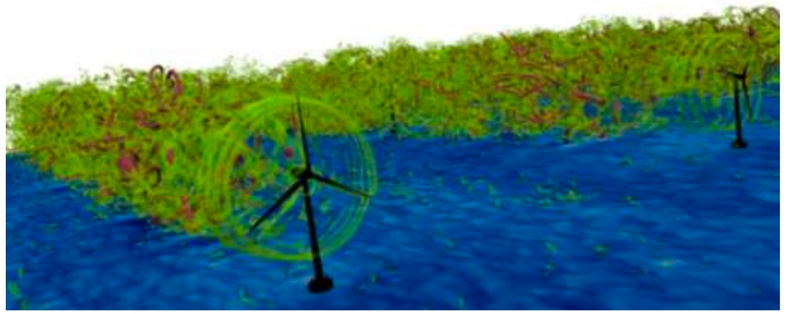
HPC·AI
ADVISORY COUNCIL
NETWORK OF EXPERTISE

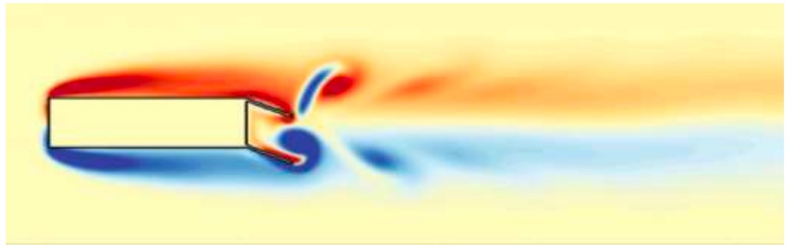
ISC2022 Coding Challenge

Filippo Spiga
filippo@hpcadvisorycouncil.com

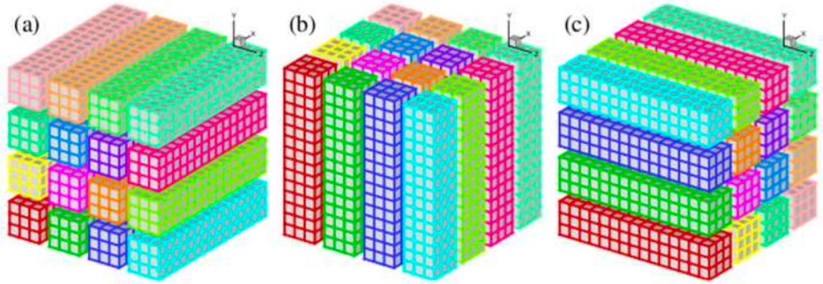
Xcompact3D

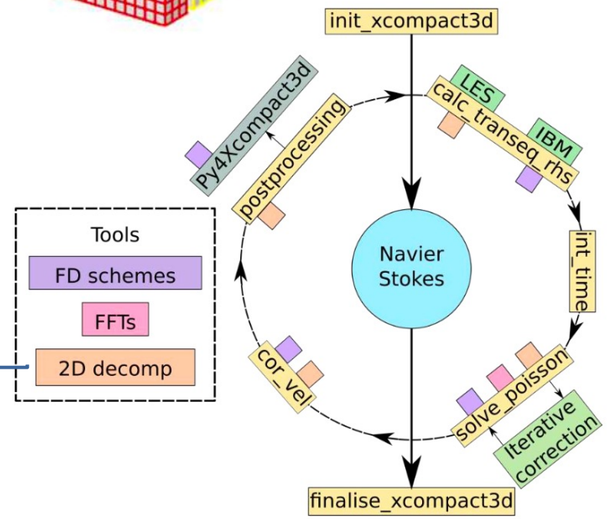
SCIENCE





IMPLEMENTATION





Tasks for the competition -- Summary

	Getting familiar with the BlueField-2 DPU	
1	Modifying the given application codebase to leverage NVIDIA DPU	30%
2	Performance assessment of original versus modified application	30%
3	Summarize all your findings and results	20%
4	Explore different inputs and configurations	20%

Task 1 -- Modifying the given application codebase to leverage NVIDIA DPU

Modify the code to use non-blocking strategy (call MPI_Ialltoall instead of MPI_Alltoall) and run the full modified xcompact3d application using DPU offload mode.

Application source code: https://github.com/xcompact3d/Incompact3d (tag: v4.0)

Submission criteria

- Submit the entire modified code plus building and submission scripts to Filippo Spiga (zip file)
- Bonus points for readability and level of comments

What is allowed?

- Modify any portion of the source code for as long as the results is correct and reproducible

What is NOT allowed?

- Reducing artificially the I/O
- Change the physics of the input files
- Change version of the code

Task 2 -- Performance assessment of original versus modified application

Run the original and modified xcompact3d application using the cylinder input case with specific problem size. Obtain performance measurements using 8 nodes with and without the DPU adapter, make sure to vary PPN (4, 8, 16, 32). Run MPI profiler (mpiP or IPM) to understand if MPI overlap is happening and how the parallel behavior of the application has changed.

Submission criteria

- Submit all building scripts and outputs
- Submit baseline scaling results (graph) using unmodified application
- Submit baseline scaling results (graph) using modified application
  - *What is the message size used for MPI_alltoall / MPI_Ialltoall? How message size related to performance improvements?*
- The modified code must be correct and return exactly the same results by any giver input and set of execution parameters (number of MPI processes, number of MPI processes per node, grid size, grid decomposition)

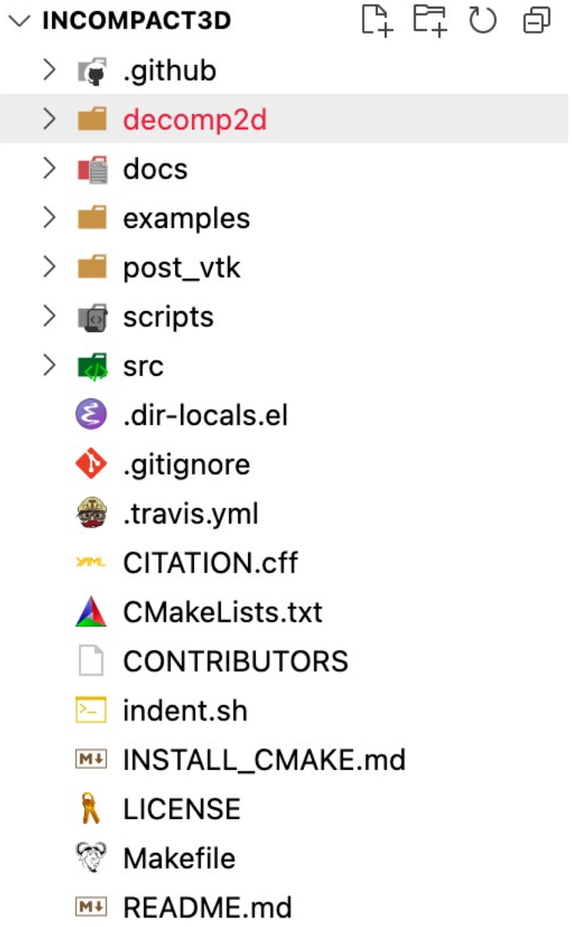
Task 3 -- Summarize all your findings and results

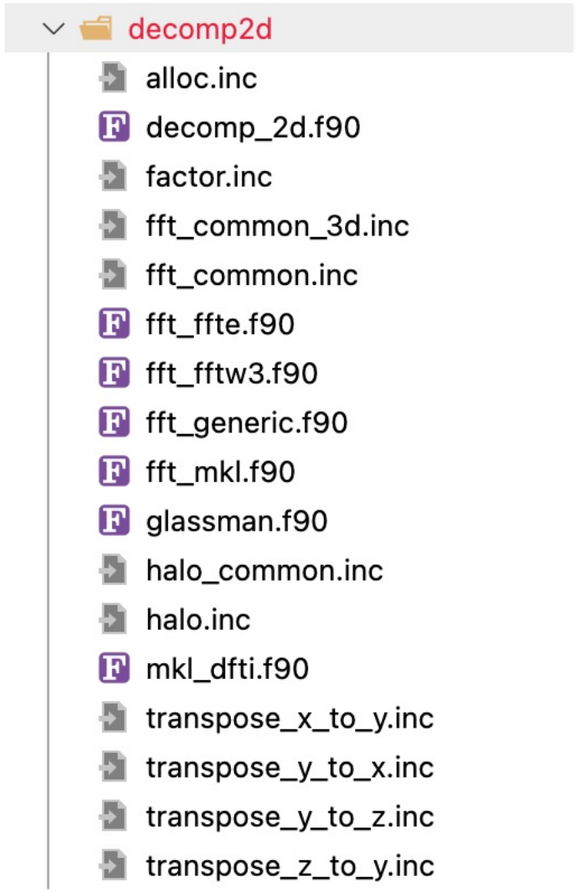
Submit a report of what you managed to achieve and learned, include a description of the code changes (better if also done as comments in the modified code) and all meaningful comparison of results with and without DPU offload. Elaborate why you get (or did not get) performance improvements.

Submission criteria

- Report - a document/slides that explain what you did and what was learned
  - Tables and graphs and MPI traces are welcome, alongside a clear description what has been done
  - Try to highlight clearly the contributions made by each team members in which tasks
- Performance improvement of your modified code vs. the original code provided
  - A ranking will be created listing all teams who successfully submitted a working code

FFT transpose prototypes (2decomp)






```
interface transpose_x_to_y
   module procedure transpose_x_to_y_real
   module procedure transpose_x_to_y_complex
end interface transpose_x_to_y

interface transpose_y_to_z
   module procedure transpose_y_to_z_real
   module procedure transpose_y_to_z_complex
end interface transpose_y_to_z

interface transpose_z_to_y
   module procedure transpose_z_to_y_real
   module procedure transpose_z_to_y_complex
end interface transpose_z_to_y

interface transpose_y_to_x
   module procedure transpose_y_to_x_real
   module procedure transpose_y_to_x_complex
end interface transpose_y_to_x
```

2decomp_FFT, blocking transpose prototypes (simplified)


```
subroutine transpose_x_to_y_real(src, dst, opt_decomp)

  implicit none

  real(mytype), dimension(:,:,:), intent(IN) :: src
  real(mytype), dimension(:,:,:), intent(OUT) :: dst
  TYPE(DECOMP_INFO), intent(IN), optional :: opt_decomp

  TYPE(DECOMP_INFO) :: decomp

  ! rearrange source array as send buffer
  call mem_split_xy_real(src, s1, s2, s3, work1_r, dims(1), &
       decomp%x1dist, decomp)

  ! transpose using MPI_ALLTOALL(V)
  call MPI_ALLTOALLV(work1_r, decomp%x1cnts, decomp%x1disp, &
       real_type, work2_r, decomp%y1cnts, decomp%y1disp, &
       real_type, DECOMP_2D_COMM_COL, ierror)

  call mem_merge_xy_real(work2_r, d1, d2, d3, dst, dims(1), &
       decomp%y1dist, decomp)

  return
end subroutine transpose_x_to_y_real
```

2decomp_FFT, non-blocking transpose prototypes

```
interface transpose_x_to_y
   module procedure transpose_x_to_y_real
   module procedure transpose_x_to_y_complex
end interface transpose_x_to_y
```

Post a MPI communication, control return to main process

NON-BLOCKING BEHAVIOUR

```
interface transpose_x_to_y_start
   module procedure transpose_x_to_y_real_start
   module procedure transpose_x_to_y_complex_start
end interface transpose_x_to_y_start
```

```
interface transpose_x_to_y_wait
   module procedure transpose_x_to_y_real_wait
   module procedure transpose_x_to_y_complex_wait
end interface transpose_x_to_y_wait
```

Process wait until MPI communication completed

BLOCKING BEHAVIOUR

Extra Suggestions

- https://github.com/numericalalgorithmsgroup/2decomp_fft
- Ignore “SHM” code path as start

Example (fake code!)

```
ux1(:,:,:) = ux1(:,:,:) + A(:,:,:)

uy1(:,:,:) = uy1(:,:,:) - B(:,:,:)

call transpose_x_to_y(ux1,ux2)

call transpose_x_to_y(uy1,uy2)

X(:,:,:) = ux2(:,:,:) * ux1(:,:,:)

Z(:,:,:) = uy2(:,:,:) * uy1(:,:,:)
```

Example (fake code!)

```
ux1(:,:,:) = ux1(:,:,:) + A(:,:,:)

uy1(:,:,:) = uy1(:,:,:) - B(:,:,:)

call transpose_x_to_y(ux1,ux2)

call transpose_x_to_y(uy1,uy2)

X(:,:,:) = ux2(:,:,:) * ux1(:,:,:)

Z(:,:,:) = uy2(:,:,:) * uy1(:,:,:)
```

Example (fake code!)

```
ux1(:,:,:) = ux1(:,:,:) + A(:,:,:)
uy1(:,:,:) = uy1(:,:,:) - B(:,:,:)
call transpose_x_to_y(ux1,ux2)
call transpose_x_to_y(uy1,uy2)
X(:,:,:) = ux2(:,:,:) * ux1(:,:,:)
Z(:,:,:) = uy2(:,:,:) * uy1(:,:,:)
```

```
ux1(:,:,:) = ux1(:,:,:) + A(:,:,:)
call transpose_x_to_y_start (ux1,ux2)
uy1(:,:,:) = uy1(:,:,:) - B(:,:,:)
call transpose_x_to_y_start(uy1,uy2)
call transpose_x_to_y_wait (ux1,ux2)
X(:,:,:) = ux2(:,:,:) * ux1(:,:,:)
call transpose_x_to_y_wait(uy1,uy2)
Z(:,:,:) = uy2(:,:,:) * uy1(:,:,:)
```

Example (fake code!)

```
ux1(:,:,:) = ux1(:,:,:) + A(:,:,:)
uy1(:,:,:) = uy1(:,:,:) - B(:,:,:)
call transpose_x_to_y(ux1,ux2)
call transpose_x_to_y(uy1,uy2)
X(:,:,:) = ux2(:,:,:) * ux1(:,:,:)
Z(:,:,:) = uy2(:,:,:) * uy1(:,:,:)
```

```
ux1(:,:,:) = ux1(:,:,:) + A(:,:,:)
call transpose_x_to_y_start (ux1,ux2)
uy1(:,:,:) = uy1(:,:,:) - B(:,:,:)
call transpose_x_to_y_start(uy1,uy2)
call transpose_x_to_y_wait (ux1,ux2)
X(:,:,:) = ux2(:,:,:) * ux1(:,:,:)
call transpose_x_to_y_wait(uy1,uy2)
Z(:,:,:) = uy2(:,:,:) * uy1(:,:,:)
```

OVERLAP COMPUTE AND COMM

SYNC POINT

Example (fake code!)

```
ux1(:,:,:) = ux1(:,:,:) + A(:,:,:)
uy1(:,:,:) = uy1(:,:,:) - B(:,:,:)
call transpose_x_to_y(ux1,ux2)
call transpose_x_to_y(uy1,uy2)
X(:,:,:) = ux2(:,:,:) * ux1(:,:,:)
Z(:,:,:) = uy2(:,:,:) * uy1(:,:,:)
```

```
ux1(:,:,:) = ux1(:,:,:) + A(:,:,:)
call transpose_x_to_y_start (ux1,ux2)
uy1(:,:,:) = uy1(:,:,:) - B(:,:,:)
call transpose_x_to_y_start(uy1,uy2)
call transpose_x_to_y_wait (ux1,ux2)
X(:,:,:) = ux2(:,:,:) * ux1(:,:,:)
call transpose_x_to_y_wait(uy1,uy2)
Z(:,:,:) = uy2(:,:,:) * uy1(:,:,:)
```

OVERLAP COMPUTE AND COMM

→ SYNC POINT

Using the DPU with Xcompact3D (submission script)

```
#!/bin/bash -l
#SBATCH -p thor
#SBATCH --nodes=8
#SBATCH -J coding_challenge
#SBATCH --time=15:00
#SBATCH --exclusive

# Loading right environment
module load ...

srun -l hostname -s | awk '{print $2}' | grep -v bf | sort > hostfile
srun -l hostname -s | awk '{print $2}' | grep bf | sort |uniq > dpufile
NPROC=$(cat hostfile |wc -l)

# The "input.i3d" needs to be in the same directory of job execution
EXE=xcompact3d

# No DPU offload
mpirun_rsh -np $NPROC -hostfile hostfile MV2_USE_DPU=0 $EXE

# DPU offload
mpirun_rsh -np $NPROC -hostfile hostfile -dpufile dpufile $EXE
```

Final recap

What you can do

- Adapt and modify any portion of the source code
- Do your development elsewhere

What you cannot do

- Change the Xcompact3D version
- Change the input (unless you are on the final task, changes need to be documented)
- Taking shortcuts even if results are reproducible
- Submit results and profiling outputs done on a different machine

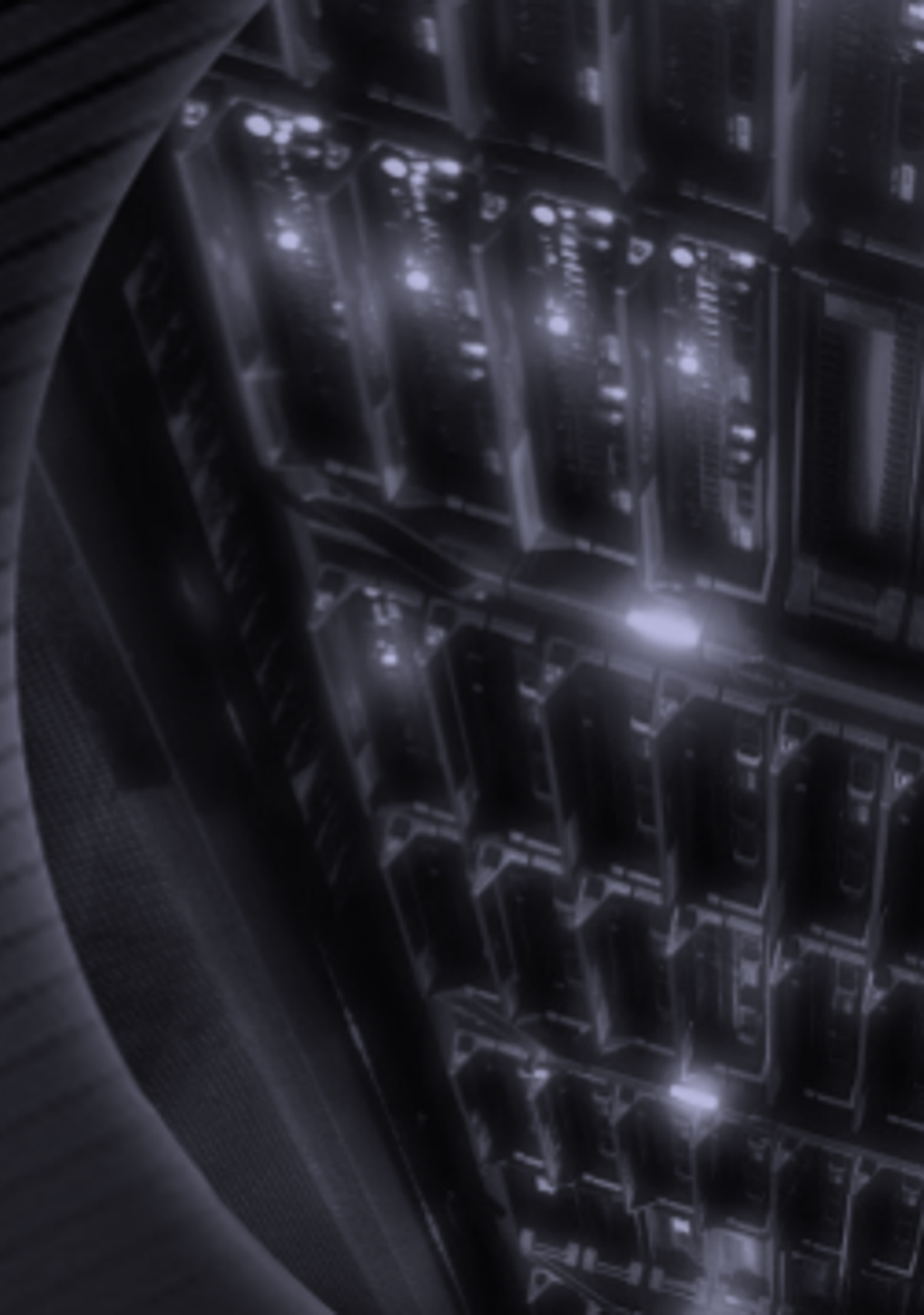
What we would like to see from you

- Clear and working code
- A report (slides or text, no preference) that covers all key points of the “journey” you took. The “hows” and the “why”.

What is not going to be evaluated

- Any time-to-solution improvement that is not linked or justified by the need of using the DPU

Questions?



Thank You