile ~/machi.dat -np 8 prime_local2.exe -nb 400
!     nodes   =            8
!     time needed :   0.2285000
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! data parallel source program for sieving primes
program prime_local
  implicit none
  integer, parameter :: n1 = 10000
  integer, parameter :: n  = n1 * n1
  logical, dimension (n)  :: a
  logical, dimension (n1) :: a1    ! replicated
  integer, dimension (number_of_processors ()) :: s1
  !hpf$ distribute (block) :: a, s1
  interface
    extrinsic (hpf_local) subroutine criba (a, n, s, a1, n1)
      integer :: n, n1
      logical, dimension (:) :: a
      integer, dimension (:) :: s
      !hpf$ distribute (block) :: a, s
      logical, dimension (n1) :: a1
    end subroutine
  end interface
  !
  integer :: i, k, s
  integer :: h1, h2, h3, ier = 0
```



```
  real    :: time
  !
  print *
  print *, "counting primes in range 2 to n, local HPF version: "
  print *, "with HPF local computation                          "
  print *, "nodes   = ", number_of_nodes   ()
  print *, "threads = ", number_of_threads ()
  print *, "cblocks = ", number_of_cblocks ()
  !
  call system_clock (count=h1)
  !
  !print *, 'replicated sieving up to ', n1
  a1 = .true.
  a1 (1) = .false.
  k = 2
  do while (k*k .le. n1)
     a1 (k*k:n1:k) = .false.
     k = k + 1
     do while (.not. a1 (k) )
       k = k + 1
     end do
  end do
  !
  !print *, 'distributed sieving up to ', n
  call criba (a, n, s1, a1, n1)
  s = sum (s1)
  !
  call system_clock (count=h2, count_rate=h3)
  h2 = h2 - h1
  time = float (h2) / float (h3)
  !
  print *, "there are ", s, " primes until ", n
  print *, "time needed : ", time
  !
end program
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
extrinsic (hpf_local) subroutine criba (a, n, s, a1, n1)
  integer :: n, n1
  logical, dimension (:) :: a
  integer, dimension (:) :: s
  !hpf$ distribute (block) :: a
  !hpf$ distribute (block) :: s
  logical, dimension (n1) :: a1
  !
  integer :: s1
  integer :: lb, ub, lb1, ub1
  integer :: i, k
  !
  lb = lbound (a, 1)
  ub = ubound (a, 1)
  s1 = 0
  do  i = lb, ub
    a (i) = .true.
  end do
  if (lb == 1) a (1) = .false.
  do  k = 2, n1
    if ( a1 (k) ) then
      if (k*k < lb) then
        lb1 = k*k + ((lb - k*k - 1) / k + 1) * k
      else
        lb1 = k*k
      end if
      ub1 = min (ub, n)
      do i = lb1, ub1, k
        if ( a(i) )  a(i) = .false.
      end do
    end if
  end do
  do i = lb, ub
```

```
    if ( a (i) ) s1 = s1 + 1
  end do
  s (:) = s1
end subroutine
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

```
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! Compilation with INTEL:
!  ifort -w -o primo.exe1 primo_hpf.f90
!
! Compilation with GNU:
!  g95 -w -o primo.exe2 primo_hpf.f90
!
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
! ADAPTOR: compilation of a HPF input source file and FREE format.
!
! Some options:
!
! 1) HPF compilation for a single node:
!    adaptor -hpf_1 -free -o primo_hpf.exe3 primo_hpf.f90
!
! 2) HPF compilation for Distributed Memory Machines using MPICH/MPI:
!    adaptor -hpf -dm -free -o primo_hpf.exe4 primo_hpf.f90
!    mpdboot -n 4 -f ~/mpd.hosts
!    mpiexec -machinefile ~/machi.dat -1 -np 3 primo_hpf.exe4
!    mpirun -machinefile ~/machi.dat -nolocal -np 3 primo_hpf.exe4
!
! 3) HPF compilation for Shared Memory Machines (SMM) using PThreads
!    adaptor -hpf -sm -free -o primo_hpf.exe5 primo_hpf.f90
!
! 4) HPF compilation for Cached Architectures
!    adaptor -hpf -cb -free -o primo_hpf.exe6 primo_hpf.f90
!    primo_hpf.exe6 -nb 2
!
! 5) HPF compilation for a cluster of SMP nodes using MPI and PThreads
!    adaptor -hpf -dm -sm -free -o primo_hpf.exe7 primo_hpf.f90
!    mpirun -machinefile ~/machi.dat -np 4 primo_hpf.exe7
!
! 6) OpenMP compilation for shared memory machines using PThreads
!    adaptor -openmp -free -o primo_hpf.exe8 primo_hpf.f90
!    export OMP_NUM_THREADS=2 ; echo $OMP_NUM_THREADS
!    primo_hpf.exe8
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
program primo
  implicit none
  logical, dimension (:), allocatable :: a
  !hpf$ distribute (block) :: a
  integer :: m=1, n, h, k, ier=0
  integer :: h1, h2, h3, p
  real    :: time
  character (13) :: arch = "primo_hpf.tim"
  character (60) :: s
  !
  write (*,*)
  write (*,*) "input n for counting primes in range 2 to n "
  write (*,*) "n ? " ; read (*,*) n
  !
  call system_clock (count=h1)
  !
  allocate (a(n), stat = ier)
  if (ier .ne. 0) stop " error aloca: a "
  !
  a = .true.                ! presuponemos que todos son primos
  a (1) = .false.           ! pero 1 no es un numero primo
  k = 2                     ! pero 2 lo es y desde ahi empezamos
  do
    p = k * k               ! cota inferior teorica para buscar
    if (p > n) exit         ! no hace falta mas revisar
    a (p:n:k) = .false.     ! los multiplos de "k" no son primos
    k = k + 1               ! pasamos al sgte numero
    do
      if (a (k)) exit       ! busca el sgte numero no marcado
      k = k + 1
    end do
  end do
```



```
  h = count (a)             ! cuenta solo los que tienen TRUE
  !
  call system_clock (count=h2,count_rate=h3)
  time = float (h2 - h1) / float (h3)
  !
  !nro de procesadores
  p = 1                                 ! secuencial
  p = number_of_processors ()           ! solo en HPF
  !
  !resumen a disco
  write (*,*)
  write (*,100) " archivo timer: " // arch
  write (*,*)
  s="  p            n m   time1           time2           time_all"
  write (*,100) s
  !
  open (1, file = arch         , &
           status = "unknown"  , &
           position = "append")
  write (*,110)  p, n, m, time
  write (1,110)  p, n, m, time
  close (1, status = 'keep')
  write (*,*)
  !
  write (*,*)
  write (*,*) "hay",h," primos hasta", n
  write (*,*) "wall time = ", time
  write (*,*)
  !
  ! formatos
  100 format (a)
  110 format (1x, i2, 1x, i12, 1x, i2, 3 (1x, e15.7))
  !
  deallocate (a, stat = ier)
  if (ier .ne. 0) stop " error dloca: a "
  !
end program
! -----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```