Parallel implementation in Chapel for the numerical solution of the 3D Poisson problem

CHIUW 2023

Anna Caroline F. S. de Jesus[1], Willian Carlos Lesinhovski[2], Livia S. Freire[1] and Nelson Luis Dias[2]

[1]University of São Paulo
Laboratory of Applied Mathematics and Scientific Computing (LMACC)
[2] Federal University of Paraná

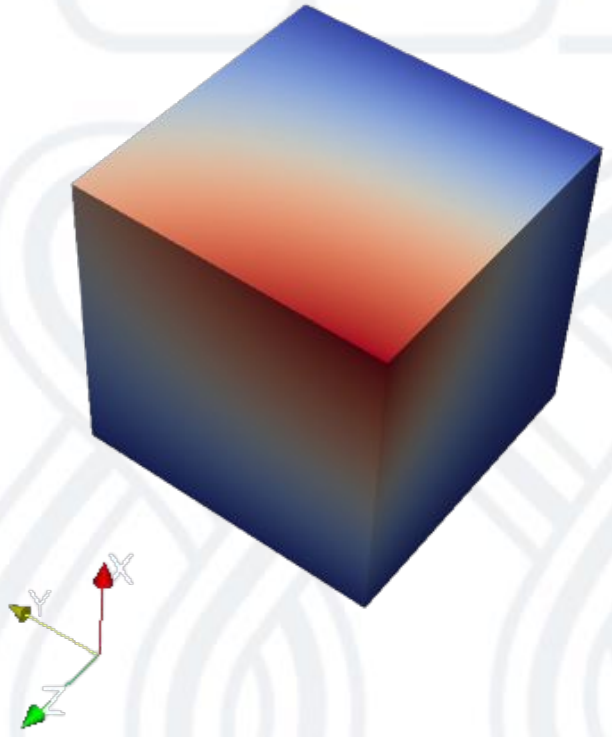
Introduction

- Finite Difference Method
- Poisson Equation
- Method of manufactured solutions

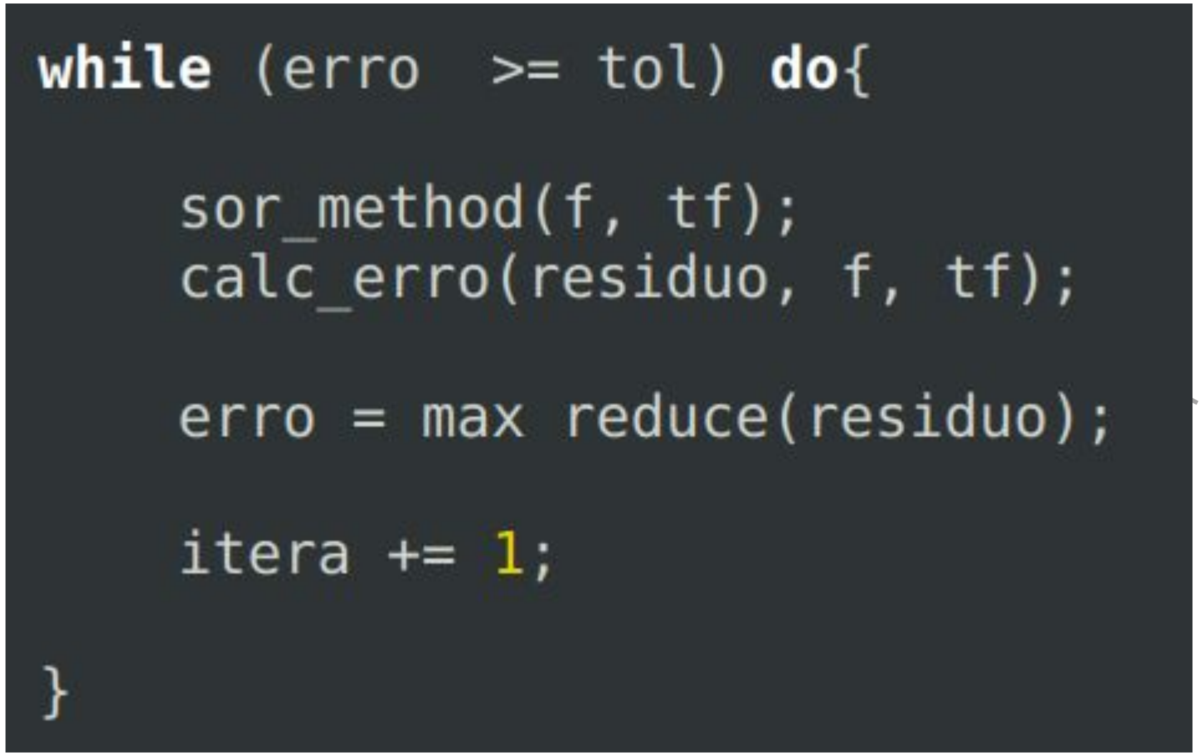
$$\left[\nabla^2 f = T\right]$$

$$\left[f^{n+1} = (1-w)\, f^{n} + w\, f^{*}\right]$$

(SOR) method

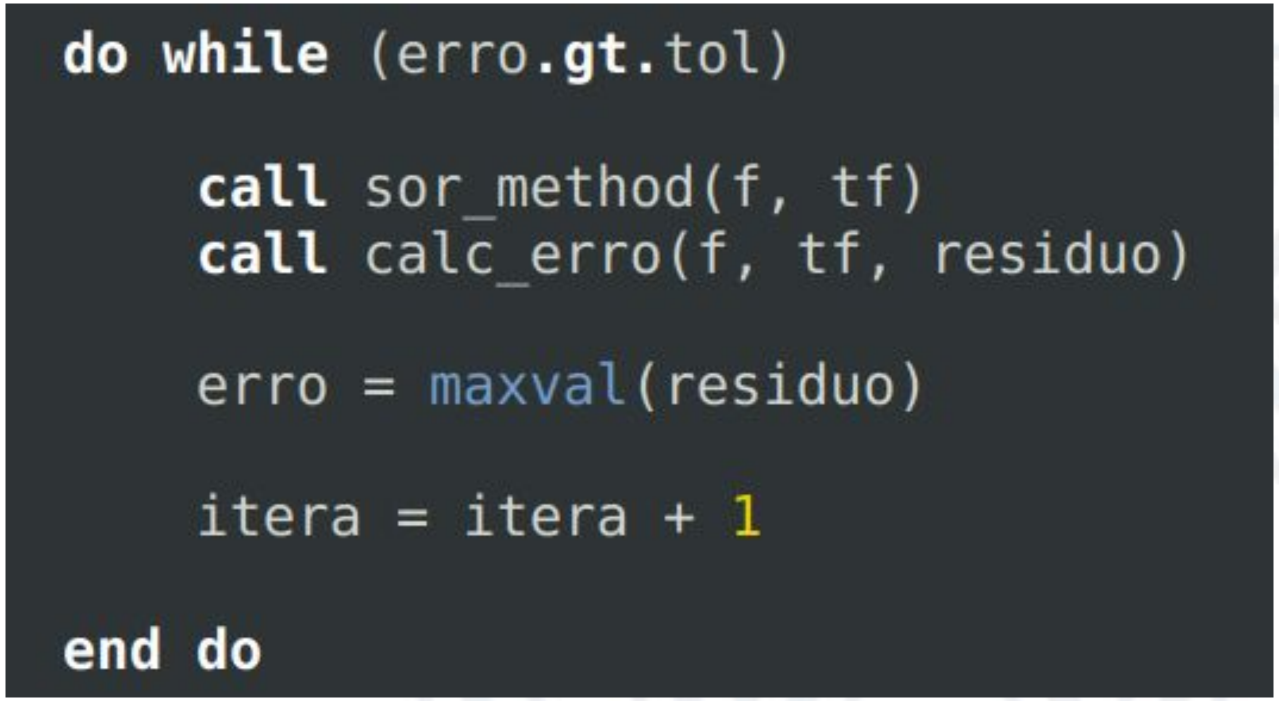


How many lines are necessary to its implementation?



```
while (erro  >= tol) do{

    sor_method(f, tf);
    calc_erro(residuo, f, tf);

    erro = max reduce(residuo);

    itera += 1;

}
```

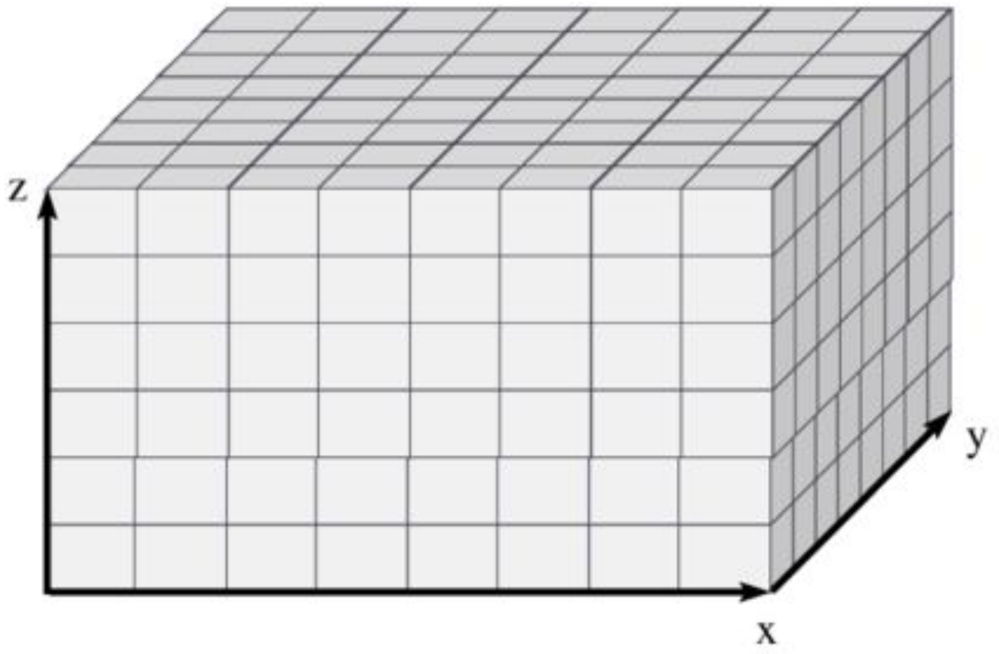
Serial Version
Chapel

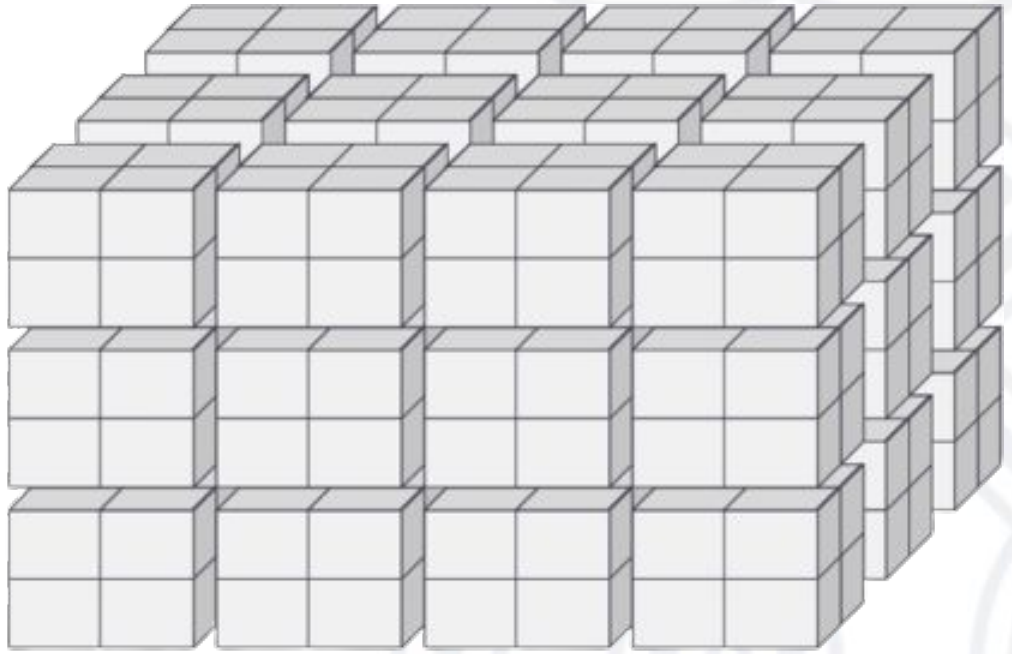


```
do while (erro.gt.tol)

    call sor_method(f, tf)
    call calc_erro(f, tf, residuo)

    erro = maxval(residuo)

    itera = itera + 1

end do
```

Serial Version
Fortran

How many lines are necessary to its implementation?

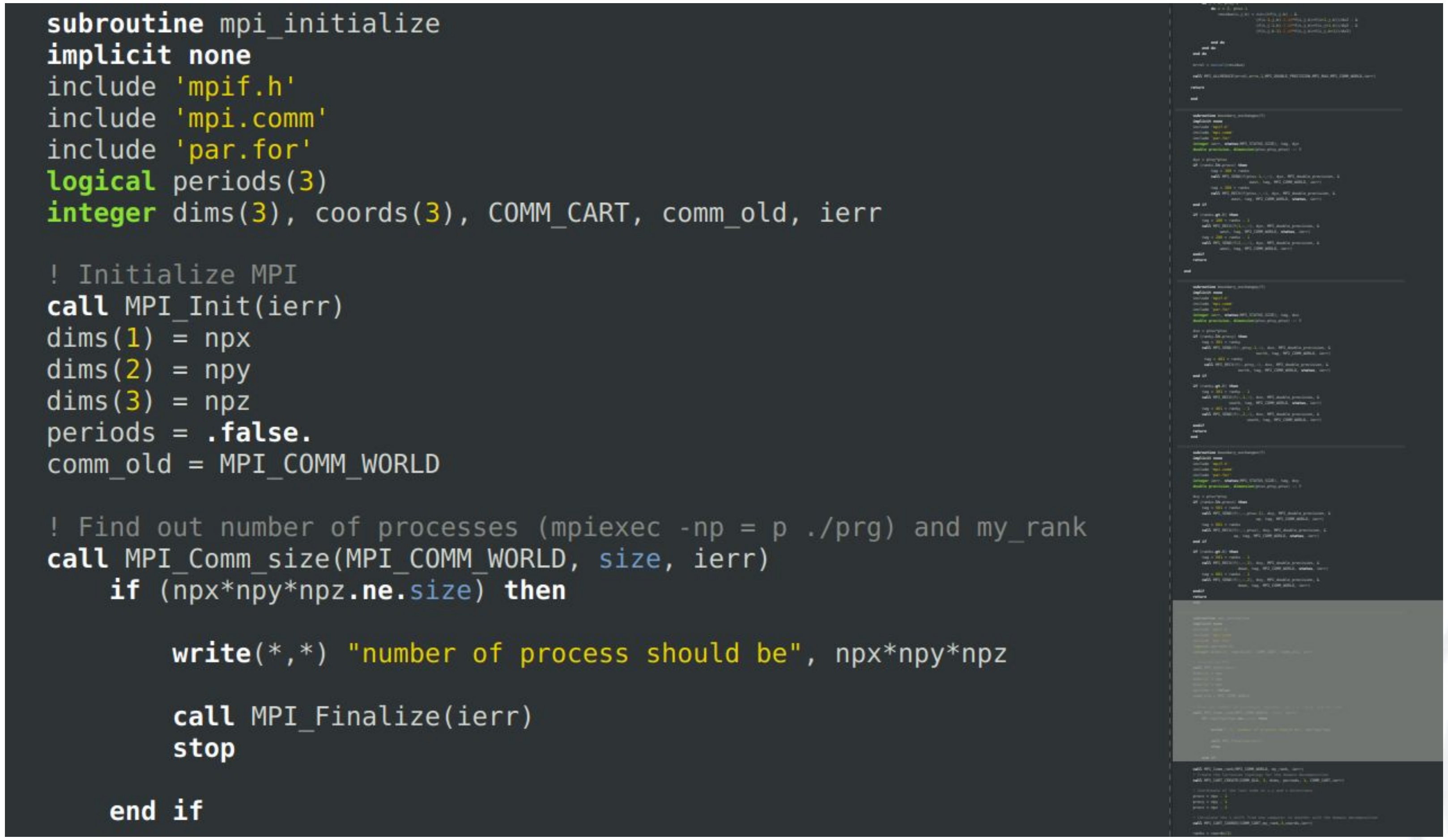




Cartesian topology of processes

How many lines are necessary to its implementation?

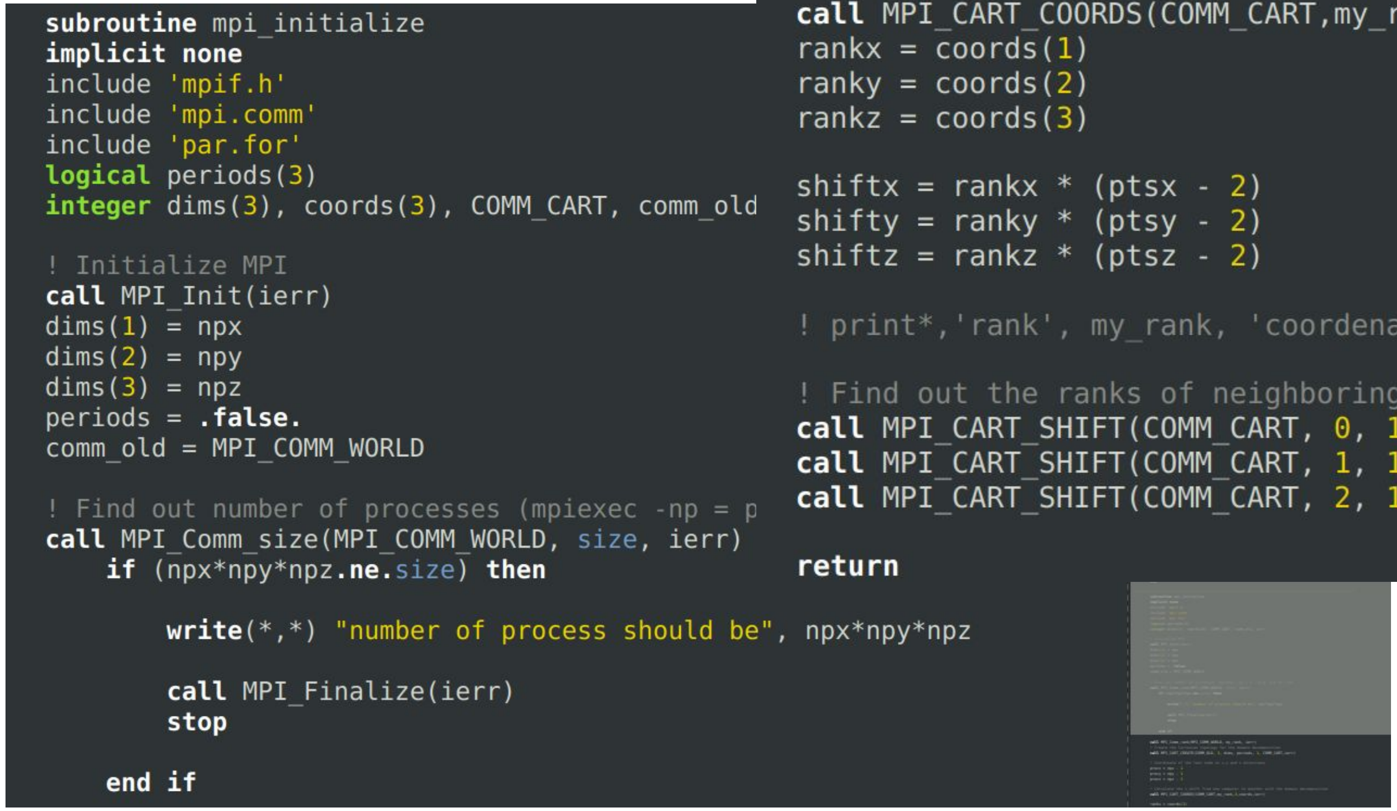
Fortran + MPI



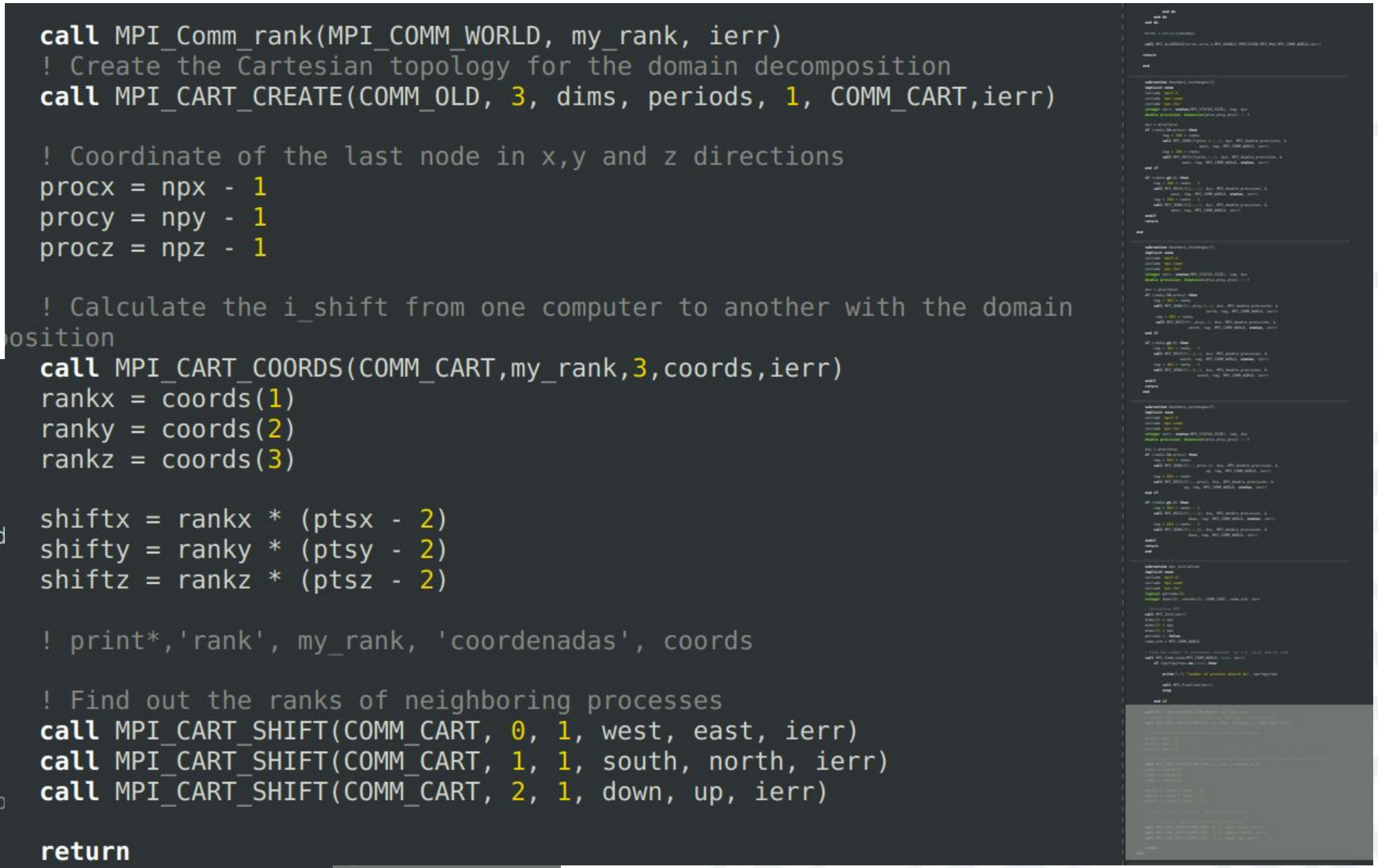
```fortran
subroutine mpi_initialize
implicit none
include 'mpif.h'
include 'mpi.comm'
include 'par.for'
logical periods(3)
integer dims(3), coords(3), COMM_CART, comm_old, ierr

! Initialize MPI
call MPI_Init(ierr)
dims(1) = npx
dims(2) = npy
dims(3) = npz
periods = .false.
comm_old = MPI_COMM_WORLD

! Find out number of processes (mpiexec -np = p ./prg) and my_rank
call MPI_Comm_size(MPI_COMM_WORLD, size, ierr)
    if (npx*npy*npz.ne.size) then

        write(*,*) "number of process should be", npx*npy*npz

        call MPI_Finalize(ierr)
        stop

    end if
```

How many lines are necessary to its implementation?

Fortran + MPI



```fortran
subroutine mpi_initialize
implicit none
include 'mpif.h'
include 'mpi.comm'
include 'par.for'
logical periods(3)
integer dims(3), coords(3), COMM_CART, comm_old

! Initialize MPI
call MPI_Init(ierr)
dims(1) = npx
dims(2) = npy
dims(3) = npz
periods = .false.
comm_old = MPI_COMM_WORLD

! Find out number of processes (mpiexec -np = p
call MPI_Comm_size(MPI_COMM_WORLD, size, ierr)
    if (npx*npy*npz.ne.size) then

        write(*,*) "number of process should be", npx*npy*npz

        call MPI_Finalize(ierr)
        stop

    end if
```

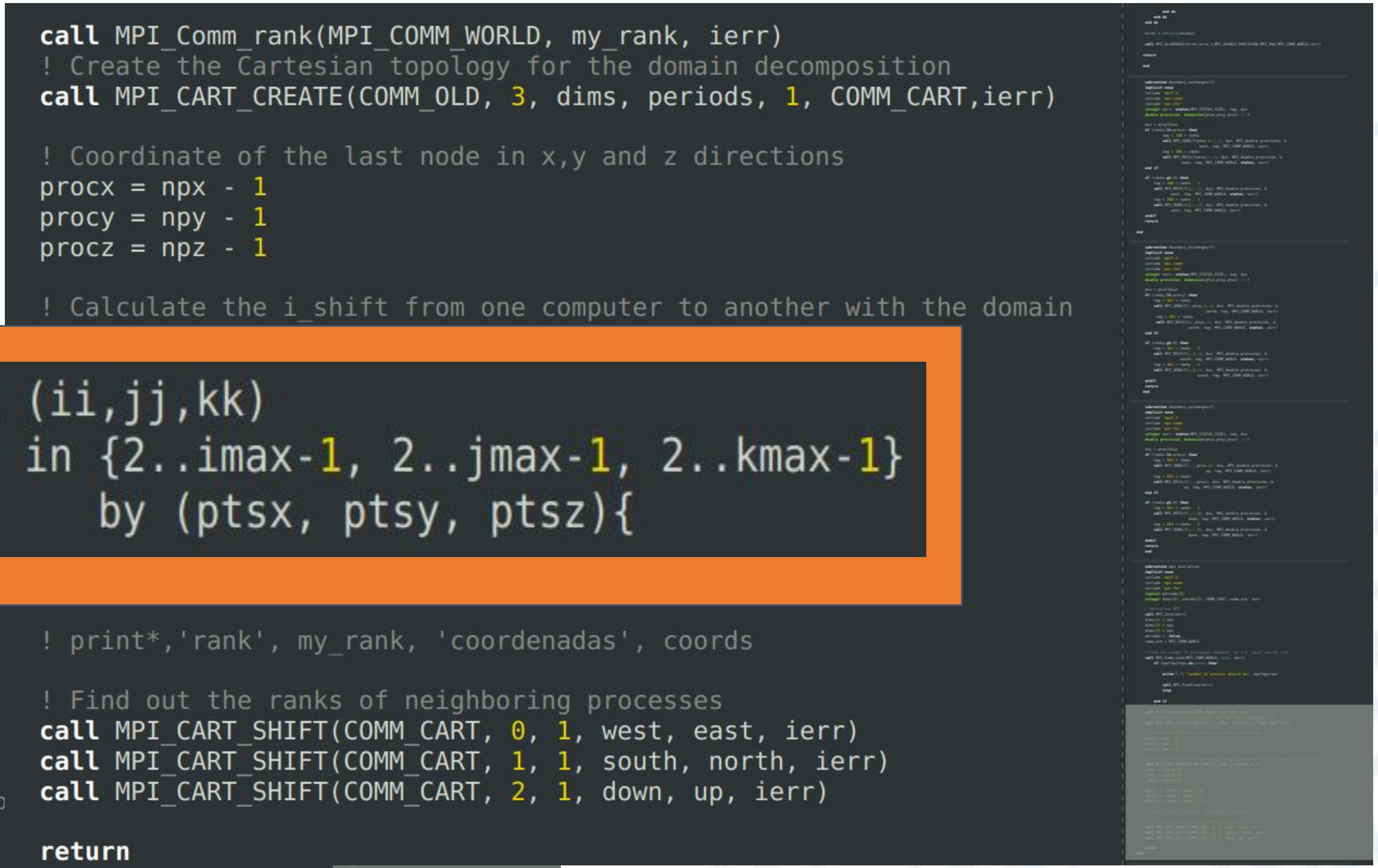


```fortran
call MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
! Create the Cartesian topology for the domain decomposition
call MPI_CART_CREATE(COMM_OLD, 3, dims, periods, 1, COMM_CART,ierr)

! Coordinate of the last node in x,y and z directions
procx = npx - 1
procy = npy - 1
procz = npz - 1

! Calculate the i_shift from one computer to another with the domain
osition
call MPI_CART_COORDS(COMM_CART,my_rank,3,coords,ierr)
rankx = coords(1)
ranky = coords(2)
rankz = coords(3)

shiftx = rankx * (ptsx - 2)
shifty = ranky * (ptsy - 2)
shiftz = rankz * (ptsz - 2)

! print*,'rank', my_rank, 'coordenadas', coords

! Find out the ranks of neighboring processes
call MPI_CART_SHIFT(COMM_CART, 0, 1, west, east, ierr)
call MPI_CART_SHIFT(COMM_CART, 1, 1, south, north, ierr)
call MPI_CART_SHIFT(COMM_CART, 2, 1, down, up, ierr)

return
```

How many lines are necessary to its implementation?

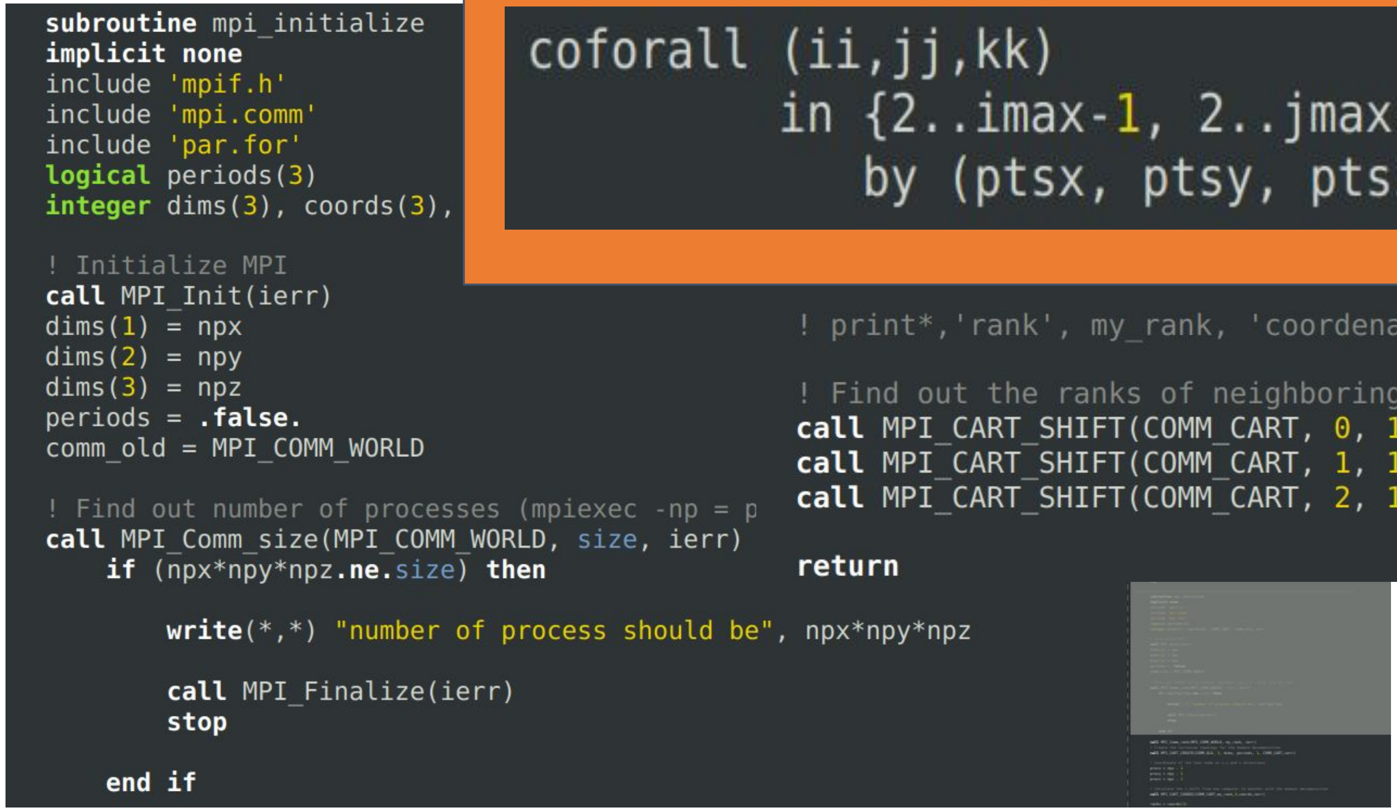
Fortran + MPI

CHAPEL

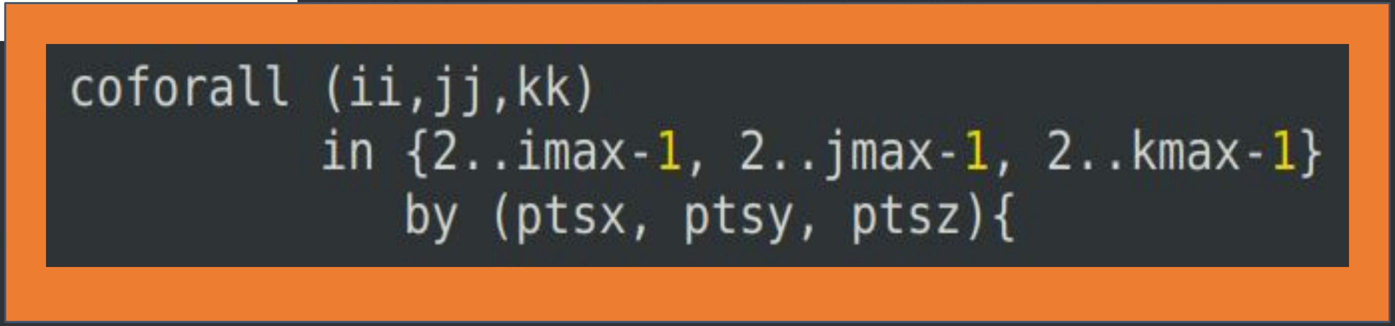


```
call MPI_Comm_rank(MPI_COMM_WORLD, my_rank, ierr)
! Create the Cartesian topology for the domain decomposition
call MPI_CART_CREATE(COMM_OLD, 3, dims, periods, 1, COMM_CART,ierr)

! Coordinate of the last node in x,y and z directions
procx = npx - 1
procy = npy - 1
procz = npz - 1

! Calculate the i_shift from one computer to another with the domain
```

```
! print*,'rank', my_rank, 'coordenadas', coords

! Find out the ranks of neighboring processes
call MPI_CART_SHIFT(COMM_CART, 0, 1, west, east, ierr)
call MPI_CART_SHIFT(COMM_CART, 1, 1, south, north, ierr)
call MPI_CART_SHIFT(COMM_CART, 2, 1, down, up, ierr)

return
```



```
subroutine mpi_initialize
implicit none
include 'mpif.h'
include 'mpi.comm'
include 'par.for'
logical periods(3)
integer dims(3), coords(3),

! Initialize MPI
call MPI_Init(ierr)
dims(1) = npx
dims(2) = npy
dims(3) = npz
periods = .false.
comm_old = MPI_COMM_WORLD

! Find out number of processes (mpiexec -np = p
call MPI_Comm_size(MPI_COMM_WORLD, size, ierr)
    if (npx*npy*npz.ne.size) then

        write(*,*) "number of process should be", npx*npy*npz

        call MPI_Finalize(ierr)
        stop

    end if
```



```
coforall (ii,jj,kk)
        in {2..imax-1, 2..jmax-1, 2..kmax-1}
           by (ptsx, ptsy, ptsz){
```

How many lines are necessary to its implementation?

Chapel

```
while (erro  >= tol) do{

    sor_method(f, tf);
    calc_erro(residuo, f, tf);

    erro = max reduce(residuo);

    itera += 1;

}
```

Fortran + MPI

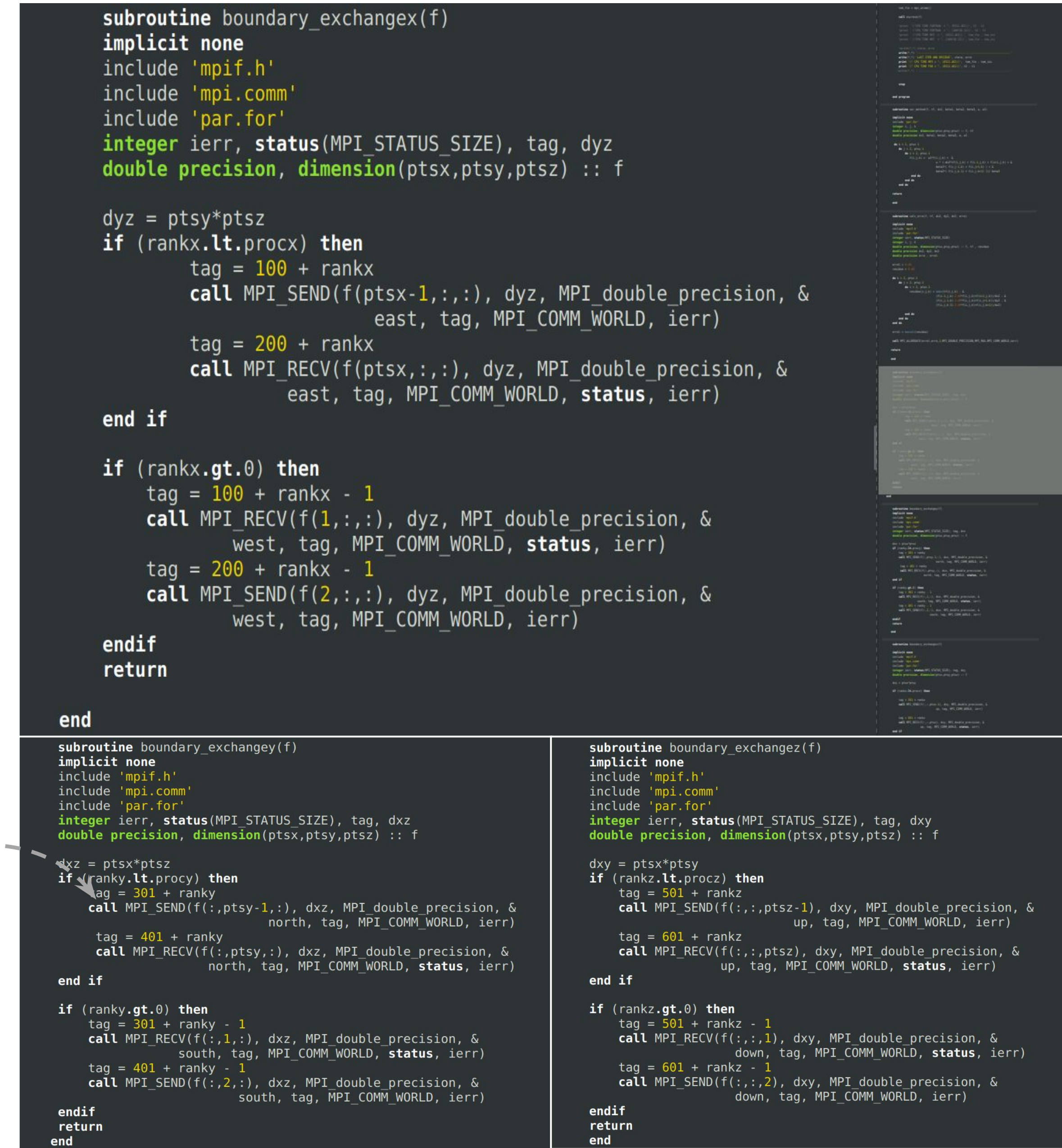
```
do while (erro.gt.tol)

    call sor_method(f, tf, dx2, beta1, beta2, beta3, w, w1)

    call boundary_exchangex(f)
    call boundary_exchangey(f)
    call boundary_exchangez(f)

    call calc_erro(f, tf, dx2, dy2, dz2, erro)

    itera = itera + 1


end do
```

How many lines are necessary to its implementation?

Chapel

```
coforall (ii,jj,kk)
         in {2..imax-1, 2..jmax-1, 2..kmax-1}
            by (ptsx, ptsy, ptsz){

 for i in ii..ii + ptsx-1 do{
        for j in jj..jj + ptsy-1 do{
            for k in  kk..kk + ptsz-1  do{
```

Fortran + MPI



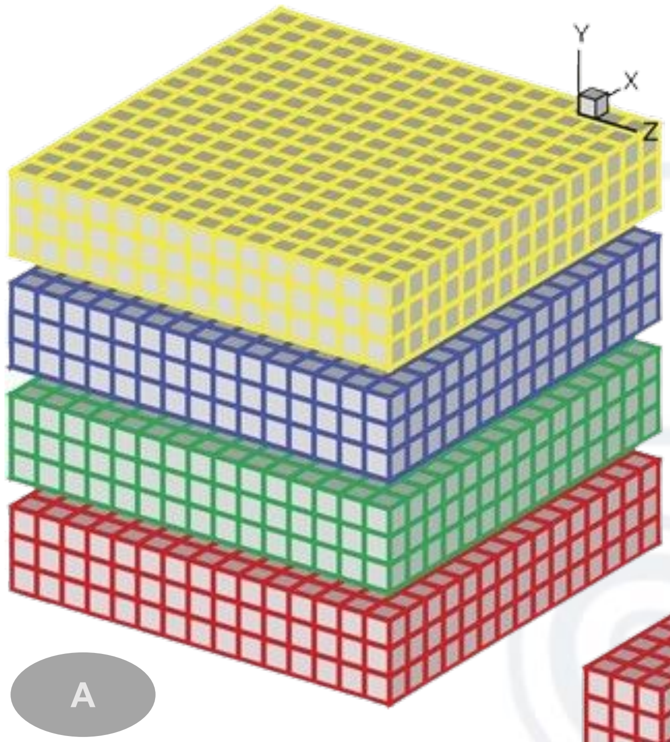
```
      subroutine boundary_exchangex(f)
      implicit none
      include 'mpif.h'
      include 'mpi.comm'
      include 'par.for'
      integer ierr, status(MPI_STATUS_SIZE), tag, dyz
      double precision, dimension(ptsx,ptsy,ptsz) :: f

      dyz = ptsy*ptsz
      if (rankx.lt.procx) then
              tag = 100 + rankx
              call MPI_SEND(f(ptsx-1,:,:), dyz, MPI_double_precision, &
                                   east, tag, MPI_COMM_WORLD, ierr)
              tag = 200 + rankx
              call MPI_RECV(f(ptsx,:,:), dyz, MPI_double_precision, &
                               east, tag, MPI_COMM_WORLD, status, ierr)
      end if

      if (rankx.gt.0) then
          tag = 100 + rankx - 1
          call MPI_RECV(f(1,:,:), dyz, MPI_double_precision, &
                  west, tag, MPI_COMM_WORLD, status, ierr)
          tag = 200 + rankx - 1
          call MPI_SEND(f(2,:,:), dyz, MPI_double_precision, &
                  west, tag, MPI_COMM_WORLD, ierr)
      endif
      return

    end
```

```
    subroutine boundary_exchangey(f)
    implicit none
    include 'mpif.h'
    include 'mpi.comm'
    include 'par.for'
    integer ierr, status(MPI_STATUS_SIZE), tag, dxz
    double precision, dimension(ptsx,ptsy,ptsz) :: f

    dxz = ptsx*ptsz
    if (ranky.lt.procy) then
        tag = 301 + ranky
        call MPI_SEND(f(:,ptsy-1,:), dxz, MPI_double_precision, &
                                north, tag, MPI_COMM_WORLD, ierr)
         tag = 401 + ranky
         call MPI_RECV(f(:,ptsy,:), dxz, MPI_double_precision, &
                               north, tag, MPI_COMM_WORLD, status, ierr)
    end if

    if (ranky.gt.0) then
        tag = 301 + ranky - 1
        call MPI_RECV(f(:,1,:), dxz, MPI_double_precision, &
                          south, tag, MPI_COMM_WORLD, status, ierr)
        tag = 401 + ranky - 1
        call MPI_SEND(f(:,2,:), dxz, MPI_double_precision, &
                                south, tag, MPI_COMM_WORLD, ierr)
    endif
    return
   end
```

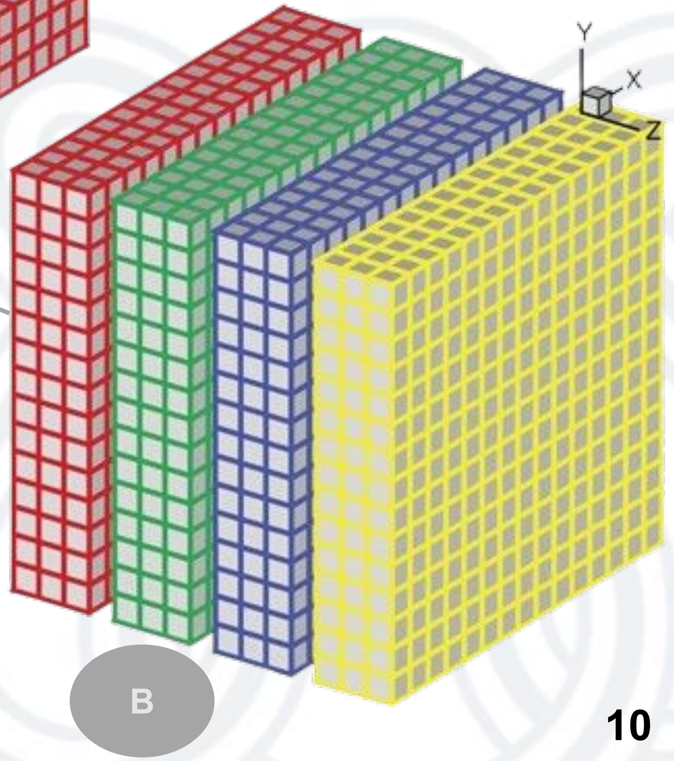
```
    subroutine boundary_exchangez(f)
    implicit none
    include 'mpif.h'
    include 'mpi.comm'
    include 'par.for'
    integer ierr, status(MPI_STATUS_SIZE), tag, dxy
    double precision, dimension(ptsx,ptsy,ptsz) :: f

    dxy = ptsx*ptsy
    if (rankz.lt.procz) then
        tag = 501 + rankz
        call MPI_SEND(f(:,:,ptsz-1), dxy, MPI_double_precision, &
                              up, tag, MPI_COMM_WORLD, ierr)
        tag = 601 + rankz
        call MPI_RECV(f(:,:,ptsz), dxy, MPI_double_precision, &
                    up, tag, MPI_COMM_WORLD, status, ierr)
    end if

    if (rankz.gt.0) then
        tag = 501 + rankz - 1
        call MPI_RECV(f(:,:,1), dxy, MPI_double_precision, &
                    down, tag, MPI_COMM_WORLD, status, ierr)
        tag = 601 + rankz - 1
        call MPI_SEND(f(:,:,2), dxy, MPI_double_precision, &
                    down, tag, MPI_COMM_WORLD, ierr)
    endif
    return
    end
```

Domain Decomposition

	Chapel			Fortran+MPI		
Threads	npx	npy	npz	npx	npy	npz
2	2	1	1	1	1	2
4	**1**	**4**	**1**	**1**	**1**	**4**
8	8	1	1	1	1	8
16	16	1	1	2	2	4
32	8	4	1	2	4	4



Chapel row-major
npx=1 npy=4 npz=1



Fortran + MPI column-major
npx=1 npy=1 npz=4

Tools/libraries

C compiler gcc 12.2.0
Chapel 1.30.0 latest version
Chapel optimization flag --fast

C compiler gcc 4.9.2
mpich 3.1.4
Fortran optimization flag -O3

Variable	Value
CHPL_TASKS	qthreads
CHPL_TARGET_CPU	native
CHPL_HOST_PLATFORM	linux64
CHPL_LLVM	none
CHPL_COMM	none

Machine and Operating System

- Machine
  - Cluster Euler
- Memory
  - 128 GB DDR3 1866MHz
- Processor Intel Xeon E5-2680v4 de 2.4 GHz
  - 1 locale - 28 cores, 56 parallel processes available.
- Operating System
  - CentOS Linux release 7.2.1511

CePID - Centro de Ciências Matemáticas Aplicadas à Indústria

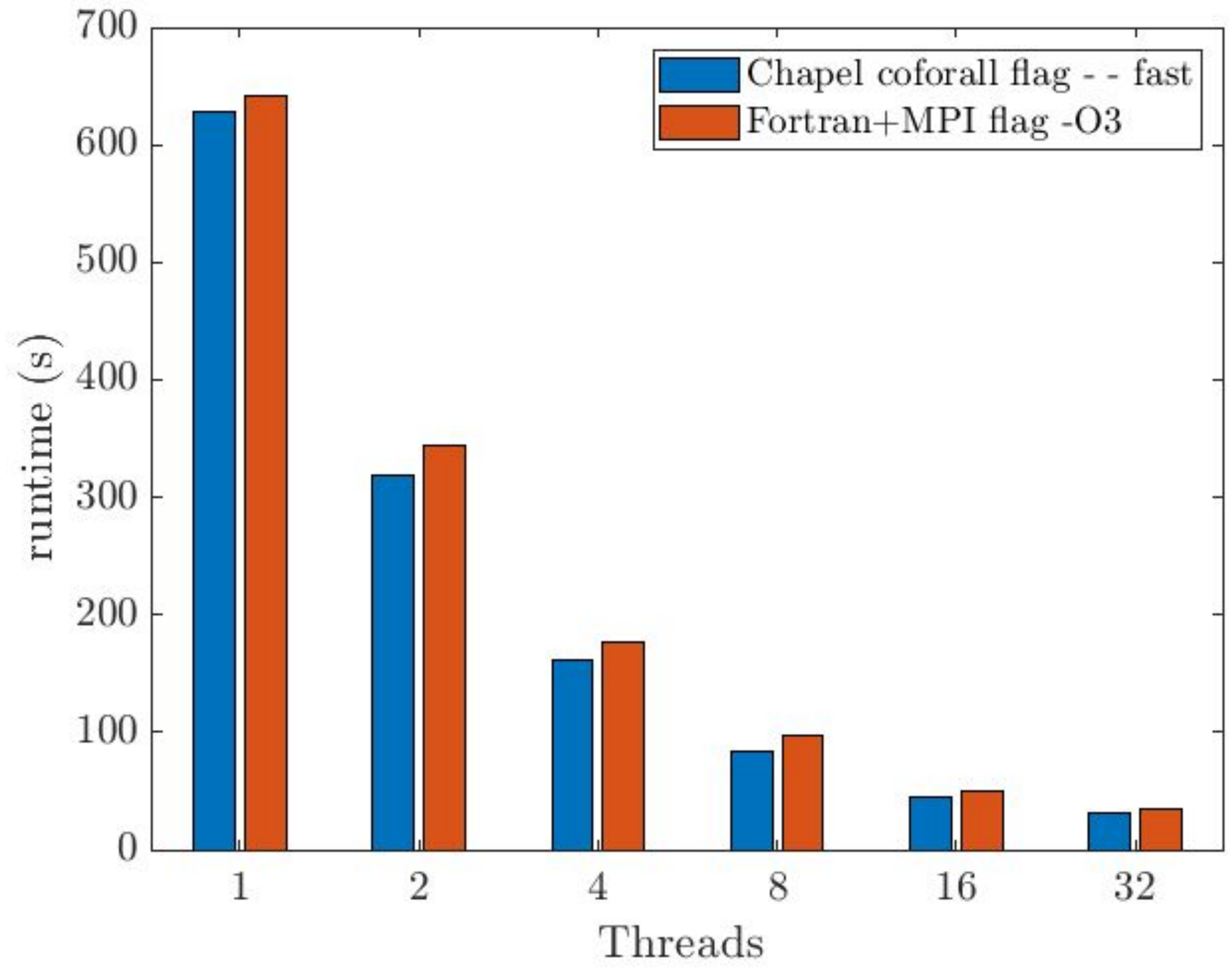
FUNDAÇÃO DE AMPARO À PESQUISA DO ESTADO DE SÃO PAULO

Cluster Euler

https://euler.cemeai.icmc.usp.br/

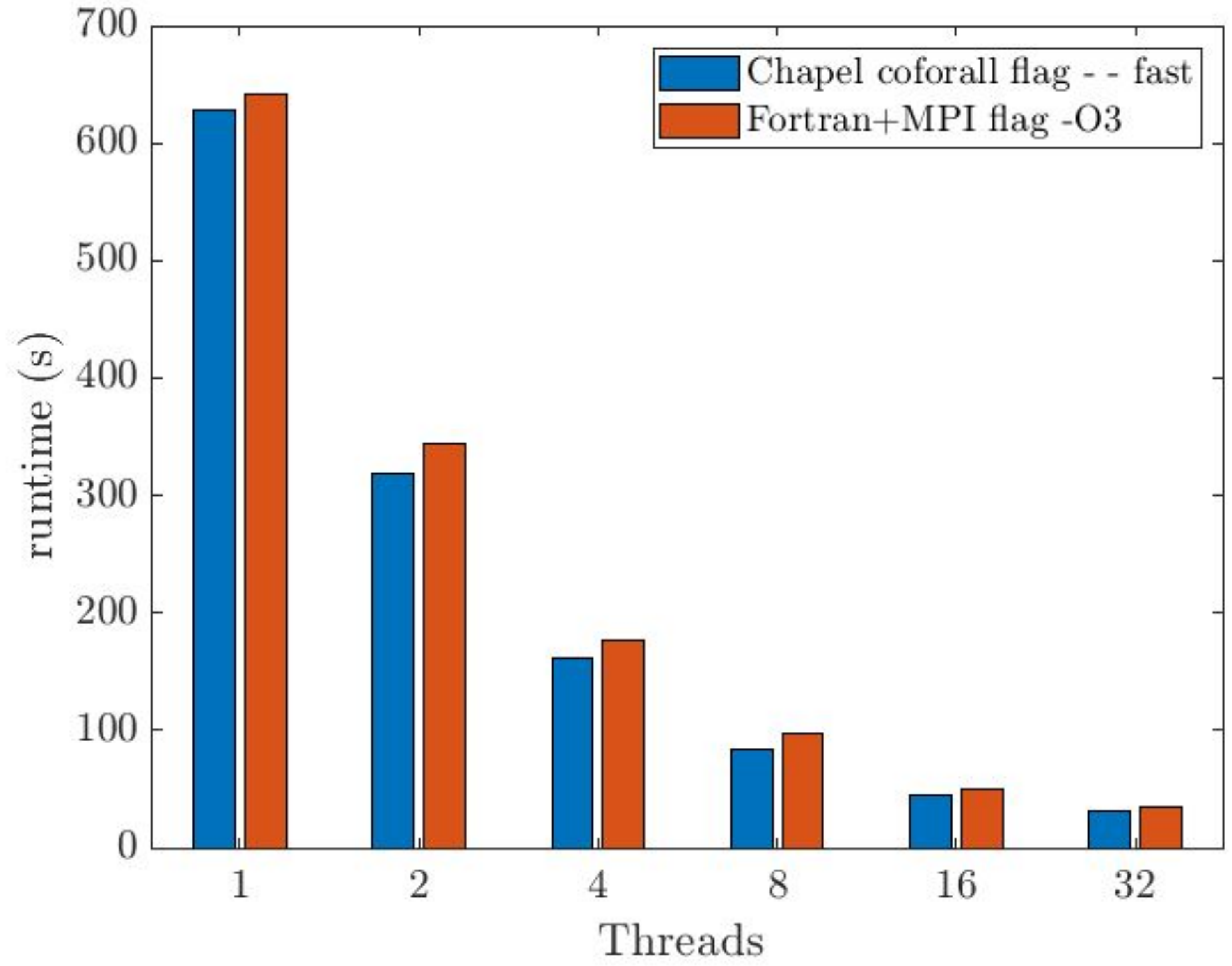
Results

Serial	2.2 %
2	7.6%
4	9.1%
8	15.9%
16	9.8%
32	9.2%



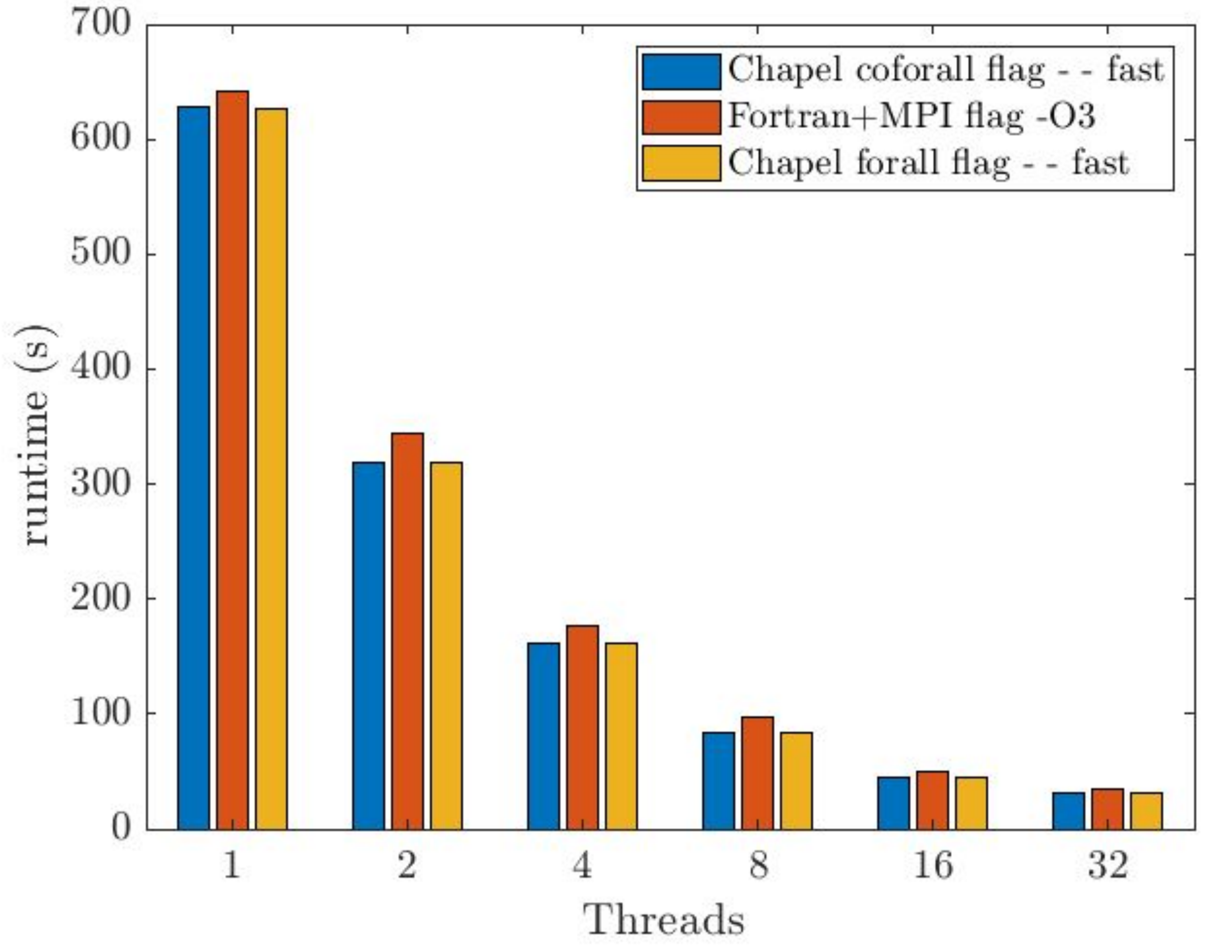
Results

Serial	2.2 %
2	7.6%
4	9.1%
8	**15.9%**
16	9.8%
32	9.2%



Results

```
const Inner = {2..imax-1, 2..jmax-1,  2..kmax-1};

forall (i,j,k) in Inner  do{
```



Conclusions

Chapel provides…

- an easy language for writing algorithms
- significantly reduces the number of lines implemented
- as fast as Fortran+MPI

and next stages…

- Work with Distributions: BlockDist, CyclicDist, BlockCycDist;
- Building Multilocale Chapel
- Compare with the latest version of Fortran+MPI

Acknowledgments

Thanks for your attention!

I thank Livia Freire and Nelson Dias, my advisors; Leonardo Martinussi, technical support; Engin Kayralki, Damian McGuckin, Jeremiah Corrado, Vass Litvinov and the all Chapel community. This study was funded by the Coordination for the Improvement of Higher Education Personnel (CAPES grant PROEX 88887.671252/2022-00) and the São Paulo Research Foundation (FAPESP grant Nº. 2018/24284-1).

Anna Caroline F. S. de Jesus
annacfelixs@gmail.com
Laboratory of Applied Mathematics and Scientific Computing (LMACC)
University of São Paulo (USP)