```
!
!  Description: Numerical solution of the nonlinear thermal conduction
!               equation in a partially insulated slab (parallel version)
!
! ----------------------------------------------------------------------

program main

   use header
   implicit none
   include "mpif.h"

   real (kind=8) :: beta, lambda, tau
   integer :: kmax
   parameter (beta = 0.25_8, tau = 1.0d-5, kmax = 20)

   type(Matrix)  :: A
   type(Vector)  :: u
   real (kind=8) :: umax
   integer       :: m, n, its
   integer       :: myid, numprocs, nrows, ibeg, iend
   integer       :: ierr

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!      Beginning of program - Initialisation of MPI context
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   call MPI_Init(ierr)
   call MPI_Comm_rank(MPI_COMM_WORLD, myid, ierr)
   call MPI_Comm_size(MPI_COMM_WORLD, numprocs, ierr)

   if ( myid == 0 ) then
      open(12,file = 'input.dat')
      print*,'Number of discretisation intervals in y direction:'
      read(12,*) m,lambda
      print*, 'Value of m =',  m
      print*, 'Value of lambda', lambda
      close(12)
   end if

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!      Broadcast m to the other processes
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   call MPI_Bcast(m,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
   call MPI_Bcast(lambda,1,mpi_double_precision,0,mpi_comm_world,ierr)
   n = (m+1)*m + m/2 + 1

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!      Calculate the start and end indices of the rows to be held locally
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   ibeg = int(myid*real(n)/numprocs)+1
   iend = int((myid+1)*real(n)/numprocs)
   nrows = iend-ibeg+1

   allocate(A%aa(5*nrows))
   allocate(A%jj(5*nrows))
   allocate(A%ii(n+1))

   A%n    = n
   A%ibeg = ibeg
   A%iend = iend

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Construct the linear part of the Jacobian
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
   call Laplace(A,m,ibeg,iend)

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Allocate space for u and set the initial guess to 0
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   allocate(u%xx(n))

   u%n    = n
   u%ibeg = ibeg
   u%iend = iend
   u%xx(u%ibeg:u%iend) = 0.0d0

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Apply Newton's method to solve the system
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   ierr = 0
   call Newton(A,u,lambda,beta,tau,kmax,its,ierr)


   !if (myid ==0) then
     ! print*,' this is u siiiiii',  u%xx
   !end if
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Calculate the maximum temperature
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   call Maximum(u,umax)

   if (myid == 0) then

      if (its > kmax) then
         print*,'The Newton Method is diverging. Maximum number',kmax,' of iterations att
ained.'
      else
         print*,'After',its,' Newton steps the maximum temperature umax =',umax
      end if

   end if

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Write the solution to a file for postprocessing in Python
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   call save_solution(u,m)

! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!     Deallocate memory
! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

   deallocate(A%aa)
   deallocate(A%jj)
   deallocate(A%ii)
   deallocate(u%xx)

   call MPI_Finalize(ierr)

end program main
```