High Performance Computing for Mathematics
Assignment 1

Jamie Burke

January 30, 2021

1 Introduction

The task was to parallelise a matrix multiplication task where the multiplication task was to be split and completed by several different processes using the message passing interface system on Cirrus. Using a procedural language such as FORTRAN, the code was written in order to split up the multiplication task evenly so that each process had to play their part equally in deducing the answer of a simple matrix multiplication problem.

The problem itself was to multiply two matrices, A and B producing matrix D, each of size $N \times N$. The matrices are built using the simple rules defined as

$$A_{ij} = (N - j + i + 1)i \tag{1}$$

$$B_{ij} = (j + i)(N - j + 1). \tag{2}$$

As simple as this task is, in order to demonstrate the MP interface the task was to employ a certain number of processors P such that:

1. The root process, $P = 0$, would construct the matrix B in its entirety. This matrix would be **broadcasted** to all other processes to use for step 3.
2. All other processes (including the root process) would compute a certain number of rows of A — depending on the size of N relative to P — and multiply these rows with matrix B to deduce their respective rows of the answer D.
3. The root process would then **gather** all these rows of D and populate them into the matrix itself to produce and print the final answer.

The caveats here are the sizes of N and P. The problem is simple when $P \geq N$ as we can allocate the first N processes to computing their individual row of A and deducing their corresponding row of D. When $P < N$ care needs to be taken on how to allocate the processes their rows of A. If N is divisible by P, i.e. for some whole number n we have $nP = N$, then we can allocate n rows of A to process i, $i = 0, \ldots P - 1$, and compute n rows of D using matrix B. This is because we will accumulate the N rows of D after all processes have run the code. If N is not divisible by P then we compute n as $n = \lceil \frac{N}{P} \rceil$ and follow the same procedure for if N was divisible by P. However, we will have some junk rows computed that don't form part of the answer in D. These rows are discarded after gathering all results outputting the correct answer, as seen in the results section.

2 Results

This section will show screenshots of tests for various values of N and P for the cases where $N = P$, $N > P$ and $N < P$. In particular, choices of $(N, P) = (3, 3), (4, 6)$ and $(5, 3)$ will be demonstrated below. In order to validate results, the inbuilt function `matmul()` is used for comparison.

The submission folder contains 3 files, `mat_mul_ext.f90`, `compile_now`, and `mat_mul_job.slurm`. The `.f90` file contains the FORTRAN code detailed in the appendix, `compile_now` contains the bash code to compile the FORTRAN code. The `.slurm` file contains the information to send to Cirrus in order to run the executable outputted from `compile_now`. For further experimentation, the two parameter to change is N in `mat_mul_ext.f90` and the `tasks-per-node` parameter (equivalent to P in this report). Below show snapshots of the output file from running the three experiments.

```
Matrix A:
           4           3           2
          10           8           6
          18          15          12
Matrix B:
           6           6           4
           9           8           5
          12          10           6

Multiplying matrices A and B using matmul():
          75          68          43
         204         184         116
         387         348         219

Multplying matrices A and B using MPI Funcs:
          75          68          43
         204         184         116
         387         348         219

Summed difference between answers:
           0           0           0
           0           0           0
           0           0           0
```

Figure 1: Correct results when $(N, P) = (3, 3)$.

```
Matrix A:
           5           4           3           2
          12          10           8           6
          21          18          15          12
          32          28          24          20
Matrix B:
           8           9           8           5
          12          12          10           6
          16          15          12           7
          20          18          14           8

Multiplying matrices A and B using matmul():
         176         174         144          86
         464         456         376         224
         864         846         696         414
        1376        1344        1104         656

Multplying matrices A and B using MPI Funcs:
         176         174         144          86
         464         456         376         224
         864         846         696         414
        1376        1344        1104         656

Summed difference between answers:
           0           0           0           0
           0           0           0           0
           0           0           0           0
           0           0           0           0
```

Figure 2: Correct results when $(N, P) = (4, 6)$.

```
Matrix A:
           6           5           4           3           2
          14          12          10           8           6
          24          21          18          15          12
          36          32          28          24          20
          50          45          40          35          30
Matrix B:
          10          12          12          10           6
          15          16          15          12           7
          20          20          18          14           8
          25          24          21          16           9
          30          28          24          18          10

Multiplying matrices A and B using matmul():
         350         360         330         260         150
         900         920         840         660         380
        1650        1680        1530        1200         690
        2600        2640        2400        1880        1080
        3750        3800        3450        2700        1550

Multplying matrices A and B using MPI Funcs:
         350         360         330         260         150
         900         920         840         660         380
        1650        1680        1530        1200         690
        2600        2640        2400        1880        1080
        3750        3800        3450        2700        1550

Summed difference between answers:
           0           0           0           0           0
           0           0           0           0           0
           0           0           0           0           0
           0           0           0           0           0
           0           0           0           0           0
```

Figure 3: Correct results when $(N, P) = (5, 3)$.

A FORTRAN Code

```fortran
!--------------------------------------------------------------
! Perform matrix multiplication using MPI functions.
! This program is able to deal with any combination of
! number of processes and size of matrices being multiplied
!--------------------------------------------------------------
      program mat_mul_ext
          implicit none

          ! Include the mpif.h module for MPI function
          include "mpif.h"

          ! Define size of matrices and iterators i, j
          integer :: N, i, j
          parameter(N = 3)
          ! Define the integer variables necessary for MPI functions
          integer :: comm, rank, N_processes, ierr

          ! Define matrices A, B, C_builtin, D and vectors
          ! row_c and row_a (to hold interim results of MPI computation
          ! note: C_builtin will hold the computation using matmul()
          ! while D will hold the result computed using MPI functions
          integer, dimension(N, N) :: A, B, C_builtin, CD_diff, D
          integer, dimension(N) :: row_a, row_c

          ! Create allocatble 2D arrays for when N_processes < N
          ! Also create integer status variable for allocation
          integer :: errCode
          integer, dimension(:,:), allocatable :: rows_a, rows_c
          integer, dimension(:,:), allocatable :: interim_D

          ! define remainder between the size of matrices and
          ! the number of processors. Also define divisor, the
          ! whole integer number when dividing N by N_processes
          integer :: div, rem

          ! Initialise MPI and extract the rank and size of process
          comm = MPI_COMM_WORLD
          call MPI_INIT(ierr)
          call MPI_COMM_RANK(comm, rank, ierr)
          call MPI_COMM_SIZE(comm, N_processes, ierr)

          ! Set up conditions depending on rank (process)
          if (rank == 0) then
             ! Create matrix B if rank 0 and form 1st row of A
             do i = 1, N
                do j = 1, N
                   B(i,j) = (j + i) * (N - j + 1)
                end do
             end do
          end if

          ! Broadcast B to all other processes
          call MPI_BCAST(B, N**2, MPI_INT, 0, &
                         comm, ierr)

          ! Compute divisor between N and N_processes for use
          ! when number of processes is less than the size of
```

```fortran
        ! matrices
        rem = mod(N, N_processes)
        div = floor(float(N)/float(N_processes))

        ! If the number of processes is greater than or equal
        ! to the size of matrices then only use up to N of the
        ! processes in dealing with the computation
        if (N_processes >= N) then
           if (rank < N) then
              ! Form rank'th row of A
              do j = 1, N
                 row_a(j) = (N - j + (rank+1) + 1)*(rank + 1)
              end do

              ! Compute vector c, i.e. compute matmul(row_a,B)
              row_c = matmul(row_a, B)
           end if

           ! Gather vectors rows_c to put in matrix D
           call MPI_GATHER(row_c, N, MPI_INT, D, N, &
                           MPI_INT, 0, comm, ierr)

        ! If the number of the processes are less than N, assign an
        ! equal number of rows of A to each process and populate D.
        else if (N_processes < N) then
           ! If non-zero remainder (N is not divisble by N_processes
           ! then add 1 to the divisor so compute ceil(N/N_processes)
           if (rem /= 0) then
              div = div + 1
           end if

           ! Allocate the necessary memory to rows_a and rows_c
           ! Also allocate maximum number of columns to interim_D
           allocate(rows_a(div, N), stat = errCode)
           allocate(rows_c(div, N), stat = errCode)
           allocate(interim_D(N, div*N_processes), stat = errCode)

           ! Build rows of A. The row number is maintained by
           ! iterating over the divisor between N and N_processes and
           ! the rank of the process.
           do i = 1, div
              do j = 1, N
                 rows_a(i,j) = (N - j + (div*rank+i) + 1)*(div*rank+i)
              end do
           end do

           ! Compute rows of answer and transpose before gathering
           ! them to populate rows of D
           rows_c = matmul(rows_a, B)
           rows_c = transpose(rows_c)

           ! Gather vector rows_c to populate D. If N is not divisible
           ! by number of processes then populate interim_D and
           ! discard the last few columns (since FORTRAN is a
           ! column-wise language) which correspond to the extra rows
           ! computed.
           if (rem /= 0) then
              call MPI_GATHER(rows_c, div*N, MPI_INT, interim_D, &
                              div*N, MPI_INT, 0, comm, ierr)
```

```fortran
         D = interim_D(:,:N)
      else
         call MPI_GATHER(rows_c, div*N, MPI_INT, D, div*N, &
                         MPI_INT, 0, comm, ierr)
      end if
    end if

    ! Print results
    if (rank == 0) then

      ! Loop over iterators to define matrices A and B
      do i = 1, N
         do j = 1, N
            A(i, j) = (N - j + i + 1) * i
         end do
      end do

      print*, "Matrix A:"
      do i = 1, N
         print*, A(i, :)
      end do

      print*, "Matrix B:"
      do i = 1, N
         print*, B(i, :)
      end do

      print*, " "
      C_builtin = matmul(A, B)
      print*,"Multiplying matrices A and B using matmul():"
      do i = 1, N
         print*, C_builtin(i, :)
      end do

      print*, " "
      print*, "Multplying matrices A and B using MPI Funcs:"
      D = transpose(D)
      do i = 1, N
         print*, D(i,:)
      end do

      print*, " "
      CD_diff = C_builtin - D
      print*, "Summed difference between answers:"
      do i = 1, N
         print*, CD_diff(i, :)
      end do

   end if

   ! Stop MPI process
   call MPI_FINALIZE(ierr)

end program mat_mul_ext
```