CAP-372 / 2019 - Quinta Lista de Exercícios

Data: 19/09/19

Este documento e os fontes estão em: https://github.com/exxxxxxxm/372

SUMÁRIO

ALGORITMO

Testando algoritmo 1 de *http://www.lac.inpe.br/~stephan/CAP-372/matrixmult_microsoft.pdf* ; compilando na máquina local (Intel i7) usando *gfortran*, sem paralelização:

```
! CAP372 - exercise 5 - 2019-09-15
! Simple check of sequential algorithm 1
! Compiling on the local machine using gfortran
! - Each subtask i hold one row of matrix A and one column of matrix B .
! - The result is stored in one element of C .
! - After that, every subtask i, 0≤i<n, transmits its column of matrix B
!   to the subtask with the number (i + 1) mod n .
! - When the number of processors ("p") is less than the number of
!   basic subtasks ("n"), each processor would execute several
!   inner products of matrix A rows and matrix B columns.
! - Each aggregated basic subtask ( = the calculation of one row of C)
!   determines several rows of the result matrix C.
program list05_algorithm
   implicit none
   integer :: mi, mj, mk               ! matrix indexes
   double precision :: t1, t2          ! time elapsed
   double precision :: temp1           ! temporary
   integer :: p                        ! emulate number of processes
   integer :: stripe_i, stripe_qtd, stripe_count, stripe_countA   ! stripe
   integer :: interation, subtask      ! looping
   ! C =  A * B, matrix initialization with example
   integer, parameter :: ms = 4        ! matrix size
   double precision, dimension(ms, ms) ::                  &
      A = transpose(reshape([ 5, 2, 6, 1,                  &
                              0, 6, 2, 0,                  &
                              3, 8, 1, 4,                  &
                              1, 8, 5, 6 ], shape(A))),    &
      B = transpose(reshape([ 7, 5, 8, 0,                  &
                              1, 8, 2, 6,                  &
                              9, 4, 3, 8,                  &
                              5, 3, 7, 9 ], shape(B))),    &
      C = 0

   print*, "=== List05 - Matrix multiplication example ==="
   call cpu_time(t1)                   ! elapsed time calculation
   p = 4                               ! emulate 4 processors
   stripe_qtd = ms / p                 ! subtasks per processor
   do interation = 0, ms / stripe_qtd - 1    ! algorithm interation
      do subtask = 0, ms / stripe_qtd - 1    ! each subtask hold one matrix row
         do stripe_countA = 0, stripe_qtd - 1      ! A count for p < n
            do stripe_count = 0,  stripe_qtd - 1   ! count for p < n
               ! stripe column calculation
               stripe_i = stripe_qtd*mod(subtask+p-interation,p)+stripe_count
               ! matrix A line calculation
               mi = subtask * stripe_qtd + 1 + stripe_countA
               temp1 = 0               ! multiplication: A line x B column
               do mj = 1, ms           ! count A column
                  mk = stripe_i + 1    ! B column index
                  temp1 = temp1 + A(mi, mj) * B(mj, mk)  ! C = A x B
                  print*, "A =", int(A(mi, mj)), "     , B =", int(B(mj, mk))
               end do
               print*, "interation =",interation,"        subtask =",subtask,   &
                        "        C(i,j) = ", int(temp1)
               C(mi, mk) = temp1       ! store C element
            end do
         end do
      end do
   end do
   print*, "Result Matrix C ="         ! show the result
   do mi = 1, ms
      print*, int(C(mi,:))
   end do
   call cpu_time(t2)                   ! elapsed time calculation
   print *, "Elapsed time [s]:", t2 - t1
end program list05_algorithm
```

Multiplicação de teste:

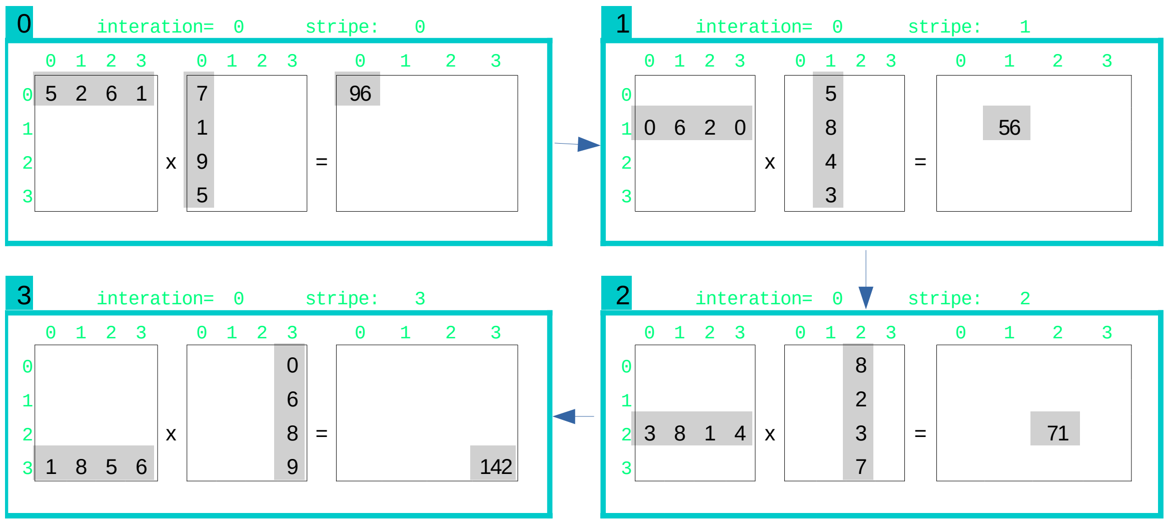
$$\begin{vmatrix} 5 & 2 & 6 & 1 \\ 0 & 6 & 2 & 0 \\ 3 & 8 & 1 & 4 \\ 1 & 8 & 5 & 6 \end{vmatrix} \text{X} \begin{vmatrix} 7 & 5 & 8 & 0 \\ 1 & 8 & 2 & 6 \\ 9 & 4 & 3 & 8 \\ 5 & 3 & 7 & 9 \end{vmatrix} = \begin{vmatrix} 96 & 68 & 69 & 69 \\ 24 & 56 & 18 & 52 \\ 58 & 95 & 71 & 92 \\ 90 & 107 & 81 & 142 \end{vmatrix}$$

SIMULAÇÃO COM 4 PROCESSADORES

INTERAÇÃO 1

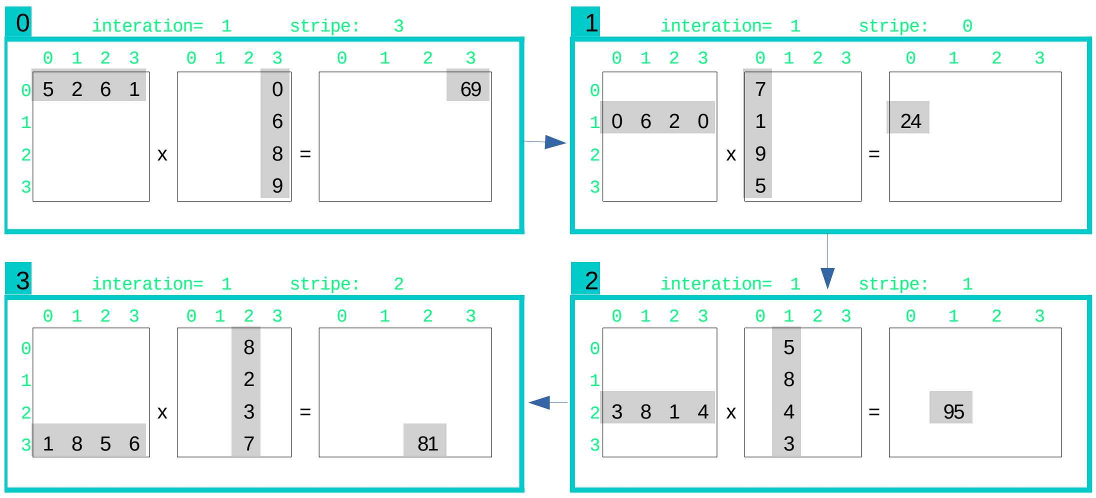
```
=== List05 - Matrix multiplication example ===
 A =             5      , B =             7
 A =             2      , B =             1
 A =             6      , B =             9
 A =             1      , B =             5
 interation =            0          subtask =           0          C(i,j) =
96
```

```
 A =            0       , B =           5
 A =            6       , B =           8
 A =            2       , B =           4
 A =            0       , B =           3
 interation =            0         subtask =           1          C(i,j) =
56
 A =            3       , B =           8
 A =            8       , B =           2
 A =            1       , B =           3
 A =            4       , B =           7
 interation =            0         subtask =           2          C(i,j) =
71
 A =            1       , B =           0
 A =            8       , B =           6
 A =            5       , B =           8
 A =            6       , B =           9
 interation =            0         subtask =           3          C(i,j) =
142
 A =            5       , B =           0
 A =            2       , B =           6
 A =            6       , B =           8
 A =            1       , B =           9
```



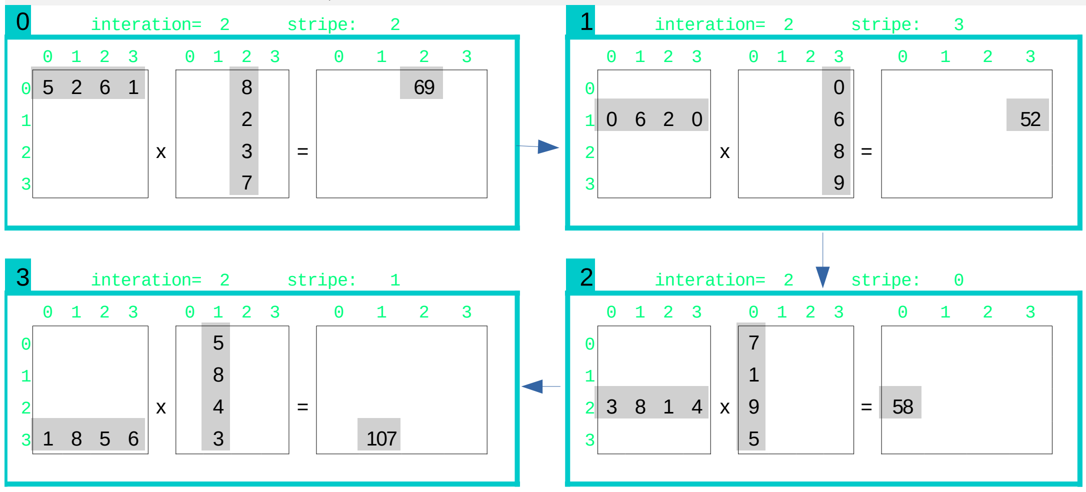
INTERAÇÃO 2

```
 interation =            1         subtask =           0          C(i,j) =
69
 A =            0       , B =           7
 A =            6       , B =           1
 A =            2       , B =           9
 A =            0       , B =           5
 interation =            1         subtask =           1          C(i,j) =
24
 A =            3       , B =           5
 A =            8       , B =           8
 A =            1       , B =           4
 A =            4       , B =           3
 interation =            1         subtask =           2          C(i,j) =
95
 A =            1       , B =           8
 A =            8       , B =           2
 A =            5       , B =           3
 A =            6       , B =           7
 interation =            1         subtask =           3          C(i,j) =
81
 A =            5       , B =           8
 A =            2       , B =           2
 A =            6       , B =           3
 A =            1       , B =           7
```



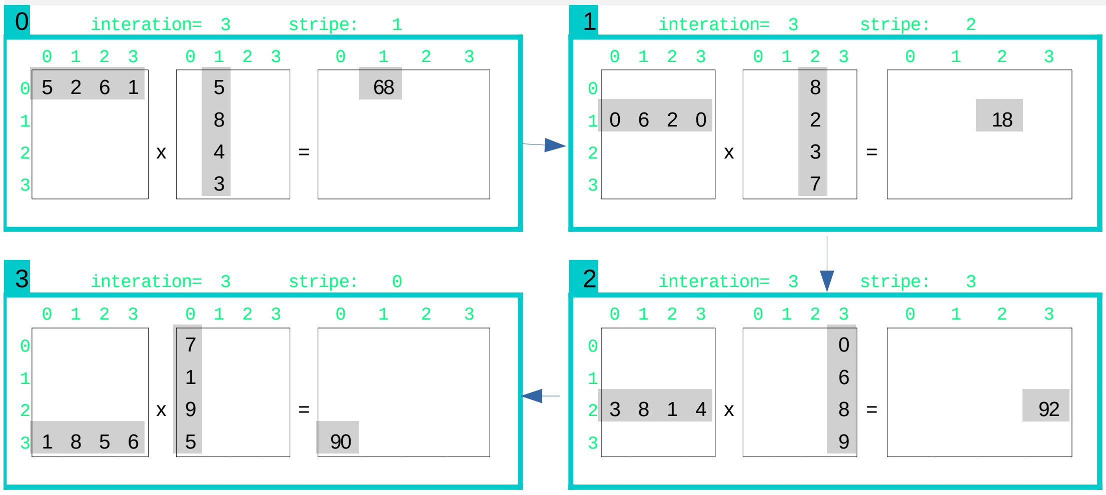
INTERAÇÃO 3

```
interation =             2          subtask =            0          C(i,j) =
69
 A =            0        , B =            0
 A =            6        , B =            6
 A =            2        , B =            8
 A =            0        , B =            9
 interation =            2          subtask =            1          C(i,j) =
52
 A =            3        , B =            7
 A =            8        , B =            1
 A =            1        , B =            9
 A =            4        , B =            5
 interation =            2          subtask =            2          C(i,j) =
58
 A =            1        , B =            5
 A =            8        , B =            8
 A =            5        , B =            4
 A =            6        , B =            3
 interation =            2          subtask =            3          C(i,j) =
107
 A =            5        , B =            5
 A =            2        , B =            8
 A =            6        , B =            4
 A =            1        , B =            3
```



INTERAÇÃO 4

```
 interation =            3          subtask =            0          C(i,j) =
68
 A =            0        , B =            8
 A =             6       , B =           2
 A =             2       , B =           3
 A =             0       , B =           7
 interation =            3          subtask =            1          C(i,j) =
18
 A =             3       , B =           0
 A =             8       , B =           6
 A =             1       , B =           8
 A =             4       , B =           9
 interation =            3          subtask =            2          C(i,j) =
92
 A =             1       , B =           7
 A =             8       , B =           1
 A =             5       , B =           9
 A =             6       , B =           5
 interation =            3          subtask =            3          C(i,j) =
90
 Result Matrix C =
          96          68          69          69
          24          56          18          52
          58          95          71          92
          90         107          81         142
```



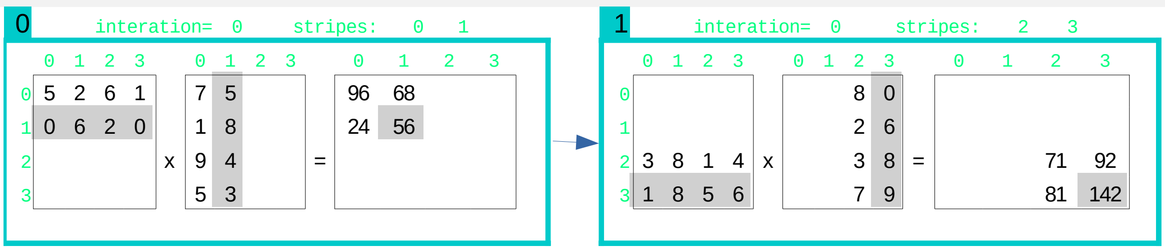
SIMULAÇÃO COM 2 PROCESSADORES

Algorithm 1: Block-Striped Decomposition…

- **Aggregating and Distributing the Subtasks among the Processors:**
  - In case when the number of processors ***p*** is less than the number of basic subtasks ***n***, calculations can be aggregated in such a way that each processor would execute several inner products of matrix ***A*** rows and matrix ***B*** columns. In this case after the completion of computation, each aggregated basic subtask determines several rows of the result matrix ***C***,
  - Under such conditions the initial matrix ***A*** is decomposed into ***p*** horizontal stripes and matrix ***B*** is decomposed into ***p*** vertical stripes,
  - Subtasks distribution among the processors have to meet the requirements of effective representation of the ring structure of subtask information dependencies

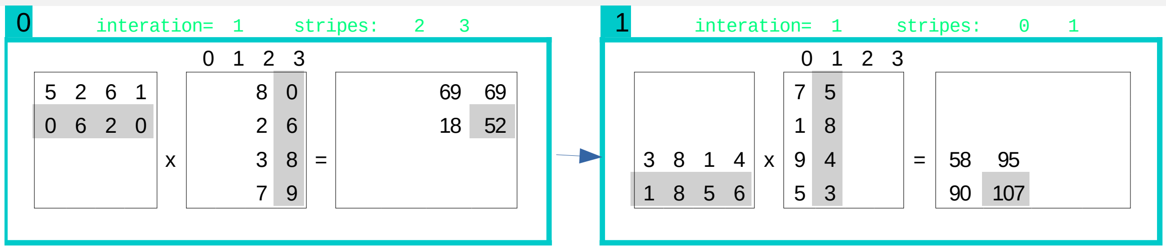
INTERAÇÃO 1

```
 === List05 - Matrix multiplication example ===
 A =             5       , B =           7
 A =             2       , B =           1
 A =             6       , B =           9
 A =             1       , B =           5
 interation =            0         subtask =            0          C(i,j) =
96
 A =             5       , B =           5
 A =             2       , B =           8
 A =             6       , B =           4
 A =             1       , B =           3
 interation =            0         subtask =            0          C(i,j) =
68
 A =             0       , B =           7
 A =             6       , B =           1
 A =             2       , B =           9
 A =             0       , B =           5
 interation =            0         subtask =            0          C(i,j) =
24
 A =             0       , B =           5
 A =             6       , B =           8
 A =             2       , B =           4
 A =             0       , B =           3
 interation =            0         subtask =            0          C(i,j) =
56
 A =             3       , B =           8
 A =             8       , B =           2
 A =             1       , B =           3
 A =             4       , B =           7
 interation =            0         subtask =            1          C(i,j) =
71
 A =             3       , B =           0
 A =             8       , B =           6
 A =             1       , B =           8
 A =             4       , B =           9
 interation =            0         subtask =            1          C(i,j) =
92
 A =             1       , B =           8
 A =             8       , B =           2
 A =             5       , B =           3
 A =             6       , B =           7
 interation =            0         subtask =            1          C(i,j) =
81
 A =             1       , B =           0
 A =             8       , B =           6
 A =             5       , B =           8
 A =             6       , B =           9
 interation =            0         subtask =            1          C(i,j) =
142
 A =             5       , B =           8
 A =             2       , B =           2
 A =             6       , B =           3
 A =             1       , B =           7
 interation =            1         subtask =            0          C(i,j) =
69
 A =             5       , B =           0
 A =             2       , B =           6
 A =             6       , B =           8
 A =             1       , B =           9
```



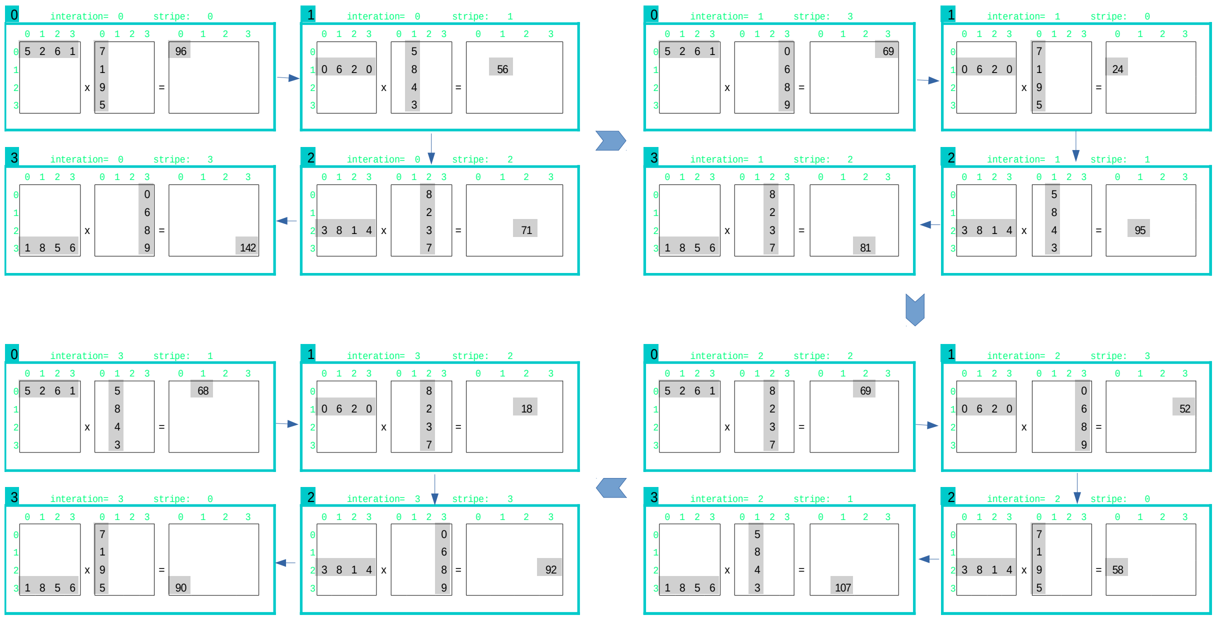
INTERAÇÃO 2

```
 interation =            1         subtask =            0          C(i,j) =
69
 A =             0       , B =           8
 A =             6       , B =           2
 A =            2       , B =            3
 A =            0       , B =            7
 interation =           1         subtask =            0          C(i,j) =
18
 A =            0       , B =            0
 A =            6       , B =            6
 A =            2       , B =            8
 A =            0       , B =            9
 interation =           1         subtask =            0          C(i,j) =
52
 A =            3       , B =            7
 A =            8       , B =            1
 A =            1       , B =            9
 A =            4       , B =            5
 interation =           1         subtask =            1          C(i,j) =
58
 A =            3       , B =            5
 A =            8       , B =            8
 A =            1       , B =            4
 A =            4       , B =            3
 interation =           1         subtask =            1          C(i,j) =
95
 A =            1       , B =            7
 A =            8       , B =            1
 A =            5       , B =            9
 A =            6       , B =            5
 interation =           1         subtask =            1          C(i,j) =
90
 A =            1       , B =            5
 A =            8       , B =            8
 A =            5       , B =            4
 A =            6       , B =            3
 interation =           1         subtask =            1          C(i,j) =
107
 Result Matrix C =
         96          68          69          69
         24          56          18          52
         58          95          71          92
         90         107          81         142
```

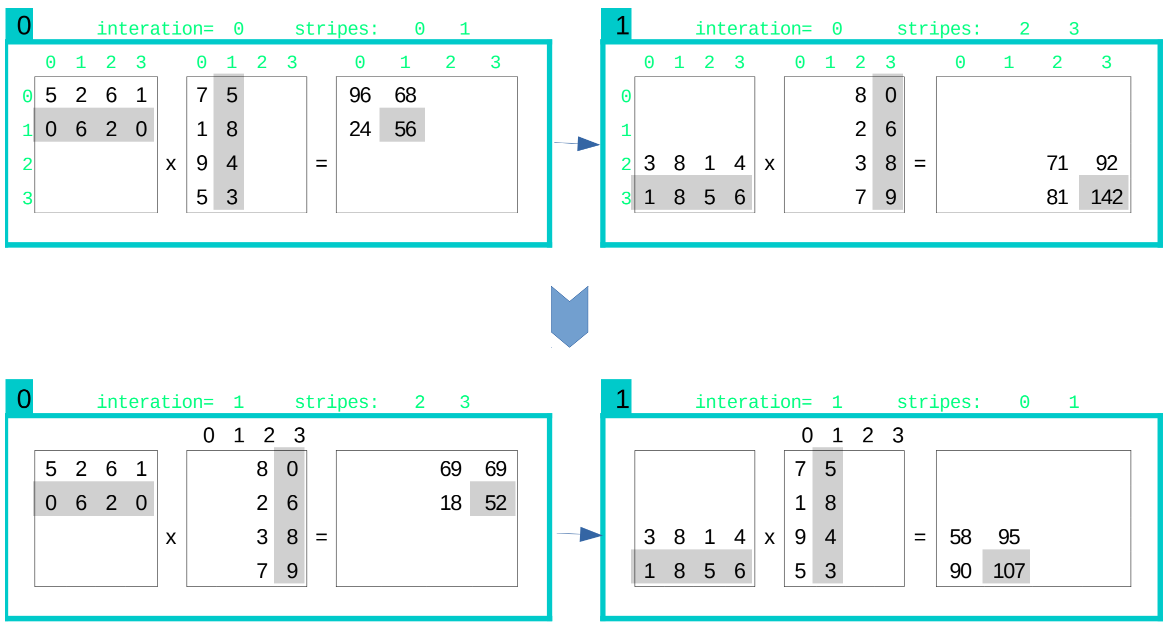


Os resultados esperados para os algoritmos são:

Simulando 4 processadores

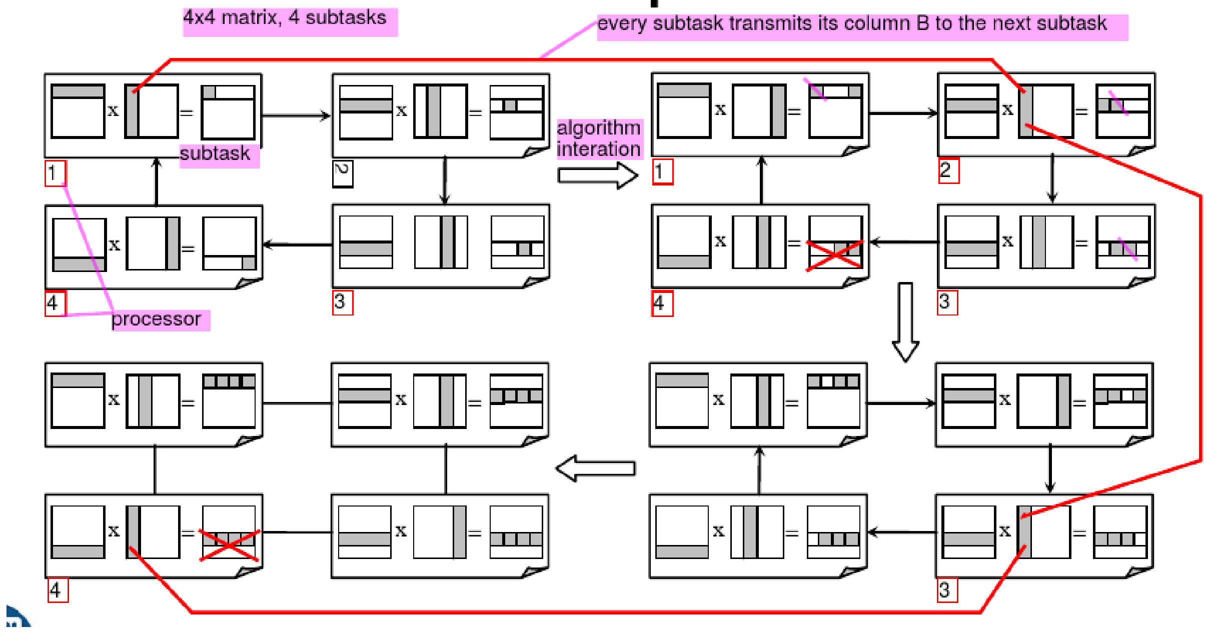


Simulando 2 processadores



As figuras anteriores derivam do slide:

❑ Scheme of Information Dependences



OBTENÇÃO DO TEMPO SERIAL

Rodando no nó de acesso do Santos Dumont

Programa:

```fortran
program ilist05s512
! CAP372 - exercise 5 - 2019-09-15
! - Each subtask i hold one row of matrix A and one column of matrix B .
! - The result is stored in one element of C .
! - After that, every subtask i, 0≤i<n, transmits its column of matrix B
!   to the subtask with the number (i + 1) mod n .
! - When the number of processors ("p") is less than the number of
!   basic subtasks ("n"), each processor would execute several
!   inner products of matrix A rows and matrix B columns.
! - Each aggregated basic subtask ( = the calculation of one row of C)
!   determines several rows of the result matrix C.
  implicit none
  integer, parameter :: ms = 512      ! matrix size
  integer, parameter :: p = 8         ! emulate the number of processes
  integer :: mi, mj, mk               ! matrix indexes
  double precision :: t1, t2          ! time elapsed
  double precision :: temp1           ! temporary
  integer :: stripe_i, stripe_qtd, stripe_count, stripe_countA   ! stripe
  integer :: interation, subtask      ! looping

  ! C =  A * B, matrix initialization
  double precision, dimension(ms, ms) :: C = 0,       &
    A = reshape([(mi, mi=1, ms*ms)], shape(A)),       &
    B = reshape([(mi, mi=ms*ms+1, ms*ms*2)], shape(A))

  call cpu_time(t1)                   ! elapsed time calculation
  stripe_qtd = ms / p                 ! subtasks per processor
  do interation = 0, ms / stripe_qtd - 1     ! algorithm interation
    do subtask = 0, ms / stripe_qtd - 1      ! each subtask hold one matrix row
      do stripe_countA = 0, stripe_qtd - 1       ! A count for p < n
        do stripe_count = 0,  stripe_qtd - 1     ! count for p < n
          ! stripe column calculation
          stripe_i = stripe_qtd*mod(subtask+p-interation,p)+stripe_count
          ! matrix A line calculation
          mi = subtask * stripe_qtd + 1 + stripe_countA
          temp1 = 0                   ! multiplication: A line x B column
          do mj = 1, ms               ! count A column
            mk = stripe_i + 1         ! B column index
            temp1 = temp1 + A(mi, mj) * B(mj, mk)   ! C = A x B
          end do
          C(mi, mk) = temp1           ! store C element
        end do
      end do
    end do
  end do
  call cpu_time(t2)                   ! elapsed time calculation
  write(*, "(A, I5, A, I3, A, F8.4)") 'Serial version: matrix size:', ms,  &
        ',  Emulated Processes:', p, ',  Elapsed time [s]:', t2 - t1
end program ilist05s512
```

Resultados:

```
Serial version: matrix size:  512,  Emulated Processes:  2,  Elapsed time [s]:
1.3891
[xxxxxxx.xxxxxxx@sdumont13 ~]$ ./ilist05s512p4
Serial version: matrix size:  512,  Emulated Processes:  4,  Elapsed time [s]:
1.3570
[xxxxxxx.xxxxxxx@sdumont13 ~]$ ./ilist05s512p8
Serial version: matrix size:  512,  Emulated Processes:  8,  Elapsed time [s]:
1.3226
```

PARALELIZANDO

Programa:

```
! CAP372 - exercise 5 - 2019-09-15 - Parallel version
! Based on "Introduction to Parallel Programming: Matrix Multiplication"
! by  Gergel V.P. :
! <http://www.lac.inpe.br/~stephan/CAP-372/matrixmult_microsoft.pdf>
! - Each subtask i hold one row of matrix A and one column of matrix B .
! - The result is stored in one element of C .
! - After that, every subtask i, 0≤i<n, transmits its column of matrix B
!   to the subtask with the number (i + 1) mod n .
! - When the number of processors ("p") is less than the number of
!   basic subtasks ("n"), each processor would execute several
!   inner products of matrix A rows and matrix B columns.
! - Each aggregated basic subtask ( = the calculation of one row of C)
!   determines several rows of the result matrix C.
! module load intel_psxe/2019
! mpiifort -o ilist05parallel list05parallel.f90
! mpirun -n 2 ./ilist05parallel

program list05parallel
  use MPI
  implicit none

  ! C =  A * B matrix initialization
  integer, parameter :: ms=4                             ! matrix size ms x ms
  integer, parameter :: ps=2                             ! expected processes
  integer, parameter :: st=ms/ps                         ! # of stripes
  double precision, dimension(ms, ms) :: A, B, C=0       ! matrix definition
  double precision, dimension(st, ms) :: AR=0            ! stripes: A Rows
  double precision, dimension(ms, st) :: BC=0            ! stripes: B Cols
  double precision, dimension(ms, st) :: CS=0            ! stripes: C Stripe
  double precision, dimension(ms)     :: AT=0, BT=0      ! temporary

  integer :: stripe_i, stripe_countA, stripe_countB      ! counters
  integer :: mi, mj, mk, source, dest, i                 ! indexes
  integer :: interation, subtask                         ! looping
  double precision :: time1, time2                       ! time elapsed
  double precision :: temp1                              ! temporary value

  ! MPI Initialization
  integer :: my_rank, sender, p, ierr                    ! mpi variables
  integer :: tagC=0                                      ! variable tag
  integer, parameter :: tag=0, tagB=99999                ! fixed tag
  integer, dimension(MPI_STATUS_SIZE) :: status          ! mpi status
  call MPI_Init(ierr)                                    ! mpi initialize
  call MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)      ! mpi processor's id
  call MPI_Comm_size(MPI_COMM_WORLD, p, ierr)            ! mpi # of processors
  if (p .ne. ps) then                                    ! abort if ps <> p
    if (my_rank .eq. 0) then
      print*, "Error: # of mpi processes is", p, " and the expected is", ps
    endif
    stop
  end if

  ! *** Each subtask hold one row|colomn of matrix A|B and return C ***
  ! The rank 0 has the entire matrix A|B .
  ! Matrix initialization:
  if (my_rank .eq. 0) then
    ! Example 4 x 4 (to check if everything is ok):
    A = transpose(reshape([ 5, 2, 6, 1,                  &
                            0, 6, 2, 0,                  &
                            3, 8, 1, 4,                  &
                            1, 8, 5, 6 ], shape(A)))
    B = transpose(reshape([ 7, 5, 8, 0,                  &
                            1, 8, 2, 6,                  &
                            9, 4, 3, 8,                  &
                            5, 3, 7, 9 ], shape(B)))
    ! The result should be:
    !   96  68  69  69
```

```fortran
    !   24  56  18  52
    !   58  95  71  92
    !   90  107 81  142
    ! Example ms x ms:
    ! A = reshape([(mi, mi=1, ms*ms)], shape(A))
    ! B = reshape([(mi, mi=ms*ms+1, ms*ms*2)], shape(A))
    ! Elapsed time calculation:
    call cpu_time(time1)
  end if


! ########## BEGIN SEND DATA TO THE FIRST INTERATION ##########################
  ! Each subtask is one processor.
  ! Row A # is the same for the same subtask #, and then it is only
  ! necessary send|recv one time.
  if (my_rank .eq. 0) then               ! Rank 0 has the entire matrix.
    do dest = 1, ms / st -1              ! Send to the subtask.
      do stripe_countA = 1, st           ! Send the stripe.
        mi = dest * st + stripe_countA  ! calculate stripe line
        ! Send row by row. Can be optimized to send a block.
        AT = A(mi, :)
        call MPI_Send(AT, ms, MPI_DOUBLE_PRECISION,    &
                      dest, tag, MPI_COMM_WORLD, ierr)
      end do  ! stripe_countA
    end do  ! dest
    do mi = 1, st                        ! Rank 0: copy A stripe to AR
        AR(mi, :) = A(mi, :)
    end do ! mi
  end if

  ! And then each subtask i, 0<i<n,receives. The i=0 is not necessary.
  if (my_rank .ne. 0) then               ! Rank > 0 receive the stripe.
    do stripe_countA = 1, st             ! Recv the A stripe.
      ! For sure it can be optimized, instead of receiving line by line.
      call MPI_Recv(AT, ms, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,    &
                    MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)
      AR(stripe_countA, :) = AT
    end do  ! stripe_countA
  end if  ! my_rank .ne. 0

  ! In the first interaction, the subtasks should receive initial B row.
  ! In the next interations, each substask send to another.
  ! First time B row send:
  if (my_rank .eq. 0) then               ! Rank 0 has the entire matrix.
    interation = 0                       ! Interation of the algorithm.
    do subtask = 0, ms / st - 1          ! Each subtask hold one matrix row.
      do stripe_countB = 0,  st - 1      ! Count for p < n .
        ! stripe column calculation
        stripe_i = st*mod(subtask+p-interation,p)+stripe_countB
        mk = stripe_i + 1                ! Matrix B column index
        if (subtask .eq. 0) then
          BC(:, mk) = B(:, mk)           ! Rank 0 only copy
        else
          BT = B(:, mk)                  ! Send to rank > 0
          dest = subtask
          call MPI_Send(BT, ms, MPI_DOUBLE_PRECISION,    &
                        dest, tag, MPI_COMM_WORLD, ierr)
        endif  ! subtask
      end do  ! stripe_countB
    end do  ! subtask
  end if  ! my_rank

  ! Recv B row that was sent
  if (my_rank .ne. 0) then
    do stripe_countB = 0,  st - 1       ! Count for p < n
      call MPI_Recv(BT, ms, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,    &
          MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)
      BC(:, stripe_countB + 1) = BT
    end do  ! stripe_countB
  end if  ! my_rank
! ########## END SEND DATA FOR THE FIRST INTERATION ############################
! ++++++++++ START INTERATION ALGORITHM ++++++++++++++++++++++++++++++++++++++++++
  do interation = 0, ms / st - 1       ! Algorithm interation.
    ! Every subtask receives its row|col of matrix A|B
    subtask = my_rank                  ! Each subtask hold one matrix A|B row|col
    ! >>>>>>>>>> START STRIPE <<<<<<<<<<
    do stripe_countA = 0, st - 1       ! A count for p < n
      do stripe_countB = 0,  st - 1    ! count for p < n
        ! stripe column calculation
        stripe_i = st * mod(subtask+p-interation,p) + stripe_countB
        ! matrix A line calculation
        mi = st * subtask + stripe_countA + 1
        temp1 = 0                      ! multiplication: A line x B column
        do mj = 1, ms                  ! count A column
          mk = stripe_i + 1            ! B column index
          ! C = A x B. Can be optimized using intrinsic functions
          temp1 = temp1 + AR(stripe_countA+1, mj) * BC(mj, stripe_countB+1)
        end do
        CS(stripe_countA+1, stripe_countB+1) = temp1       ! store C element
      end do
    end do    ! >>>>>>>>>> END STRIPE <<<<<<<<<<


    ! $$$$$$$$$$ send C stripe to rank 0 $$$$$$$$$$
    ! Each subtask should return the calculated C to rank 0
    if (my_rank .ne. 0) then      ! Rank 0 does not need to send
      tagC = interation
      do mj = 1,  st                  ! Send one stripe.
        call MPI_Send(CS(:,mj), st, MPI_DOUBLE_PRECISION, 0, tagC, &
                      MPI_COMM_WORLD, ierr)
      end do
    end if

    ! Rank 0 is responsible for completing the matrix C
    if (my_rank .eq. 0) then
      ! For sure there is a whay to optimize this part of code
      ! SEND C stripe
      do stripe_countA = 0, st - 1
        do stripe_countB = 0, st - 1
          stripe_i = st * mod(0 + p - interation, p) + stripe_countB + 1
          mi = stripe_countA + 1
          C(mi, stripe_i) = CS(stripe_countA + 1, stripe_countB + 1)
        end do
      end do
      ! RECV C stripe
      do source = 1,  ps - 1      ! recv from ranks > 0 and store in C
        ! Receive the corresponding C
        do mj = 1,  st                 ! Recv one stripe.
          call MPI_Recv(CS(:, mj), st * st, MPI_DOUBLE_PRECISION, source, &
                        MPI_ANY_TAG, MPI_COMM_WORLD, status, ierr)
        end do
        sender   = status(MPI_SOURCE)
        i        = status(MPI_TAG)
        ! For sure there is a whay to optimize this part of code
        do stripe_countA = 0, st - 1
          do stripe_countB = 0,  st - 1
            mi = st * sender + stripe_countA + 1  ! Calculate C line index
            stripe_i = st * mod(sender + p - i, p) + stripe_countB
            mk = stripe_i + 1      ! B column index
            C(mi , mk) = CS(stripe_countA + 1, stripe_countB + 1)
          end do  ! stripe_countA
        end do  ! stripe_countB
      end do  ! source
    end if  ! $$$$$$$$$$ END OF SEND C $$$$$$$$$$


    ! @@@@@@@@@@ send and recv B @@@@@@@@@@
    ! At the end of one interation, every subtask transmits its column
    ! of matrix B to the subtask with the number (i + 1) mod n .
    ! NOTE: the count index in Intel MPI implementation is a 4-byte integer,
    ! so the maximum allowed value is 2^31-1 : 2.147.483.647 .
    ! I had a lot o trouble with this..........
    if (interation .lt. (ms / st - 1)) then   ! Not need in the last interation.
      dest = mod(my_rank + 1, ps)
      do mj = 1,  st                  ! Send one stripe.
```

```
        ! tagAB ensures no confusion with matrix C
        call MPI_Send(BC(:,mj), ms, MPI_DOUBLE_PRECISION, dest,   &
                      tagB, MPI_COMM_WORLD, ierr)
      end do
      ! Each subtask recv
      do mj = 1,  st     ! Recv one stripe.
        ! tagAB ensures no confusion with matrix C
        call MPI_Recv(BC(:,mj), ms, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,  &
                      tagB, MPI_COMM_WORLD, status, ierr)
      end do
    end if

  end do  ! ++++++++++ END INTERATION ++++++++++


  ! %%%%%%%%%% SHOW THE RESULT %%%%%%%%%%
  if (my_rank .eq. 0) then
    ! Uncomment to show the multiplication result C
        print*, "Result Matrix C ="
        do mi = 1, ms
          print*, int(C(mi,:))
        end do
    call cpu_time(time2)                    ! elapsed time calculation
    write(*, "(A, I5, A, I3, A, F8.4)") "Matrix size:", ms, &
          ",   Processors:", p, ",   Elapsed time [s]: ", time2 - time1
  end if

  call MPI_Finalize(ierr)
end program list05parallel
```

Rodando a matriz de teste para verificar o funcionamento:

```
$ mpif90 -Og -Wall -fcheck=all list05parallel.f90
$ mpirun -n 2 ./a.out
 Result Matrix C =
          96          68          69          69
          24          56          18          52
          58          95          71          92
          90         107          81         142
Matrix size:    4,   Processors:  2,   Elapsed time [s]:   0.0001
```

O programa foi feito de forma rápida e pode ser melhorado. O foco foi fazer funcionar em uma semana. Destes 7 dias, 1 dia foi gasto com Fortran e algoritmo. Os demais 6 dias foram gastos com o MPI, sendo 1 dia só com um problema intermitente com um MPI_Send que em algus casos travava silenciosamente e não emitia nenhuma mensagem....
A ideia do programa era fazer um passo-a-passo tentando seguir o que está escrito nos slides. Ele ainda não está finalizado. É apenas uma primeira versão para atender o prazo de entrega.
O programa começa definindo variáveis, inicializando o MPI, e construindo as matrizes. Em seguida envia os dados para a primeira interação do algoritmo. Na sequência executa a primeira interação. Depois envia o resultado parcial para o rank 0 colocar na matriz C. Na sequência envia o "stripe" da matriz B para o próximo rank conforme descrição do algoritmo. O próximo passo é repetir a interação até que todos os elementos de C sejam calculados. O resultado final aparece na matriz C.

Testes na máquina local (Intel i7) usando ifort :

```
$ mpiifort -g -check all -fpe0 -warn -traceback -debug extended -o ilist05parallel-
512-2 list05parallel.f90
$ mpirun -n 2 ./ilist05parallel-512-2
Matrix size:  512,   Processors:  2,   Elapsed time [s]:   0.5320
```

```
$ mpiifort -g -check all -fpe0 -warn -traceback -debug extended list05parallel.f90
-o ilist05parallel-512-4
$ mpirun -n 4 ./ilist05parallel-512-4
Matrix size:  512,   Processors:  4,   Elapsed time [s]:   0.2669
```

```
$ mpiifort -g -check all -fpe0 -warn -traceback -debug extended list05parallel.f90
-o ilist05parallel-512-8
$ mpirun -n 8 ./ilist05parallel-512-8
Matrix size:  512,   Processors:  8,   Elapsed time [s]:   0.1552
```

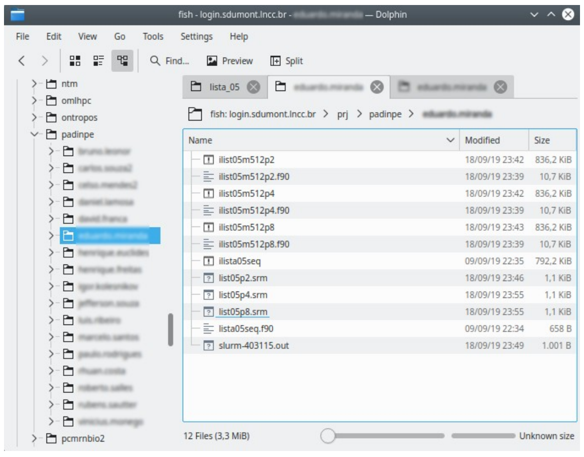
```
$ mpiifort -g -check all -fpe0 -warn -traceback -debug extended list05parallel.f90
-o ilist05parallel-1024-2
$ mpirun -n 2 ./ilist05parallel-1024-2
Matrix size: 1024,   Processors:  2,   Elapsed time [s]:   9.8530
```

```
$ mpiifort -g -check all -fpe0 -warn -traceback -debug extended list05parallel.f90
-o ilist05parallel-1024-4
$ mpirun -n 4 ./ilist05parallel-1024-4
Matrix size: 1024,   Processors:  4,   Elapsed time [s]:   2.3534
```

```
$ mpiifort -g -check all -fpe0 -warn -traceback -debug extended list05parallel.f90
-o ilist05parallel-1024-8
$ mpirun -n 8 ./ilist05parallel-1024-8
Matrix size: 1024,   Processors:  8,   Elapsed time [s]:   1.1655
```

RODANDO NO SANTOS DUMONT

Arquivos

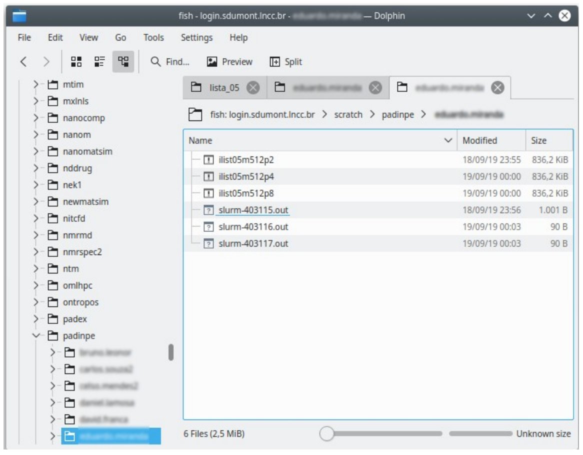


fish - login.sdumont.lncc.br - [redacted] — Dolphin

fish: login.sdumont.lncc.br > prj > padinpe > [redacted]

Name	Modified	Size
ilist05m512p2	18/09/19 23:42	836,2 KiB
ilist05m512p2.f90	18/09/19 23:39	10,7 KiB
ilist05m512p4	18/09/19 23:42	836,2 KiB
ilist05m512p4.f90	18/09/19 23:39	10,7 KiB
ilist05m512p8	18/09/19 23:43	836,2 KiB
ilist05m512p8.f90	18/09/19 23:39	10,7 KiB
ilista05seq	09/09/19 22:35	792,2 KiB
list05p2.srm	18/09/19 23:46	1,1 KiB
list05p4.srm	18/09/19 23:55	1,1 KiB
list05p8.srm	18/09/19 23:55	1,1 KiB
lista05seq.f90	09/09/19 22:34	658 B
slurm-403115.out	18/09/19 23:49	1.001 B

12 Files (3,3 MiB)



fish - login.sdumont.lncc.br - [redacted] — Dolphin

fish: login.sdumont.lncc.br > scratch > padinpe > [redacted]

Name	Modified	Size
ilist05m512p2	18/09/19 23:55	836,2 KiB
ilist05m512p4	19/09/19 00:00	836,2 KiB
ilist05m512p8	19/09/19 00:00	836,2 KiB
slurm-403115.out	18/09/19 23:56	1.001 B
slurm-403116.out	19/09/19 00:03	90 B
slurm-403117.out	19/09/19 00:03	90 B

6 Files (2,5 MiB)

slurm-403115.out

```
dumont1292
sdumont1292
        linux-vdso.so.1 =>  (0x00007fff87daa000)
        libmpifort.so.12 =>
/opt/intel/parallel_studio_xe_2019/compilers_and_libraries_2019.3.199/linux/mpi/
intel64/lib/libmpifort.so.12 (0x00002b47c0382000)
        libmpi.so.12 =>
/opt/intel/parallel_studio_xe_2019/compilers_and_libraries_2019.3.199/linux/mpi/
intel64/lib/release/libmpi.so.12 (0x00002b47c0740000)
        libdl.so.2 => /usr/lib64/libdl.so.2 (0x00002b47c18ce000)
        librt.so.1 => /usr/lib64/librt.so.1 (0x00002b47c1ad2000)
        libpthread.so.0 => /usr/lib64/libpthread.so.0 (0x00002b47c1cda000)
        libm.so.6 => /usr/lib64/libm.so.6 (0x00002b47c1ef6000)
        libc.so.6 => /usr/lib64/libc.so.6 (0x00002b47c21f8000)
        libgcc_s.so.1 => /usr/lib64/libgcc_s.so.1 (0x00002b47c25c5000)
        libfabric.so.1 =>
/opt/intel/parallel_studio_xe_2019/compilers_and_libraries_2019.3.199/linux/mpi/
intel64/libfabric/lib/libfabric.so.1 (0x00002b47c27db000)
        /lib64/ld-linux-x86-64.so.2 (0x00002b47c015e000)
Matrix size:  512,   Processors:  2,   Elapsed time [s]:   0.6700
```

slurm-403116.out

```
sdumont1292
sdumont1292
Matrix size:  512,   Processors:  4,   Elapsed time [s]:   0.3713
```

slurm-403117.out

```
sdumont1292
sdumont1292
Matrix size:  512,   Processors:  8,   Elapsed time [s]:   0.2085
```

TEMPORIZAÇÃO DO PROCESSAMENTO REFERENTE AOS CÁLCULOS E ÀS COMUNICAÇÕES

Foi utilizado o seguinte programa, que é o mesmo anterior acrescentado de “MPI_Wtime”:

```fortran
program list05p
! CAP372 - exercise 5 - 2019-09-15 - Parallel version
! Based on "Introduction to Parallel Programming: Matrix Multiplication"
! by  Gergel V.P. :
! <http://www.lac.inpe.br/~stephan/CAP-372/matrixmult_microsoft.pdf>
! - Each subtask i hold one row of matrix A and one column of matrix B .
! - The result is stored in one element of C .
! - After that, every subtask i, 0≤i<n, transmits its column of matrix B
!   to the subtask with the number (i + 1) mod n .
! - When the number of processors ("p") is less than the number of
!   basic subtasks ("n"), each processor would execute several
!   inner products of matrix A rows and matrix B columns.
! - Each aggregated basic subtask ( = the calculation of one row of C)
!   determines several rows of the result matrix C.
! module load intel_psxe/2019
! mpiifort -g -check all -fpe0 -warn -traceback -debug extended  &
!           -o ilist05p ilist05p.f90
! mpirun -n 2 ./ilist05p
! or if using gnu: mpif90 -Og -fcheck=all list05p.f90

  use MPI
  implicit none

  integer, parameter :: ms=512                         ! matrix size ms x ms
  integer, parameter :: ps=2                           ! expected processes
  integer, parameter :: st=ms/ps                       ! # of stripes
  double precision, dimension(ms, ms) :: A, B, C=0     ! matrix definition
  double precision, dimension(st, ms) :: AR=0          ! stripes: A Rows
  double precision, dimension(ms, st) :: BC=0          ! stripes: B Cols
  double precision, dimension(ms, st) :: CS=0          ! stripes: C Stripe
  double precision, dimension(ms)     :: AT=0, BT=0    ! temporary
  integer :: stripe_i, stripe_countA, stripe_countB    ! counters
  integer :: mi, mj, mk, source, dest, i               ! indexes
  integer :: interation, subtask                       ! looping
  double precision :: t1, t2, ttot, tcal=0, tcom=0     ! time elapsed
  double precision :: temp1                            ! temporary value

  ! MPI Initialization
  integer :: my_rank, sender, p, ierr                  ! mpi variables
  integer :: tagC=0                                    ! variable tag
  integer, parameter ::  &
    tag=0,   &                        ! general use tag
    tagB=99999,   &                   ! tag for B matrix send
    tagTcom=99998,   &                ! tag for communication time elapsed
    tagTcal=99997                     ! tag for calculation time elapsed
  integer, dimension(MPI_STATUS_SIZE) :: status        ! mpi status
  call MPI_Init(ierr)                                  ! mpi initialize
  call MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)    ! mpi processor's id
  call MPI_Comm_size(MPI_COMM_WORLD, p, ierr)          ! mpi # of processors
  if (p .ne. ps) then                                  ! abort if ps <> p
    if (my_rank .eq. 0) then
      print*, "Error: # of mpi processes is", p, " and the expected is", ps
    endif
    stop
  end if
  if (my_rank .eq. 0) then
    ttot = MPI_Wtime()                ! Elapsed time calculation.
    ! *** Each subtask hold one row|colomn of matrix A|B and return C ***
    ! The rank 0 has the entire matrix A|B .
    A = reshape([(mi, mi=1, ms*ms)], shape(A))
    B = reshape([(mi, mi=ms*ms+1, ms*ms*2)], shape(B))
  end if

  !########## BEGIN SEND DATA TO THE FIRST INTERATION #########################
```

```
! Each subtask is one processor.
! Row A # is the same for the same subtask #, and then it is only
! necessary send|recv one time.
if (my_rank .eq. 0) then              ! Rank 0 has the entire matrix.
  t1 = MPI_Wtime()                    ! Start elapsed time calculation.
  do dest = 1, ms / st -1             ! Send to the subtask.
    do stripe_countA = 1, st          ! Send the stripe.
      mi = dest * st + stripe_countA  ! calculate stripe line
      ! Send row by row. Can be optimized to send a block.
      AT = A(mi, :)
      call MPI_Send(AT, ms, MPI_DOUBLE_PRECISION,   &
                    dest, tag, MPI_COMM_WORLD, ierr)
    end do  ! stripe_countA
  end do  ! dest
  do mi = 1, st                       ! Rank 0: copy A stripe to AR
      AR(mi, :) = A(mi, :)
  end do ! mi
end if
! And then each subtask i, 0<i<n,receives. The i=0 is not necessary.
if (my_rank .ne. 0) then              ! Rank > 0 receive the stripe.
  do stripe_countA = 1, st            ! Recv the A stripe.
    ! For sure it can be optimized, instead of receiving line by line.
    call MPI_Recv(AT, ms, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,   &
                  MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)
    AR(stripe_countA, :) = AT
  end do  ! stripe_countA
  t2 = MPI_Wtime() - t1        ! Elapsed time to send initial data
  tcom = tcom + t2
end if  ! my_rank .ne. 0

! In the first interaction, the subtasks should receive initial B row.
! In the next interations, each substask send to another.
! First time B row send:
if (my_rank .eq. 0) then              ! Rank 0 has the entire matrix.
  interation = 0                      ! Interation of the algorithm.
  do subtask = 0, ms / st - 1         ! Each subtask hold one matrix row.
    do stripe_countB = 0,  st - 1     ! Count for p < n .
      ! stripe column calculation
      stripe_i = st * mod(subtask + p - interation, p) + stripe_countB
      mk = stripe_i + 1               ! Matrix B column index
      if (subtask .eq. 0) then
        BC(:, mk) = B(:, mk)          ! Rank 0 only copy
      else
        BT = B(:, mk)                 ! Send to rank > 0
        dest = subtask
        call MPI_Send(BT, ms, MPI_DOUBLE_PRECISION,   &
                      dest, tag, MPI_COMM_WORLD, ierr)
      endif  ! subtask
    end do  ! stripe_countB
  end do  ! subtask
end if  ! my_rank

! Recv B row that was sent
if (my_rank .ne. 0) then
  do stripe_countB = 0,  st - 1         ! Count for p < n
    call MPI_Recv(BT, ms, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,   &
        MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE, ierr)
    BC(:, stripe_countB + 1) = BT
  end do  ! stripe_countB
end if  ! my_rank

!++++++++++ START INTERATION ALGORITHM ++++++++++++++++++++++++++++++++++++++
do interation = 0, ms / st - 1        ! Algorithm interation.
  t1 = MPI_Wtime()                    ! Start elapsed time calculation.
  ! Every subtask receives its row|col of matrix A|B
  subtask = my_rank                   ! Each subtask hold one matrix A|B row|col
  ! >>>>>>>>>> START STRIPE <<<<<<<<<<
  do stripe_countA = 0, st - 1        ! A count for p < n
    do stripe_countB = 0,  st - 1     ! count for p < n
      ! stripe column calculation
      stripe_i = st * mod(subtask+p-interation,p) + stripe_countB
      ! matrix A line calculation
      mi = st * subtask + stripe_countA + 1
      temp1 = 0                       ! multiplication: A line x B column
```

```fortran
        do mj = 1, ms                    ! count A column
          mk = stripe_i + 1              ! B column index
          ! C = A x B. Can be optimized using intrinsic functions
          temp1 = temp1 + AR(stripe_countA+1, mj) * BC(mj, stripe_countB+1)
        end do
        CS(stripe_countA+1, stripe_countB+1) = temp1  ! store C element
      end do
    end do   ! >>>>>>>>>> END STRIPE <<<<<<<<<<
    ! Each rank has their partial time and must send to the rank 0
    t2 = MPI_Wtime() - t1
    if (my_rank .ne. 0) then             ! rank 0 accumulate
      call MPI_Send(t2, 1, MPI_DOUBLE_PRECISION, 0,  &
                   tagTcal, MPI_COMM_WORLD, ierr)
    else
      tcal = tcal + t2                 ! rank 0 own time
      call MPI_Recv(t2, 1, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,  &
                   tagTcal, MPI_COMM_WORLD, status, ierr)
      tcal = tcal + t2                 ! other rank time
    endif  ! End calculation elapsed time calculation

    ! $$$$$$$$$$ send C stripe to rank 0 $$$$$$$$$$
    ! Each subtask should return the calculated C to rank 0
    t1 = MPI_Wtime()                     ! Start elapsed time calculation.
    if (my_rank .ne. 0) then             ! Rank 0 does not need to send
      tagC = interation
      do mj = 1,  st                       ! Send one stripe.
        call MPI_Send(CS(:,mj), st, MPI_DOUBLE_PRECISION, 0, tagC, &
                     MPI_COMM_WORLD, ierr)
      end do
    end if
    ! Rank 0 is responsible for completing the matrix C
    if (my_rank .eq. 0) then
      ! For sure there is a whay to optimize this part of code
      ! SEND C stripe
      do stripe_countA = 0, st - 1
        do stripe_countB = 0, st - 1
          stripe_i = st * mod(0 + p - interation, p) + stripe_countB + 1
          mi = stripe_countA + 1
          C(mi, stripe_i) = CS(stripe_countA + 1, stripe_countB + 1)
        end do
      end do
      ! RECV C stripe
      do source = 1,  ps - 1             ! recv from ranks > 0 and store in C
        ! Receive the corresponding C
        do mj = 1,  st                   ! Recv one stripe.
          call MPI_Recv(CS(:, mj), st * st, MPI_DOUBLE_PRECISION, source, &
                       MPI_ANY_TAG, MPI_COMM_WORLD, status, ierr)
        end do
        sender  = status(MPI_SOURCE)
        i       = status(MPI_TAG)
        ! For sure there is a whay to optimize this part of code
        do stripe_countA = 0, st - 1
          do stripe_countB = 0,  st - 1
            mi = st * sender + stripe_countA + 1      ! Calculate C line index
            stripe_i = st * mod(sender + p - i, p) + stripe_countB
            mk = stripe_i + 1              ! B column index
            C(mi , mk) = CS(stripe_countA + 1, stripe_countB + 1)
          end do  ! stripe_countA
        end do  ! stripe_countB
      end do  ! source
    end if  ! $$$$$$$$$$ END OF SEND C $$$$$$$$$$

    ! @@@@@@@@@@ send and recv B @@@@@@@@@@
    ! At the end of one interation, every subtask transmits its column
    ! of matrix B to the subtask with the number (i + 1) mod n .
    ! NOTE: the count index in Intel MPI implementation is a 4-byte integer,
    ! so the maximum allowed value is 2^31-1 : 2.147.483.647 .
    ! I had a lot o trouble with this..........
    if (interation .lt. (ms / st - 1)) then   ! Not need in the last interation.
      dest = mod(my_rank + 1, ps)
      do mj = 1,  st                     ! Send one stripe.
        ! tagAB ensures no confusion with matrix C
        call MPI_Send(BC(:,mj), ms, MPI_DOUBLE_PRECISION, dest,  &
                     tagB, MPI_COMM_WORLD, ierr)
      end do
      ! Each subtask recv
      do mj = 1,  st                    ! Recv one stripe.
        ! tagAB ensures no confusion with matrix C
        call MPI_Recv(BC(:,mj), ms, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,  &
                      tagB, MPI_COMM_WORLD, status, ierr)
      end do
    end if

    ! Each rank has their partial time and must send to the rank 0
    t2 = MPI_Wtime() - t1
    if (my_rank .ne. 0) then            ! rank 0 accumulate
      call MPI_Send(t2, 1, MPI_DOUBLE_PRECISION, 0,  &
                    tagTcom, MPI_COMM_WORLD, ierr)
    else
      tcom = tcom + t2                ! rank 0 own time
      call MPI_Recv(t2, 1, MPI_DOUBLE_PRECISION, MPI_ANY_SOURCE,  &
                    tagTcom, MPI_COMM_WORLD, status, ierr)
      tcom = tcom + t2                ! other rank time
    endif  ! End communication elapsed time calculation

  end do  ! ++++++++++ END INTERATION ++++++++++


  !%%%%%%%%%% SHOW THE RESULT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  if (my_rank .eq. 0) then
    ttot = MPI_Wtime() - ttot           ! Elapsed time calculation.
    write(*, '(a, i5, 3x, a, i2, 3x, a, f7.4, 3x, a, f7.4, 3x, a, f7.4)')   &
      'Matrix size:', ms,  &
      'Processors:', p,  &
      'Total time:', ttot,  &
      'Comm time:', tcom, &
      'Calc time:', tcal
  end if

  call MPI_Finalize(ierr)               ! mpi finalize
end program list05p
```

Resultado obtido rodando na máquina local:

```
$ mpif90 -Og -fcheck=all list05p.f90
$ mpirun -n 2 ./a.out
Matrix size: 512   Processors: 2   Total time: 0.2387   Comm time: 0.0058   Calc time: 0.4517
```

onde:

Total time	-	tempo medido do início ao final do Rank 0
Comm time	-	soma de todos os tempos de comunicação dos Ranks
Calc time	-	soma de todos os tempos de todos os Ranks (soma dos Ranks que estão rodando em paralelo)

SPEED UP

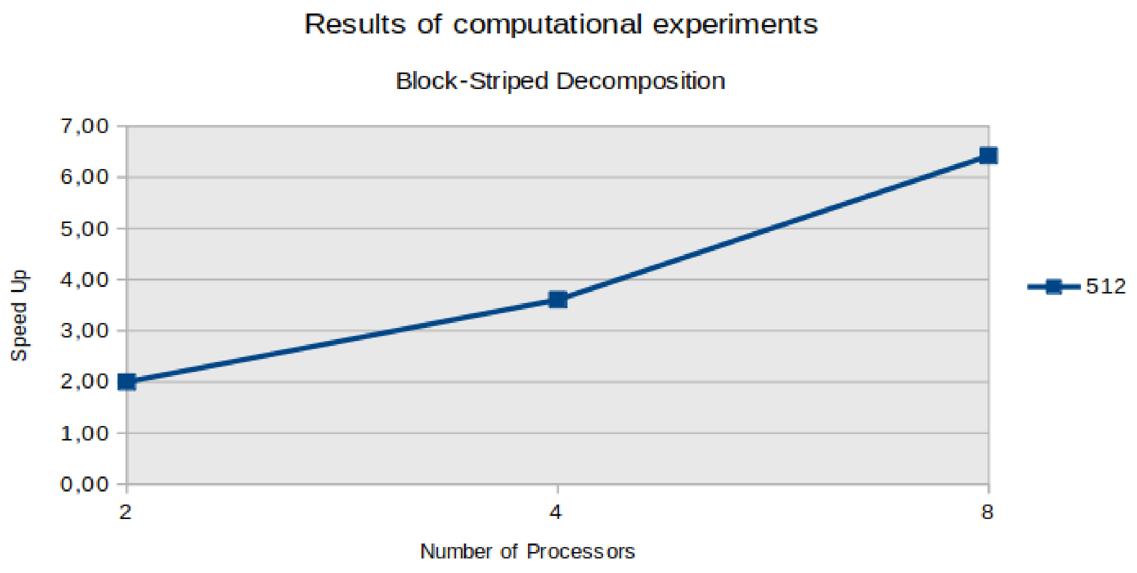
O Speedup é o tempo de um programa série dividido pelo tempo do programa paralelo:

S = 1.3891 / 0.6700 = 2,07

A Eficiência é o Speedup dividido pelo nro. de processadores:

E = 2,07 / 2 = 1,04

Matrix size	Serial Algorithm	2 processors		4 processors		8 processors	
		Time	Seep UP	Time	Seep UP	Time	Seep UP
512	1,33891	0,67	2,00	0,3713	3,61	0,2085	6,42



REFERÊNCIAS

- http://www.lac.inpe.br/~stephan/CAP-372/
- https://stackoverflow.com
- https://annefou.github.io/Fortran/
- http://www.mathcs.emory.edu/~cheung/Courses/561/Syllabus/syl.html
- https://web.stanford.edu/class/me200c/tutorial_90/
- http://userweb.eng.gla.ac.uk/peter.smart/com/fortran.htm
- https://software.intel.com/en-us/fortran-compilers
- http://fortranwiki.org
- https://events.prace-ri.eu/event/176/contributions/57/attachments/148/296/Advanced_MPI_I.pdf
- https://people.sc.fsu.edu/~jburkardt/f_src/mpi/matvec_mpi.f90
- https://www.dartmouth.edu/~rc/classes/intro_mpi/
- https://en.wikibooks.org/wiki/Fortran/
- http://people.ds.cam.ac.uk/nmm1/MPI
- https://events.prace-ri.eu/event/176/contributions/71/attachments/162/317/Advanced_MPI_I.pdf