Introduction to High-Performance Computing

Giorgio Amati

Corso di dottorato in Ingegneria Aeronautica e Spaziale 2024

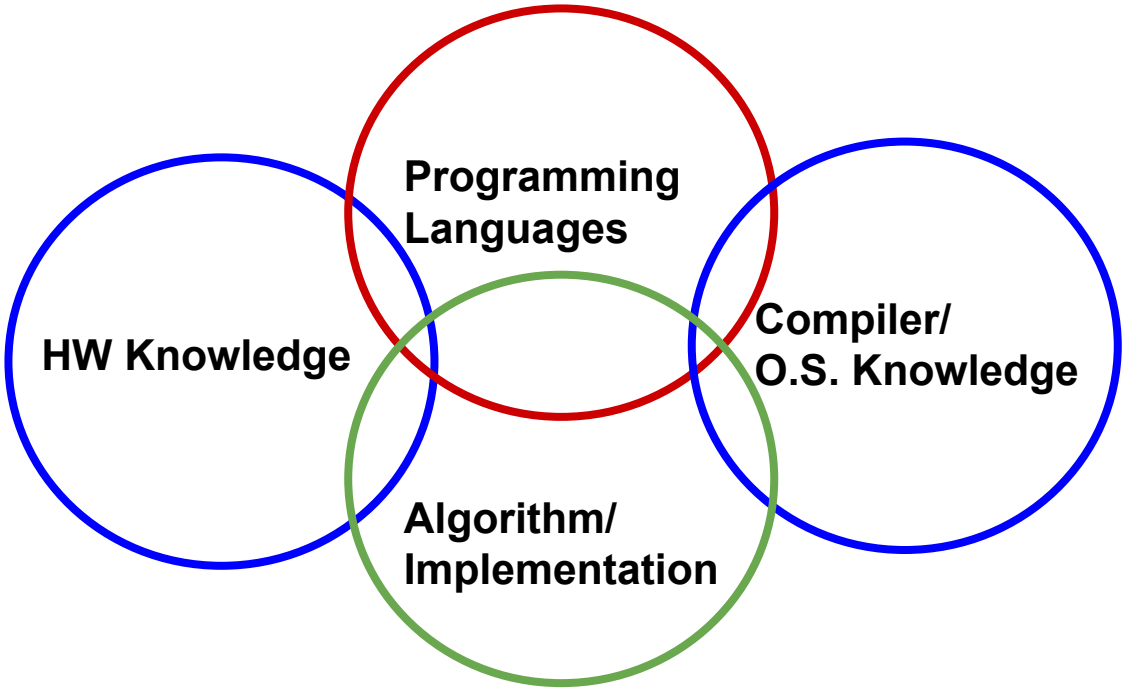
g.amati@cineca.it / g.amaticode@gmail.com

Agenda

- ✓ **Lattice Boltzmann Method**
  - **Single Node performance**
  - **Single GPU**
- ✓ **Synthetic MPI performance**
  - **Multinode performance**

HPC: what it is?

✓ These are the main skills for an efficient HPC



What I'd like to show/CAVEAT

- ✓ How different approaches impact on performances for a "real world code"
- ✓ Just to show "qualitative figures" of what could be your job….
- ✓ Probably you don't have to face (compex) HPC issues in your career, but it's always better to "know your enemy".
- ✓ For sure, sooner or later, some HPC issues will be present!

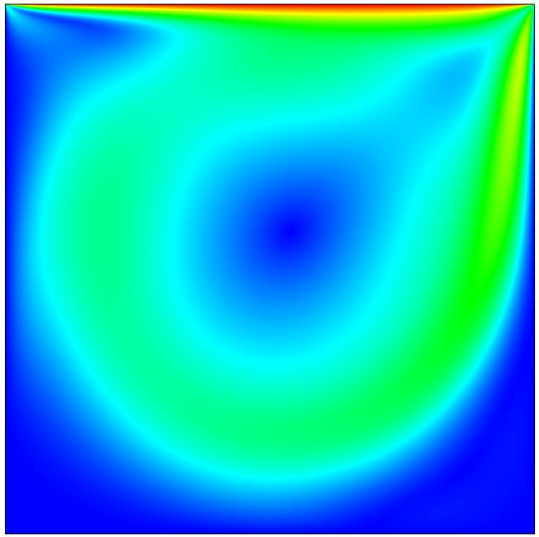
Test #1

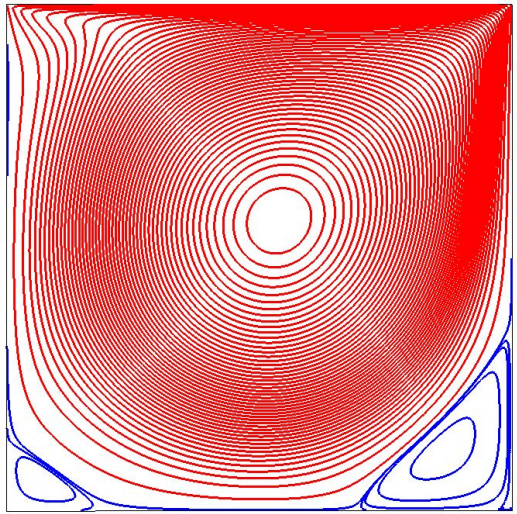
- ✓ CFD code
- ✓ Kinetic code (Lattice Boltzmann Method)
- ✓ 2D “shrink” of a 3D multi-gpu production code
- ✓ available at https://github.com/gamati01/BGK2D_GPU (under MIT license, still working in progress)

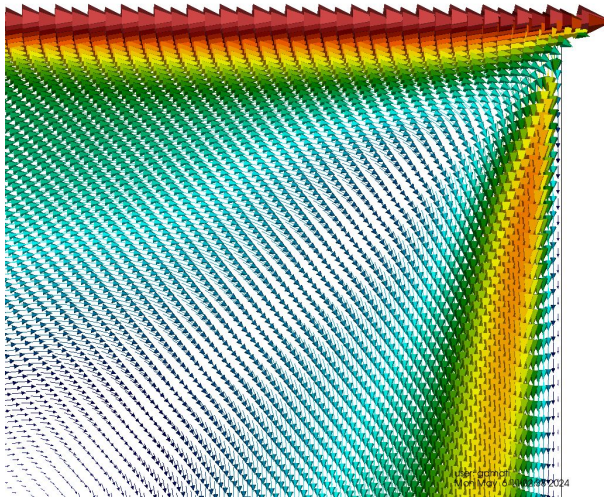
- ✓ References:
  a. https://www.nature.com/articles/s41586-021-03658-1
  b. https://www.sciencedirect.com/science/article/abs/pii/S004579301830851X
  c. https://www.sciencedirect.com/science/article/abs/pii/S187775032100128

The code

- ✓ Written in Fortran90
- ✓ Different physical test case
  - Driven cavity
  - Flow around a Cylinder
  - Taylor Green Vortices

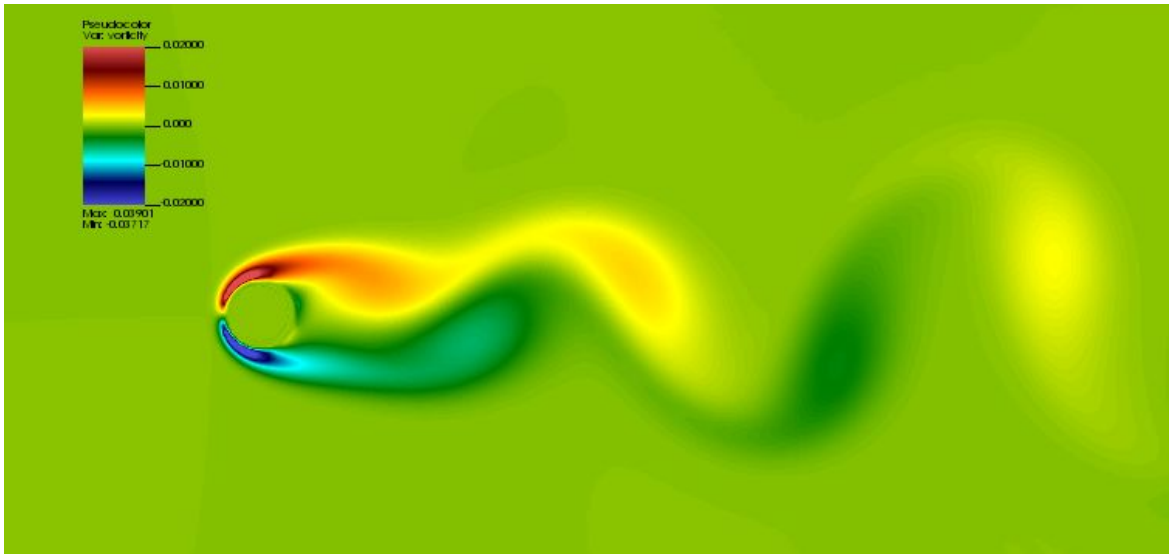


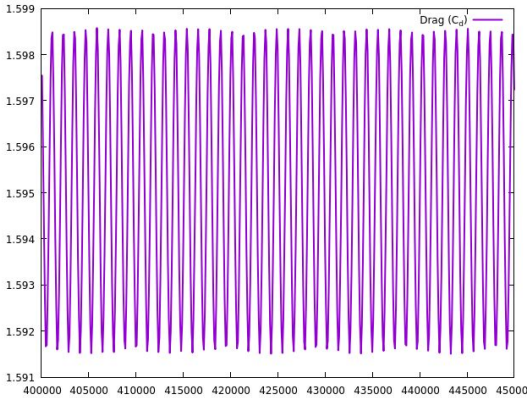


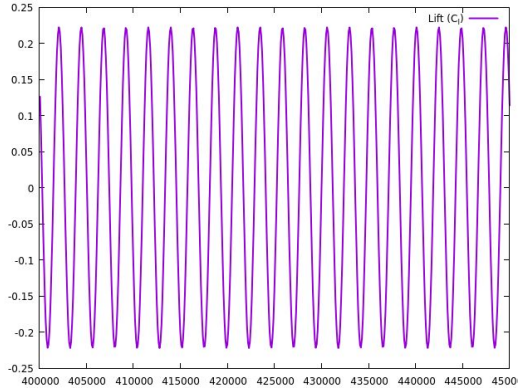


The code/2

- ✓ Written in Fortran90
- ✓ Different physical test case
  - Driven cavity
  - **Flow around a Cylinder**
  - Taylor Green Vortices







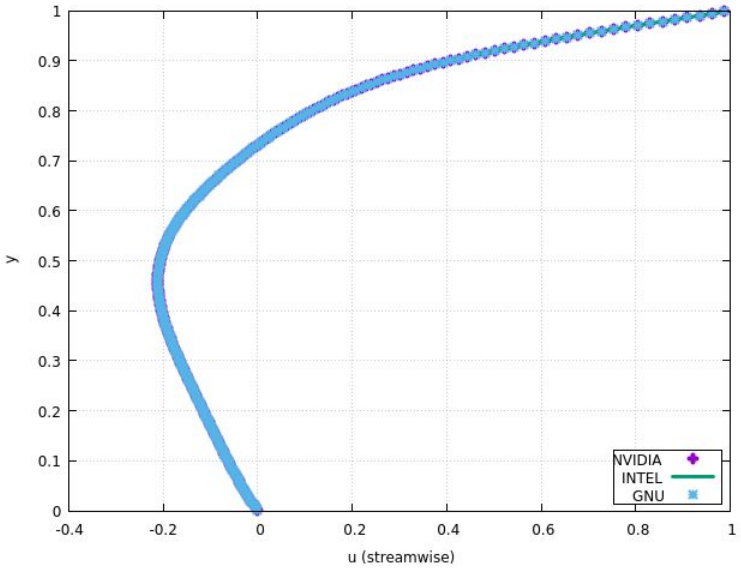
GPU vs. CPU

- ✓ Which are the performances for different compilers?
- ✓ Which are the performance between CPU vs. GPU
- ✓ Different parallel models
  - OpenACC
  - OpenMP Offload
  - DO concurrent

CPU: serial performance-1

- ✓ Intel Intel(R) Xeon(R) Platinum 8480+ (Sapphire Rapids)
- ✓ Driven Cavity, Re=100, Double precision
- ✓ Compilers
  - INTEL: Rel. 2023.2.1
  - NVIDIA: Rel. 23.11
  - GNU: Rel. 12.2.0

Compiler	Double
INTEL	96
NVIDIA	96
GNU	73



CPU: serial performance-2

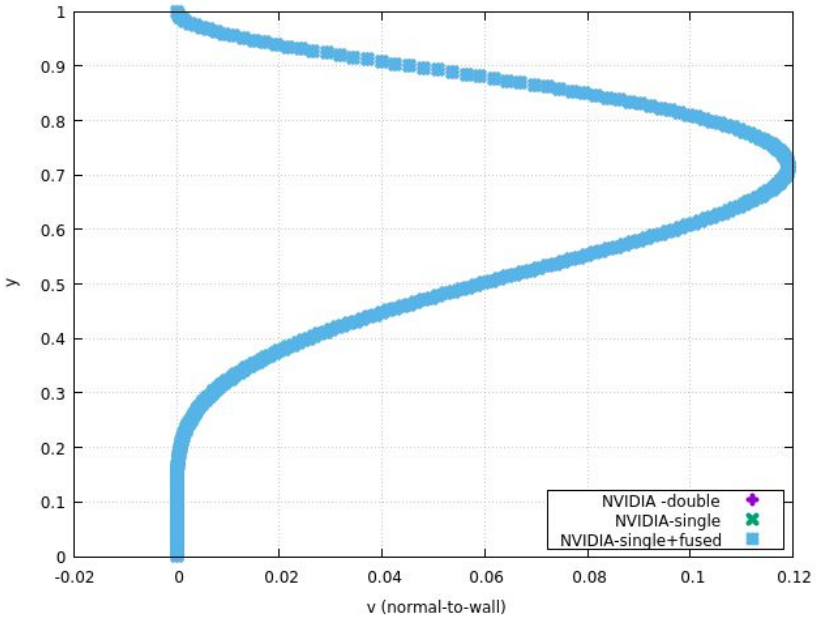
- ✓ Intel Intel(R) Xeon(R) Platinum 8480+ (Sapphire Rapids)
- ✓ Driven Cavity, Re=100, Single precision
- ✓ Compilers
  - INTEL: Rel. 2023.2.1
  - NVIDIA: Rel. 23.11
  - GNU: Rel. 12.2.0

Compiler	Double	Single
INTEL	96	180
NVIDIA	96	171
GNU	73	139

CPU: serial performance-3

- ✓ Intel Intel(R) Xeon(R) Platinum 8480+ (Sapphire Rapids)
- ✓ Driven Cavity, Re=100, Single precision, fused implementation
- ✓ DO Concurrent implementation
- ✓ Compilers
  - INTEL: Rel. 2023.2.1
  - NVIDIA: Rel. 23.11
  - GNU: Rel. 12.2.0

Compiler	Double	Single	Fused
INTEL	96	180	296
NVIDIA	96	171	334
GNU	73	139	104



CPU: node performance

- ✓ Intel Intel(R) Xeon(R) Platinum 8480+ (Sapphire Rapids)
- ✓ Driven Cavity, Re=1000, Single precision, fused implementation
- ✓ DO concurrent
- ✓ Compilers
  - INTEL: Rel. 2023.2.1
  - NVIDIA: Rel. 23.11

Compiler	32 threads	64 threads	112 threads
INTEL	973	1675	1217
NVIDIA	4255	3541	2393

GPU implementation

```
#ifdef OFFLOAD
!$OMP target teams distribute parallel do simd collapse(2)
    do j = 1,m
       do i = 1,l
#elif OPENACC
 !$acc parallel
 !$acc loop independent collapse(2)
    do j = 1,m
       do i = 1,l
#else
    do concurrent (j=1:m, i=1:l)
#endif
         … do something …
```

GPU implementation

```
        … do something …
        enddo
#ifdef OFFLOAD
     end do
!$OMP end target teams distribute parallel do simd
#elif OPENACC
     end do
     !$acc end parallel
#endif
```

single GPU performance

- ✓ Nvidia A100@64GB
- ✓ Driven Cavity, Single precision, fused implementation
  - Re=1’000 (512^2)
  - Re=10’000 (1024^2)
- ✓ Parallel paradigm
  - DO concurrent
  - OpenACC
  - OpenMP Offload
- ✓ Compiler: NVIDIA, Rel. 23.11

Paradigm	R=1000	R=10000
DO concurrent	5034	9869
Openacc	5034	10147
Offload	2534	7355
CPU (max)	4255	-

Some comments

- ✓ Different compilers behave in a different way, difference can be quite big (e.g. **`gfortran`**)
- ✓ Take care on parallel performance (e.g. **`ifx`** on multicore), could be even a problem of “maturity”
- ✓ Always verify if results are correct and “compatible” with the assumption (e.g. single precision vs. double precision)
- ✓ The most “flexible” approach could not be the most efficient (i.e. openMP offload)

Agenda

- ✓ Lattice Boltzmann Method
  - Single Node performance
  - Single GPU
- ✓ **Synthetic MPI performance**
  - **Multinode performance**

Test #2

- ✓ Synthetic code
- ✓ Multi-GPU mpi based code
  a. Working with CPU or GPU
- ✓ available at https://github.com/gamati01/Check_MPI (under MIT license, still working in progress)

- ✓ Written in Fortran90
- ✓ Synthetic code to explore MPI communication patterns
  a. Tri-periodic boundary condition
  b. domain decomposition in three directions
  c. rigid shifting of three fields x,y,z direction (do_something_GPU)

The code/1

Available 11 different patterns

0. Step0: 1 call to sendrecv for single field, with mpi_datatype
1. Step1: 1 call to sendrecv for single field, with explicit data packing/unpacking
2. Step2: 1 call to sendrecv for 3 fields, with explicit data packing/unpacking
3. Step3: 1 call to sendrecv for 3 fields, with explicit data packing/unpacking (for GPU)
4. Step4: 1 call to sendrecv for 3 fields (cudaware)
5. Step5: skip….
6. Step6: skip ….
7. Step7: skip (call isend/recv)
8. Step8: call to isend/recv + overlap
9. Step9: kernel async
10. Step10: masked loop

The code/2

Available 11 different patterns

Version	production	CPU	GPU
Step0	yes	yes	no
Step1	yes	yes	no
Step2	yes	yes	no
Step3	yes	yes	yes
Step4	yes	yes	yes
Step5	no	yes	yes
Step6	no	yes	yes
Step7	no	yes	yes
Step8	yes	yes	yes
Step9	yes	yes	yes
Step10	yes	yes	yes

Step0 (default)

- ✓ One call to **`sendrecv,`** one for each field to “propagate”
  - 3x3x2 calls every timestep
- ✓ using mpi-datatype

```
tag = 04
call mpi_sendrecv(field1(0,0,n), 1, xyplane, up(2), tag,    &
                  field1(0,0,0), 1, xyplane, down(2), tag,  &
                  lbecomm, status,ierr)
tag = 06
call mpi_sendrecv(field2(0,0,n), 1, xyplane, up(2), tag,    &
                  field2(0,0,0), 1, xyplane, down(2), tag,  &
                  lbecomm, status,ierr)
tag = 07
call mpi_sendrecv(field3(0,0,n), 1, xyplane, up(2), tag,    &
                  field3(0,0,0), 1, xyplane, down(2), tag,  &
                  lbecomm, status,ierr)
```

Step0 (Default)

- ✓ 1024^3, 64 tasks
- ✓ 1’000 timestep
- ✓ Intel Intel(R) Xeon(R) Platinum 8480+ (Sapphire Rapids)

```
# Time for section
# loop    time   0.750499E+02    0.750508E+02
# comp.   time   0.560540E+02    0.559492E+02
# diag.   time   0.000000E+00    0.000000E+00
# MPI     time   0.189785E+02    0.190547E+02
#--------------------------------
# Ratio
# Ratio Coll   0.747    0.745
# Ratio Dg.    0.000    0.000
# Ratio MPI    0.253    0.254
# Check        0.000    0.001
#--------------------------------
# Z-MPI time, BW (MB/s) -->  0.828125        306.6222
# Y-MPI time, BW (MB/s) -->  2.097656        121.0501
# X-MPI time, BW (MB/s) -->  15.84766        16.02265
```

Step1

- ✓ One call to **`sendrecv,`** one for each field to “propagate”
  - 3x3x2 calls every timestep
- ✓ no mpi-datatype, explicit packing/unpacking

```
        do k = 0,n+1
             do j = 0,m+1
                bufferXIN(j,k)=field1(1,j,k)
             enddo
        enddo
!
        call mpi_sendrecv(bufferXIN(0,0),msgsizeX,MYMPIREAL,front(2),tag,&
                          bufferXOUT(0,0),msgsizeX,MYMPIREAL,rear(2),tag,&
                          lbecomm,status,ierr)
!
        do k = 0,n+1
             do j = 0,m+1
                field1(0,j,k) = bufferXOUT(j,k)
             enddo
        enddo
```

Step1

- ✓ 1024^3, 64 tasks
- ✓ 1’000 timestep
- ✓ Intel Intel(R) Xeon(R) Platinum 8480+ (Sapphire Rapids)

```
# Time for section
# loop    time   0.727114E+02    0.727109E+02
# comp.   time   0.558754E+02    0.558281E+02
# diag.   time   0.000000E+00    0.000000E+00
# MPI     time   0.168161E+02    0.167930E+02
#--------------------------------
# Ratio
# Ratio Coll   0.768    0.768
# Ratio Dg.    0.000    0.000
# Ratio MPI    0.231    0.231
# Check        0.000    0.001
#--------------------------------
# Z-MPI time, BW (MB/s) -->  0.648437        391.5898
# Y-MPI time, BW (MB/s) -->  2.125000        119.4925
# X-MPI time, BW (MB/s) -->  13.79297        18.40949
```

Step2

- ✓ One call to **`sendrecv,`** one for all three field to “propagate”
  - 3x1x2 calls every time step
- ✓ no mpi-datatype, explicit packing/unpacking

```
      msgsizeY = (l+2)*(n+2)*3
!
      do k = 0,n+1
          do i = 0,l+1
              bufferYIN(i,k,1)=field1(i,m,k)
              bufferYIN(i,k,2)=field2(i,m,k)
              bufferYIN(i,k,3)=field3(i,m,k)
          enddo
      enddo
!
      call mpi_sendrecv(bufferYIN(0,0,1),msgsizeY,MYMPIREAL,right(2),tag,&
                        bufferYOUT(0,0,1),msgsizeY,MYMPIREAL,left(2),tag,&
                        lbecomm,status,ierr)
!
```

Performance figure (CPU)

- ✓ 1024^3,
- ✓ 64 tasks
- ✓ 1’000 timestep
- ✓ Intel Sapphire Rapids

Version	HW	Time	MPI	Comp
Step0	CPU	75”	25%	74%
Step1	CPU	72”	23%	77%
Step2	CPU	72”	22%	78%

Step3

- ✓ One call to **`sendrecv,`** one for all three field to “propagate”
  - 3x1x2 calls every time step
- ✓ no mpi-datatype, explicit packing/unpacking, ported to GPU

```
!$acc kernels
     do k = 0,n+1
          do i = 0,l+1
               bufferYIN(i,k,1)=field1(i,m,k)
               bufferYIN(i,k,2)=field2(i,m,k)
               bufferYIN(i,k,3)=field3(i,m,k)
          enddo
     enddo
!$acc end kernels
!
     call mpi_sendrecv(bufferYIN(0,0,1),msgsizeY,MYMPIREAL,right(2),tag,&
                       bufferYOUT(0,0,1),msgsizeY,MYMPIREAL,left(2),tag,&
                       lbecomm,status,ierr)
!
_
```

Step3

- ✓ 1024^3, 8 tasks
- ✓ 10’000 timestep
- ✓ Nvidia A100@64GB (Leonardo)

```
# Time for section
# loop    time  0.119129E+03   0.119129E+03
# comp.   time  0.479769E+02   0.480273E+02
# diag.   time  0.000000E+00   0.000000E+00
# MPI     time  0.654978E+02   0.654102E+02
#--------------------------------
# Ratio
# Ratio Coll  0.403   0.403
# Ratio Dg.   0.000   0.000
# Ratio MPI   0.550   0.549
# Check       0.047   0.048
#--------------------------------
# Z-MPI time, BW (MB/s) -->  19.58984       514.4644
# Y-MPI time, BW (MB/s) -->  19.83594       508.0817
# X-MPI time, BW (MB/s) -->  20.90625       482.0701
```

Step4

- ✓ One call to **`sendrecv,`** one for all three field to “propagate”
  - 3x1x2 calls every time step
- ✓ no mpi-datatype, explicit packing/unpacking, ported to GPU
- ✓ Cuda-aware MPI

```
!$acc kernels
    do k = 0,n+1
        do i = 0,l+1
            bufferYIN(i,k,1)=field1(i,m,k)
            bufferYIN(i,k,2)=field2(i,m,k)
            bufferYIN(i,k,3)=field3(i,m,k)
        enddo
    enddo
!$acc end kernels
!
!$acc host_data use_device(bufferYIN,bufferYOUT)
    call mpi_sendrecv(bufferYIN(0,0,1),msgsizeY,MYMPIREAL,right(2),tag,&
                      bufferYOUT(0,0,1),msgsizeY,MYMPIREAL,left(2),tag,&
                      lbecomm,status,ierr)
!$acc end host_data
```

Step4

- ✓ 1024^3, 8 tasks
- ✓ 10’000 timestep
- ✓ Nvidia A100@64GB (Leonardo)

```
# Time for section
# loop    time  0.730247E+02   0.730234E+02
# comp.   time  0.480949E+02   0.476836E+02
# diag.   time  0.000000E+00   0.000000E+00
# MPI     time  0.193703E+02   0.197539E+02
#--------------------------------
# Ratio
# Ratio Coll  0.659   0.653
# Ratio Dg.   0.000   0.000
# Ratio MPI   0.265   0.271
# Check       0.076   0.076
#--------------------------------
# Z-MPI time, BW (MB/s) -->  1.667969       6042.246
# Y-MPI time, BW (MB/s) -->  1.875000       5375.081
# X-MPI time, BW (MB/s) -->  15.82422       636.8894
```

Step8

- ✓ Using non blocking comms
  - **`isend/recv`**
  - 3x1x2 calls every time step
- ✓ overlapping some computation

```
! First pack data ...
! Second send data  ...
! Third receive data ...
! do something “else”
! Fourth unpack data ...
! Fifth wait ...
```

Step8

```
        tag = 10
!$acc host_data use_device(bufferXINM)
        call mpi_isend(bufferXINM(0,0,1),msgsizex,MYMPIREAL,rear(2),tag, &
                       lbecomm, reqs_rear(1), ierr)
!$acc end host_data
!
        tag = 11
!
!$acc host_data use_device(bufferXOUTP)
        call mpi_recv(bufferXOUTP(0,0,1),msgsizeX,MYMPIREAL,rear(2), tag, &
                       lbecomm, status_front, ierr)
!$acc end host_data
…
< do something else >
…
! Fourth  wait...
        call mpi_wait(reqs_front(1), status_front, ierr)
```

Step8

- ✓ 1024^3, 8 tasks
- ✓ 10’000 timestep
- ✓ Nvidia A100@64GB (Leonardo)

```
# Time for section
# loop    time   0.679699E+02    0.679688E+02
# comp.   time   0.242806E+02    0.242852E+02
# diag.   time   0.000000E+00    0.000000E+00
# MPI     time   0.382523E+02    0.382109E+02
#-------------------------------
```

Step9

- ✓ Using non blocking comms
  - **isend/recv**
  - 3x1x2 calls every time step
- ✓ overlapping some computation
- ✓ asynchronous

```
!$acc kernels async
    do j = 0,m+1
        do i = 0,l+1
! z+ direction
            bufferZINP(i,j,1)=field1(i,j,n)
            bufferZINP(i,j,2)=field2(i,j,n)
            bufferZINP(i,j,3)=field3(i,j,n)
        enddo
    enddo
!$acc end kernels
```

Step9

- ✓ 1024^3, 8 tasks
- ✓ 10’000 timestep
- ✓ Nvidia A100@64GB (Leonardo)

```
# Time for section
# loop    time   0.599483E+02    0.599492E+02
# comp.   time   0.246177E+02    0.247109E+02
# diag.   time   0.000000E+00    0.000000E+00
# MPI     time   0.299210E+02    0.297891E+02
#-------------------------------
```

Step10

- ✓ Using non blocking comms
  - **`isend/recv`**
  - 3x1x2 calls every time step
- ✓ overlapping some computation via a matrix (border/bulk points)
- ✓ asynchronous
- ✓ More simple to manage borders

```
! First pack data ...
! Second send data  ...
! Third receive data ...
! do something (bulk only)
! Fourth unpack data ...
! Fifth wait ...
! do something (border only)
```

Step10

```
! first set everything to border
    do k = 1, n
        do j = 1, m
            do i = 1, l
                mask(i,j,k) = uno
            end do
        end do
    end do
!
! second set the bulk
    do k = 1+border, n-border
        do j = 1+border, m-border
            do i = 1+border, l-border
                mask(i,j,k) = zero
            end do
        end do
    end do
```

Step9

- ✓ 1024^3, 8 tasks
- ✓ 10’000 timestep
- ✓ Nvidia A100@64GB (Leonardo)

```
# Time for section
# loop    time   0.625515E+02    0.625508E+02
# comp.   time   0.434290E+02    0.436133E+02
# diag.   time   0.000000E+00    0.000000E+00
# MPI     time   0.428037E+02    0.426250E+02
#-------------------------------
```

Performance figure (GPU)

- ✓ 1024^3,
- ✓ 8 tasks
- ✓ 10’000 timestep
- ✓ NVIDIA A100@64 GB (Leonardo)

Version	HW	Time	%MPI	%Comp
Step3	GPU	119”	55%	40%
Step4	GPU	73”	27%	65%
Step8	GPU	68”’	-	-
Step9	GPU	59”	-	-
Step10	GPU	62”	-	-

Performance figure (GPU)

- ✓ 1024^3,
- ✓ 64 tasks
- ✓ 100’000 timestep
- ✓ NVIDIA A100@64 GB (Leonardo)

Version	HW	Time	%MPI	%Comp
Step3	GPU	376”	80%	20%
Step9	GPU	124”	-	-
Step10	GPU	149”	-	-

Performance figure (GPU)

- ✓ 1024^3,
- ✓ 512 tasks
- ✓ 1’000’000 timestep
- ✓ NVIDIA A100@64 GB (Leonardo)

Version	HW	Time	%MPI	%Comp
Step3	GPU	3900”	-	-
Step9	GPU	569”	-	-
Step10	GPU	656”	-	-