并行算法试题答题卷

林宇健 2018202296

2019 年 1 月 28 日

1 并行矩阵向量乘法

1.1 问题描述和分析

编程计算Ax，其中A是$m\times n$的稠密矩阵，x是n维列向量，分别采用$1,4,8,16$台处理机计算。给出并行算法，及并行效率分析。

记$y=Ax$。为简便起见，取A是$n\times n$的方阵，$n=2048$可以整除$1,4,8,16$，从而保证每个进程储存的向量块维度相同。对于不能整除$4,8,16$的n，可通过循环存储等方式为各个进程分配矩阵和向量的数据。在计算时编程随机生成了$matrix_{2048\times 2048}$和$vector_{2048\times 1}$作为待计算的矩阵和向量（计算程序略去）。

我们采用一维行划分的方式并行计算矩阵向量乘法。假设矩阵A按逐行一维块划分为p个块（p表示进程数），即$A=[A_1,A_2,\cdots,A_p]^T, A_k=[A_{k,0},A_{k,1},\cdots,A_{k,p}]$。其中

$$A_{k,j}=\begin{bmatrix} \mathrm{a}_{k\times n/p+1,j\times n/p+1} & \mathrm{a}_{k\times n/p+1,j\times n/p+2} & \cdots & \mathrm{a}_{k\times n/p+1,j\times n/p+n/p} \\ \mathrm{a}_{k\times n/p+2,j\times n/p+1} & \mathrm{a}_{k\times n/p+2,j\times n/p+2} & \cdots & \mathrm{a}_{k\times n/p+2,j\times n/p+n/p} \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{a}_{k\times n/p+n/p,j\times n/p+1} & \mathrm{a}_{k\times n/p+n/p,j\times n/p+2} & \cdots & \mathrm{a}_{k\times n/p+n/p,j\times n/p+n/p} \end{bmatrix}$$

$$\begin{array}{cccccc} & A & & & X & \\ \hline A_{1,1} & A_{1,2} & \cdots & A_{1,p} & x_1 & P_1 \\ \hline A_{2,1} & A_{2,2} & \cdots & A_{2,p} & x_2 & P_2 \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \hline A_{p,1} & A_{p,2} & \cdots & A_{p,p} & x_p & P_p \end{array}$$

```
!****************************************************************************
!
!  matrix_mul_vector_parallel.f90
!  并行矩阵向量乘法，基于行划分方法进行并行化
!
!****************************************************************************

program parallel_Mat_mul_Vec

    use mpi
    implicit none

    integer, parameter :: N = 2048
    integer :: my_left, my_right
    integer :: IERR, NPROC, NSTATUS(MPI_STATUS_SIZE)
    integer :: myrank, myleft, myright, myfile, buf_size, cnt
```

```fortran
    real(4) :: startwtime, endwtime, wtime
    real(4), allocatable :: matrix(:, :), vector(:, :), answer(:, :)
    real(4), allocatable :: matrix_buf(:, :), vector_buf(:, :), answer_buf(:, :)
    character(len = 2) :: sTemp

    call cpu_time(startwtime)

    call mpi_init(IERR)
    call mpi_comm_rank(MPI_COMM_WORLD, myrank, IERR)
    call mpi_comm_size(MPI_COMM_WORLD, NPROC, IERR)

    buf_size = N / NPROC
    myleft = my_left(myrank, NPROC)
    myright = my_right(myrank, NPROC)

    allocate(matrix_buf(N, buf_size)) !  Fortran的矩阵储存方式为列储存，需要
                                      !  进行一次转置，因此设置读取缓存空间
    allocate(vector_buf(buf_size,1))  !  接收其他进程储存的向量所需要的缓存空间
    allocate(matrix(buf_size, N))
    allocate(vector(buf_size, 1))
    allocate(answer(N, 1))
    allocate(answer_buf(N, 1))

    !  读取矩阵
    call mpi_file_open(MPI_COMM_WORLD, "matrix", MPI_MODE_RDONLY, MPI_INFO_NULL, &
    &  myfile, IERR)
    call mpi_file_seek(myfile, myrank*N*buf_size*sizeof(MPI_REAL), MPI_SEEK_SET, &
    &  IERR)
    call mpi_file_read(myfile, matrix_buf, N*buf_size, MPI_REAL, NSTATUS, IERR)
    call mpi_file_close(myfile, IERR)
    matrix = transpose(matrix_buf)

    !  读取向量
    call mpi_file_open(MPI_COMM_WORLD, "vector", MPI_MODE_RDONLY, MPI_INFO_NULL, &
    &  myfile, IERR)
    call mpi_file_seek(myfile, myrank*buf_size*sizeof(MPI_REAL), MPI_SEEK_SET, IERR)
    call mpi_file_read(myfile, vector, buf_size, MPI_REAL, NSTATUS, IERR)
    call mpi_file_close(myfile, IERR)

    answer = 0
    deallocate(matrix_buf) !  释放矩阵缓存空间用于储存每一次计算时的矩阵块
    allocate(matrix_buf(buf_size, buf_size))
    !  循环进程中储存的所有矩阵块
    do cnt = 0, NPROC
        !  计算对应矩阵块与向量的乘积
        matrix_buf = matrix(:, mod(myrank+cnt, NPROC)*buf_size+1:(mod(myrank+cnt, NPROC) &
        &  +1)*buf_size)
        answer(myrank*buf_size+1:(myrank+1)*buf_size, :) = matmul(matrix_buf,vector) &
        &  + answer(myrank*buf_size+1:(myrank+1)*buf_size, :)
        !  进行一次向量块的传递(向上)
        call mpi_send(vector, buf_size, MPI_REAL, myleft, myrank, MPI_COMM_WORLD, IERR)
        call mpi_recv(vector_buf, buf_size, MPI_REAL, myright, myright, &
        &  MPI_COMM_WORLD, NSTATUS, IERR)
        vector = vector_buf
    end do

    !  全规约结果向量，并行输出到文件
    call mpi_allreduce(answer, answer_buf, N, MPI_REAL, MPI_SUM, MPI_COMM_WORLD, IERR)
    call mpi_file_open(MPI_COMM_WORLD, "answer", MPI_MODE_CREATE+MPI_MODE_WRONLY, &
    &  MPI_INFO_NULL, myfile, IERR)
    call mpi_file_seek(myfile, myrank*buf_size*sizeof(MPI_REAL), MPI_SEEK_SET, IERR)
    call mpi_file_write(myfile, answer_buf(myrank*buf_size+1, 1), buf_size, MPI_REAL, &
    &  MPI_STATUS_IGNORE, IERR)
    call mpi_file_close(myfile, IERR)

    !  将各进程的运行时间记录到文件中
    call cpu_time(endwtime)
    wtime = (endwtime - startwtime) * 1000
    write(sTemp, '(i2)') NPROC
    call mpi_file_open(MPI_COMM_WORLD, "walltime"//trim(adjustl(sTemp)), MPI_MODE_CREATE &
    &  +MPI_MODE_WRONLY, MPI_INFO_NULL, myfile, IERR)
    call mpi_file_seek(myfile, myrank*sizeof(MPI_REAL), MPI_SEEK_SET, IERR)
    call mpi_file_write(myfile, wtime, 1, MPI_REAL, MPI_STATUS_IGNORE, IERR)
    call mpi_file_close(myfile, IERR)

    deallocate(matrix)
    deallocate(vector)
    deallocate(answer)
    deallocate(matrix_buf)
    deallocate(vector_buf)
    deallocate(answer_buf)
```

```
    call mpi_finalize(IERR)

end program parallel_Mat_mul_Vec


!------------------子程序与函数部分------------------------------------------
integer function my_left(myrank, nproc) result(ans)

    implicit none
    integer, intent(in) :: myrank, nproc

    ans = myrank - 1
    if (0 == myrank) ans = nproc - 1

end function my_left


integer function my_right(myrank, nproc) result(ans)

    implicit none
    integer, intent(in) :: myrank, nproc

    ans = myrank + 1
    if (nproc-1 == myrank) ans = 0

end function my_right
```

```
!******************************************************************************
!
!  matrix_mul_vector_serial.f90
!  串行矩阵向量乘法
!
!******************************************************************************

program serial_Mat_mul_Vec

    implicit none

    integer, parameter :: N = 2048
    integer :: i, j
    real(4) :: startwtime, endwtime
    real(4) :: matrix(N, N), vector(N, 1), answer(N, 1) = 0

    call cpu_time(startwtime)

    ! 读取矩阵
    open(10, file = 'matrix', access = 'direct', form = 'unformatted', recl = 4*N*N)
    read(10, rec = 1)  ((matrix(i,j), j = 1, N), i = 1, N)
    close(10)
    matrix = transpose(matrix)  ! Fortran的矩阵储存方式为列储存，需要
                                ! 进行一次转置

    ! 读取向量
    open(20, file = 'vector', access = 'direct', form = 'unformatted', recl = 4*N)
    read(20, rec = 1) (vector(i, 1), i = 1, N)
    close(20)

    answer = matmul(matrix, vector)

    ! 输出结果到向量文件
    open(30, file = 'answer', access = 'direct', form = 'unformatted', recl = 4)
    do i = 1, N
        write(30, rec = i) answer(i, 1)
    end do

    call cpu_time(endwtime)
    open(40, file = 'walltime', access = 'direct', form = 'unformatted', recl = 4)
    write(40, rec = 1) (endwtime - startwtime) * 1000
    close(40)

end program serial_Mat_mul_Vec
```

```
#!/bin/bash
# nohup sh autoexec.sh  > /dev/null 2>&1 &

gfortran random_matrix.f90 -o matrix.out
gfortran random_vector.f90 -o vector.out
./matrix.out
```

```
./vector.out
mpif90 matrix_mul_vector_parallel.f90
mpirun -np 1 a.out
mpirun -np 4 a.out
mpirun -np 8 a.out
mpirun -np 16 a.out
gfortran matrix_mul_vector_serial.f90 -o b.out
./b.out
gfortran walltime.for -o c.out
./c.out
```

```
C**********************************************************************
C
C      walltime.for
C      计算串行程序和并行程序的运行时间并显示
C
C**********************************************************************

      PROGRAM WALLTIME_FOR

      REAL*4 T, T1, T4(4), T8(8), T16(16)

      OPEN(8, FILE = 'walltime', ACCESS = 'DIRECT', FORM = 'UNFORMATTED
     & ', RECL = 4)
      READ(8, REC = 1) T
      CLOSE(8)
      PRINT *, "serial program walltime: ", T, "ms"

      OPEN(8, FILE = 'walltime1', ACCESS = 'DIRECT', FORM = 'UNFORMATTED
     & ', RECL = 4)
      READ(8, REC = 1) T1
      CLOSE(8)
      PRINT *, "parallel program walltime (1 process): ", T1, "ms"

      OPEN(8, FILE = 'walltime4', ACCESS = 'DIRECT', FORM = 'UNFORMATTED
     & ', RECL = 4*4)
      READ(8, REC = 1) T4
      CLOSE(8)
       SUM4 = .0
      DO 40 I = 1, 4
        SUM4 = SUM4 + T4(I)
   40 CONTINUE
      PRINT *, "parallel program walltime (4 process): ", SUM4/4, "ms"

      OPEN(8, FILE = 'walltime8', ACCESS = 'DIRECT', FORM = 'UNFORMATTED
     & ', RECL = 4*8)
      READ(8, REC = 1) T8
      CLOSE(8)
       SUM8 = .0
      DO 80 I = 1, 8
        SUM8 = SUM8 + T8(I)
   80 CONTINUE
      PRINT *, "parallel program walltime (8 process): ", SUM8/8, "ms"

      OPEN(8, FILE = 'walltime16', ACCESS = 'DIRECT', FORM =
     & 'UNFORMATTED', RECL = 4*16)
      READ(8, REC = 1) T16
      CLOSE(8)
       SUM16 = .0
      DO 160 I = 1, 16
        SUM16 = SUM16 + T16(I)
  160 CONTINUE
      PRINT *, "parallel program walltime (16 process): ", SUM16/16,
     & "ms"

      END PROGRAM WALLTIME_FOR
```