Calling Microphysical Schemes from WRF and libcloudph++ using Python

Dorota Jarecka

University of Warsaw
NCAR

Workshop on Eulerian vs. Lagrangian methods for cloud microphysics
2015 April 20th; Warsaw, Poland

talk outline

1. introduction
2. binding to libcloudph++ library
3. bindings to WRF model
4. bindings libcloudph++ and atmospheric models
5. summary

talk outline

programming languages binding

- bindings from a programming language to a library or operating system service

programming languages binding

- bindings from a programming language to a library or operating system service
- programming interfaces providing glue code to use the library in a particular programming language

programming languages binding

- bindings from Python language to the libcloudph++ library or WRF model
- programming interfaces providing glue code to use the libcloudphxx++ or WRF in Python language

Python language - why using?

general-purpose, high-level programming language

Python language - why using?

general-purpose, high-level programming language

- developed with emphasis on code readability
- concise syntax

Python language - why using?

general-purpose, high-level programming language

- developed with emphasis on code readability
- concise syntax
- many scientific libraries: NumPy, SciPy, Matplotlib, ...

Python language - why using?

general-purpose, high-level programming language

- developed with emphasis on code readability
- concise syntax
- many scientific libraries: NumPy, SciPy, Matplotlib, ...
- growing scientific community
  - AMS Annual Meeting: Symposium on Advances in Modeling and Analysis Using Python
  - UCAR Software Engineering Assembly Conference: Python in Scientific Computing

Python language - why using?

general-purpose, high-level programming language

- developed with emphasis on code readability
- concise syntax
- many scientific libraries: NumPy, SciPy, Matplotlib, ...
- growing scientific community
  - AMS Annual Meeting: Symposium on Advances in Modeling and Analysis Using Python
  - UCAR Software Engineering Assembly Conference: Python in Scientific Computing
- large availability of trained personnel

Python language - why using?

general-purpose, high-level programming language

- developed with emphasis on code readability
- concise syntax
- many scientific libraries: NumPy, SciPy, Matplotlib, ...
- growing scientific community
  - AMS Annual Meeting: Symposium on Advances in Modeling and Analysis Using Python
  - UCAR Software Engineering Assembly Conference: Python in Scientific Computing
- large availability of trained personnel
- easy to learn and teach

bindings to Python language: what for?

bindings to Python language: what for?

- to use existing software

bindings to Python language: what for?

- to use existing software
- to compare with existing software

bindings to Python language: what for?

- to use existing software
- to compare with existing software
- to speed up most expensive part

bindings to Python language: what for?

- to use existing software
- to compare with existing software
- to speed up most expensive part
- to bind various languages together, eg., C++ with Fortran via Python

bindings to Python language: what for?

- to use existing software
- to compare with existing software
- to speed up most expensive part
- to bind various languages together, eg., C++ with Fortran via Python
- to use the same language for modelling, analysis, plotting, etc.

bindings to Python language: why?

bindings to Python language: why?

- a relatively new language in science

bindings to Python language: why?

- a relatively new language in science
- slower than Fortran, C, C++ due to language overhead

bindings to Python language: why?

- a relatively new language in science
- slower than Fortran, C, C++ due to language overhead
  - although problem often overestimated - Arabas S., D. Jarecka, A. Jaruga, M. Fijałkowski, 2014: Formula translation in Blitz++, NumPy and modern Fortran: A case study of the language choice tradeoffs, *Scien. Program.*

bindings to Python language: why?

- a relatively new language in science
- slower than Fortran, C, C++ due to language overhead
  - although problem often overestimated - Arabas S., D. Jarecka, A. Jaruga, M. Fijałkowski, 2014: Formula translation in Blitz++, NumPy and modern Fortran: A case study of the language choice tradeoffs, *Scien. Program.*
- issues with parallelization

bindings to Python language: why?

- a relatively new language in science
- slower than Fortran, C, C++ due to language overhead
  - although problem often overestimated - Arabas S., D. Jarecka, A. Jaruga, M. Fijałkowski, 2014: Formula translation in Blitz++, NumPy and modern Fortran: A case study of the language choice tradeoffs, *Scien. Program.*
- issues with parallelization
- good as a glue language

talk outline

1. introduction
2. binding to libcloudph++ library
3. bindings to WRF model
4. bindings libcloudph++ and atmospheric models
5. summary

talk outline

C++ language

general-purpose programming language

C++ language

general-purpose programming language

- comparable performance to Fortran

C++ language

general-purpose programming language

- comparable performance to Fortran
- large availability of trained personnel
- boost.units for checking physical units

C++ language

general-purpose programming language

- comparable performance to Fortran
- large availability of trained personnel
- boost.units for checking physical units
- thrust library for implementations for CPU and GPU

C++ language

general-purpose programming language

- comparable performance to Fortran
- large availability of trained personnel
- boost.units for checking physical units
- thrust library for implementations for CPU and GPU
- but not so easy syntax...

libcloudph++ library and Python bindings

Python

C++

libcloudph++ library and Python bindings

Python

C++

- numerically-intensive algorithms
- including concurrency
- implementation for CPU and GPU

libcloudph++ library and Python bindings

C++

- numerically-intensive algorithms
- including concurrency
- implementation for CPU and GPU

Python

- user interface (no need to interact with the native C++ interface)
- rapid-development of new features
- interfacing with other languages

libcloudph++ library and Python bindings

C++

- numerically-intensive algorithms
- including concurrency
- implementation for CPU and GPU

Python

- user interface (no need to interact with the native C++ interface)
- rapid-development of new features
- interfacing with other languages

Jarecka D., S. Arabas, D. Del Vento: Python bindings for libcloudph++, *http://arxiv.org/abs/1504.01161*

libcloudph++ with Python: a general structure

your_code.py

```
print "Hello World"
```

has access to:

"core" Python language

libcloudph++ with Python: a general structure

your_code.py

```
print "Hello World"

import numpy
```

has access to:

"core" Python language

NumPy - part of the standard library

libcloudph++ with Python: a general structure

your_code.py

```
print "Hello World"

import numpy

import libcloudphxx as libcl
```

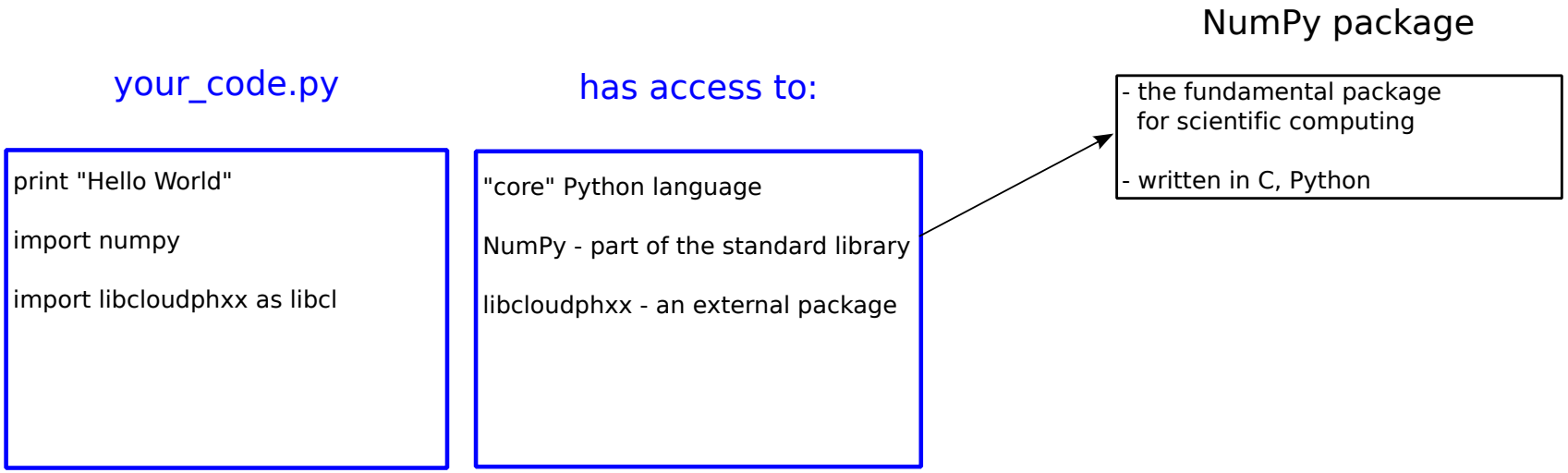
has access to:

"core" Python language

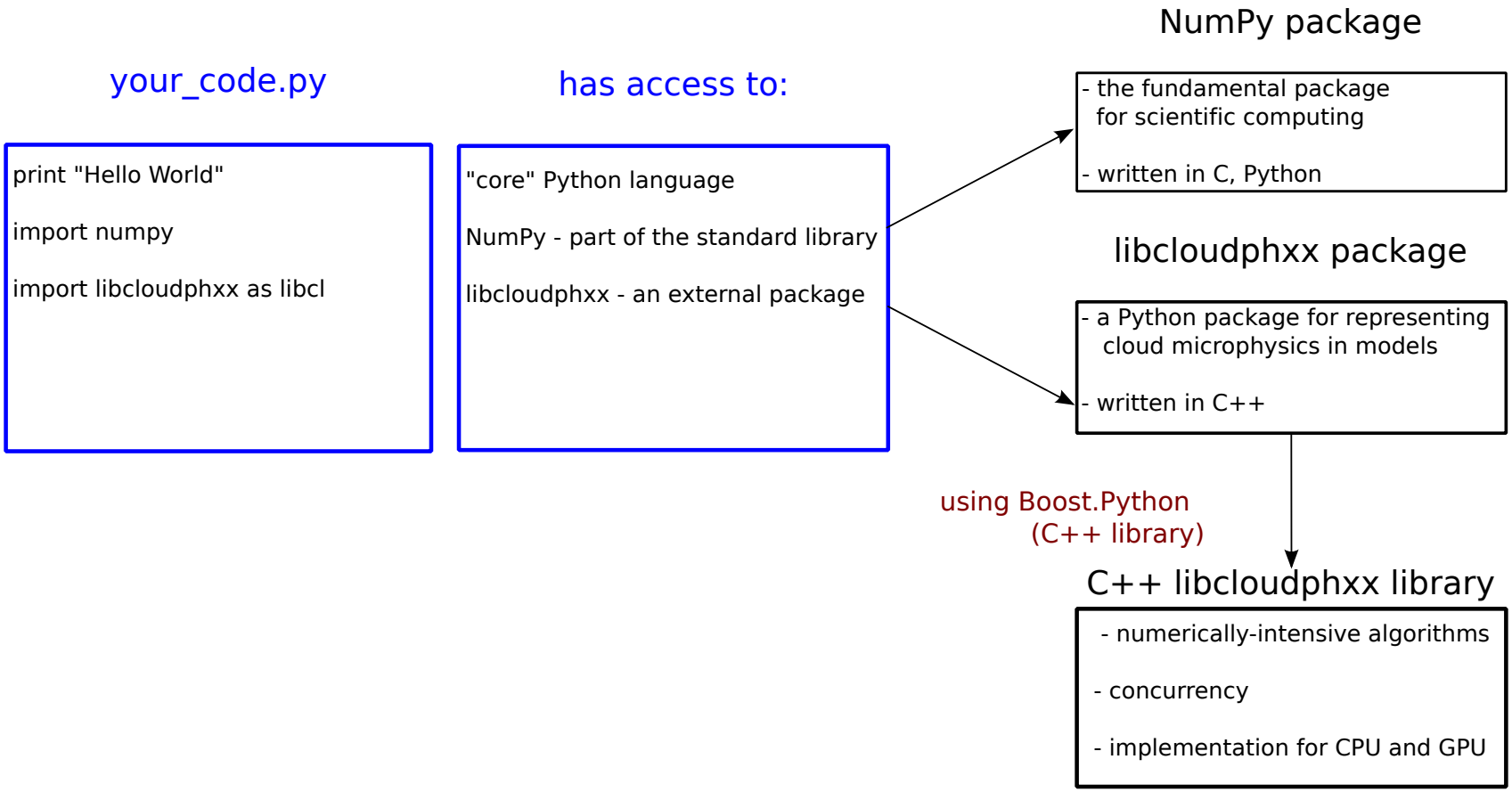
NumPy - part of the standard library

libcloudphxx - an external package

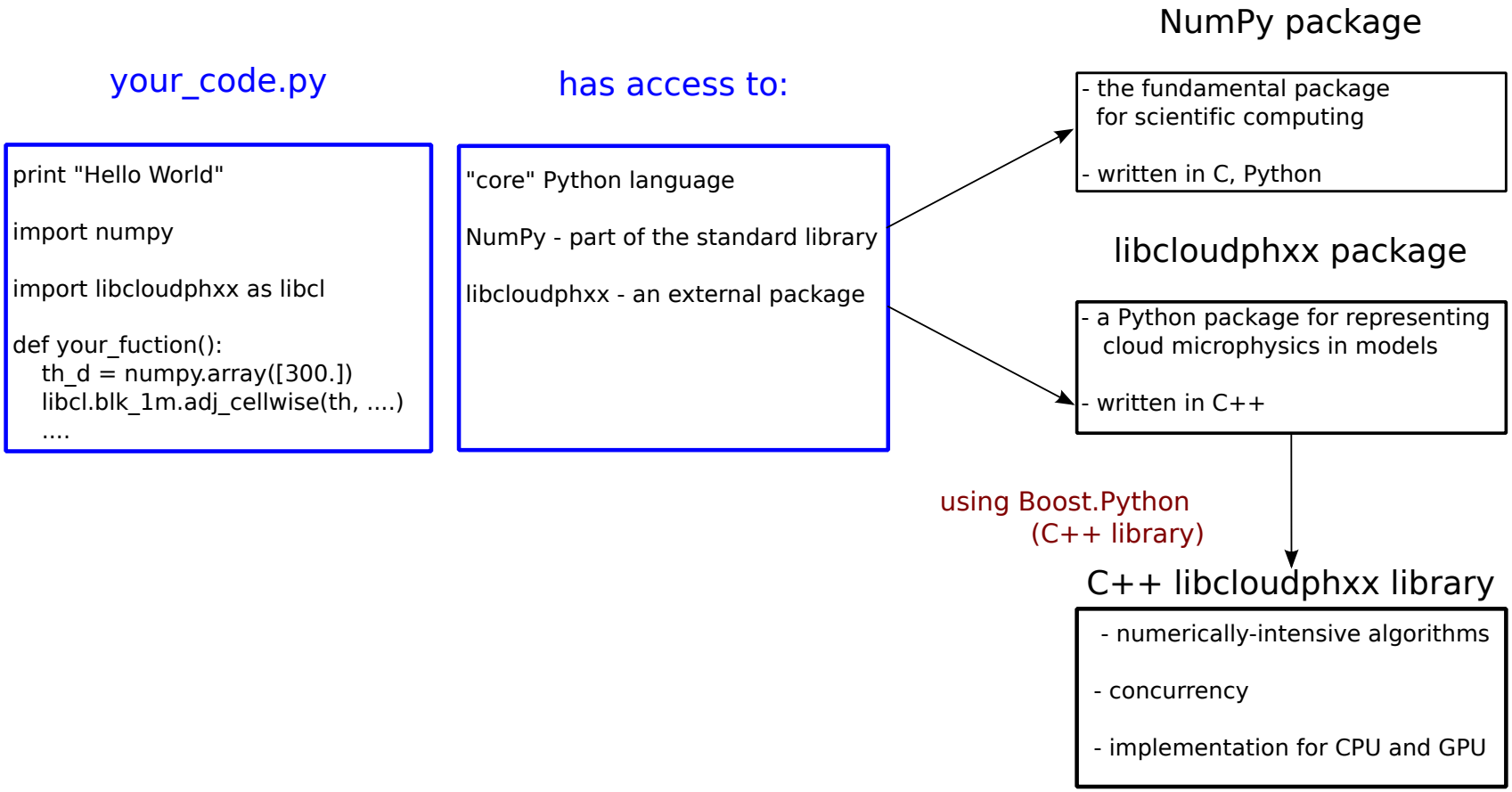
libcloudph++ with Python: a general structure



libcloudph++ with Python: a general structure



libcloudph++ with Python: a general structure



libcloudph++ with Python: examples

- calling saturation adjustment procedure

libcloudph++ with Python: examples

- calling saturation adjustment procedure

```
import numpy
import libcloudphxx as libcl

opts = libcl.blk_1m.opts_t()

rhod = numpy.array([1.   ])
th_d = numpy.array([305. ])
r_v  = numpy.array([0.01 ])
r_c  = numpy.array([0.001])
r_r  = numpy.array([0.001])
dt   = 1

libcl.blk_1m.adj_cellwise(opts,
        rhod,                  # array, read-only
        th_d, r_v, r_c, r_r, # arrays, read-write
        dt)                    # scalar
```

talk outline

1. introduction
2. binding to libcloudph++ library
3. bindings to WRF model
4. bindings libcloudph++ and atmospheric models
5. summary

talk outline

bindings to WRF: comparing with existing software

WRF - The Weather Research and Forecasting

bindings to WRF: comparing with existing software

WRF - The Weather Research and Forecasting

- mesoscale numerical weather prediction system

bindings to WRF: comparing with existing software

WRF - The Weather Research and Forecasting

- mesoscale numerical weather prediction system
- designed to serve both atmospheric research and operational forecasting needs

bindings to WRF: comparing with existing software

WRF - The Weather Research and Forecasting

- mesoscale numerical weather prediction system
- designed to serve both atmospheric research and operational forecasting needs
- a large worldwide community of registered users (over 25,000 in over 130 countries)

bindings to WRF: comparing with existing software

WRF - The Weather Research and Forecasting

- mesoscale numerical weather prediction system
- designed to serve both atmospheric research and operational forecasting needs
- a large worldwide community of registered users (over 25,000 in over 130 countries)
- many microphysical schemes to compare with

Fortran with Python: a general structure

your_code.py

```
import numpy
```

has access to:

"core" Python language

NumPy - part of the standard library

Fortran with Python: a general structure

your_code.py

```
import numpy

from cffi import FFI
ffi = FFI()
```

has access to:

"core" Python language

NumPy - part of the standard library

CFFI -C Foreign Function Interface

Fortran with Python: a general structure

your_code.py

```
import numpy

from cffi import FFI
ffi = FFI()

lib_F = ffi.dlopen("fortran_code.so")

ffi.cdef("void function_F(double th[]);")
```

has access to:

"core" Python language

NumPy - part of the standard library

CFFI -C Foreign Function Interface

Fortran modul or whole model

specific function from the module

Fortran with Python: a general structure

your_code.py

```
import numpy

from cffi import FFI
ffi = FFI()

lib_F = ffi.dlopen("fortran_code.so")

ffi.cdef("void function_F(double th[]);")
```

has access to:

"core" Python language

NumPy - part of the standard library

CFFI -C Foreign Function Interface

Fortran modul or whole model

specific function from the module

NumPy package

CFFI package

- Python package for binding to compiled C (or Fortran) code

Fortran with Python: a general structure

your_code.py

```
import numpy

from cffi import FFI
ffi = FFI()

lib_F = ffi.dlopen("fortran_code.so")

ffi.cdef("void function_F(double th[]);")
```

has access to:

"core" Python language

NumPy - part of the standard library

CFFI -C Foreign Function Interface

Fortran modul or whole model

specific function from the module

using CFFI
(Python library)

NumPy package

CFFI package

- Python package for binding to compiled C (or Fortran) code

fortran_code.so library

- shared Fortran library
- might use a C wrapper

original Fortran code

- written entirely in Fortran

Fortran with Python: a general structure

your_code.py

```
import numpy

from cffi import FFI
ffi = FFI()

lib_F = ffi.dlopen("fortran_code.so")

ffi.cdef("void function_F(double th[]);")

def your_fuction():
    th = numpy.array([300.])
    th_ffi = ffi.cast(th,...)
    lib_F.bfunction_F(th_ffi)
    ....
```

has access to:

"core" Python language

NumPy - part of the standard library

CFFI -C Foreign Function Interface

Fortran modul or whole model

specific function from the module

using CFFI
(Python library)

NumPy package

CFFI package

- Python package for binding to compiled C (or Fortran) code

fortran_code.so library

- shared Fortran library
- might use a C wrapper

original Fortran code

- written entirely in Fortran

WRF with Python: examples

- creating a shared library from Kessler module

```
gfortran -c -fPIC module_mp_kessler.f90 -o module_mp_kessler.o

gfortran -shared -fPIC module_mp_kessler.o wraper.f90 -o libkessler.so
```

WRF with Python: examples

- calling Kessler scheme from the shared library

```
import numpy as np
from cffi import FFI
ffi = FFI()

# function creates cdata variables of a type "double *" from a numpy array
def as_pointer(numpy_ar):
    return ffi.cast("double*", numpy_ar.__array_interface__['data'][0])

# define a python function - binding a C function
def kessler(nx, ny, nz, dt_in, variable_nparr):
    # provide a signature for the C function
    ffi.cdef("void c_kessler(double t[], double qv[], double qc[], ....
              int its, int ite, int jts, int jte, int kts, int kte);")

    # load a library with the C function
    lib = ffi.dlopen('libkessler.so')
    ......
    lib.c_kessler(CFFI_ar["t"], CFFI_ar["qv"], CFFI_ar["qc"], ...
                  its, ite, jts, jte, kts, kte)
```

talk outline

1. introduction
2. binding to libcloudph++ library
3. bindings to WRF model
4. bindings libcloudph++ and atmospheric models
5. summary

talk outline

accessing libcloudph++ from Fortran: why?

accessing libcloudph++ from Fortran: why?

- to join comparison studies

accessing libcloudph++ from Fortran: why?

- to join comparison studies
  - KiD_A project - using the Kinematic Driver model (KiD) to compare detailed and bulk microphysics schemes

accessing libcloudph++ from Fortran: why?

- to join comparison studies
  - KiD_A project - using the Kinematic Driver model (KiD) to compare detailed and bulk microphysics schemes
- to compare results to microphysical schemes used in other atmospheric models

accessing libcloudph++ from Fortran: why?

- to join comparison studies
  - KiD_A project - using the Kinematic Driver model (KiD) to compare detailed and bulk microphysics schemes
- to compare results to microphysical schemes used in other atmospheric models
  - Dutch Atmospheric Large Eddy Simulation (DALES)

accessing libcloudph++ from Fortran: why?

- to join comparison studies
  - KiD_A project - using the Kinematic Driver model (KiD) to compare detailed and bulk microphysics schemes
- to compare results to microphysical schemes used in other atmospheric models
  - Dutch Atmospheric Large Eddy Simulation (DALES)
- to extend a group of users

accessing libcloudph++ from Fortran: how?

accessing libcloudph++ from Fortran: how?

- without changes to the libcloudph++ library

accessing libcloudph++ from Fortran: how?

- without changes to the libcloudph++ library
- with only minimal changes to other models

accessing libcloudph++ from Fortran: how?

- without changes to the libcloudph++ library
- with only minimal changes to other models
- using existing Python bindings

accessing libcloudph++ from Fortran: how?

- without changes to the libcloudph++ library
- with only minimal changes to other models
- using existing Python bindings
  - to the C++ libcloudph++ library
  - to exemplary Fortran code

calling the libcloudph++ library from Fortran: examples

- Python code that initialize the Fortran model

```
from cffi import FFI
from libcloudphxx import common

ffi = FFI()
lib = ffi.dlopen("test.so")
ffi.cdef("void main(void*,void*);")

@ffi.callback("double(double,double,double)")
def rw3_cr(rd3, kappa, T):
  return common.rw3_cr(rd3, kappa, T)

@ffi.callback("double(double,double,double)")
def S_cr(rd3, kappa, T):
  return common.S_cr(rd3, kappa, T)

lib.main(rw3_cr, S_cr)
```

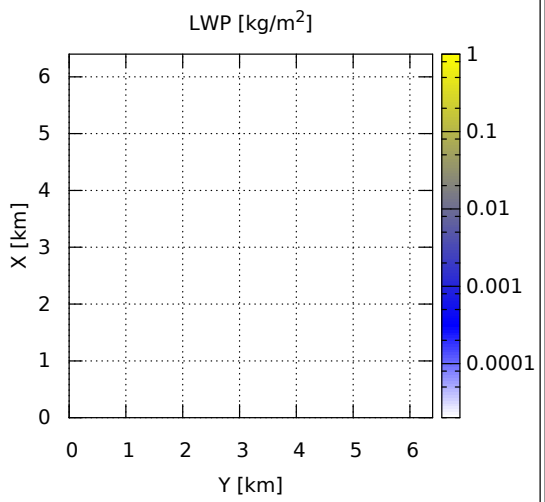
calling the libcloudph++ library from Fortran: examples

- Fortran code with main function

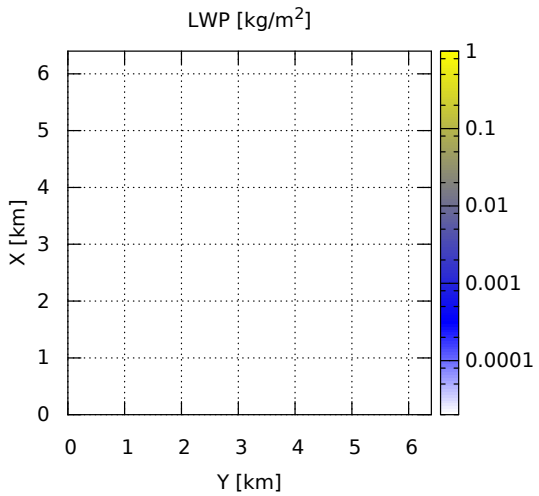
```
module test
  interface
    function f3arg(a1,a2,a3) bind(c)
      use iso_c_binding
      real(c_double) :: f3arg
      real(c_double), value :: a1,a2,a3
    end
  end interface

  contains

  subroutine main(rw3_cr_p, S_cr_p) bind(c)
    use iso_c_binding
    type(c_funptr), value :: rw3_cr_p, S_cr_p

    procedure(f3arg), pointer :: rw3_cr, S_cr
    call c_f_procpointer(rw3_cr_p, rw3_cr)
    call c_f_procpointer(S_cr_p, S_cr)
    ....
    print*, rw3_cr(rd3, kappa, T), S_cr(rd3, kappa, T)
    ....
```

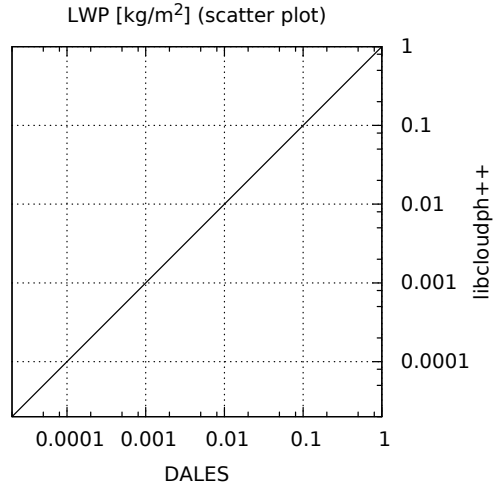
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=0m)



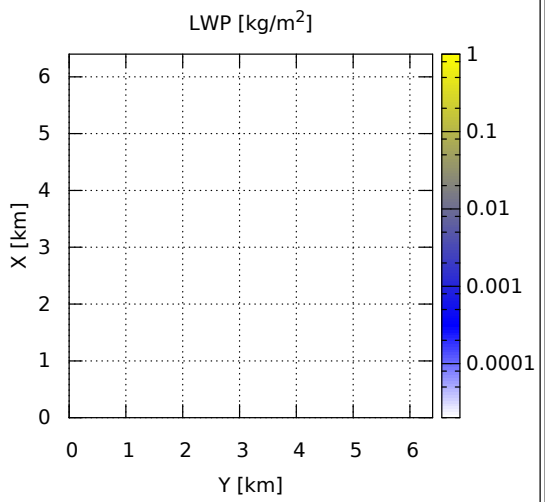
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



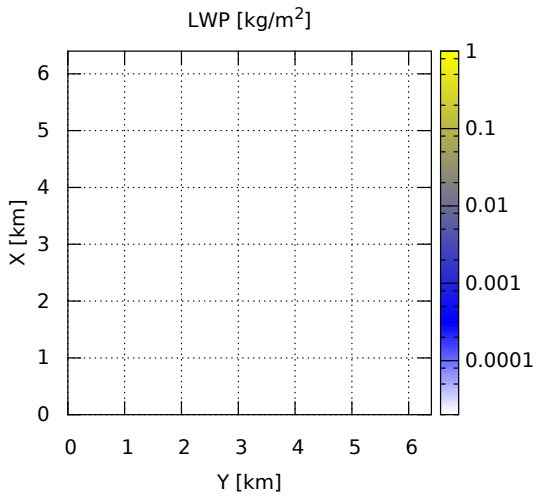


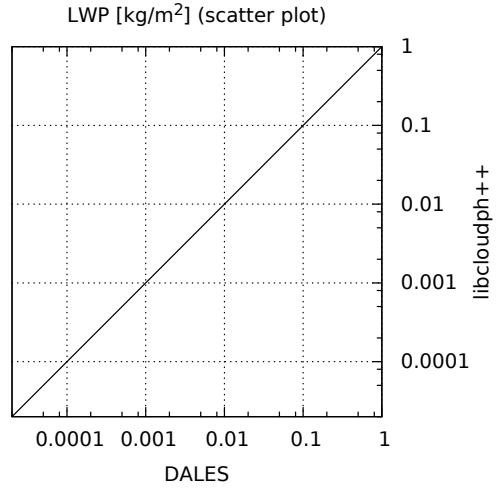
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=1m)



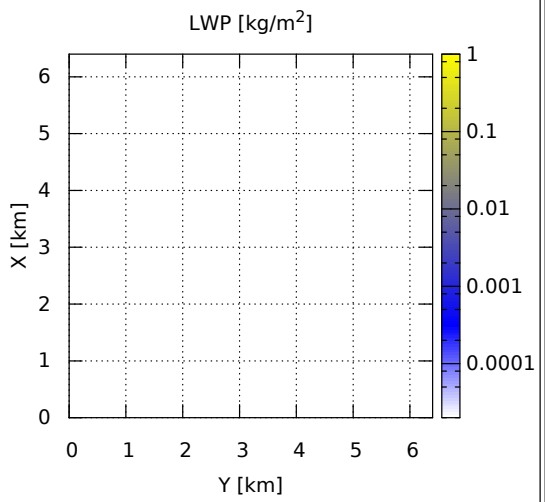
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



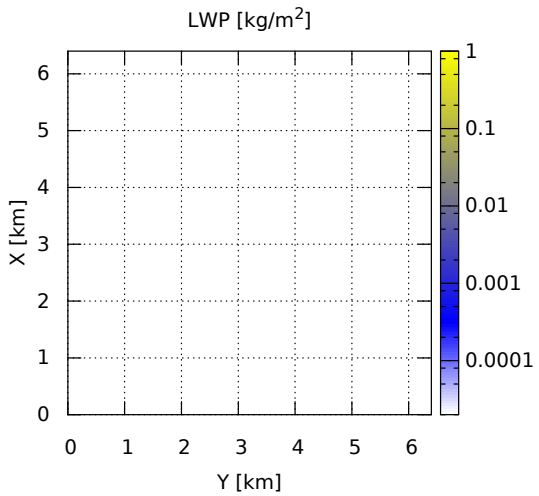


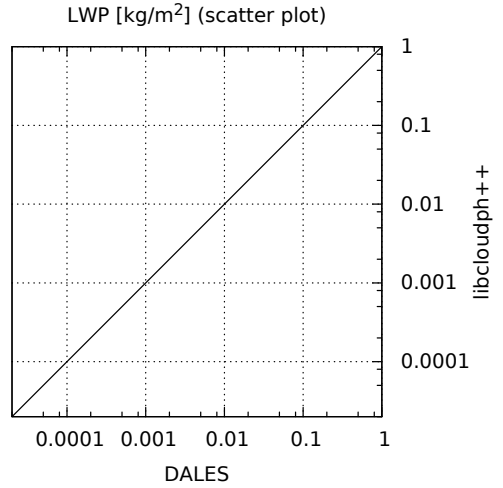
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=2m)



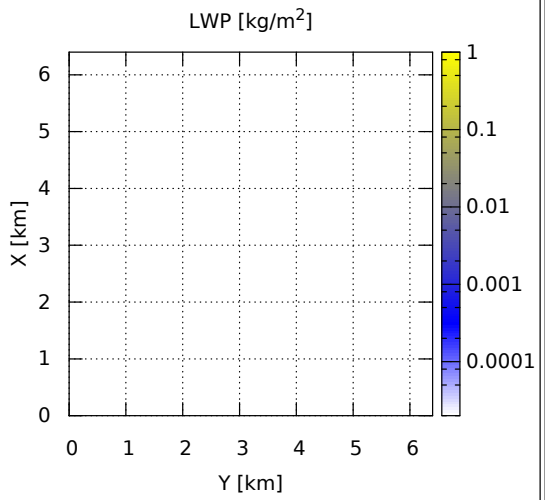
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



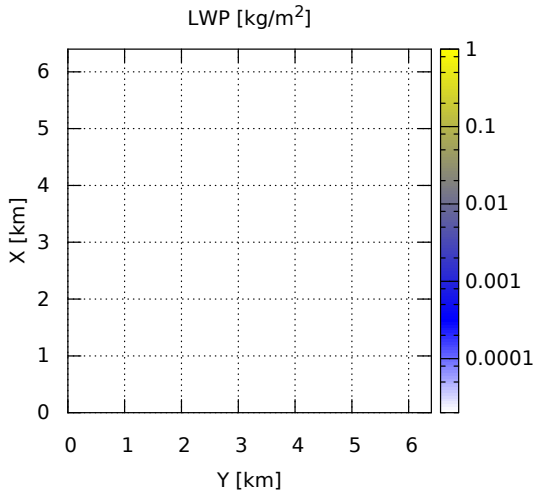


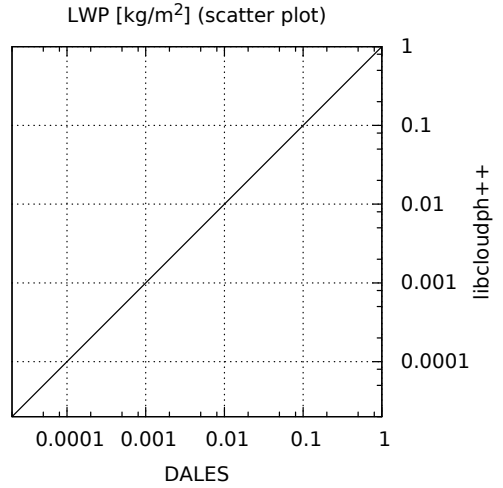
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=3m)



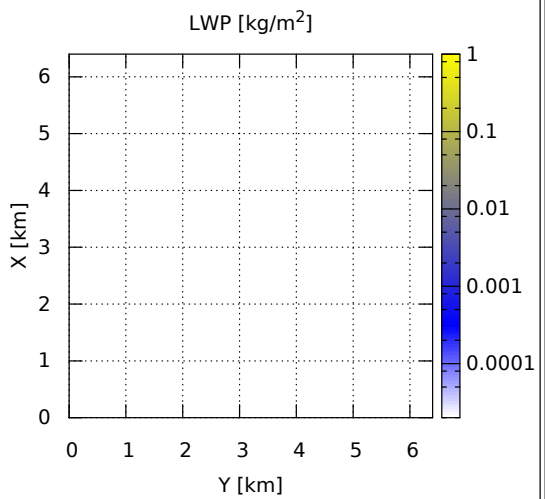
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



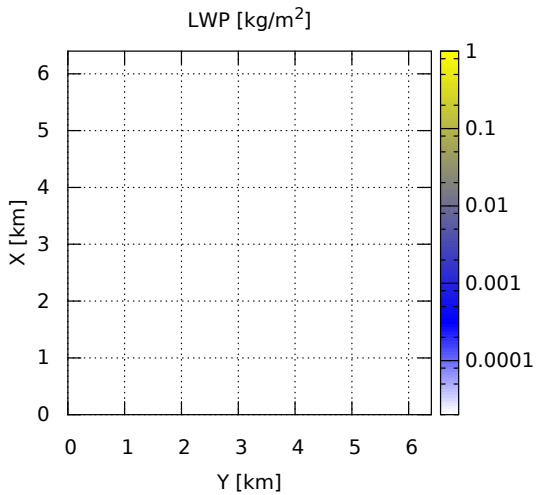


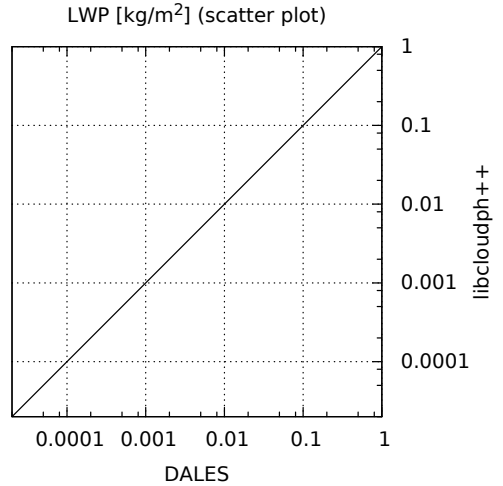
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=4m)



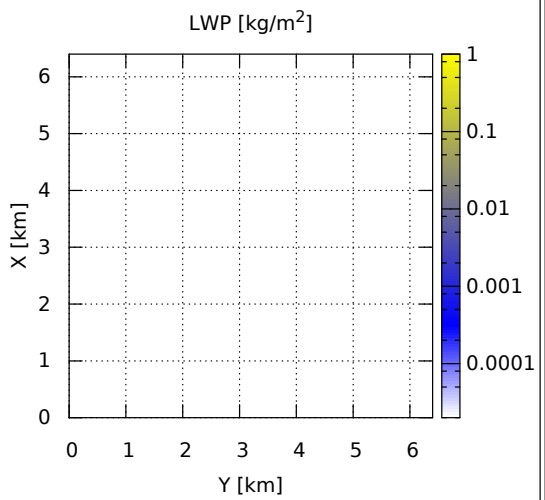
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



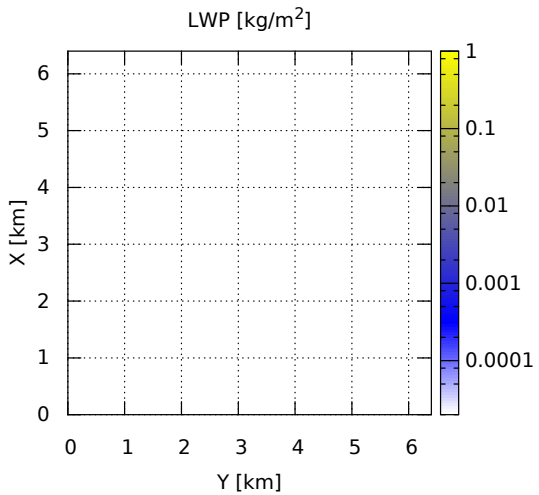


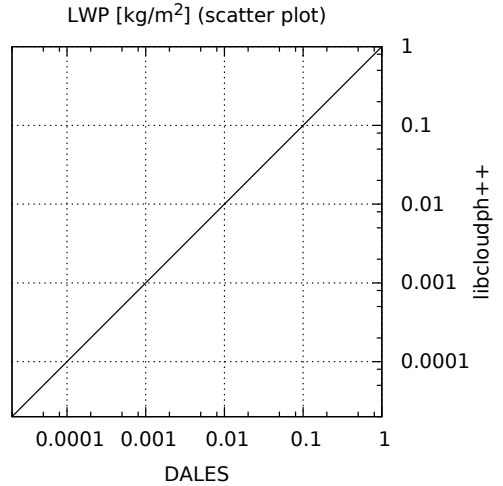
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=5m)



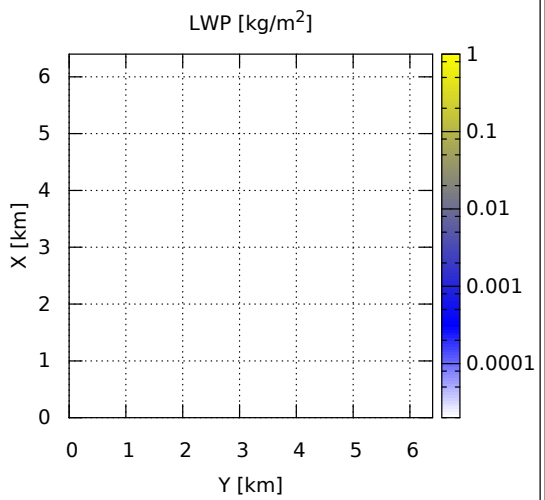
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



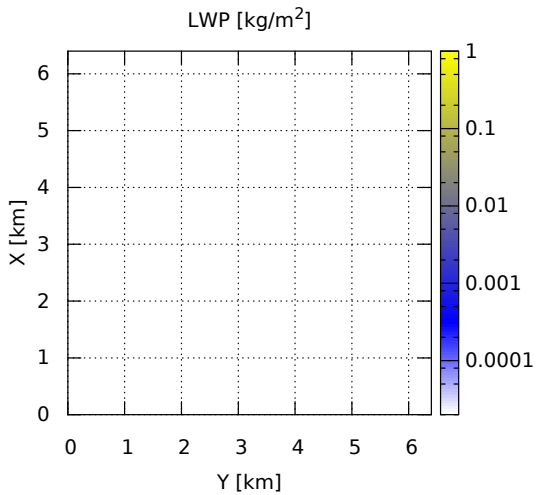


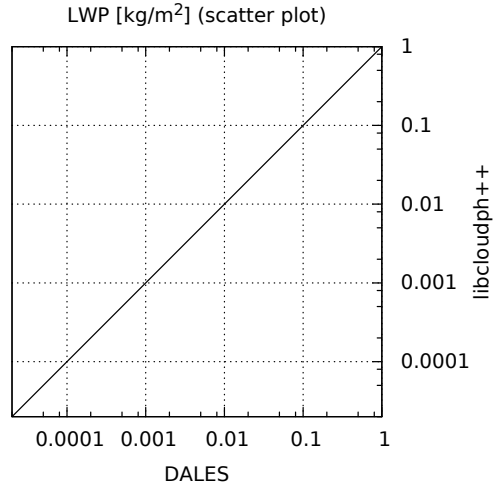
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=6m)



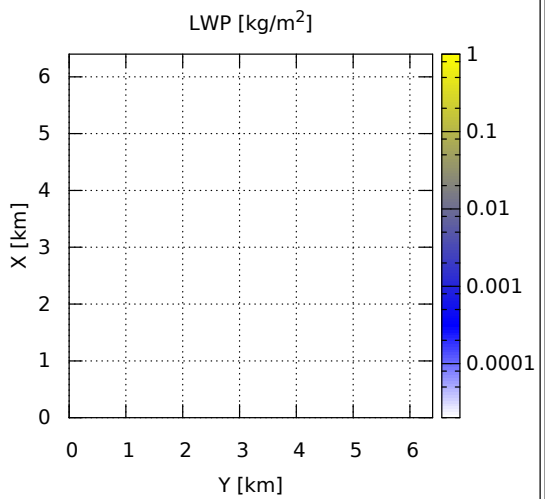
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



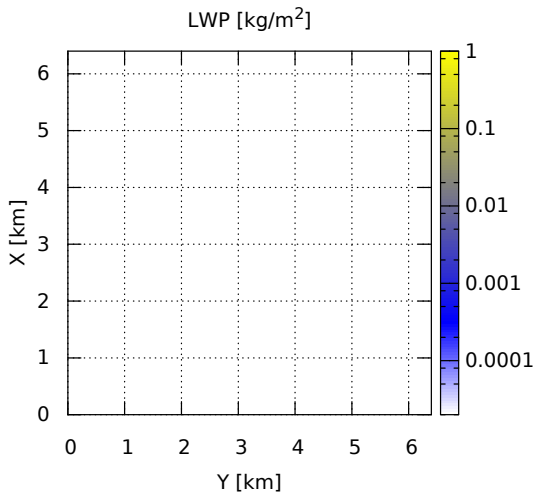


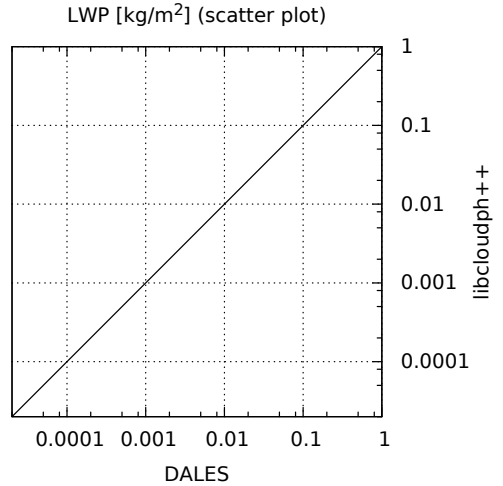
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=7m)



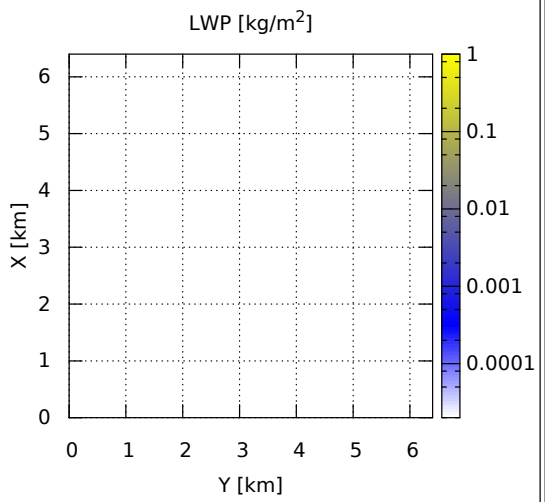
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



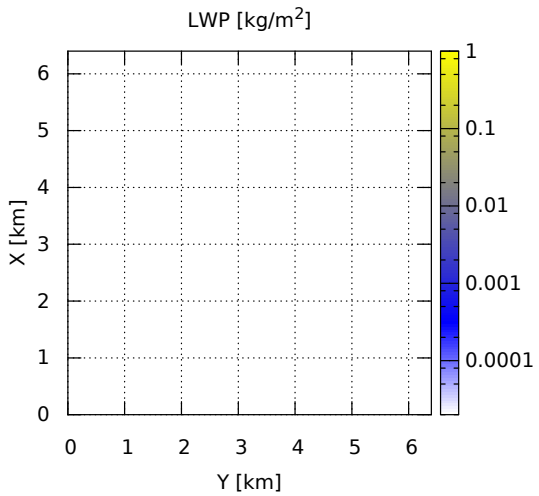


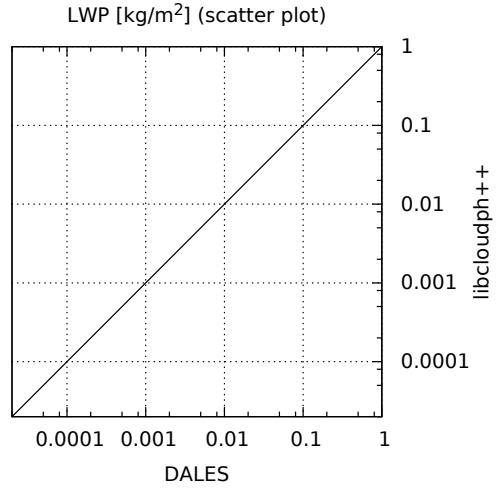
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=8m)



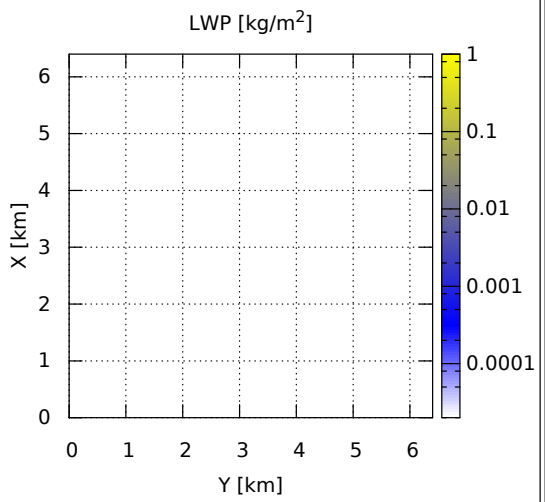
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



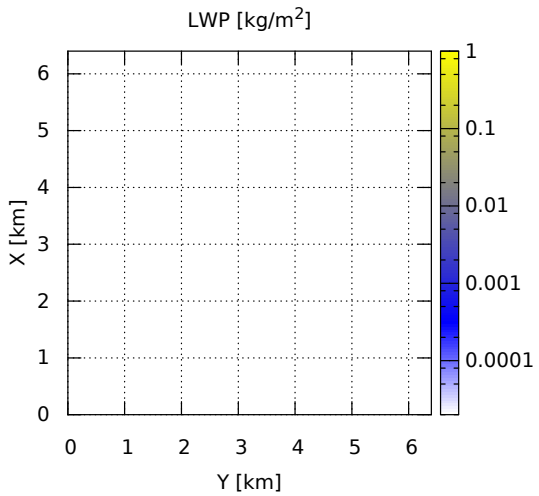


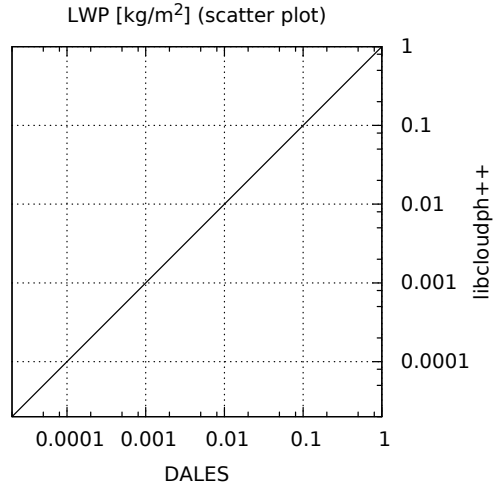
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=9m)



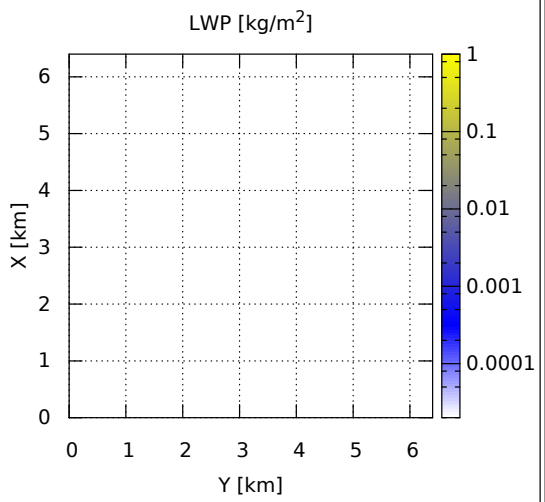
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



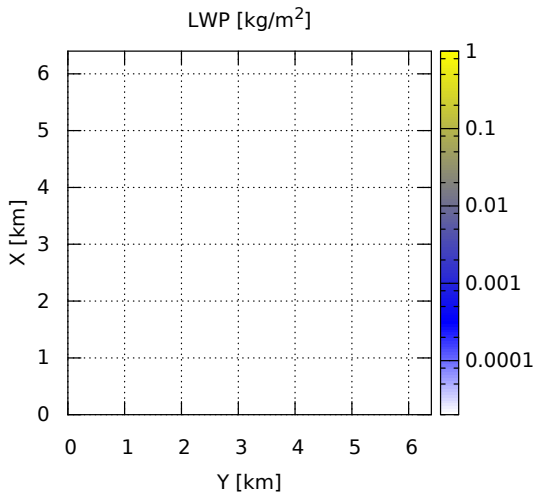


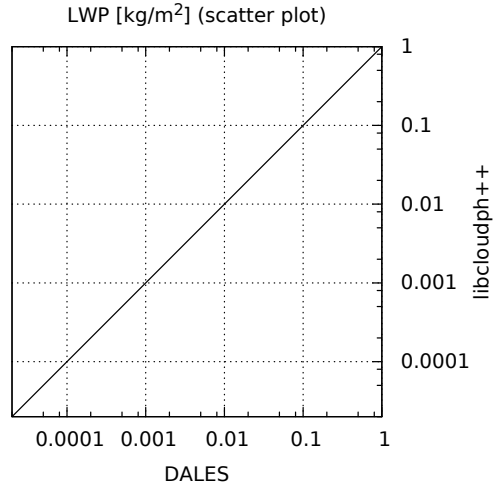
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=10m)



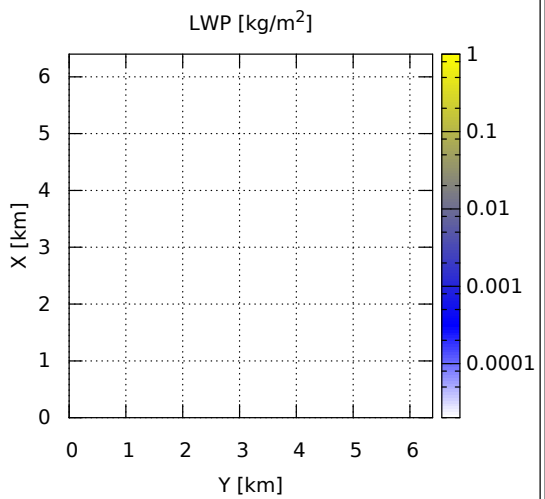
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



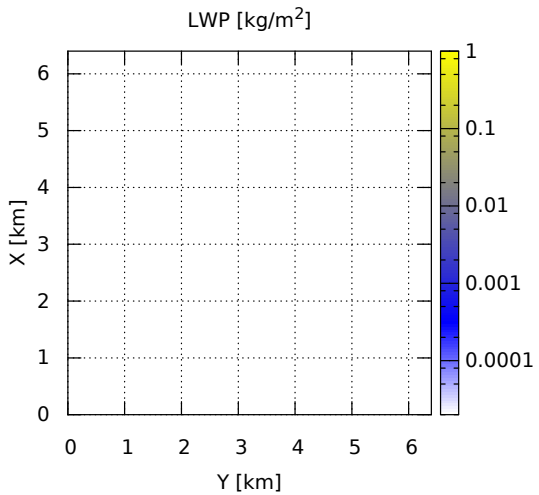


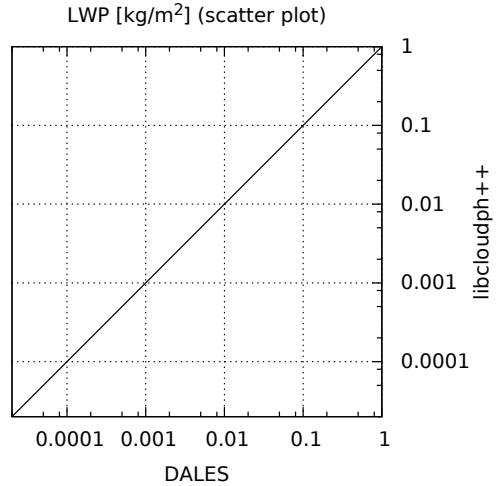
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=11m)



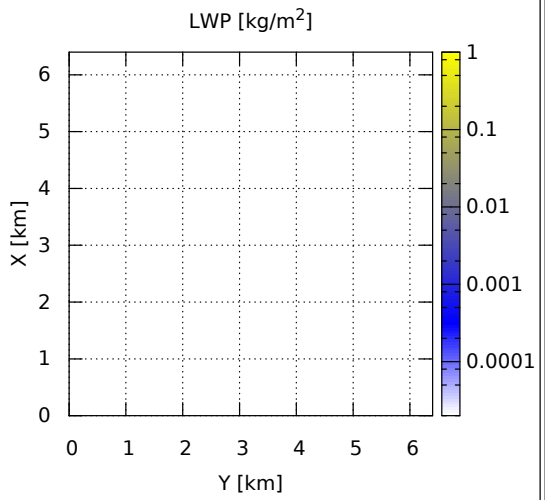
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



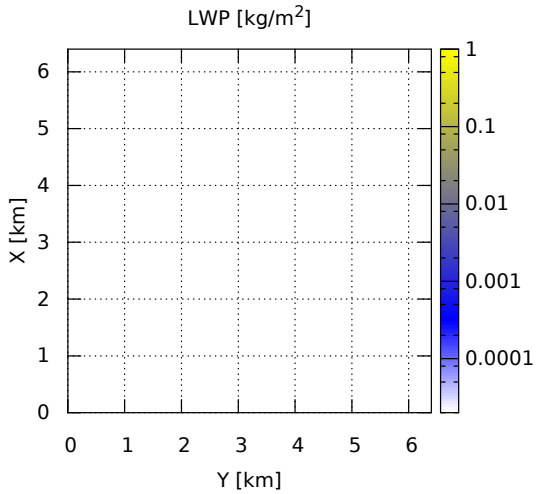


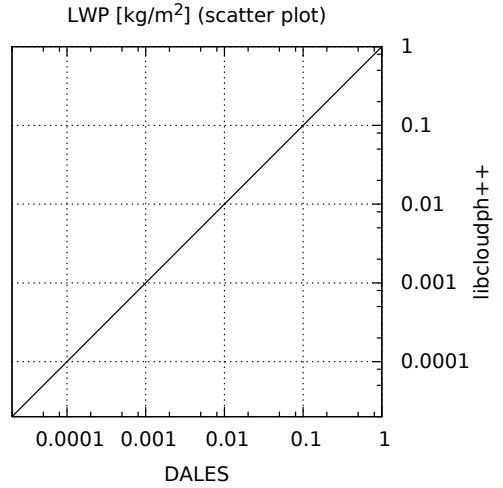
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=12m)



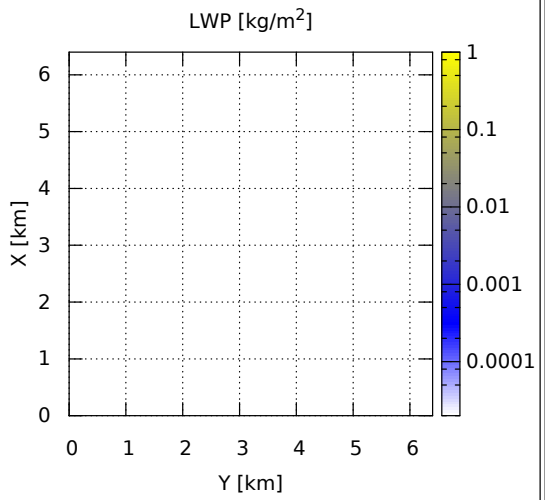
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



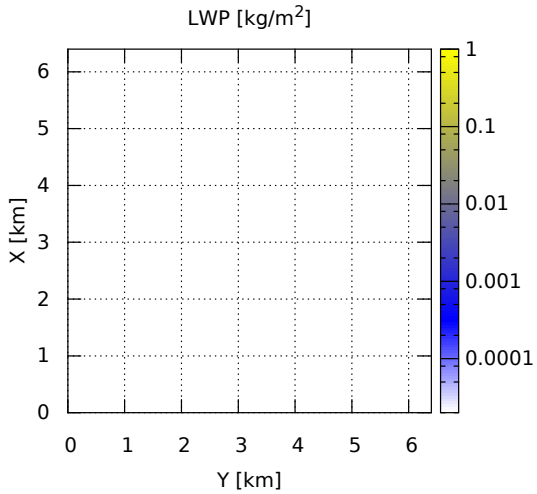


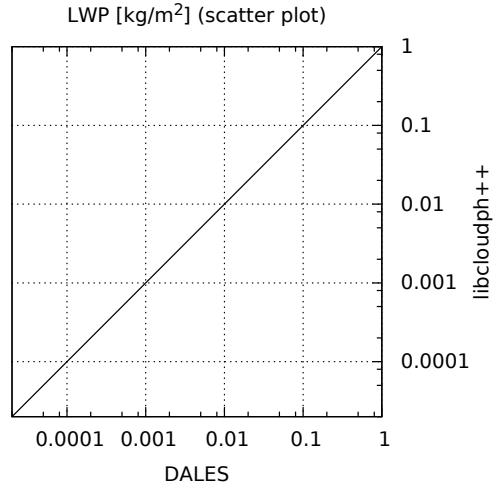
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=13m)



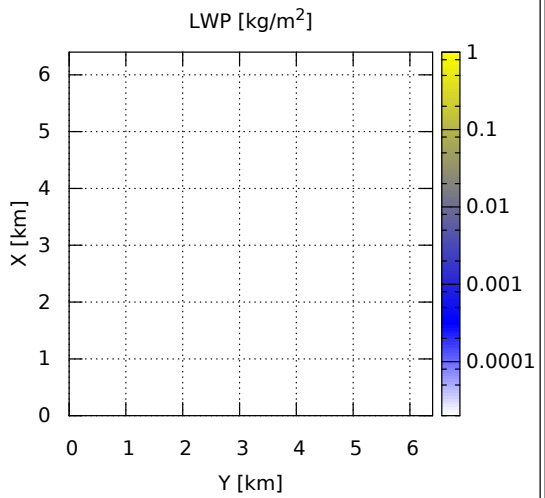
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



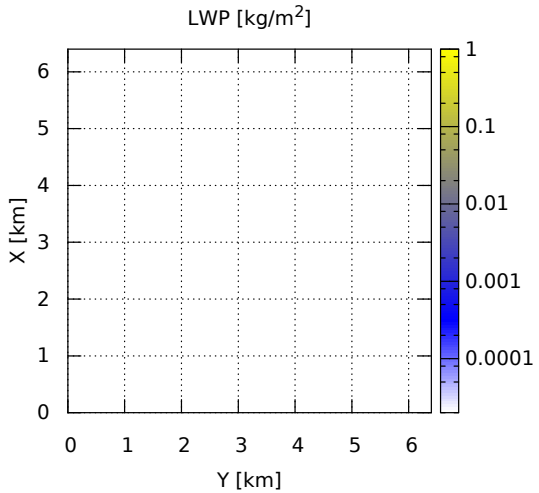


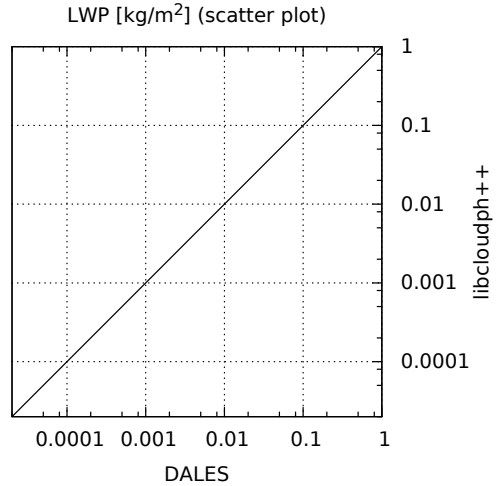
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=14m)



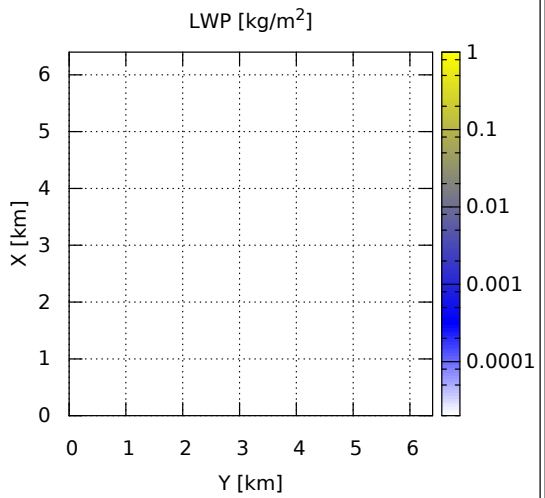
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



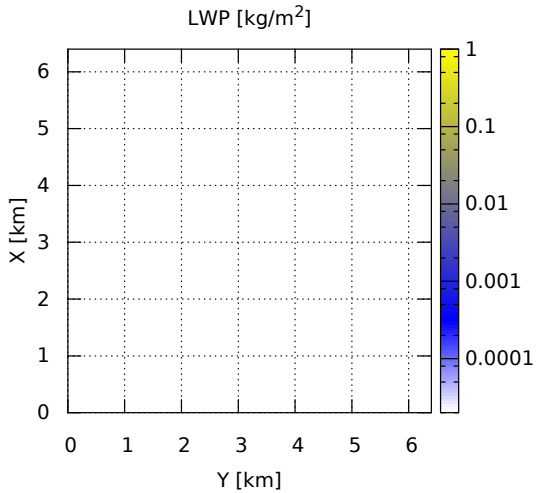


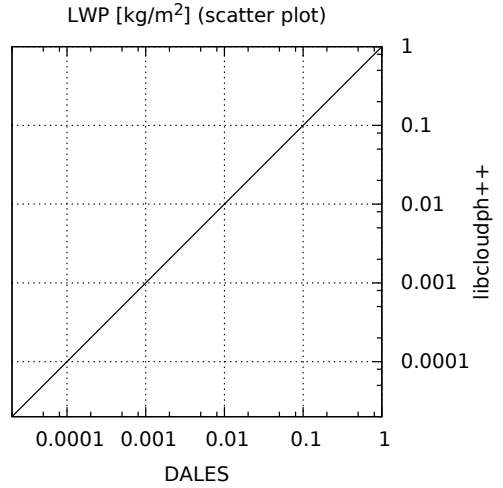
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=15m)



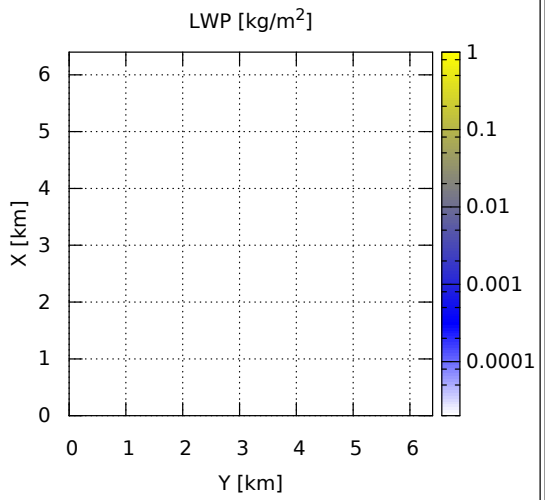
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



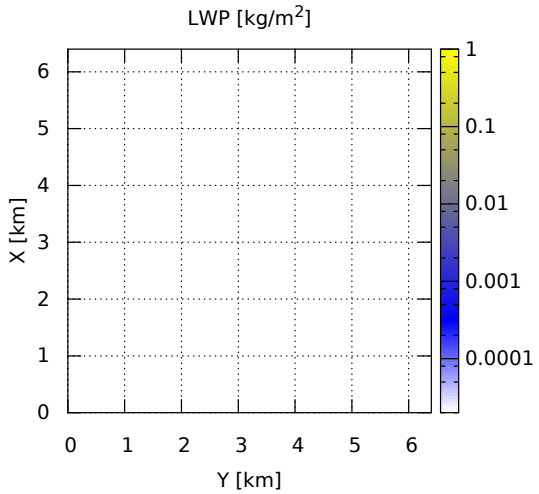


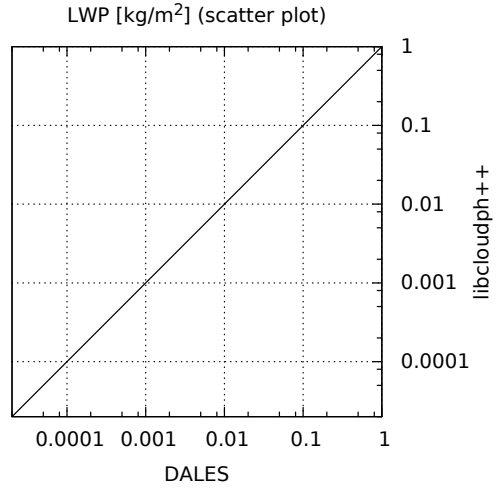
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=16m)



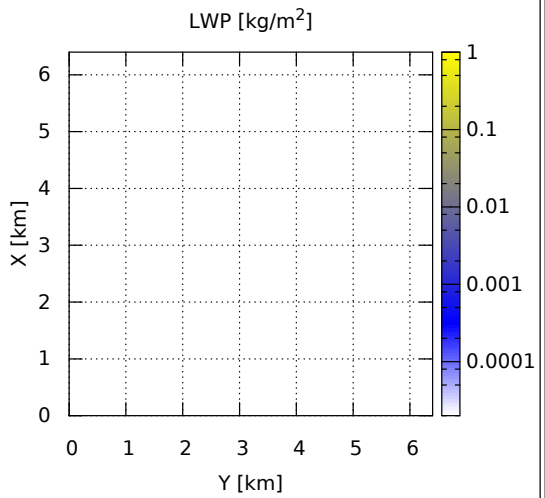
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



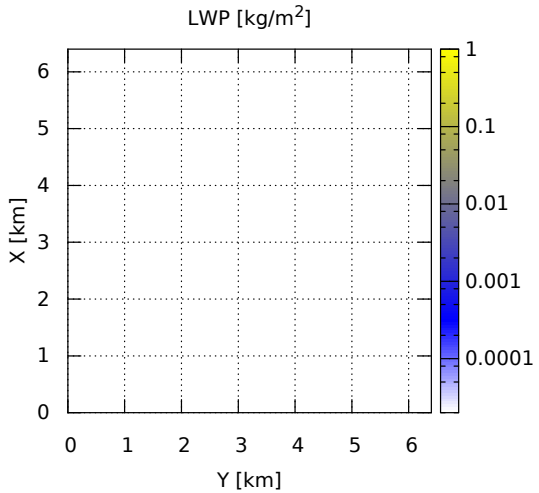


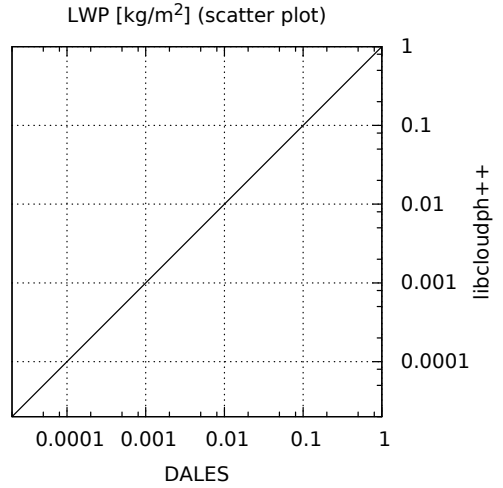
example: DALES/libclouph++ coupling

DALES (BOMEX, bulk μ-physics, t=17m)

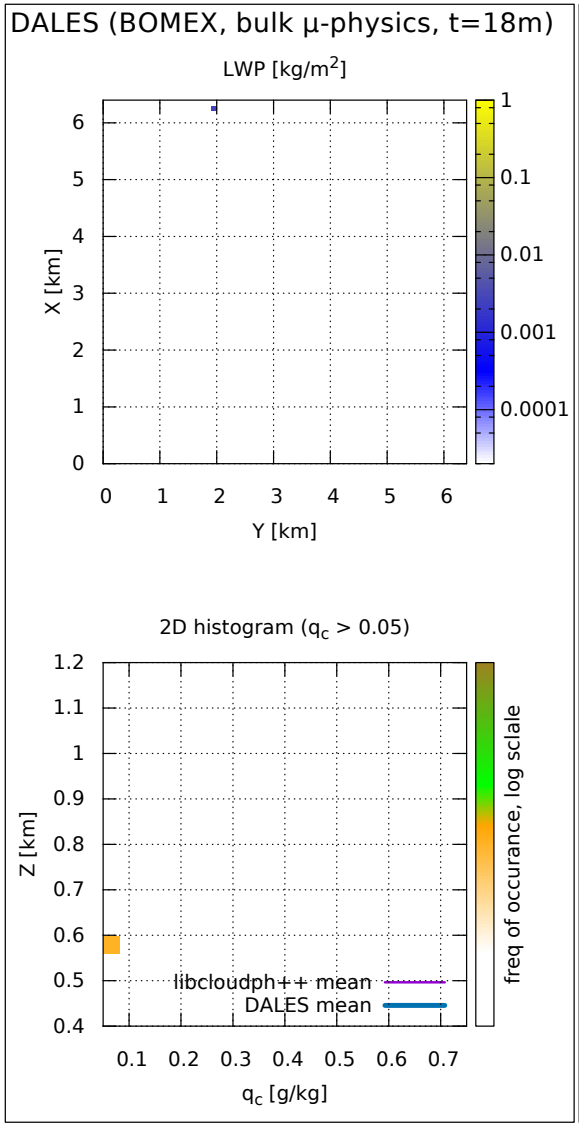


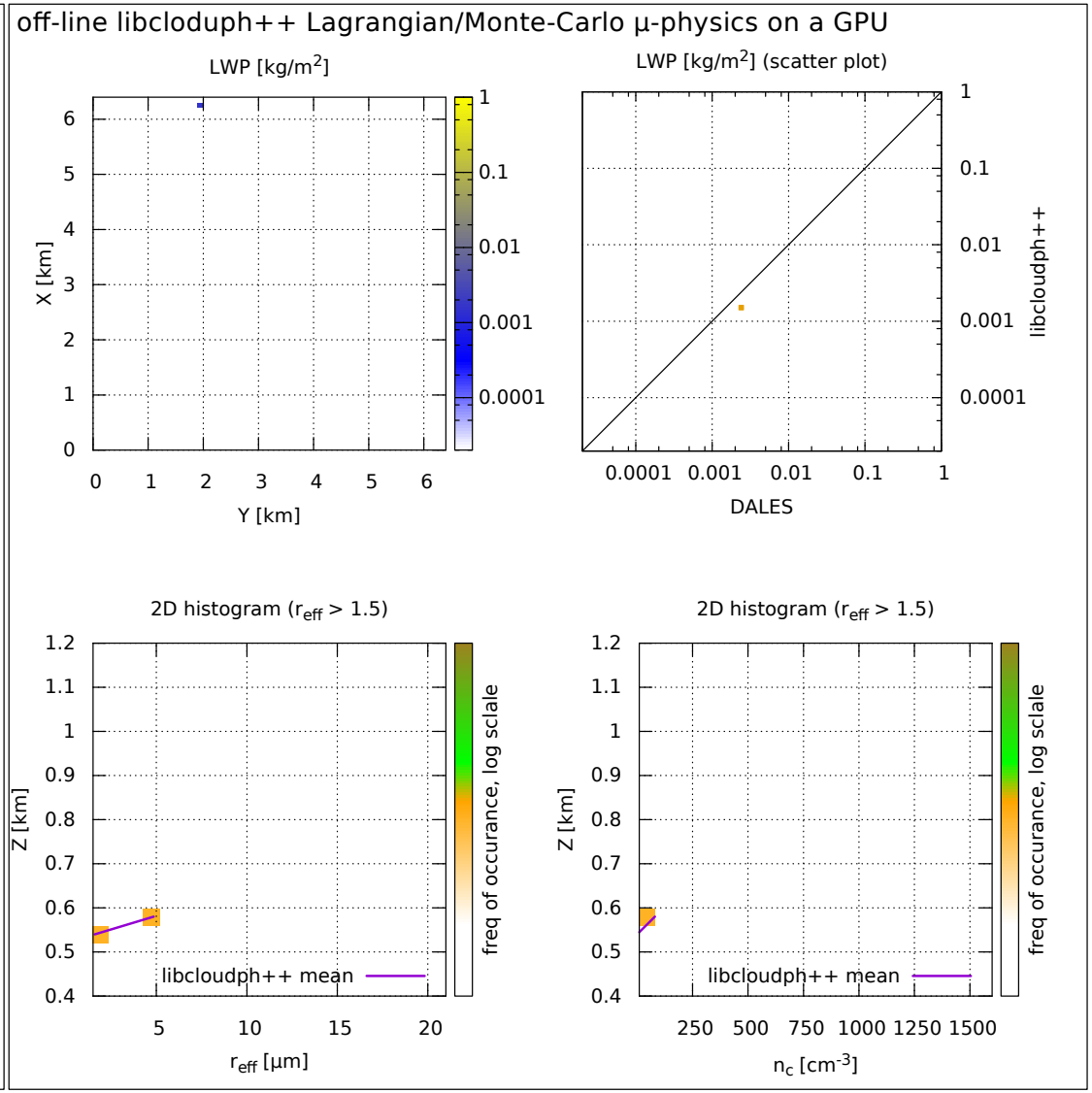
off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU



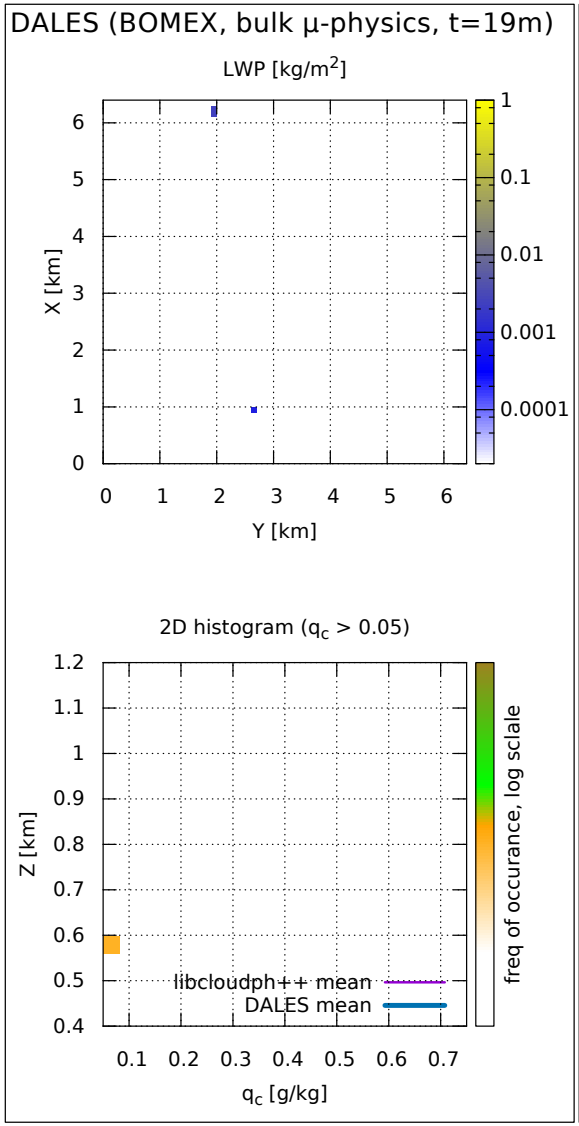


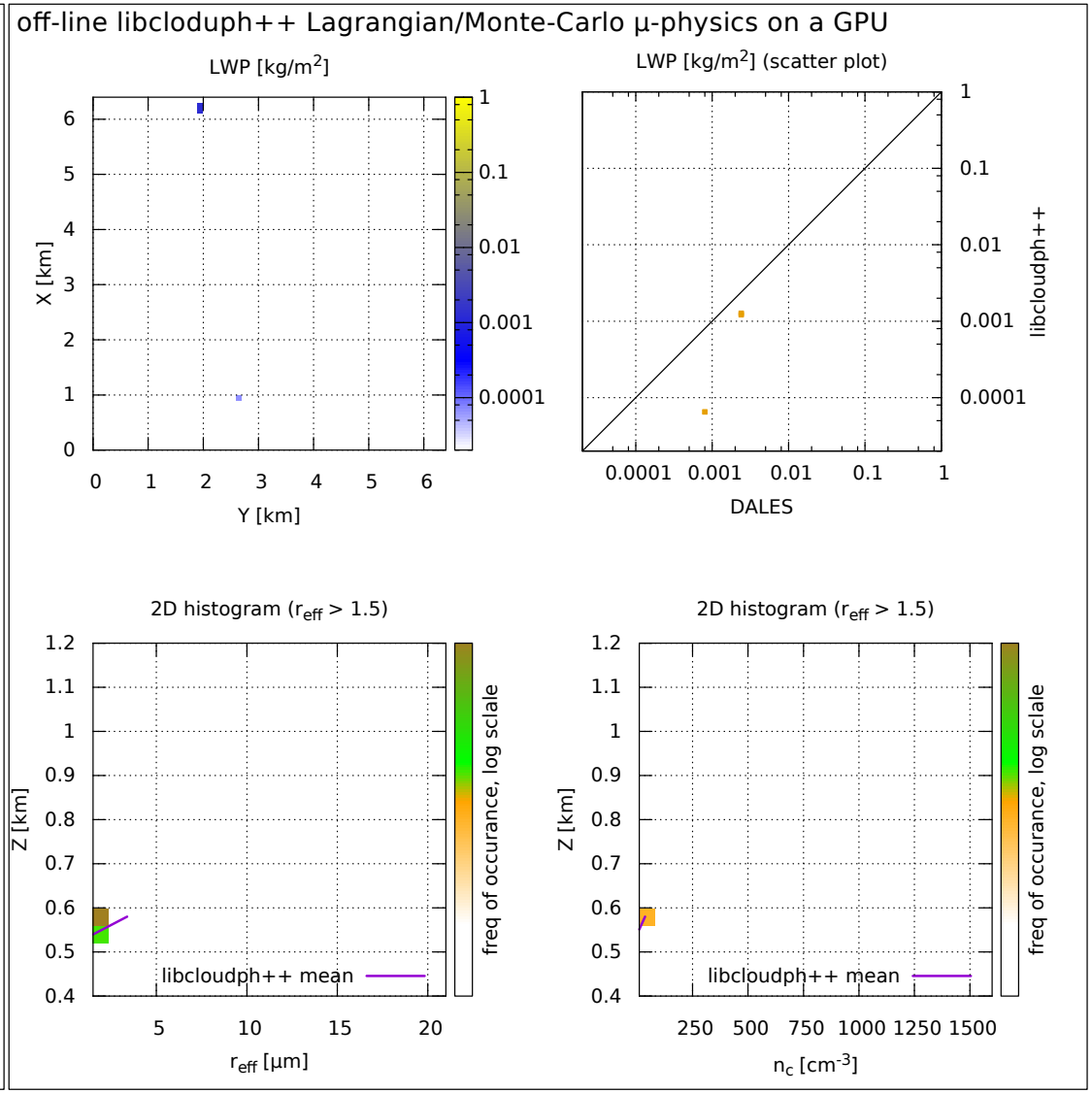
example: DALES/libclouph++ coupling



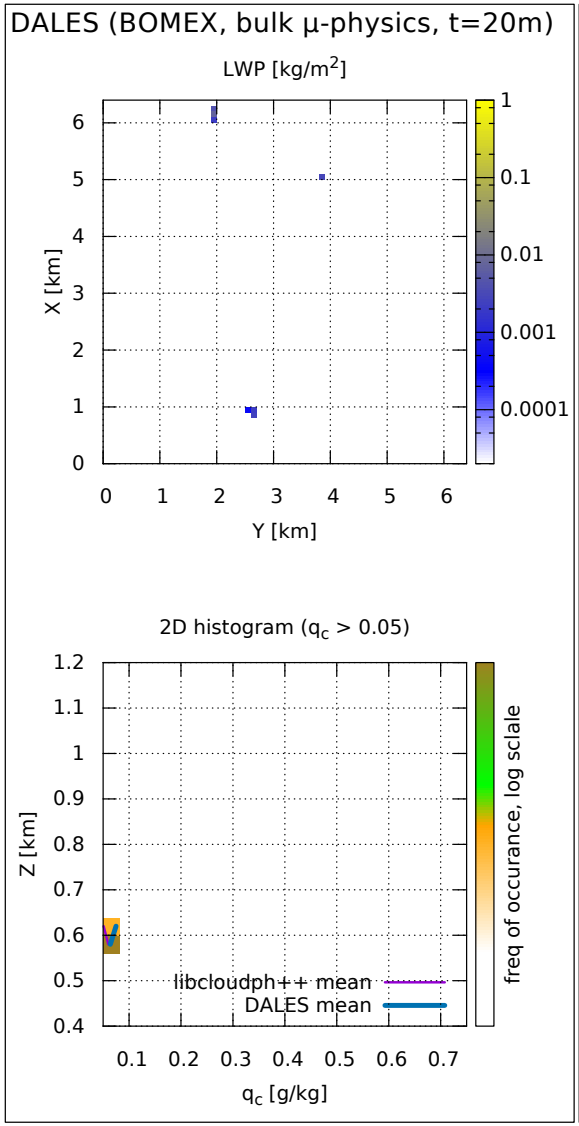


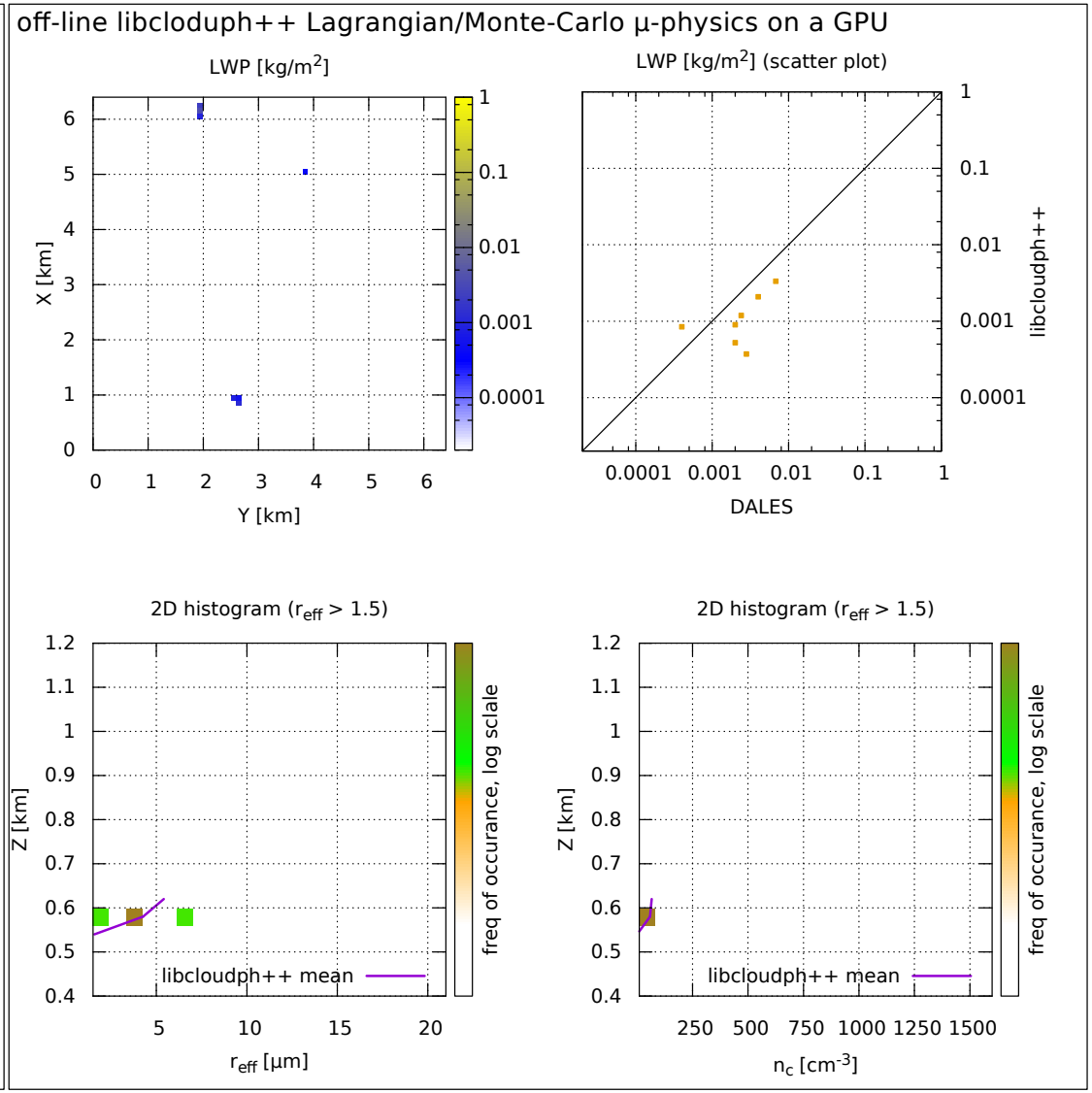
example: DALES/libclouph++ coupling



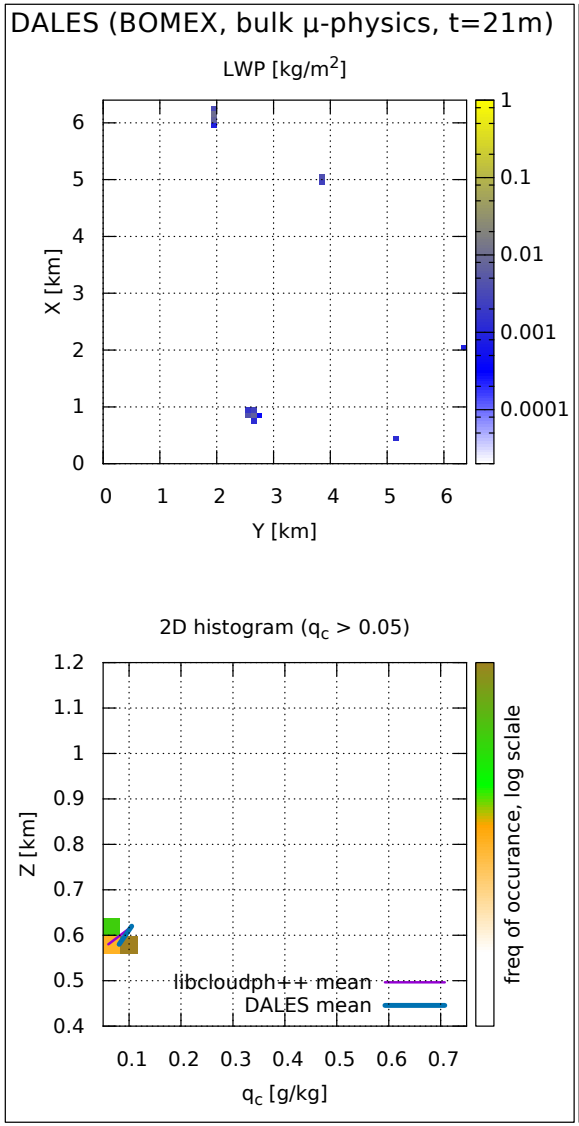


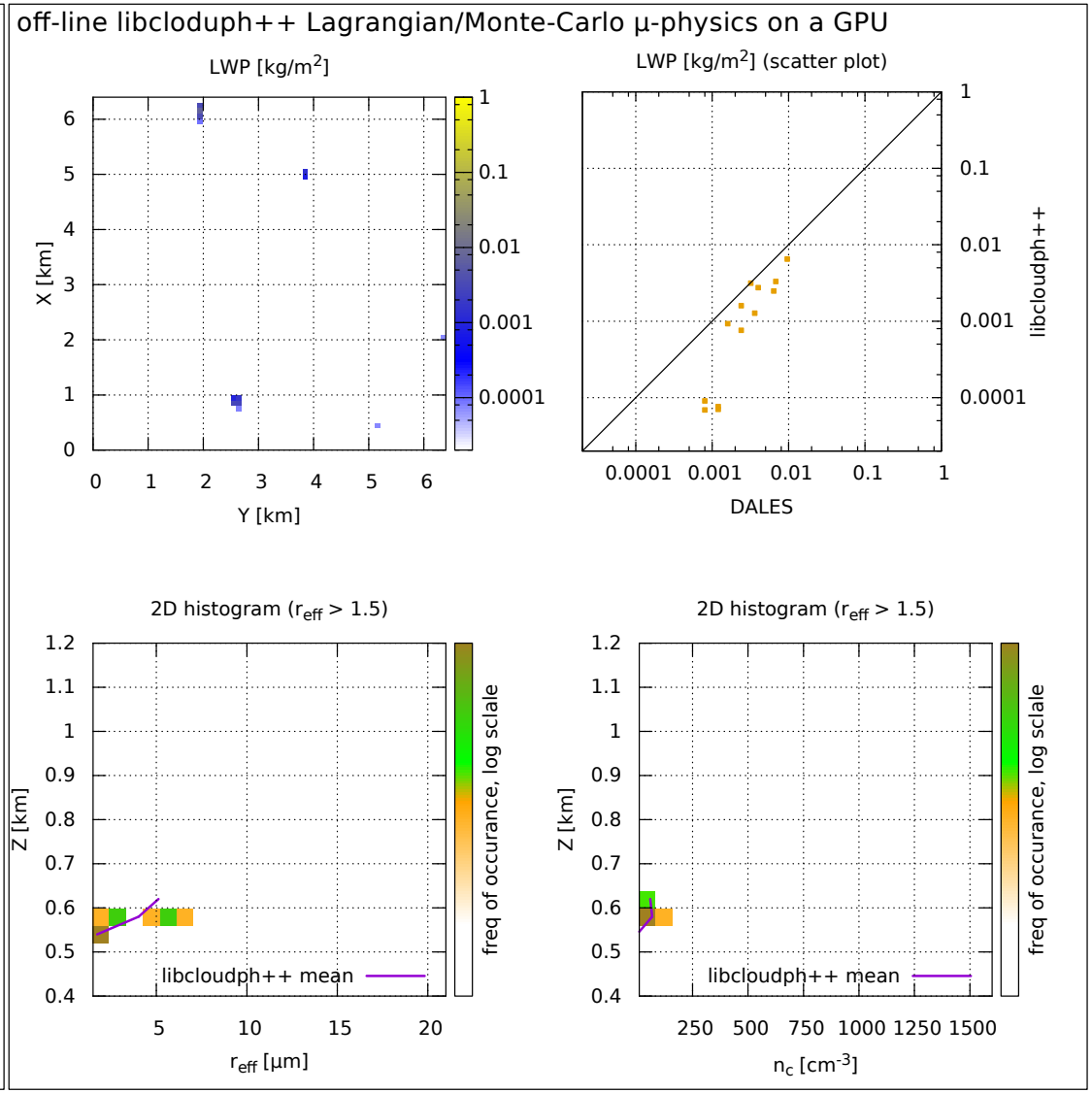
example: DALES/libclouph++ coupling



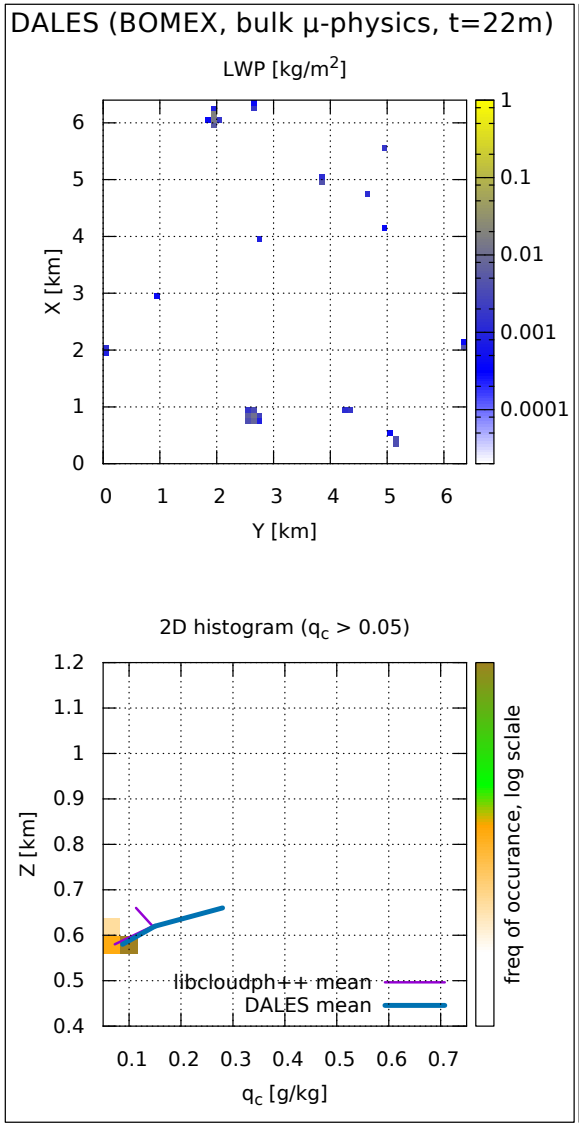


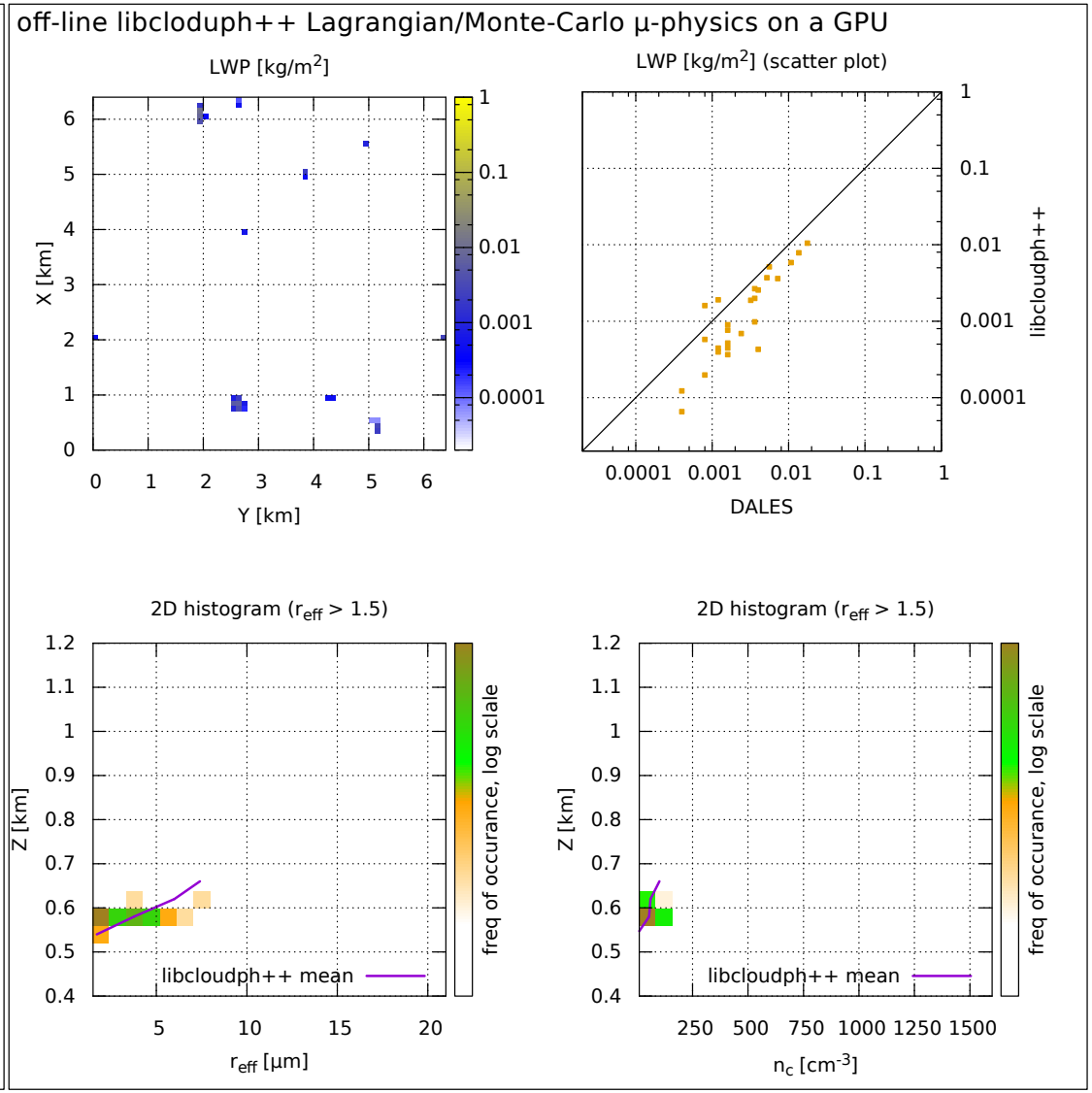
example: DALES/libclouph++ coupling



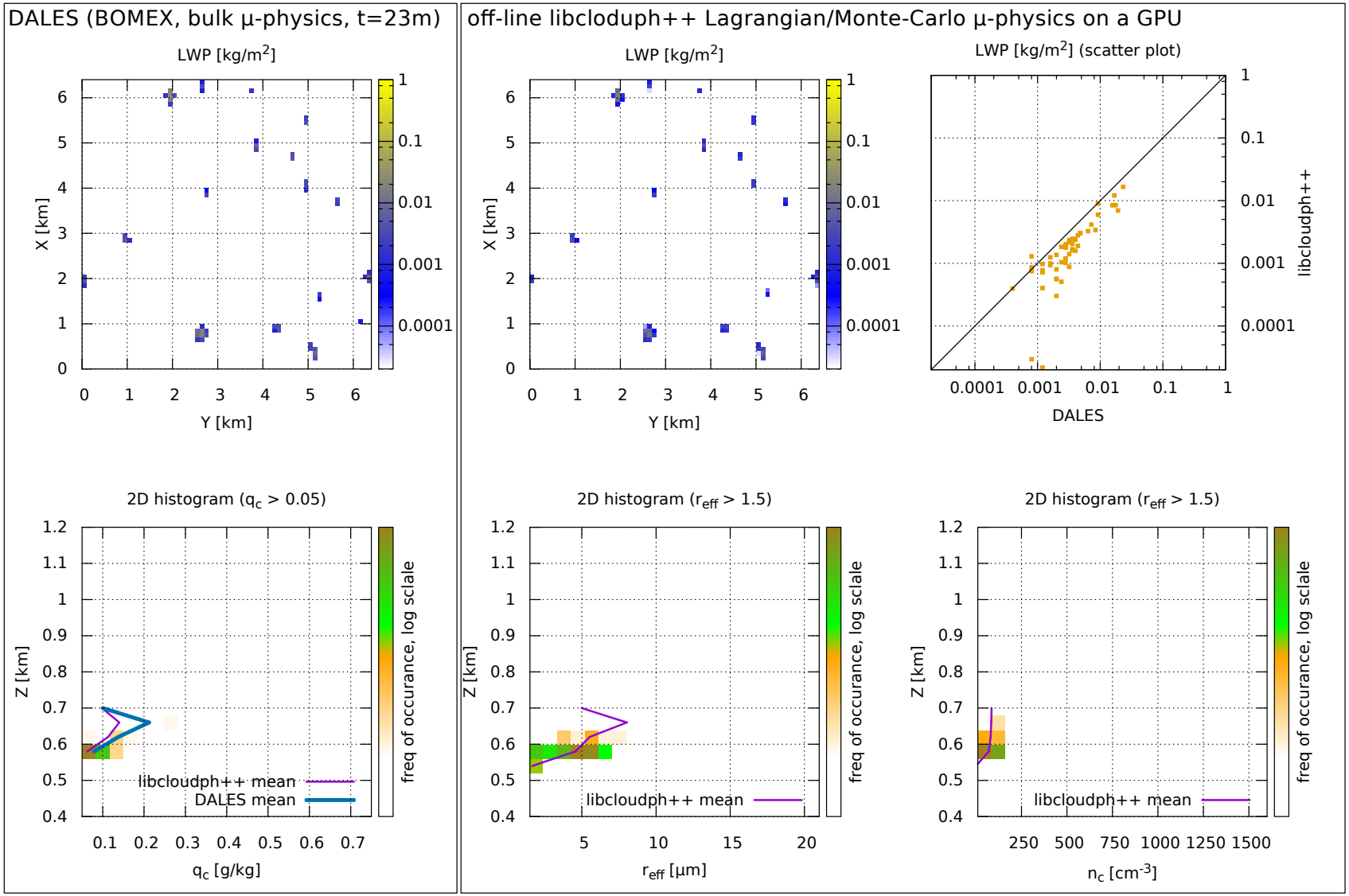


example: DALES/libclouph++ coupling

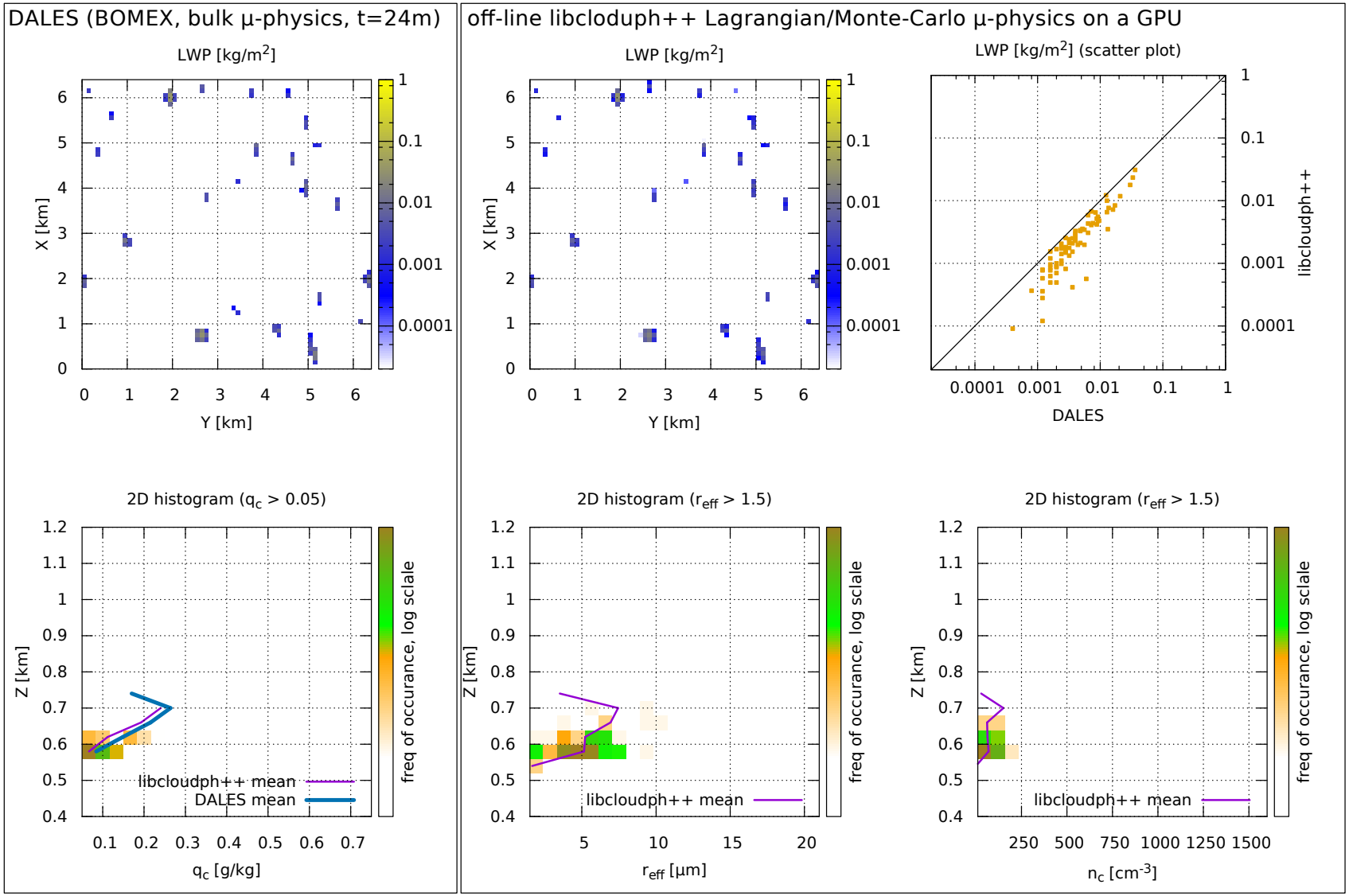




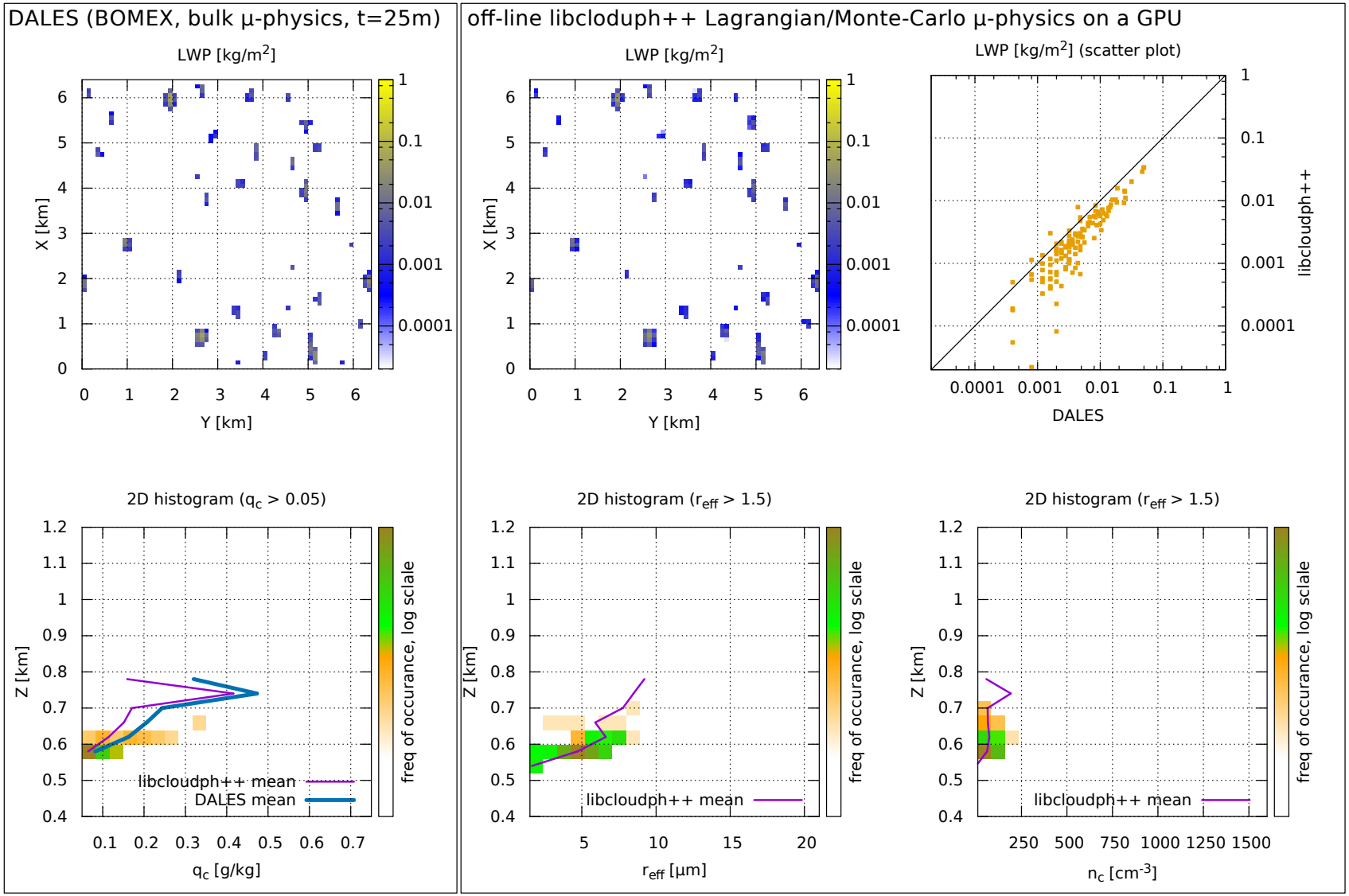
example: DALES/libclouph++ coupling



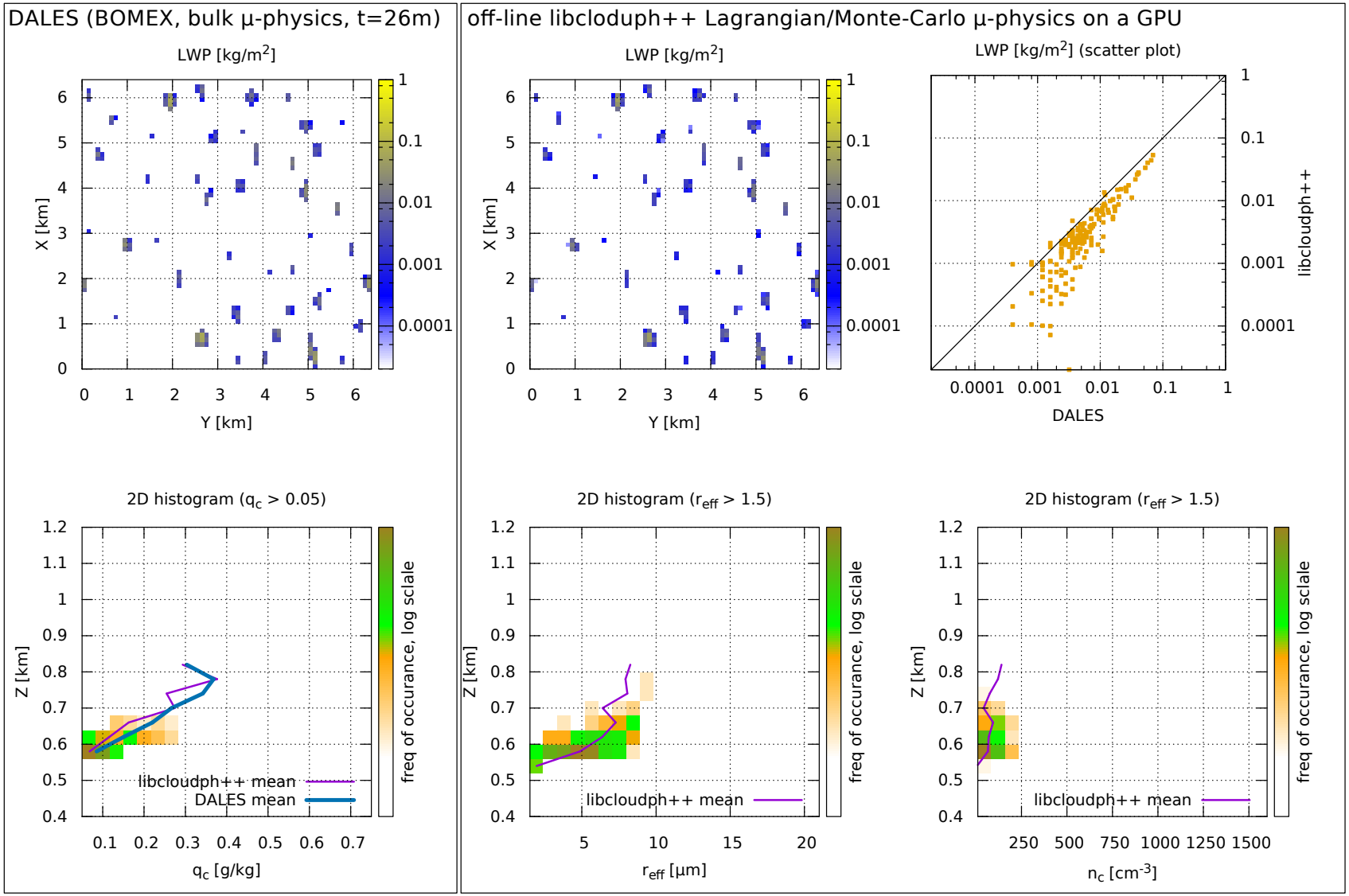
example: DALES/libclouph++ coupling



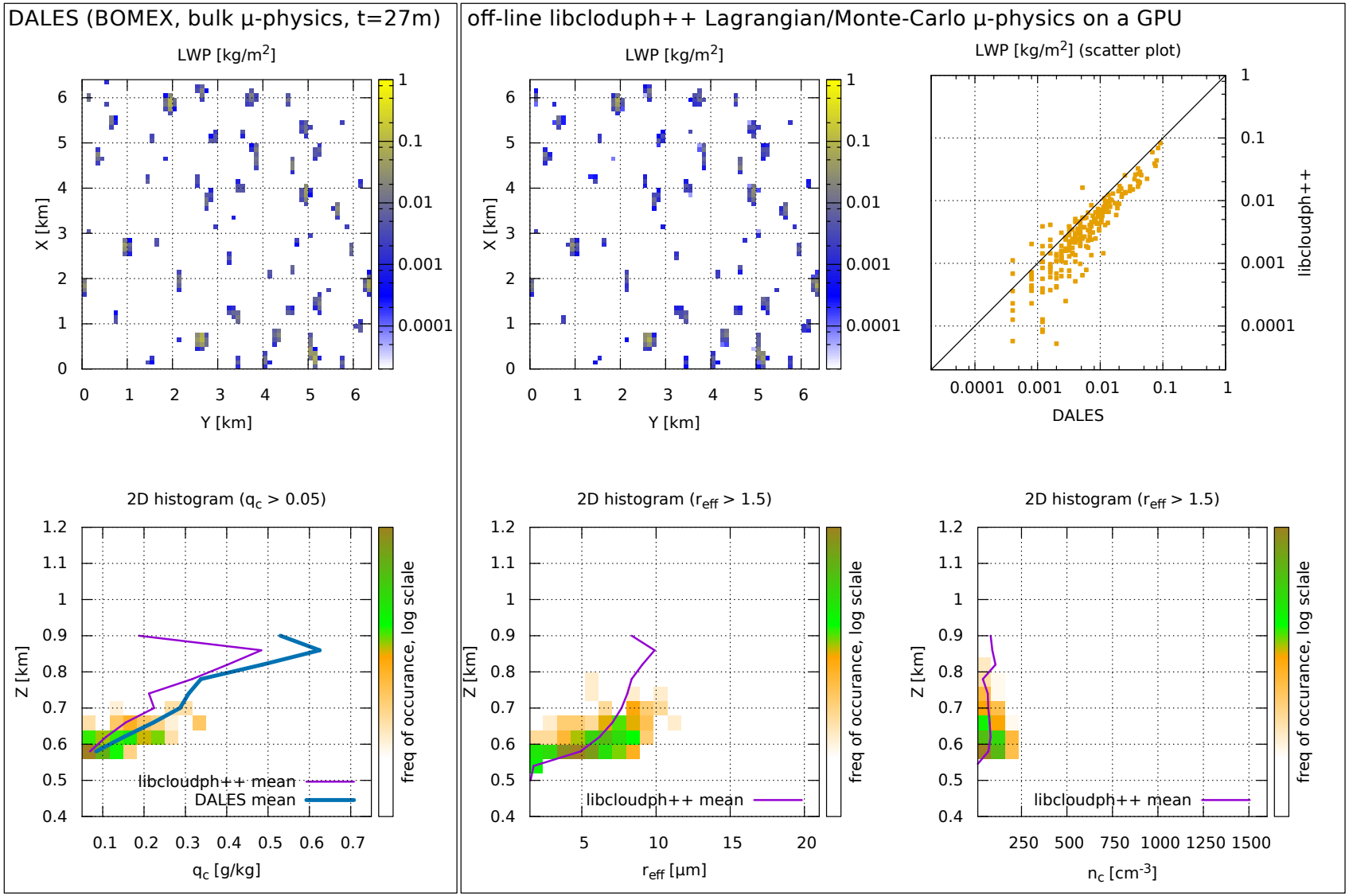
example: DALES/libclouph++ coupling



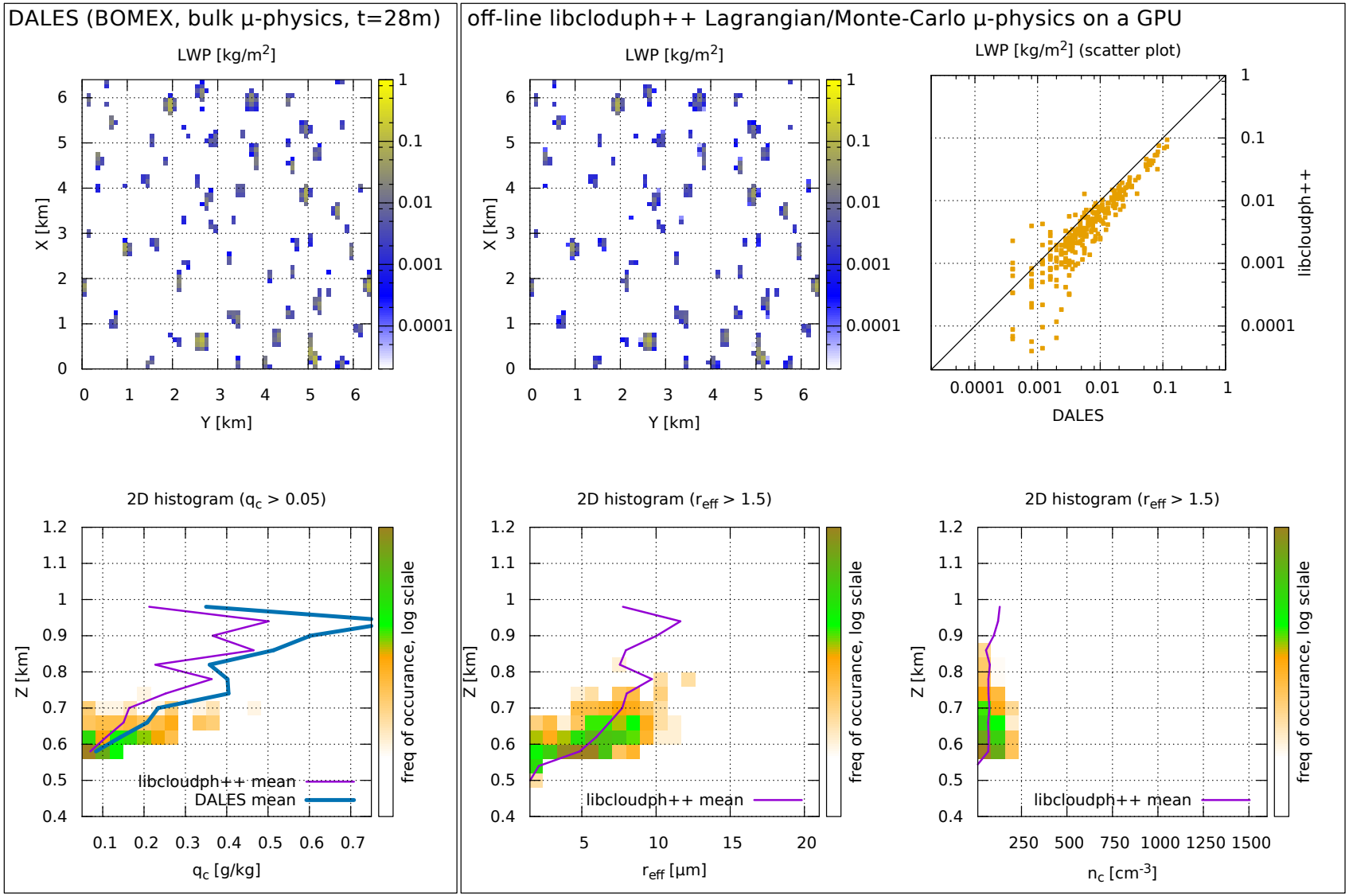
example: DALES/libclouph++ coupling



example: DALES/libclouph++ coupling

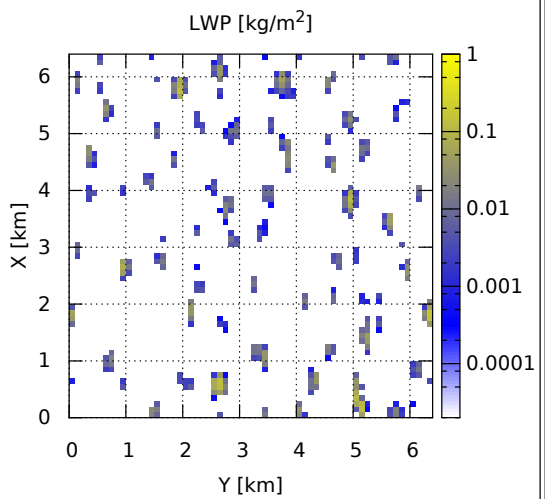


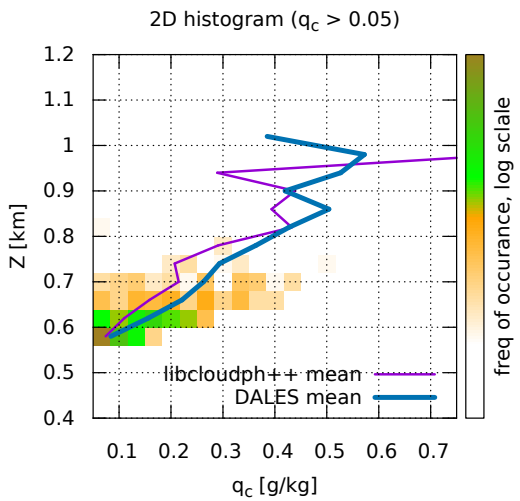
example: DALES/libclouph++ coupling



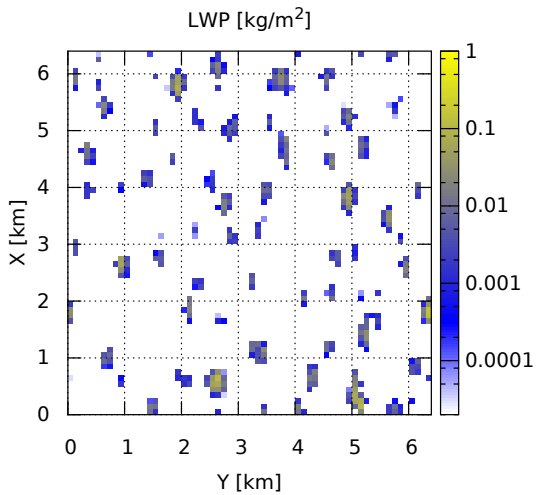
example: DALES/libclouph++ coupling

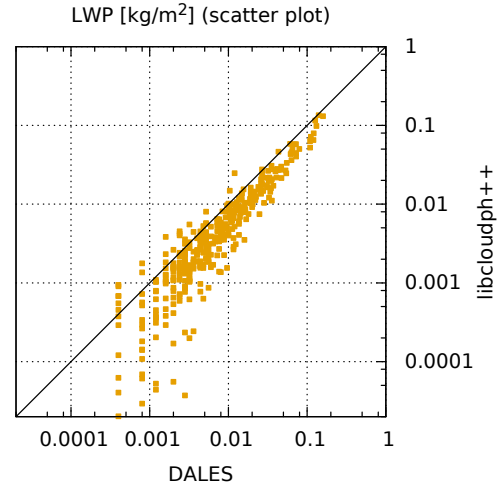
DALES (BOMEX, bulk μ-physics, t=29m)

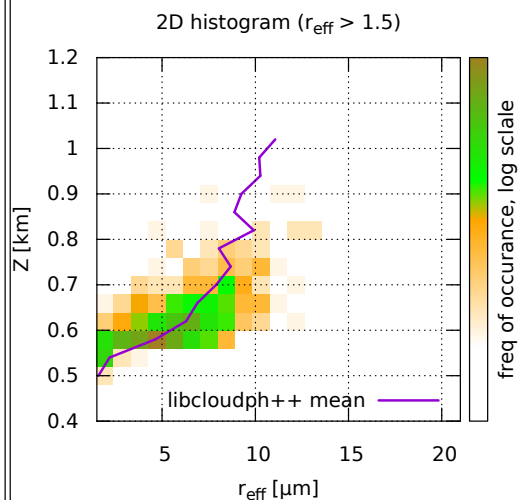


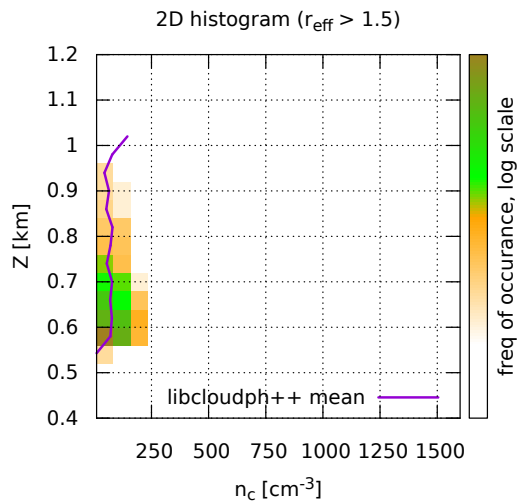


off-line libcloduph++ Lagrangian/Monte-Carlo μ-physics on a GPU

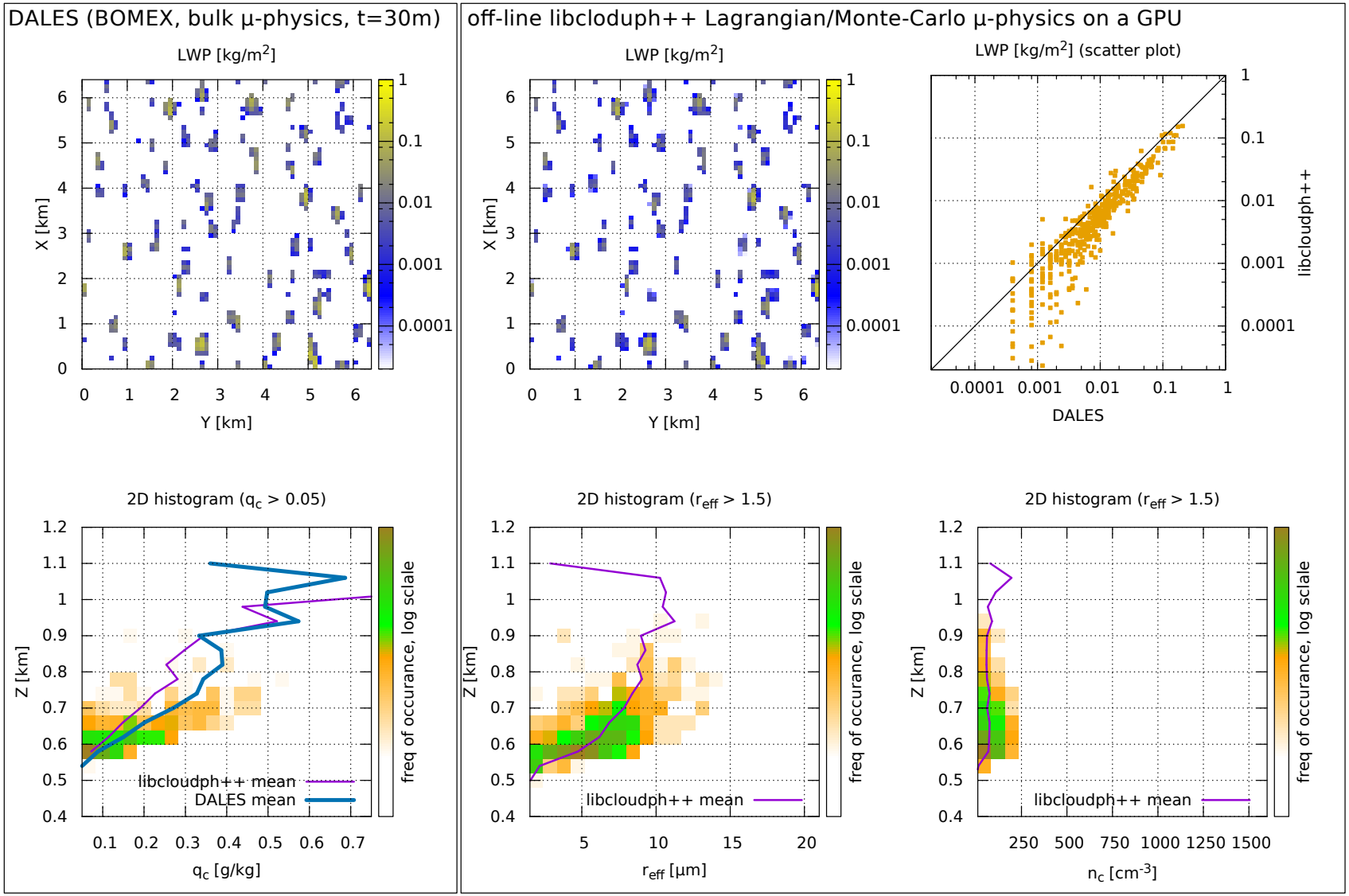




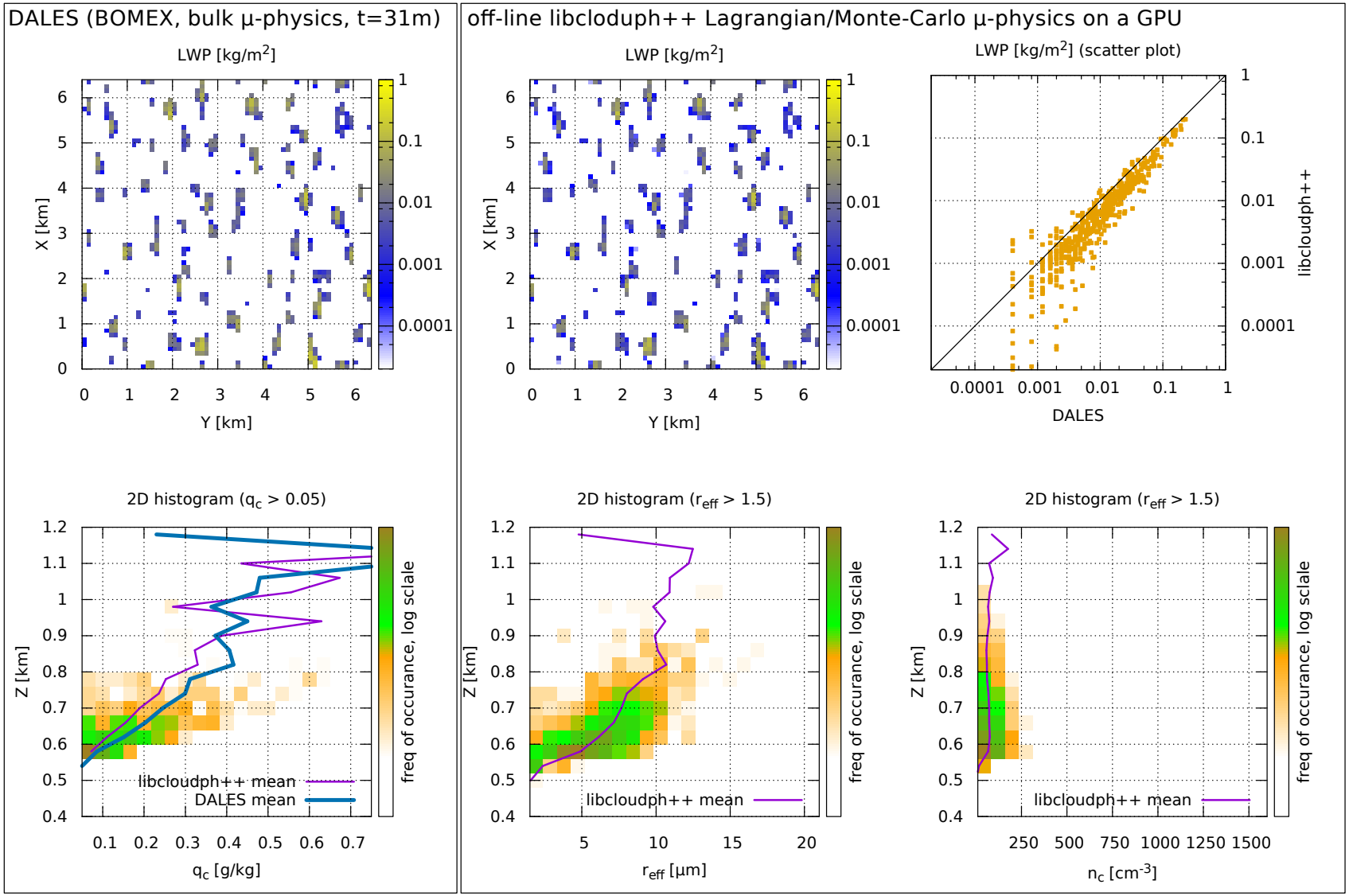




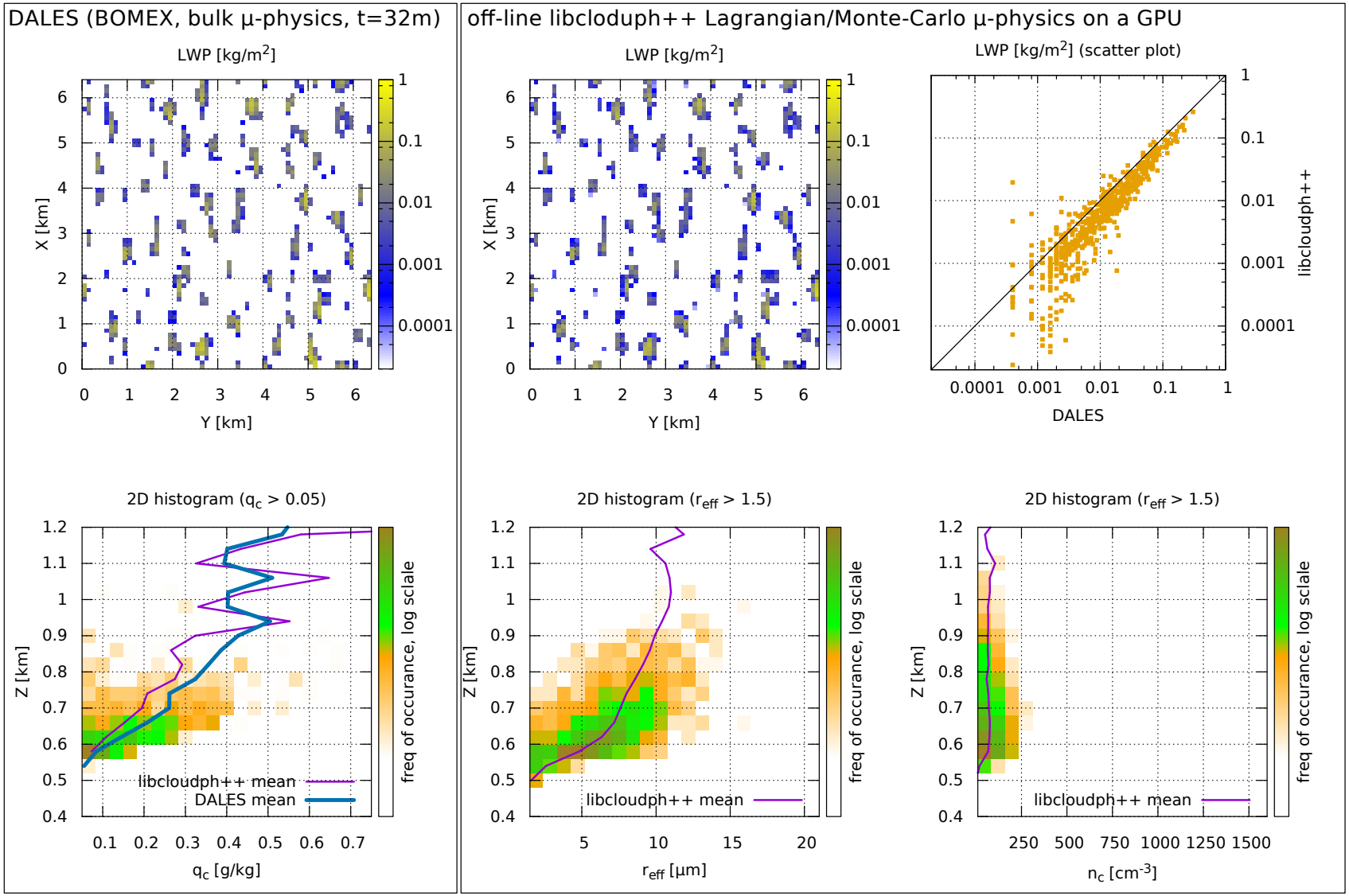
example: DALES/libclouph++ coupling



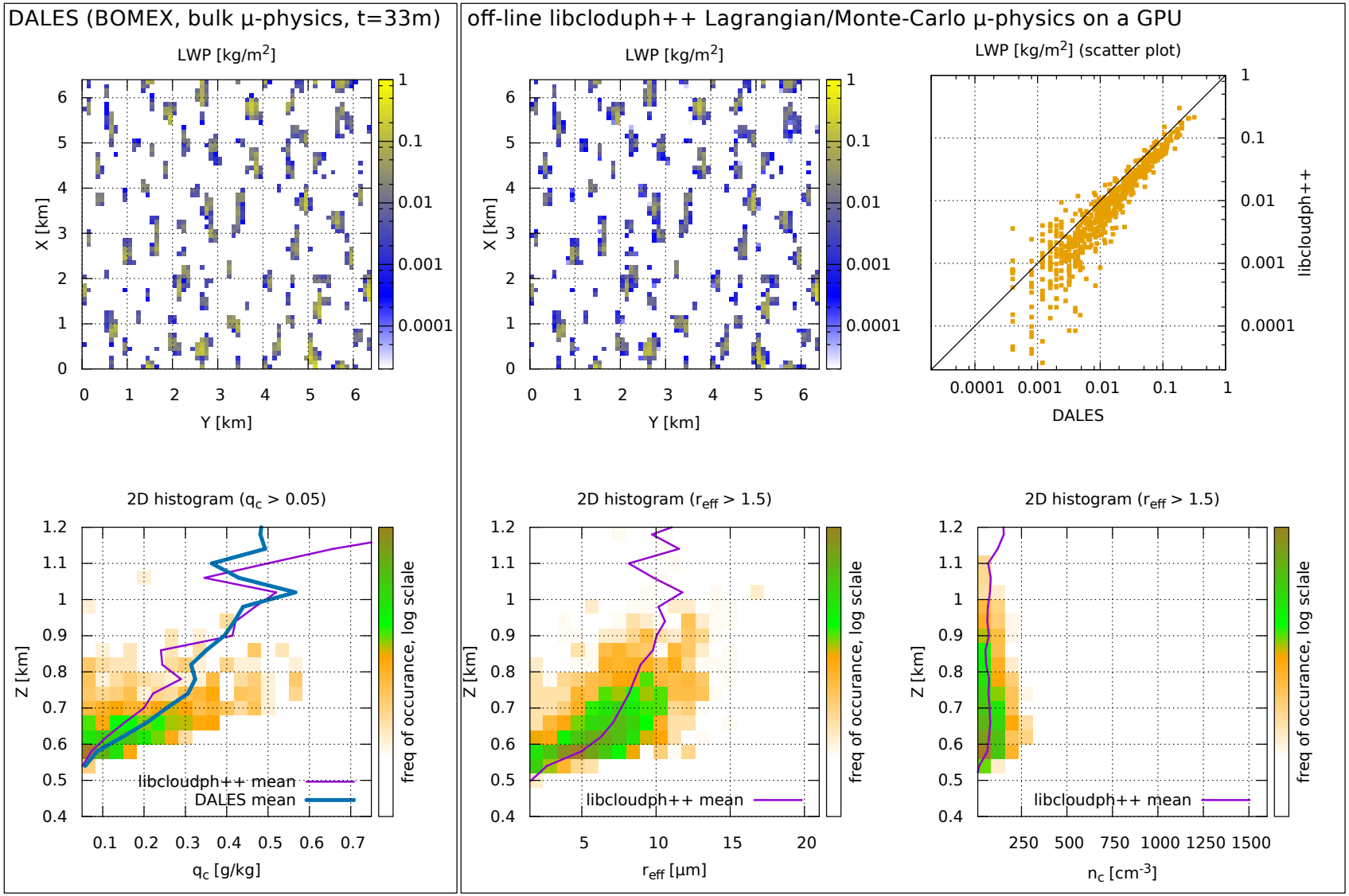
example: DALES/libclouph++ coupling



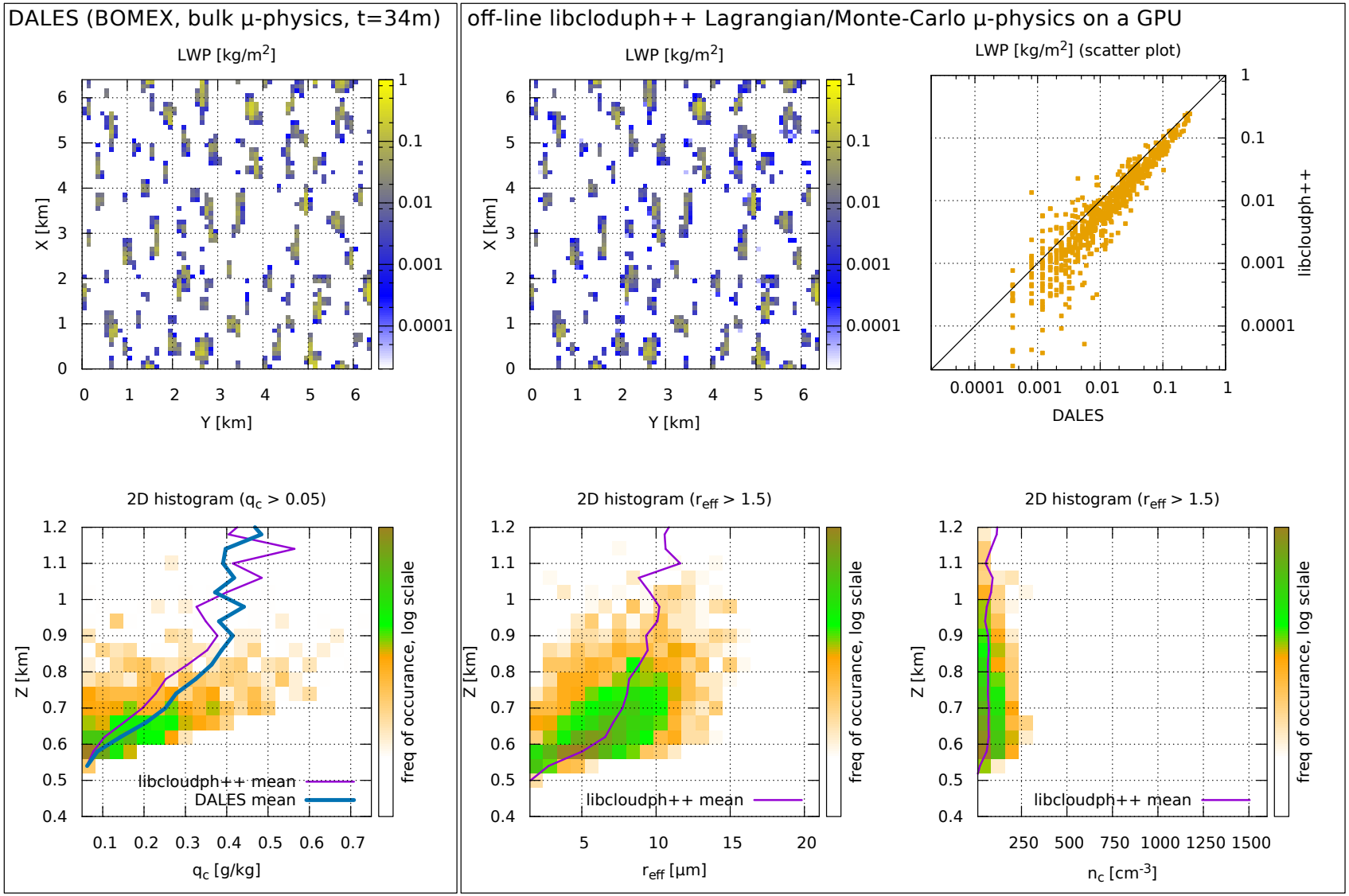
example: DALES/libclouph++ coupling



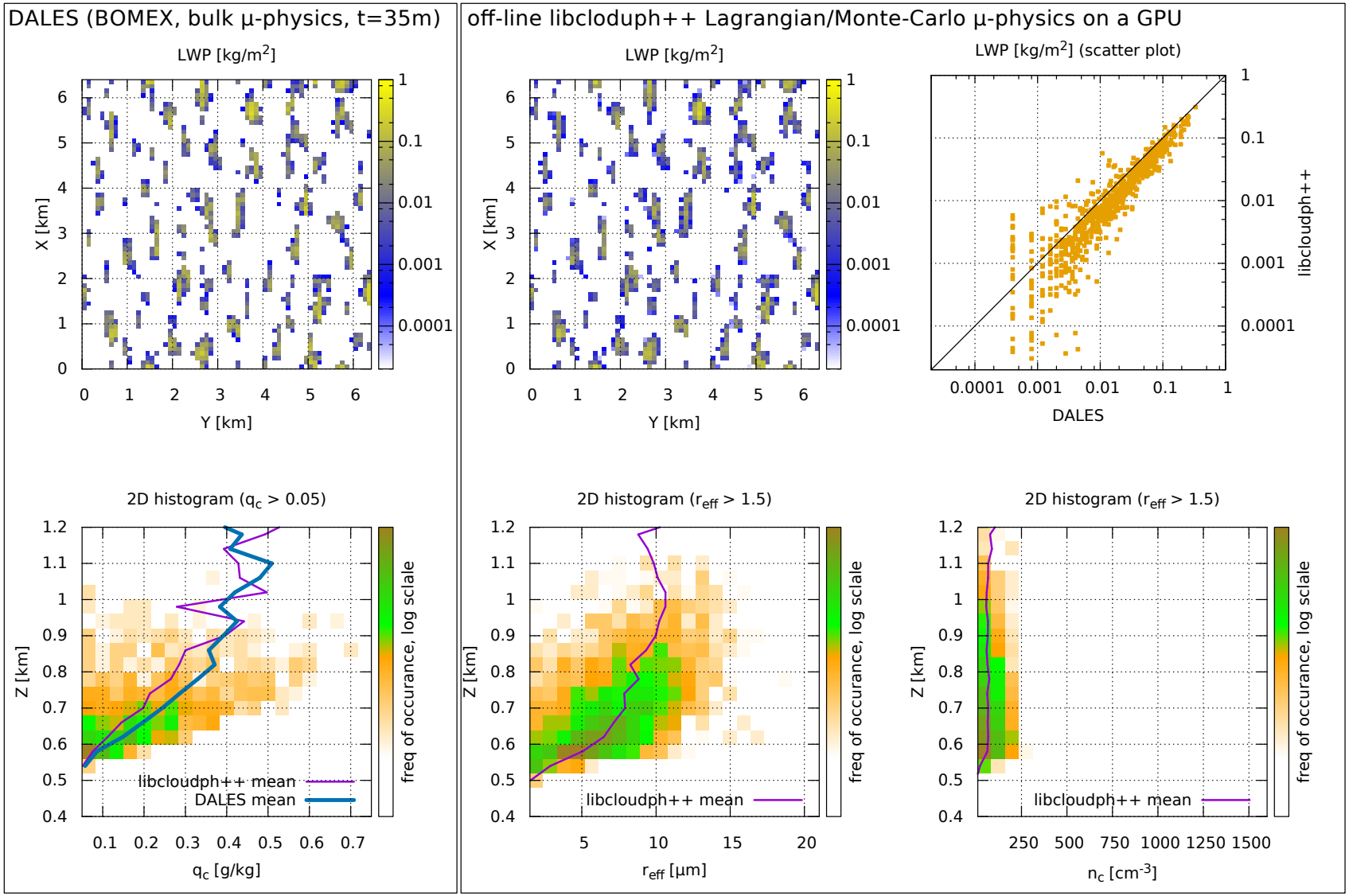
example: DALES/libclouph++ coupling



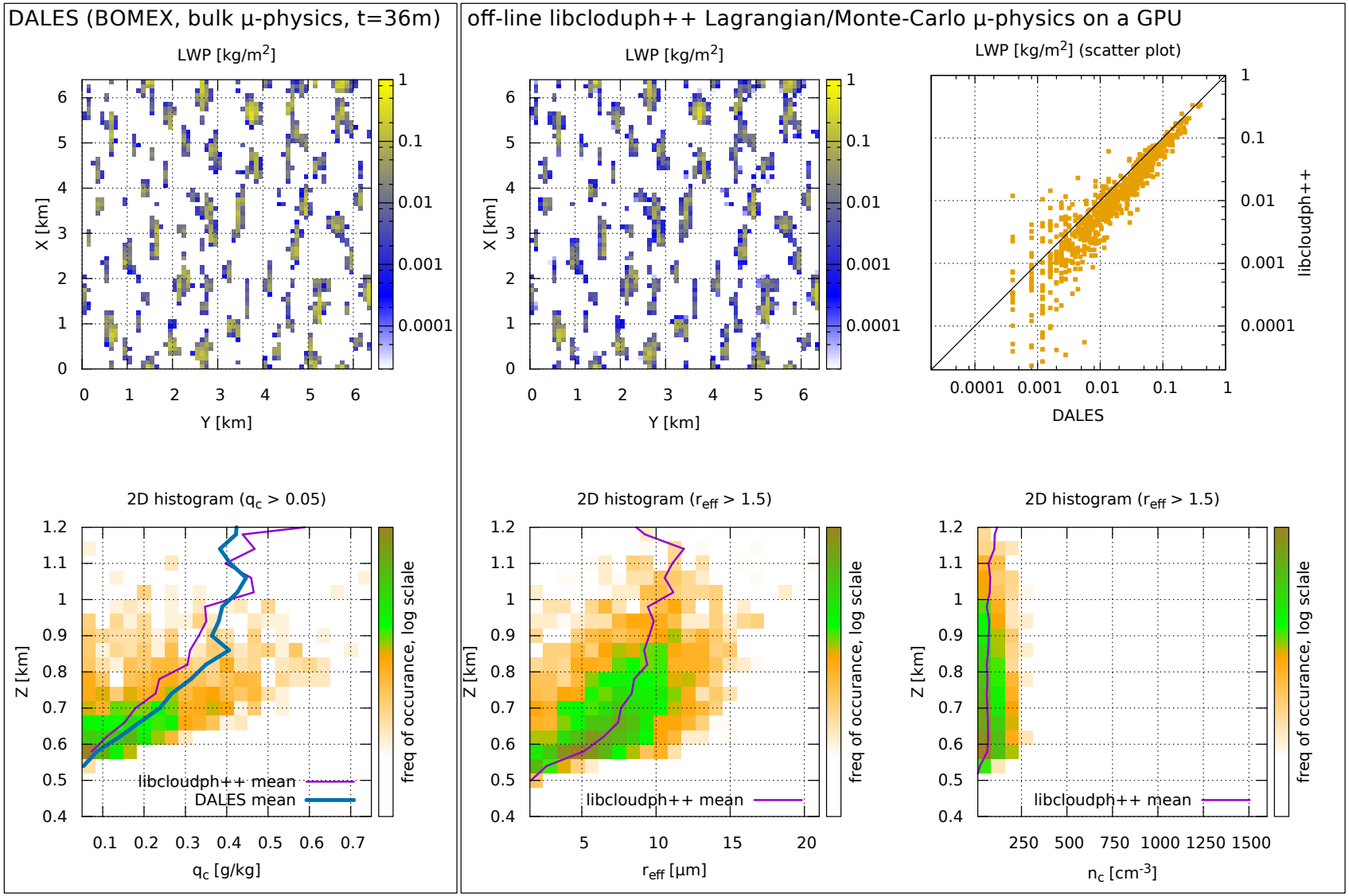
example: DALES/libclouph++ coupling



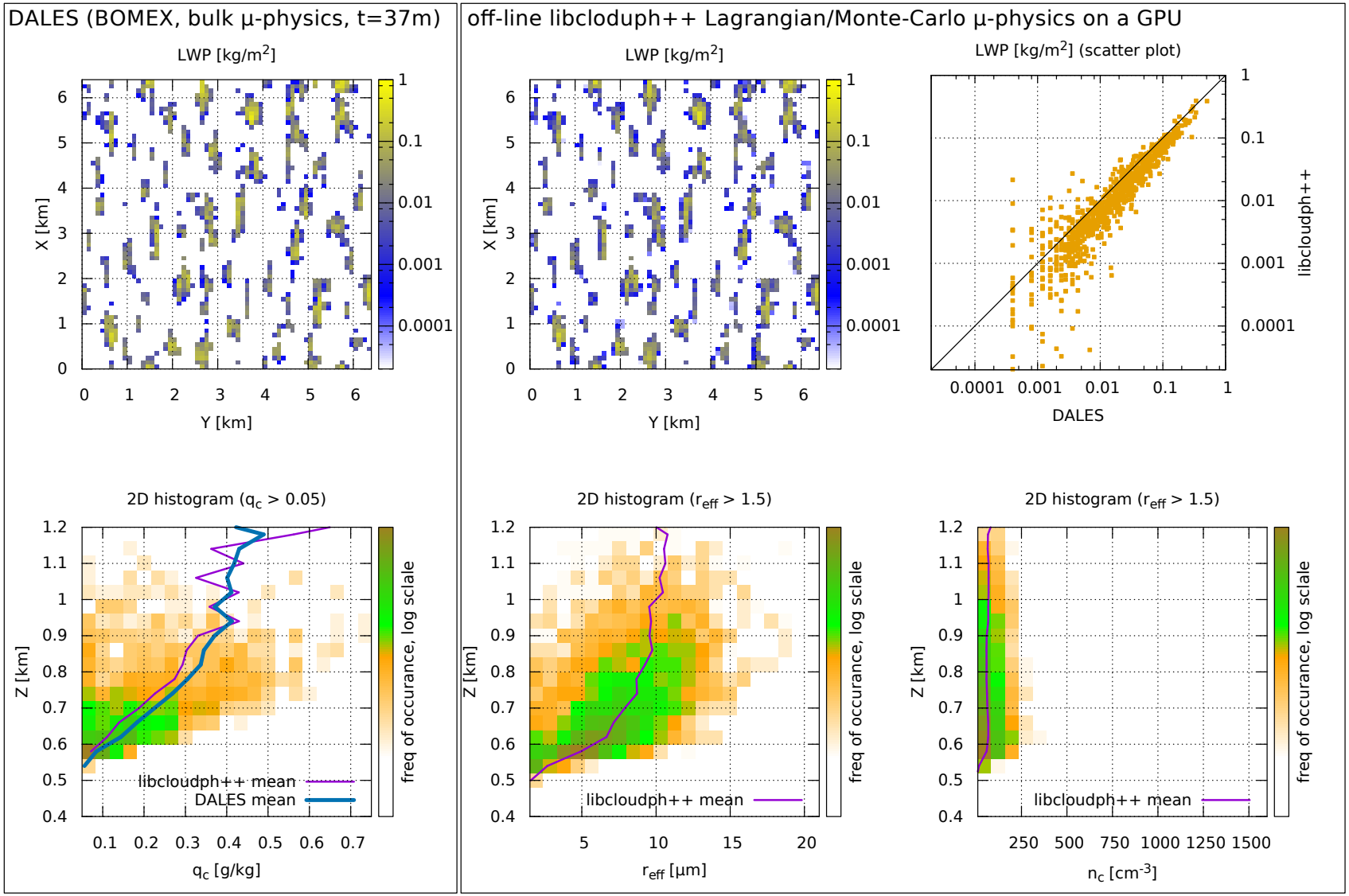
example: DALES/libclouph++ coupling



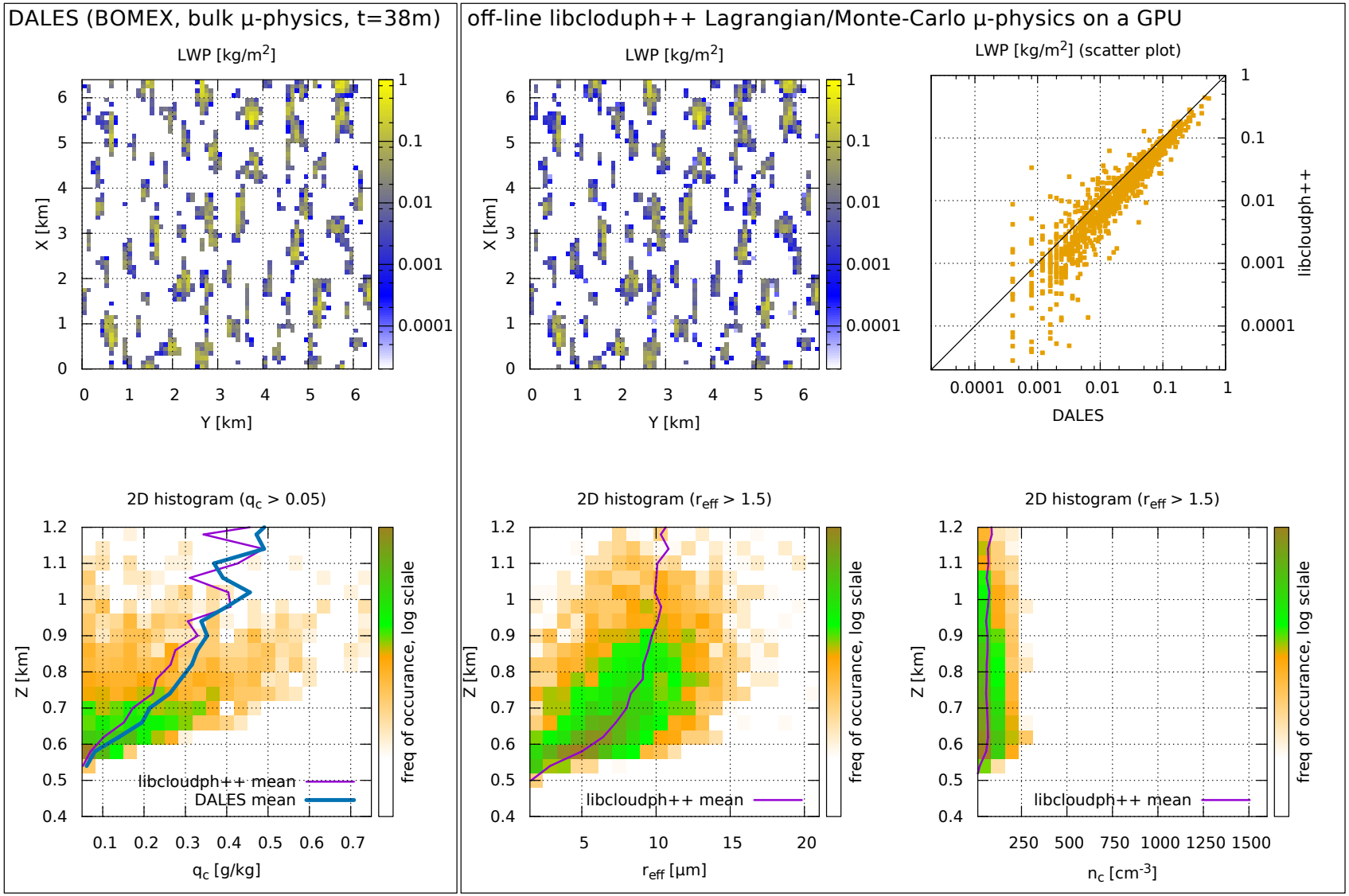
example: DALES/libclouph++ coupling



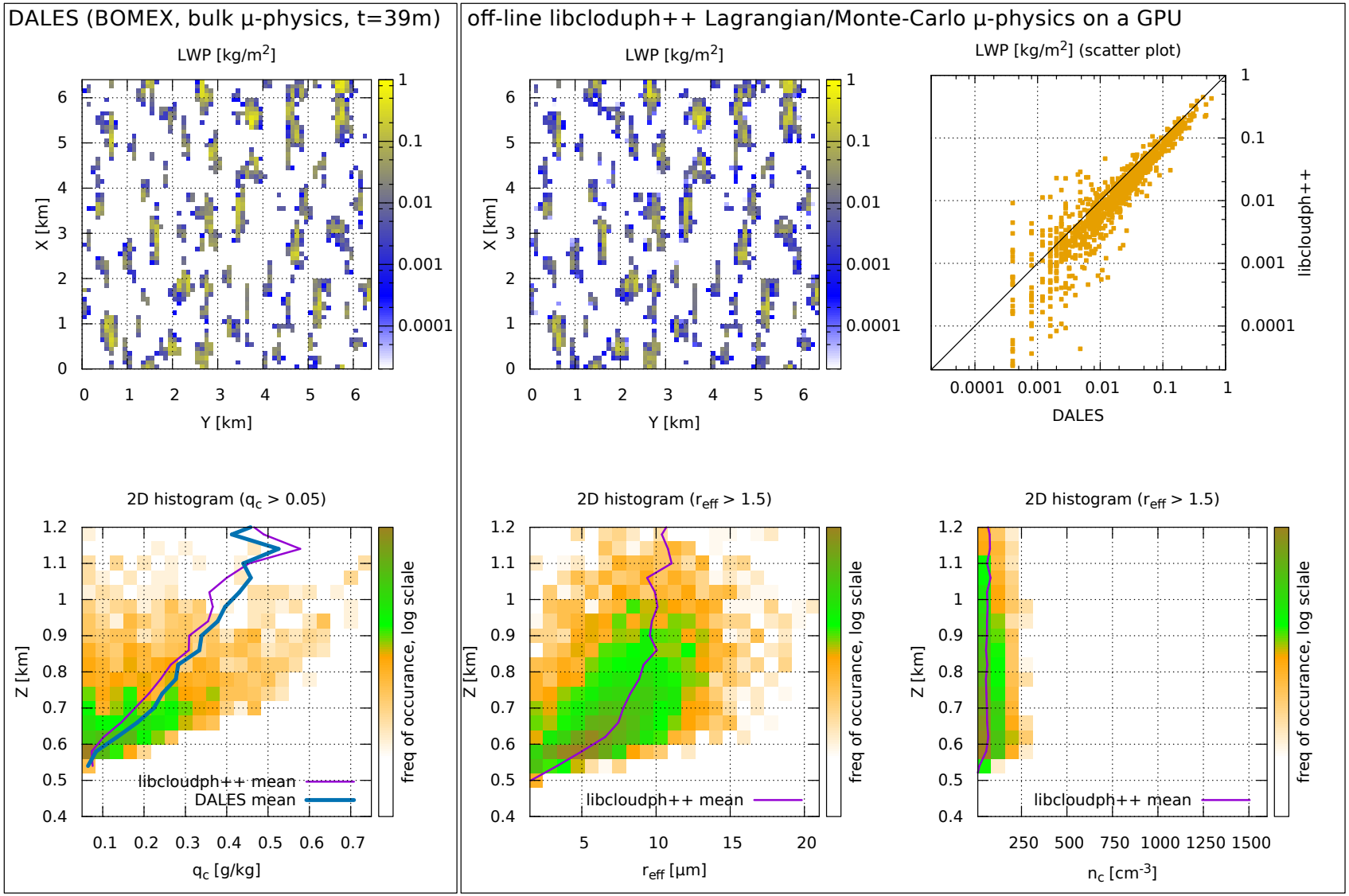
example: DALES/libclouph++ coupling



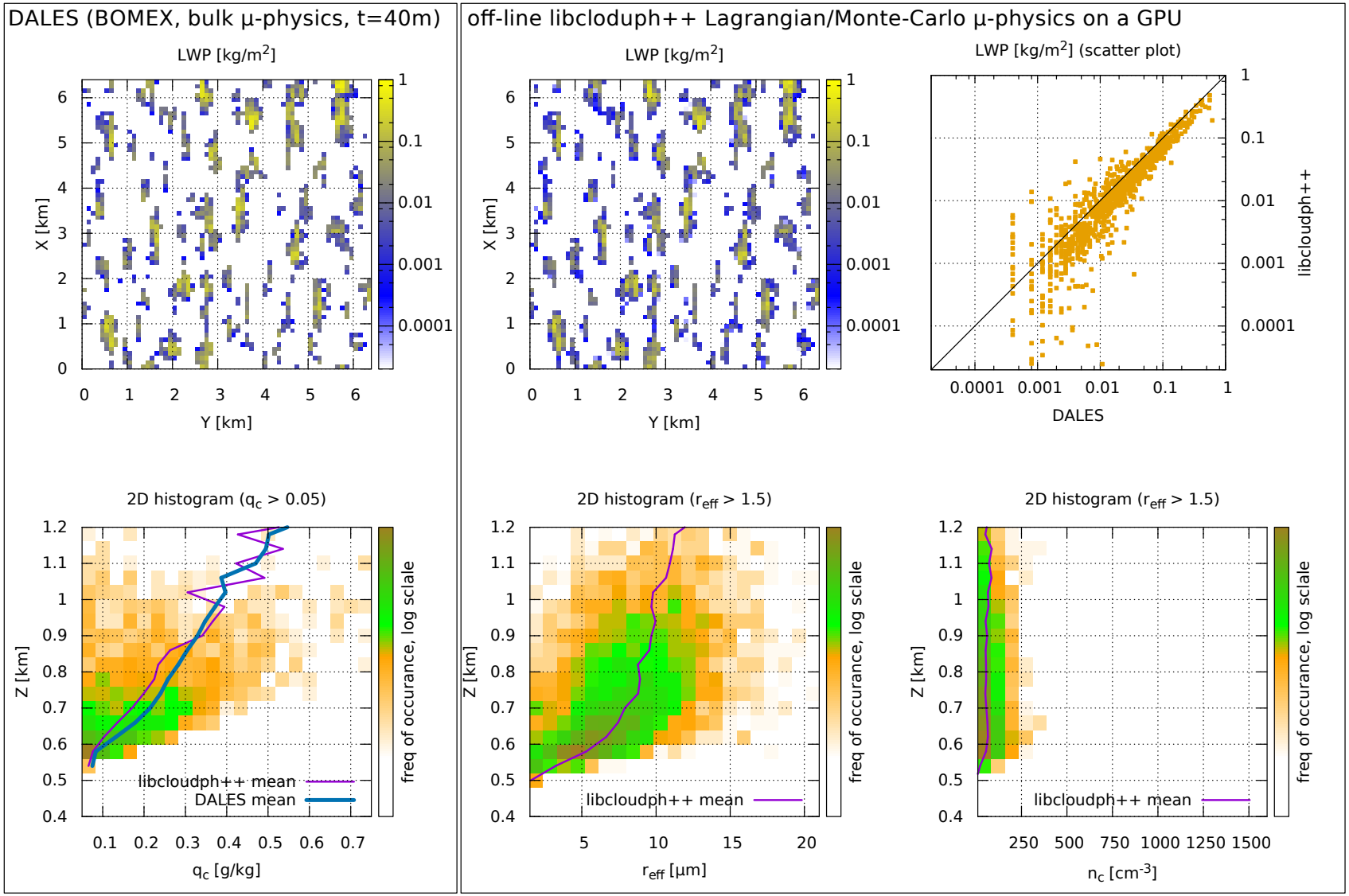
example: DALES/libclouph++ coupling



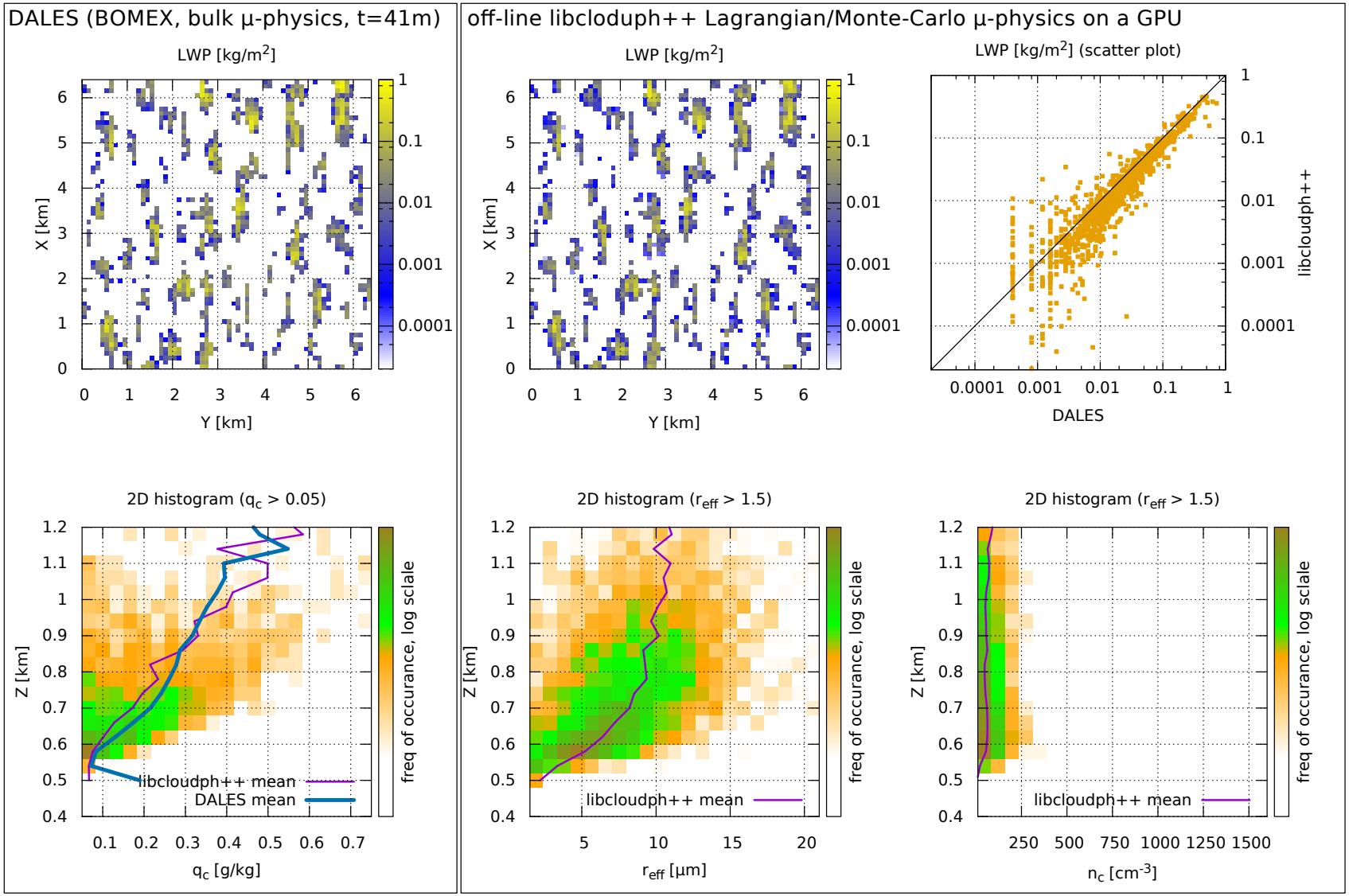
example: DALES/libclouph++ coupling



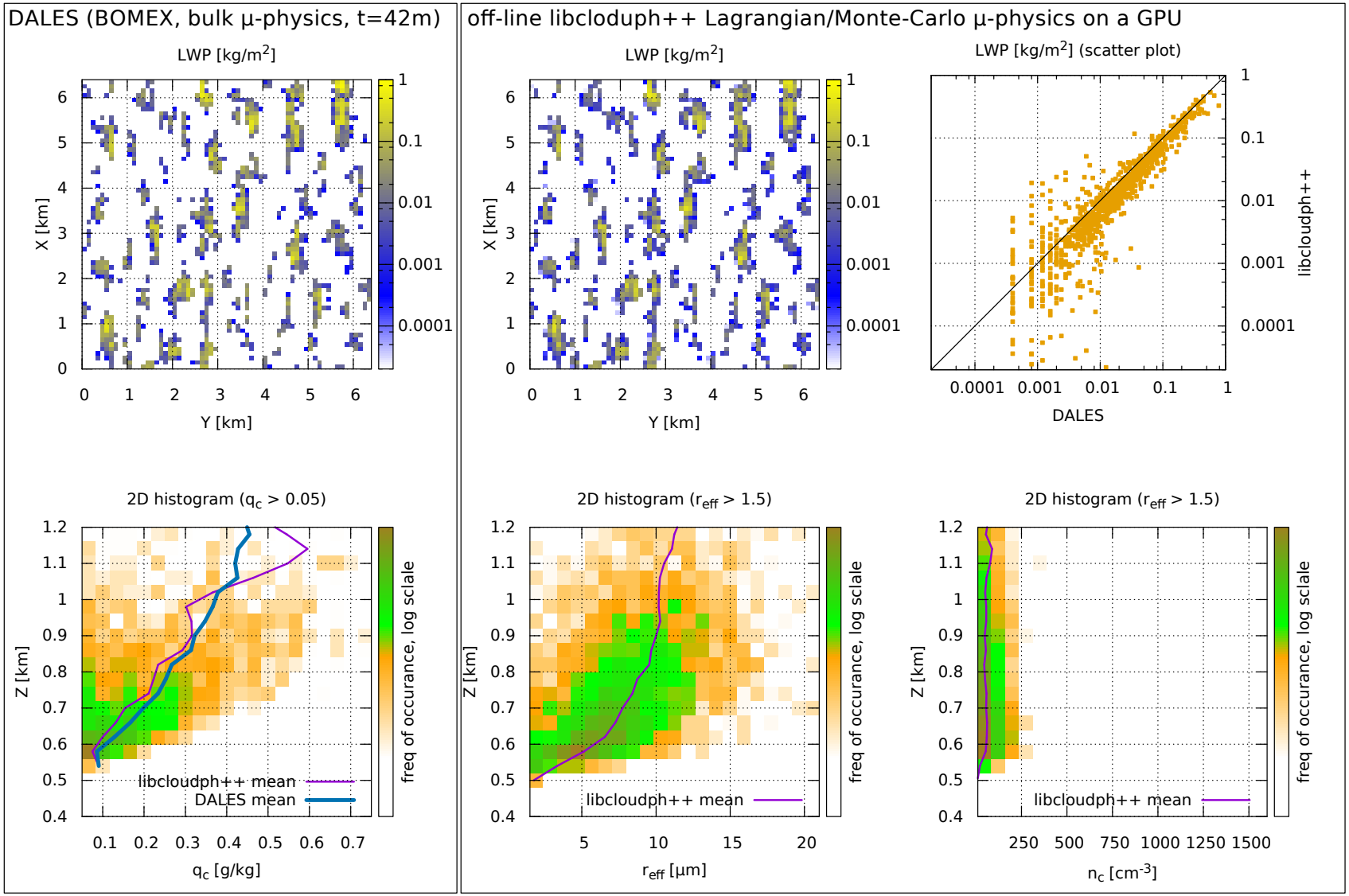
example: DALES/libclouph++ coupling



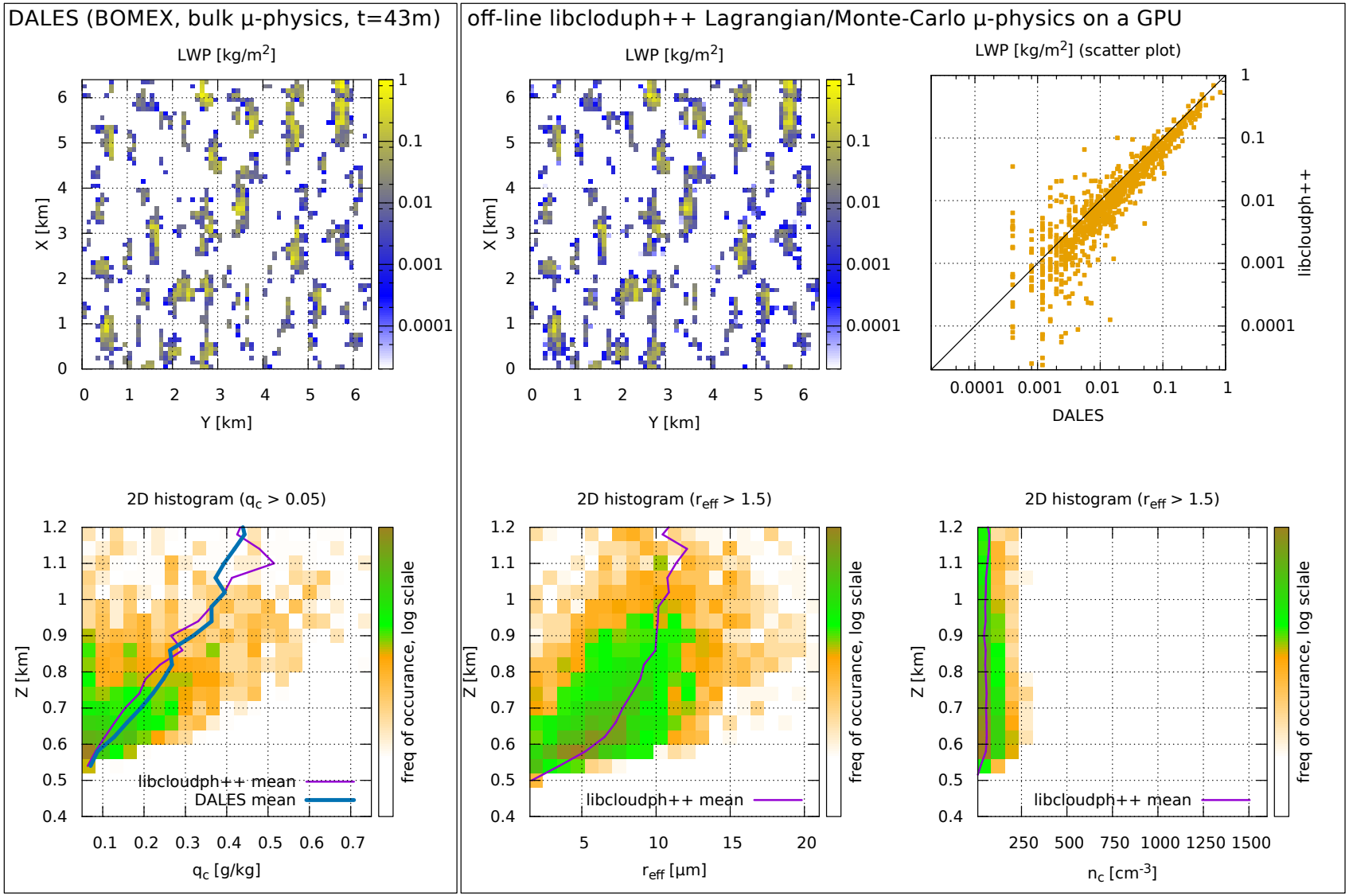
example: DALES/libclouph++ coupling



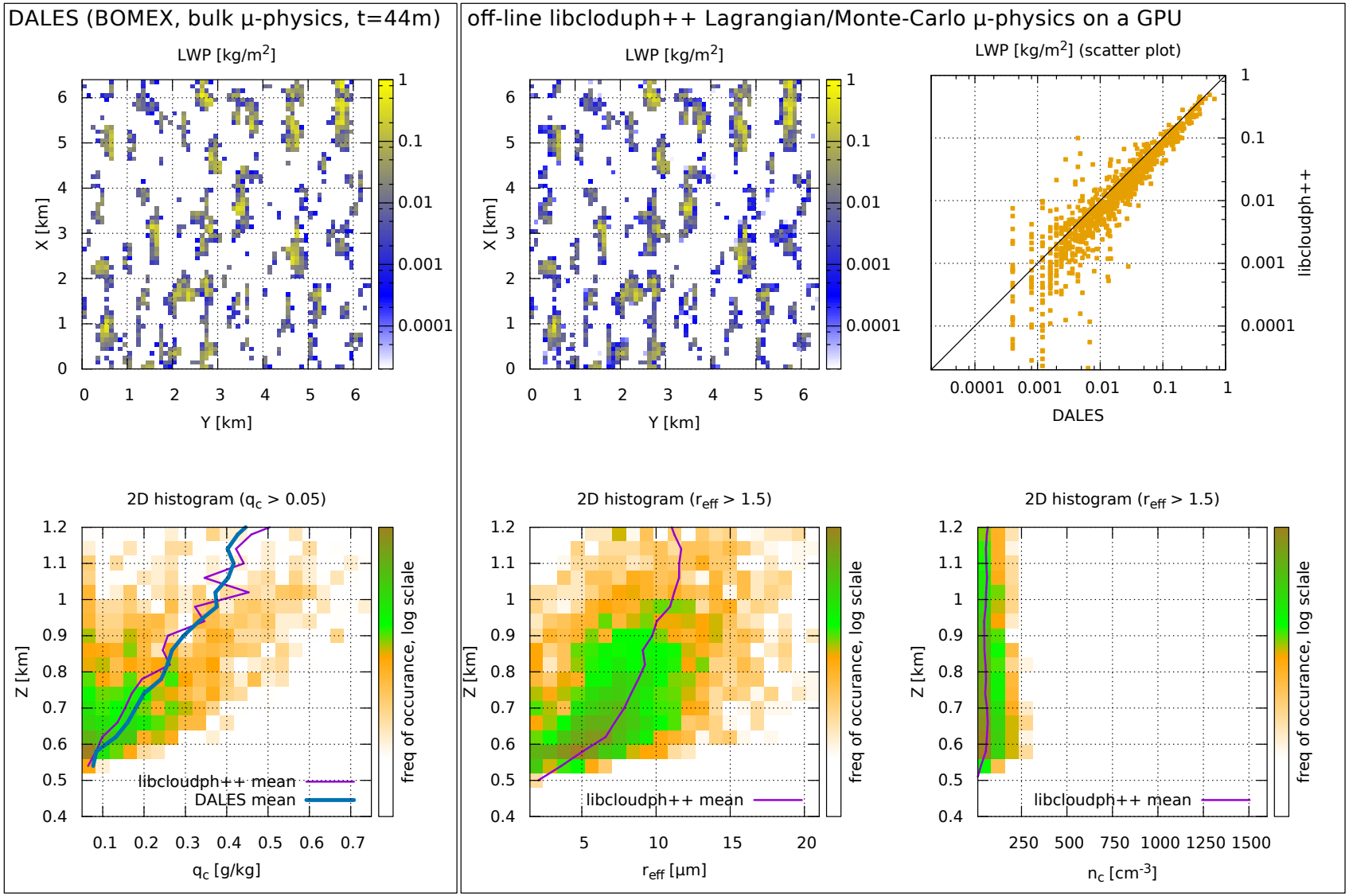
example: DALES/libclouph++ coupling



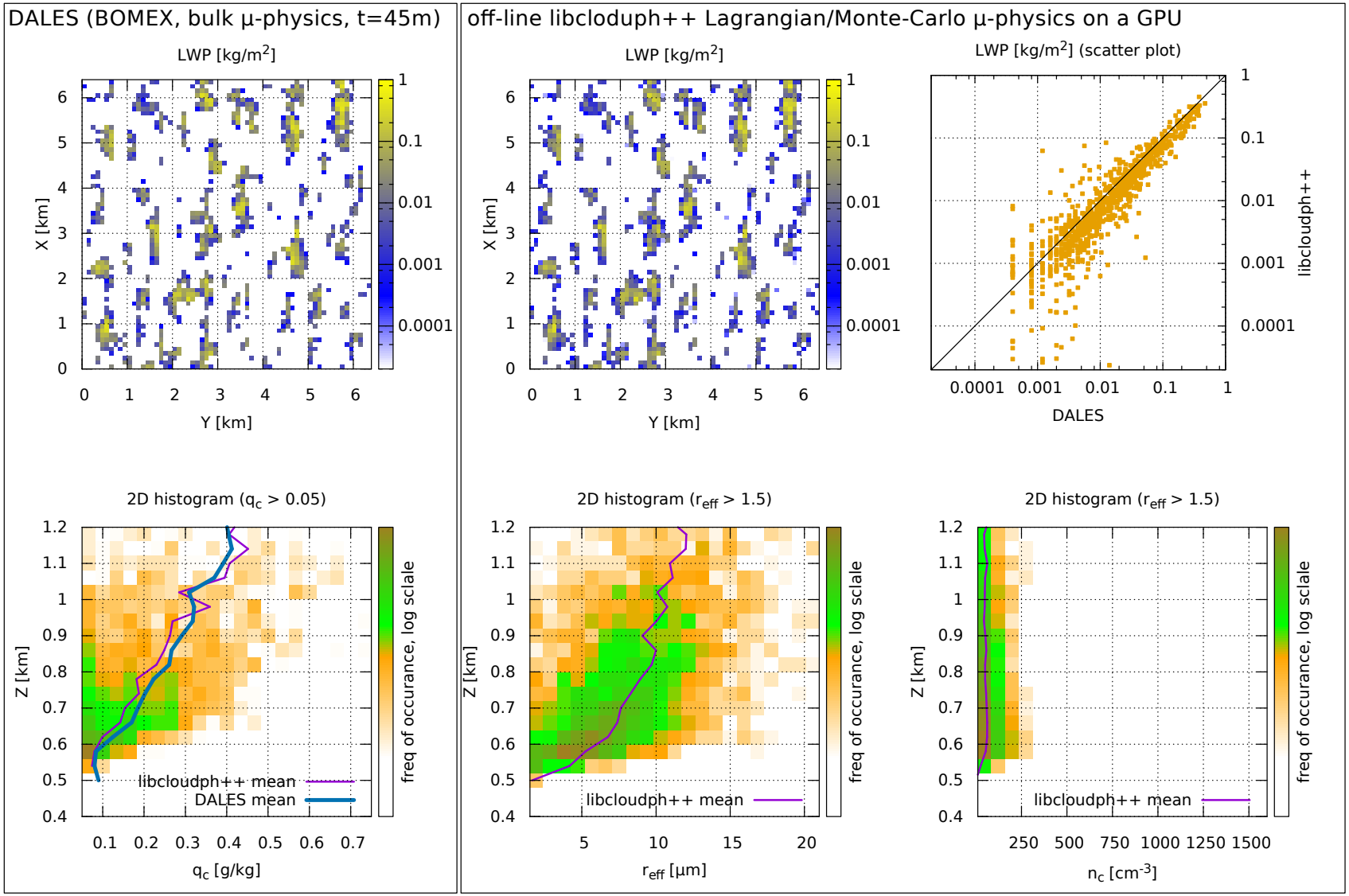
example: DALES/libclouph++ coupling



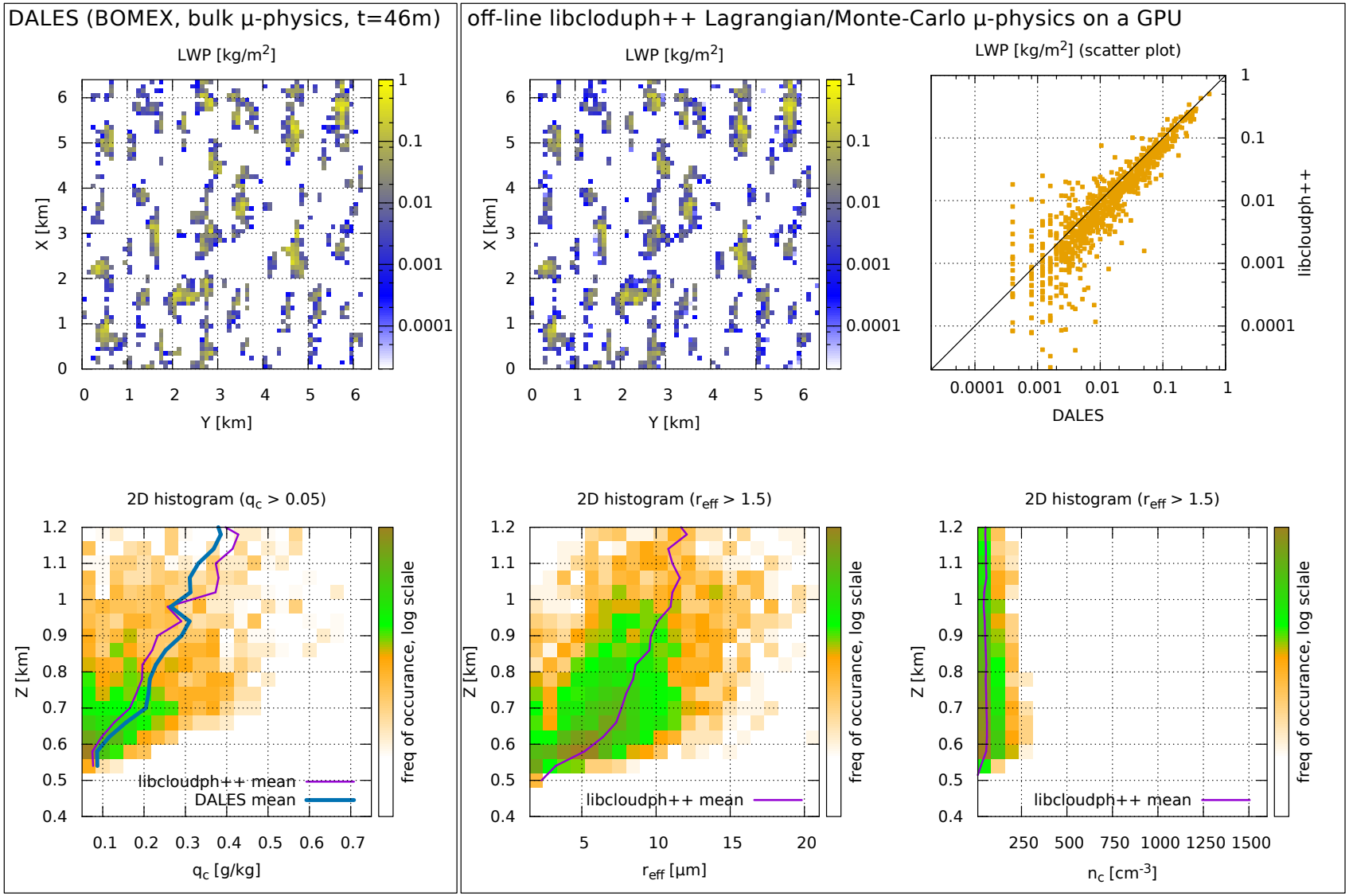
example: DALES/libclouph++ coupling



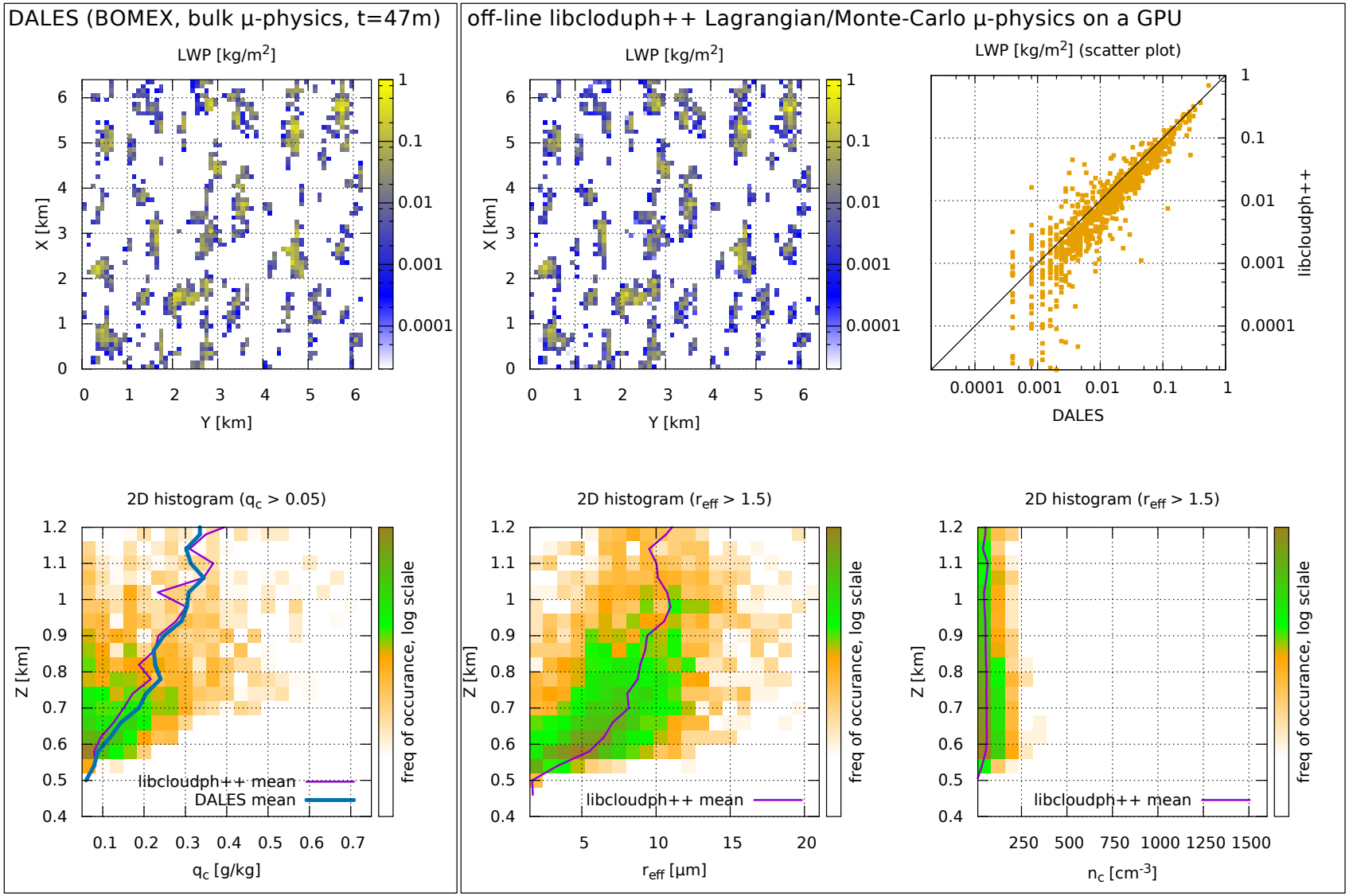
example: DALES/libclouph++ coupling



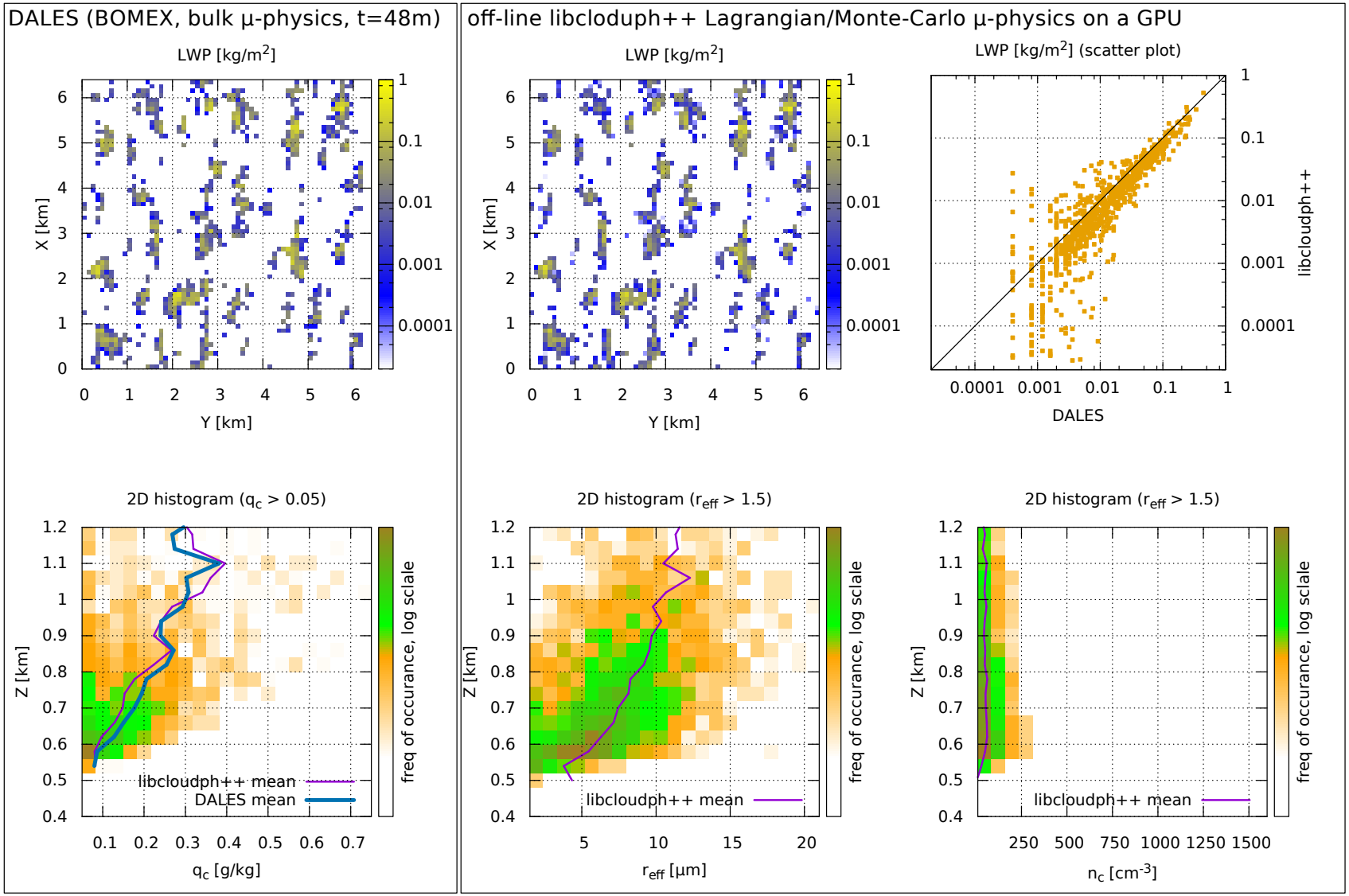
example: DALES/libclouph++ coupling



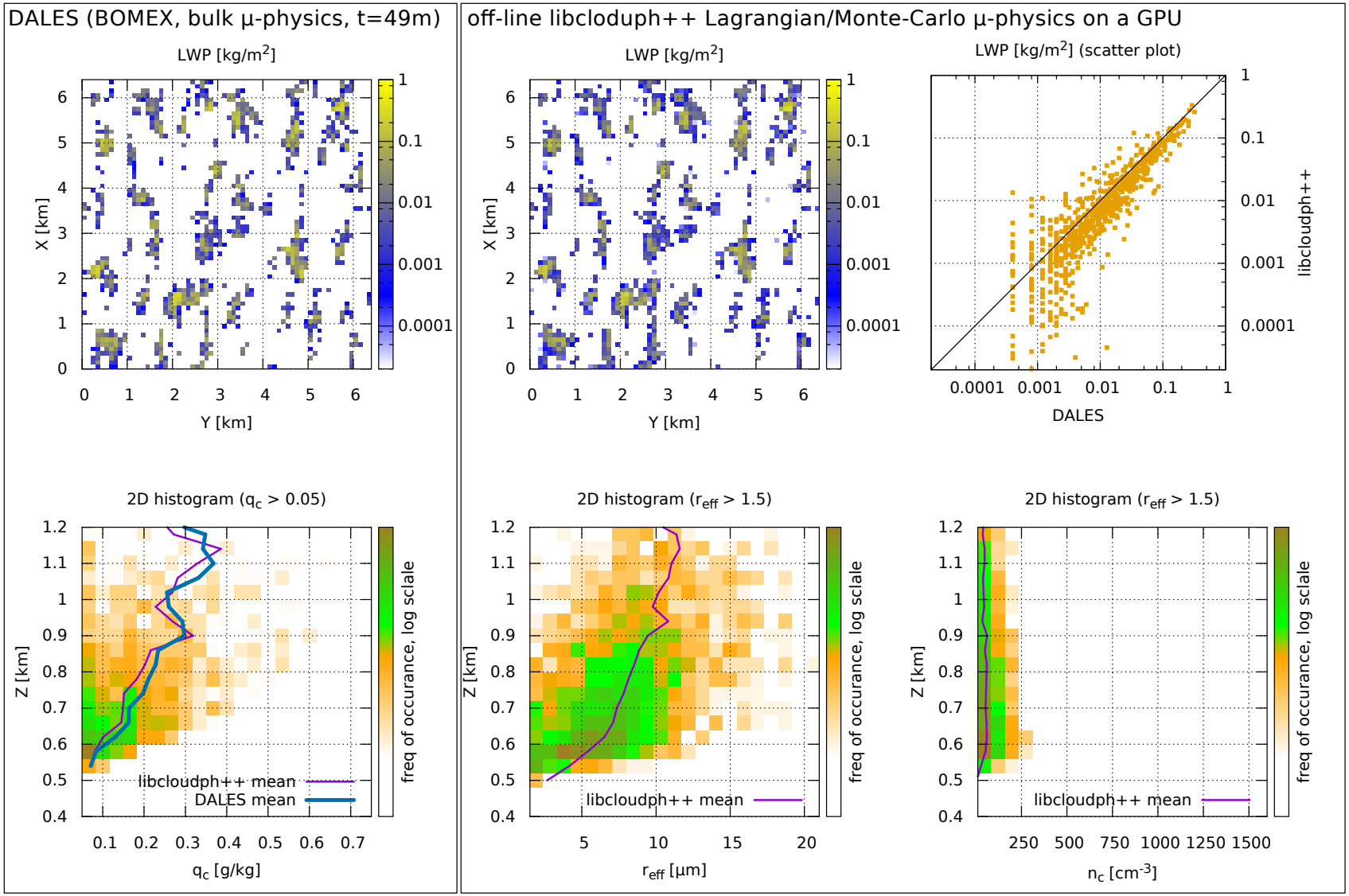
example: DALES/libclouph++ coupling



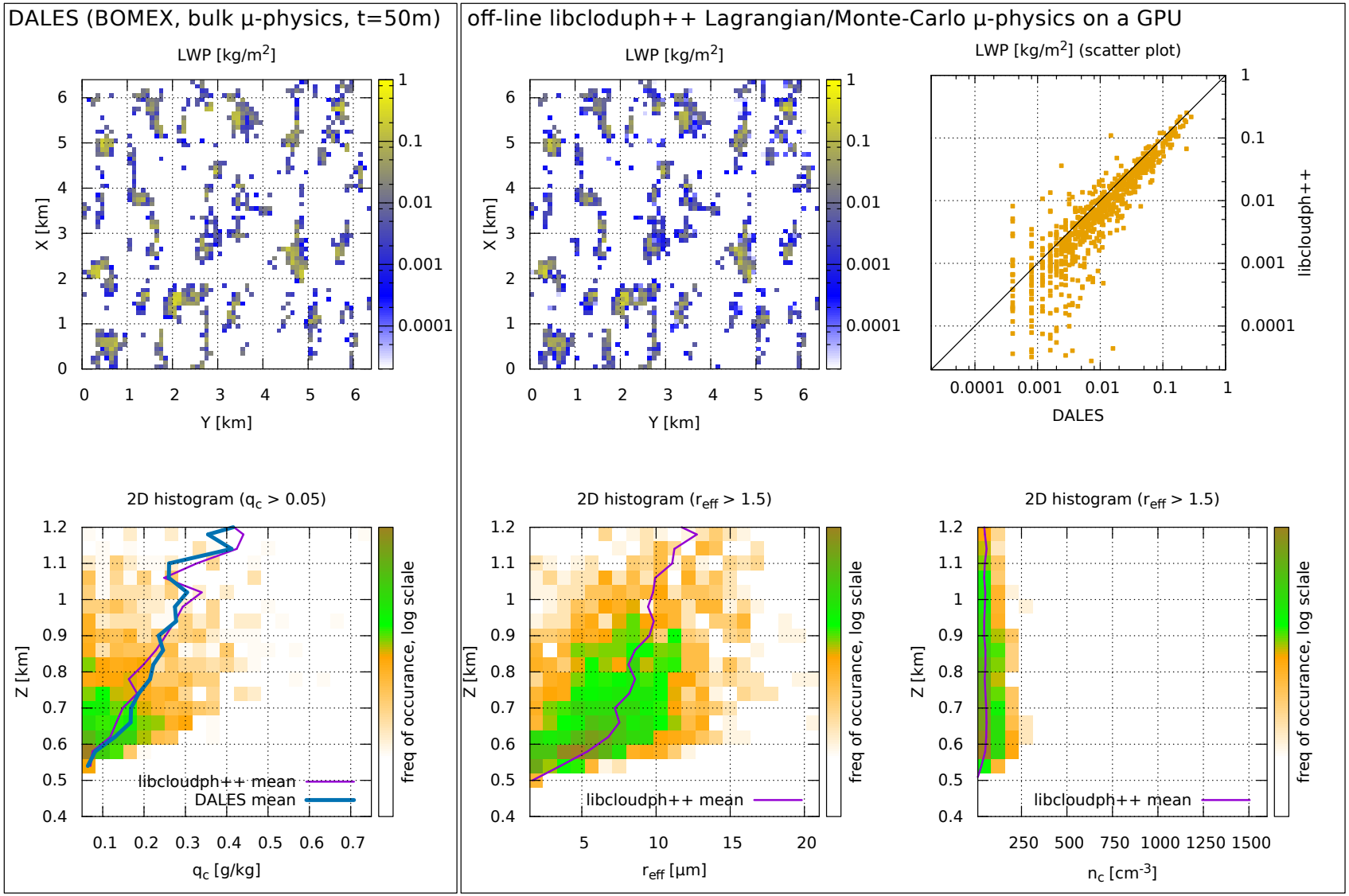
example: DALES/libclouph++ coupling



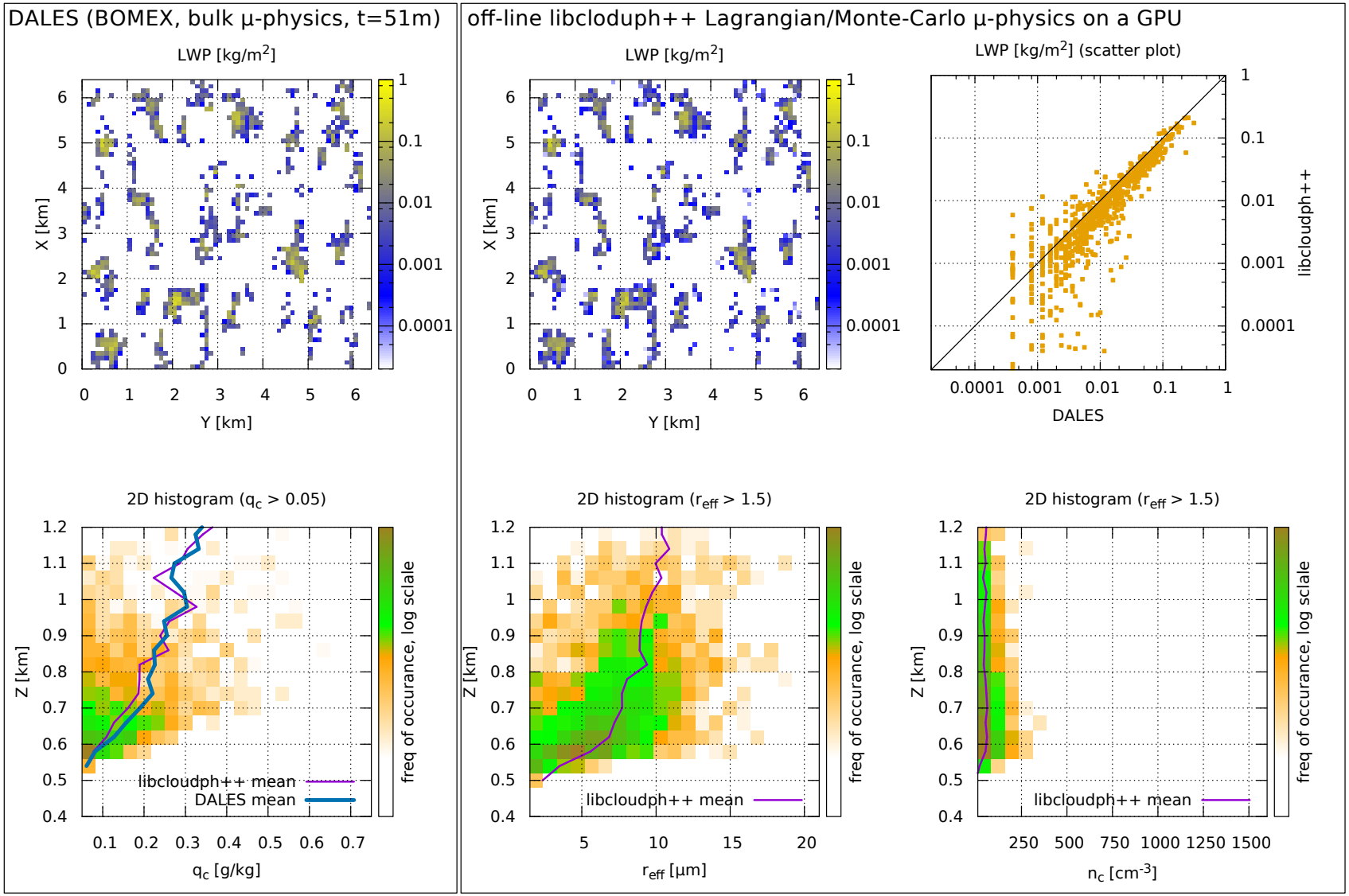
example: DALES/libclouph++ coupling



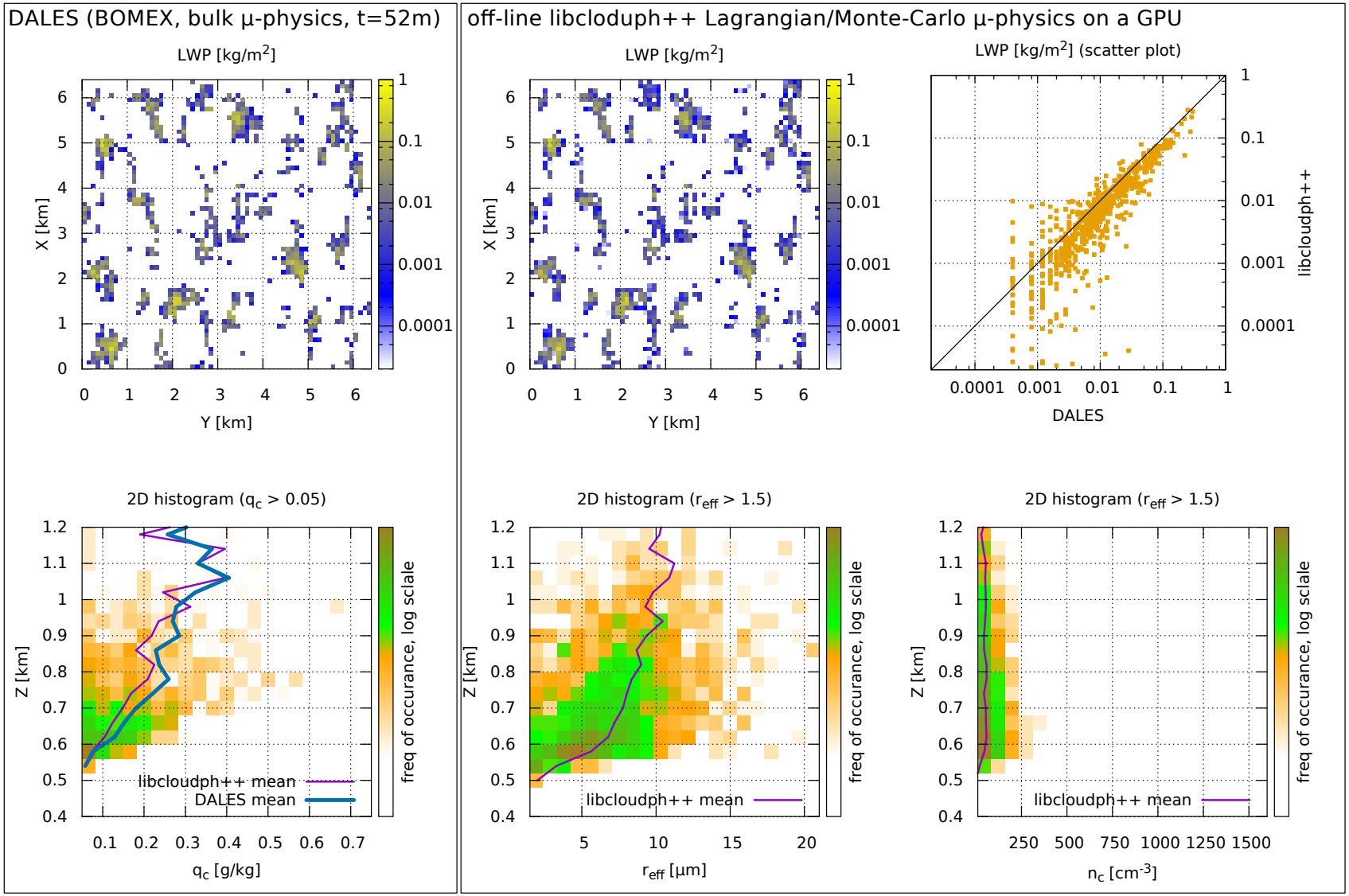
example: DALES/libclouph++ coupling



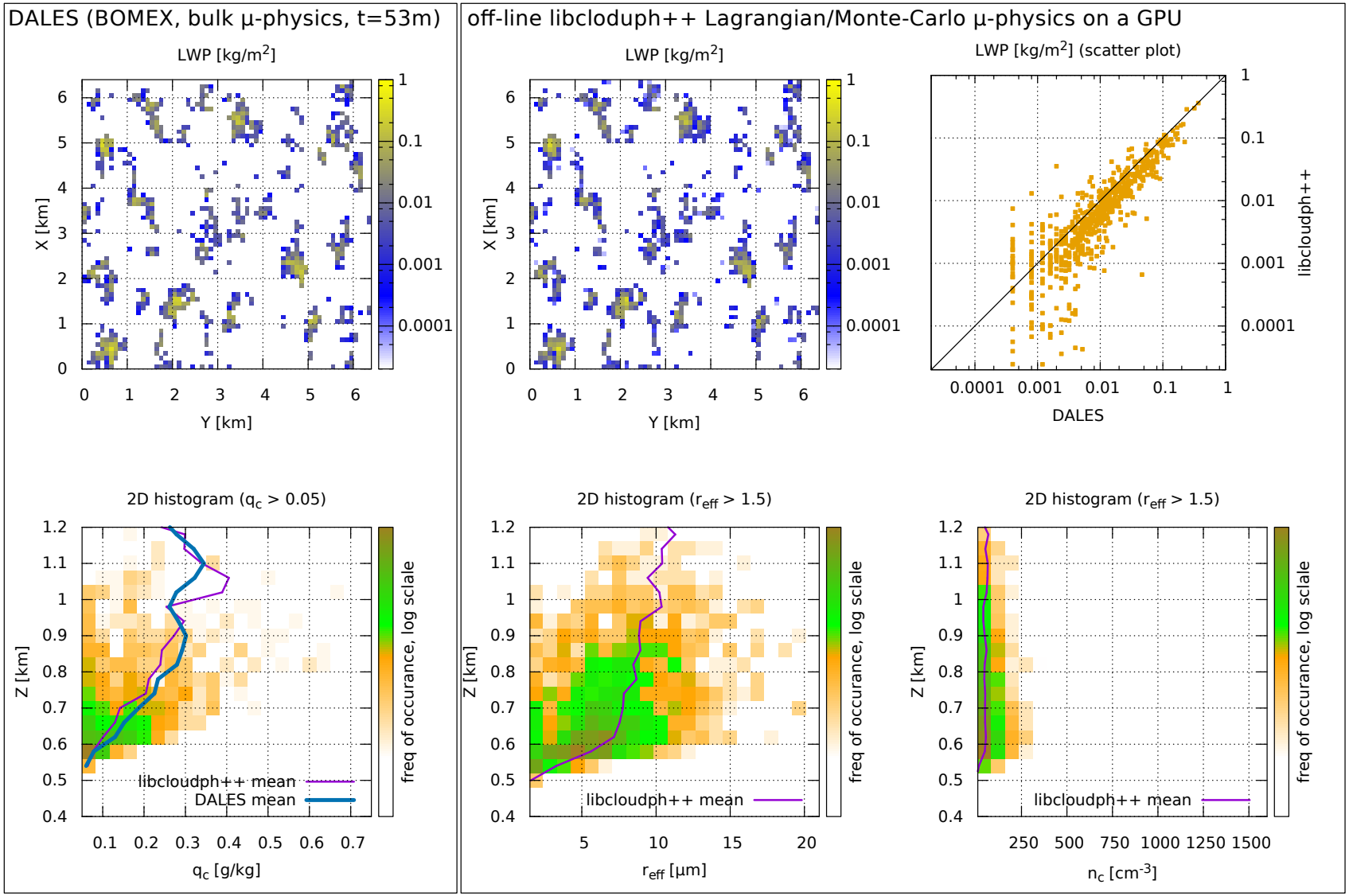
example: DALES/libclouph++ coupling



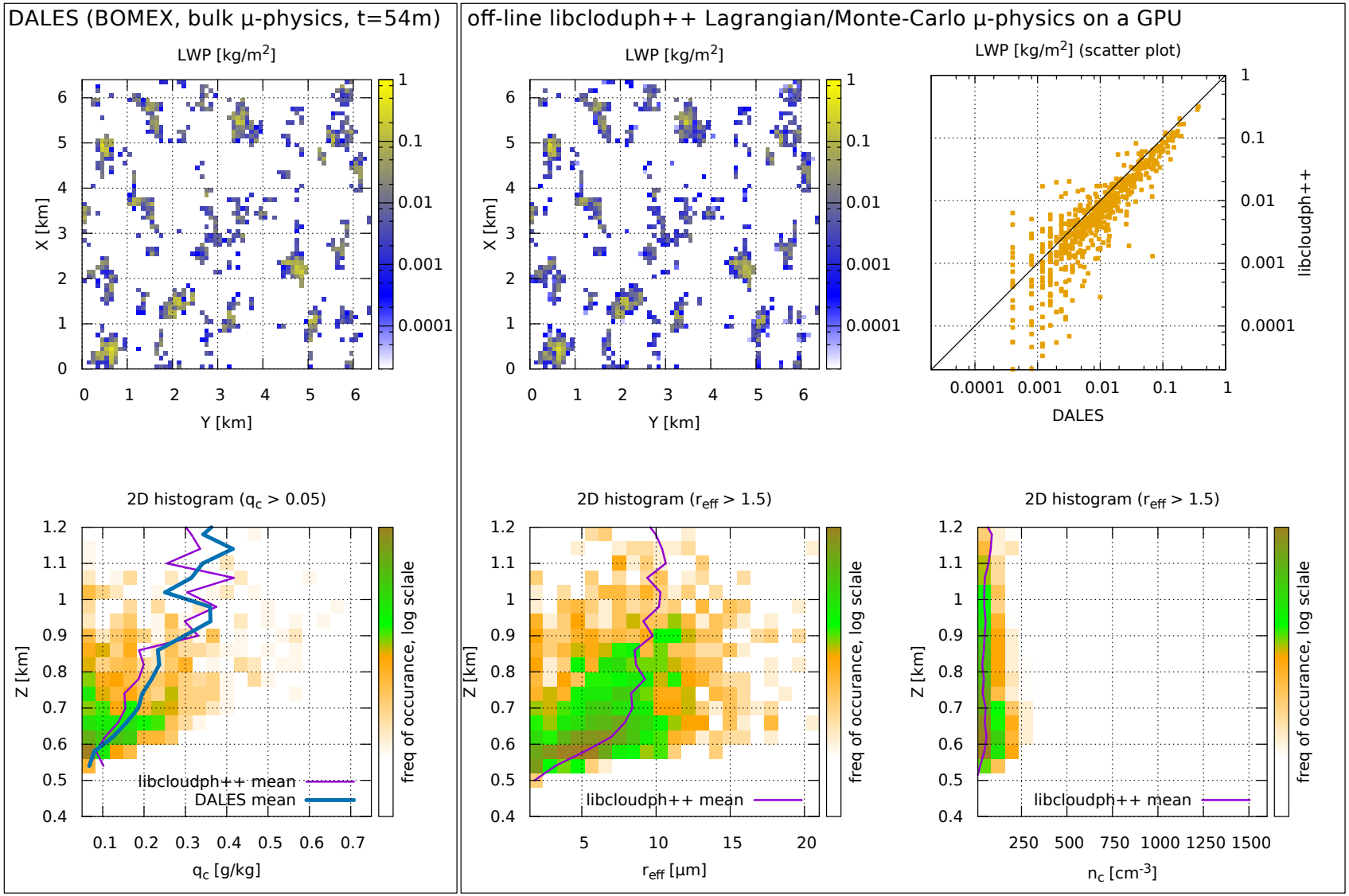
example: DALES/libclouph++ coupling



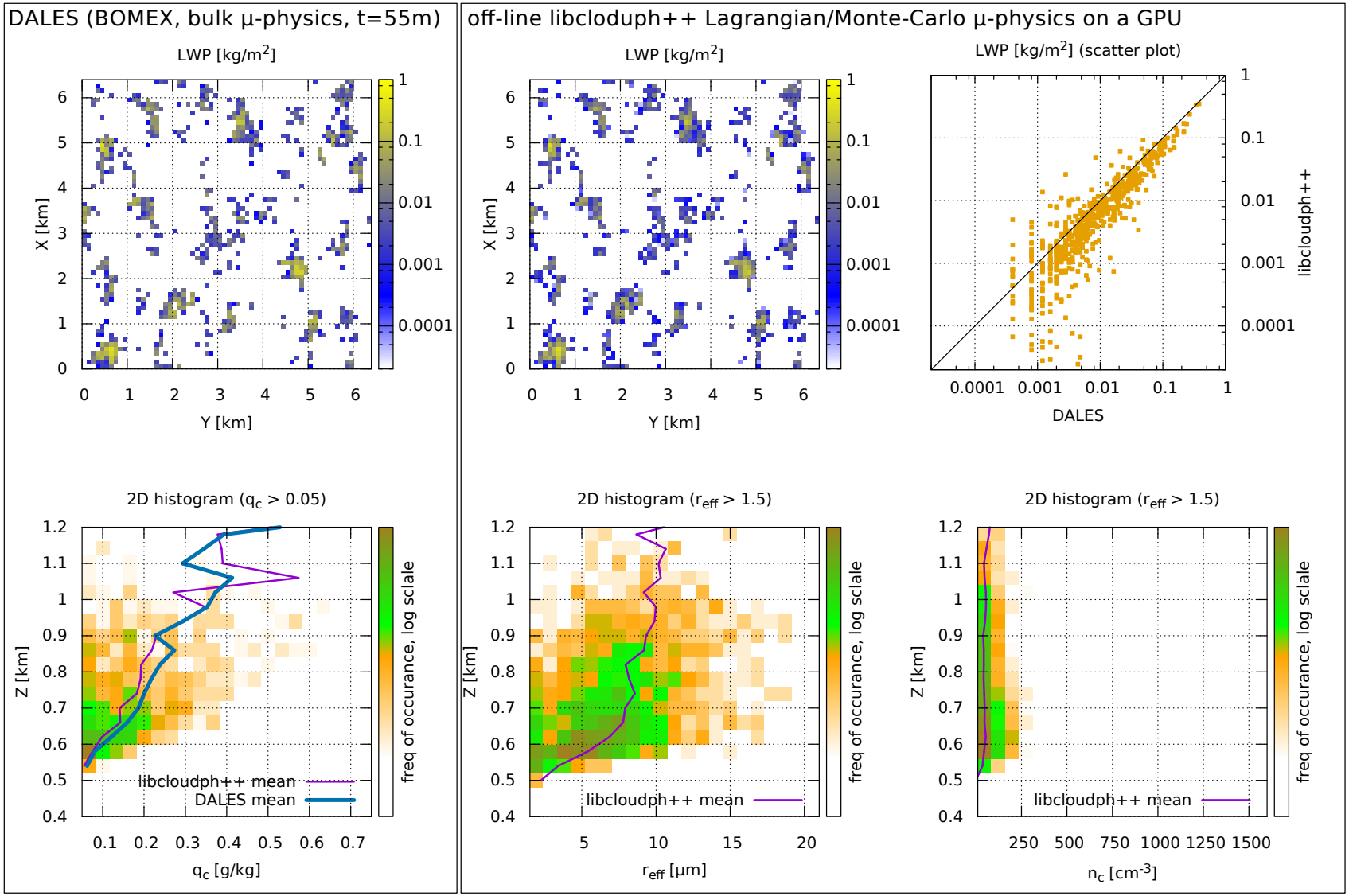
example: DALES/libclouph++ coupling



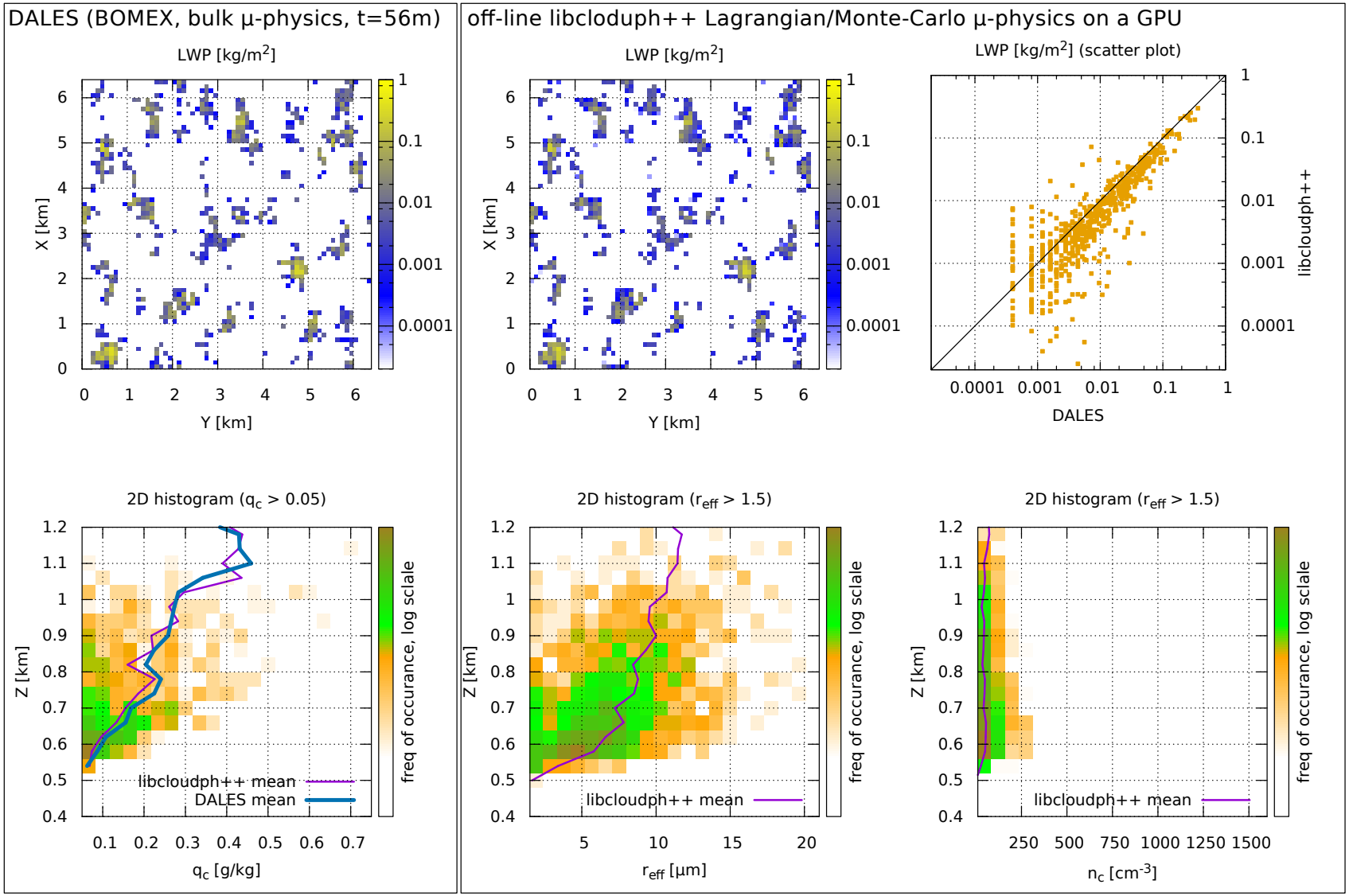
example: DALES/libclouph++ coupling



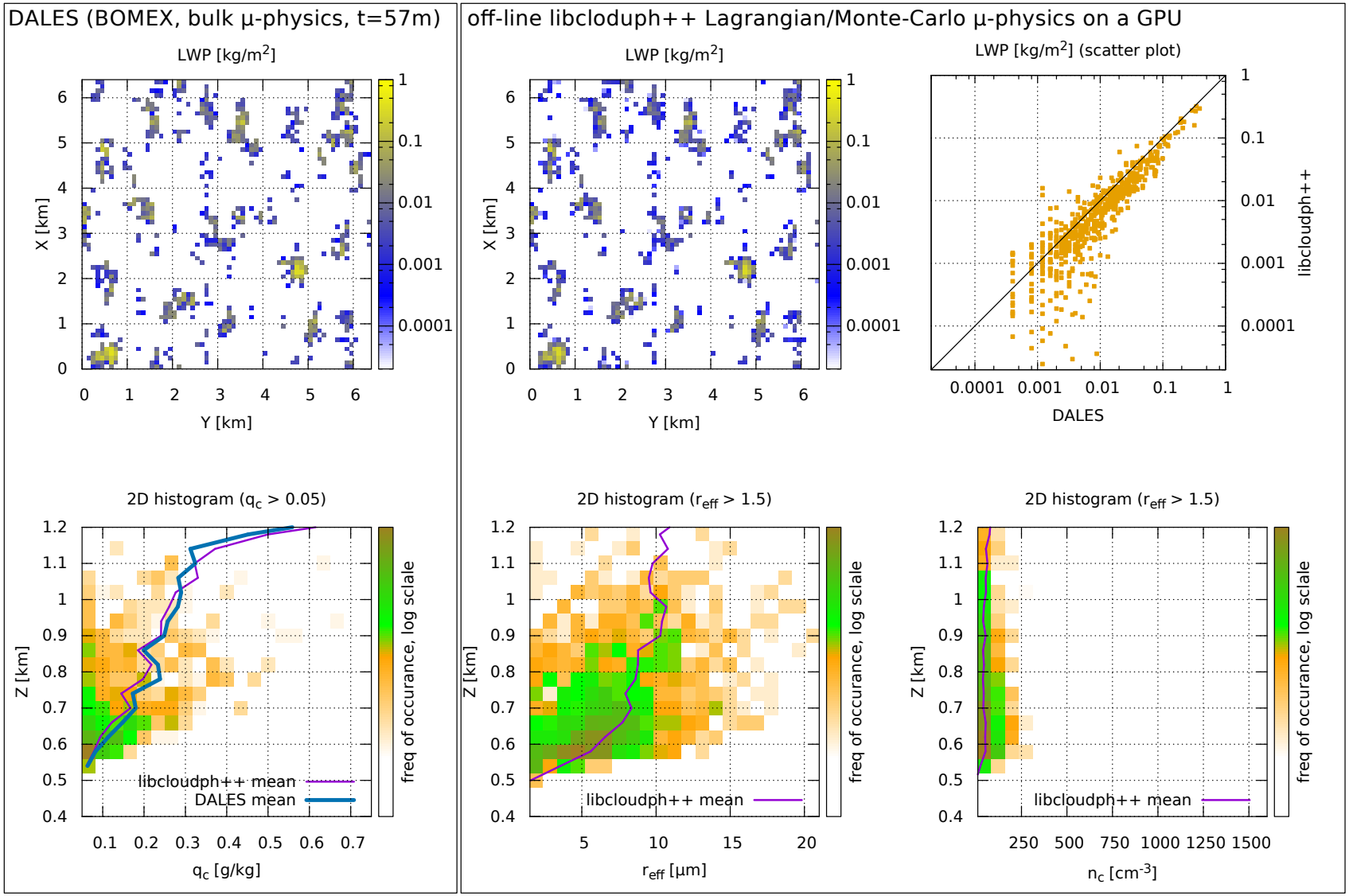
example: DALES/libclouph++ coupling



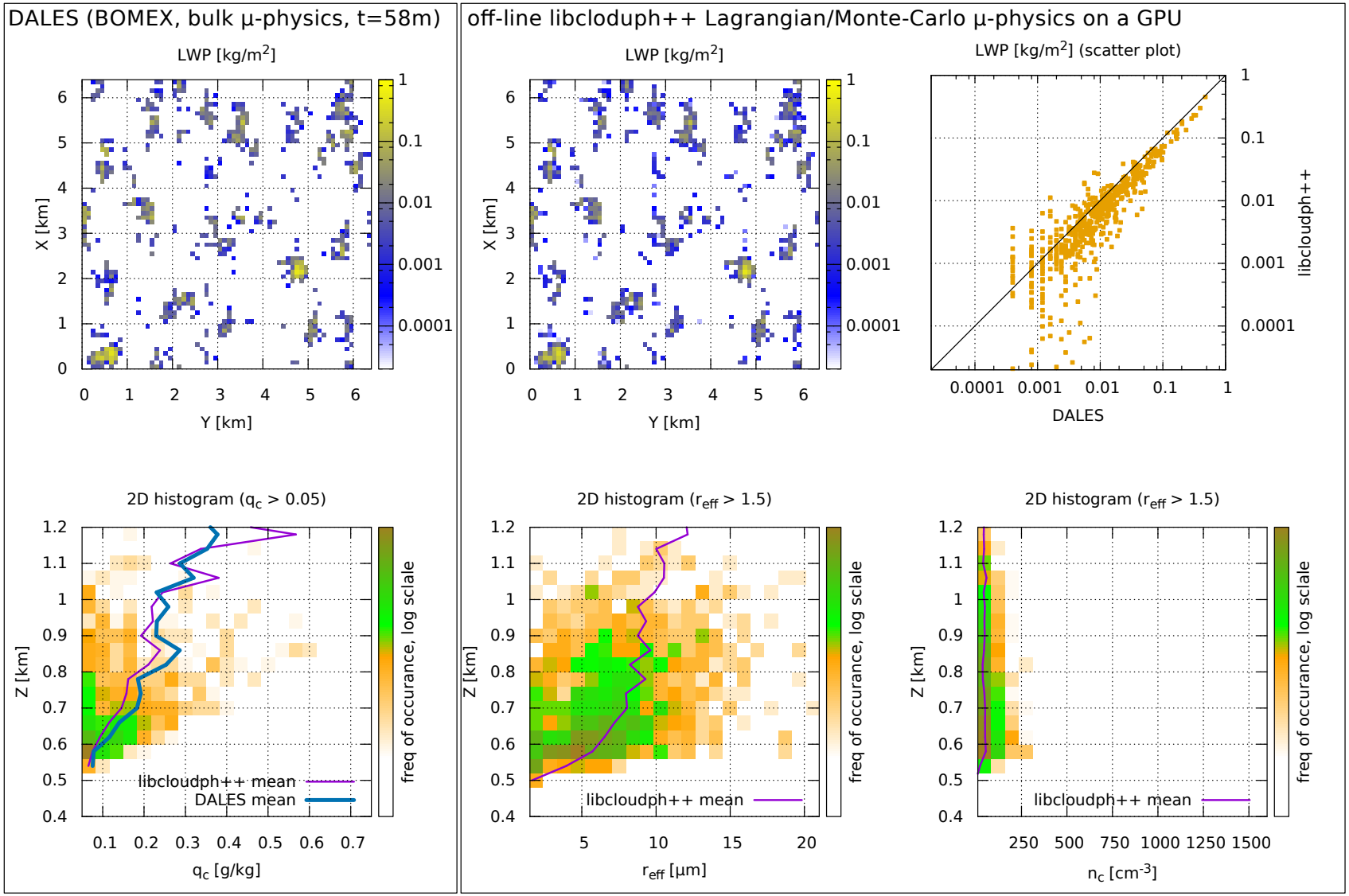
example: DALES/libclouph++ coupling



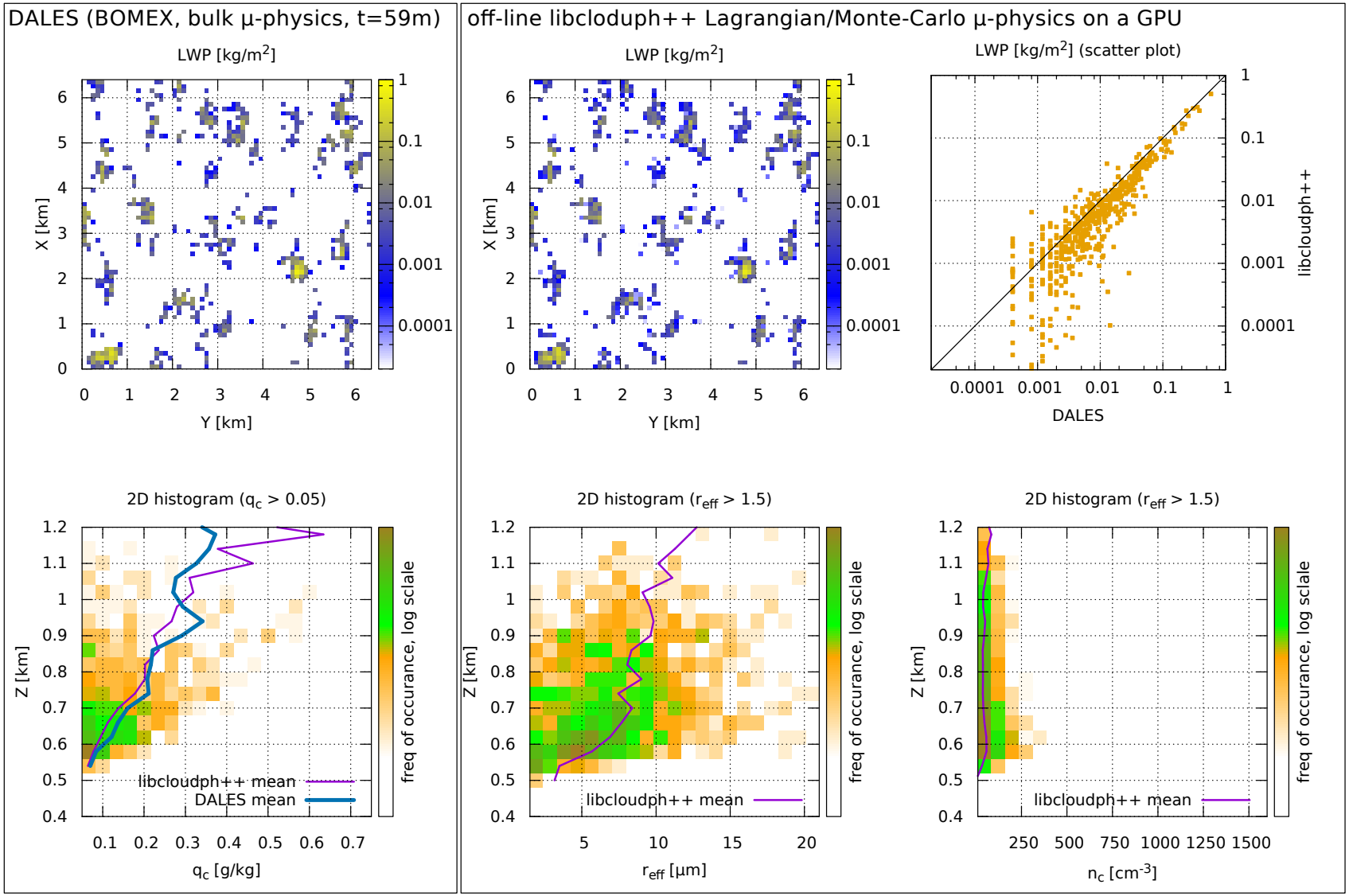
example: DALES/libclouph++ coupling



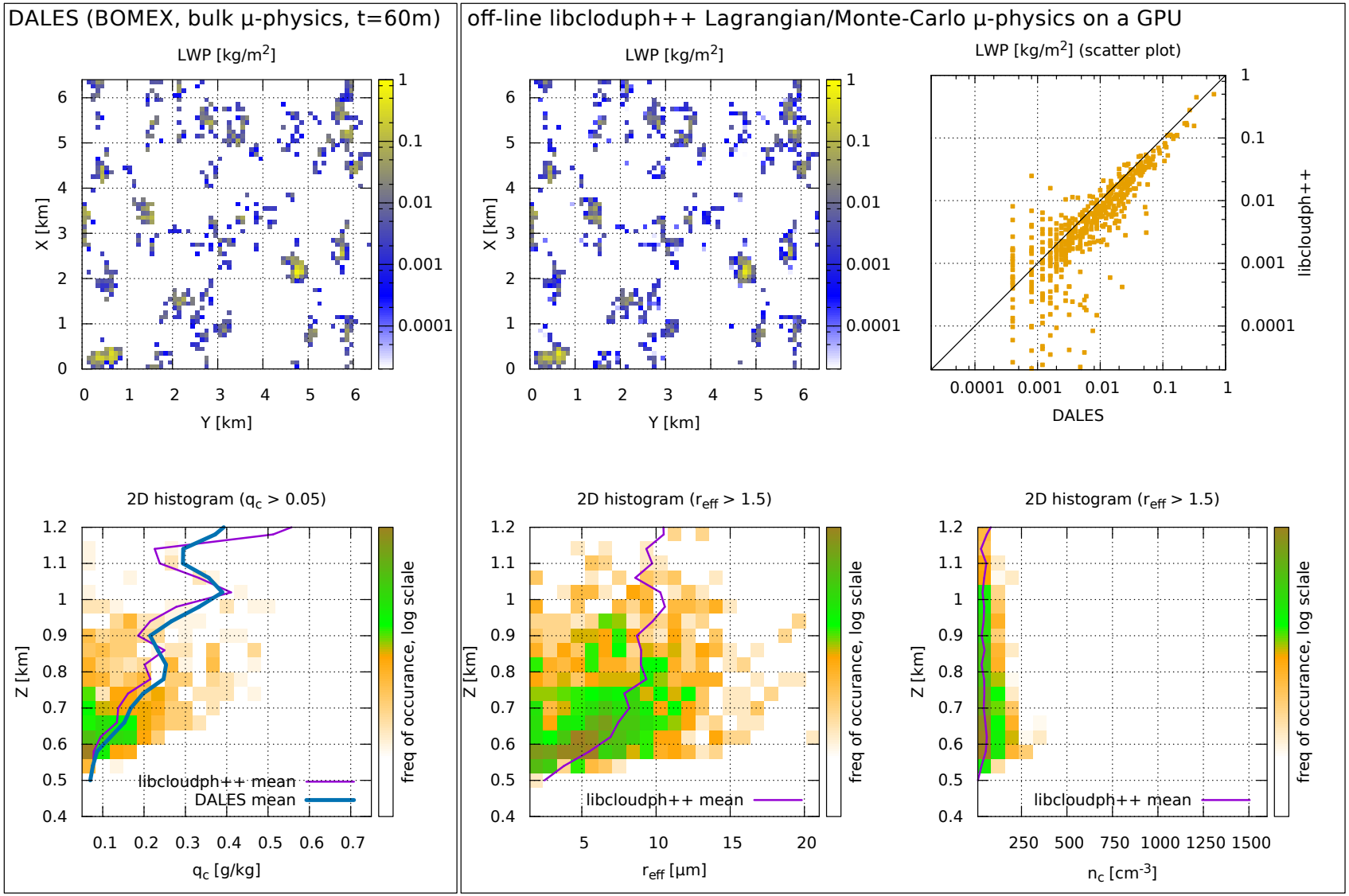
example: DALES/libclouph++ coupling



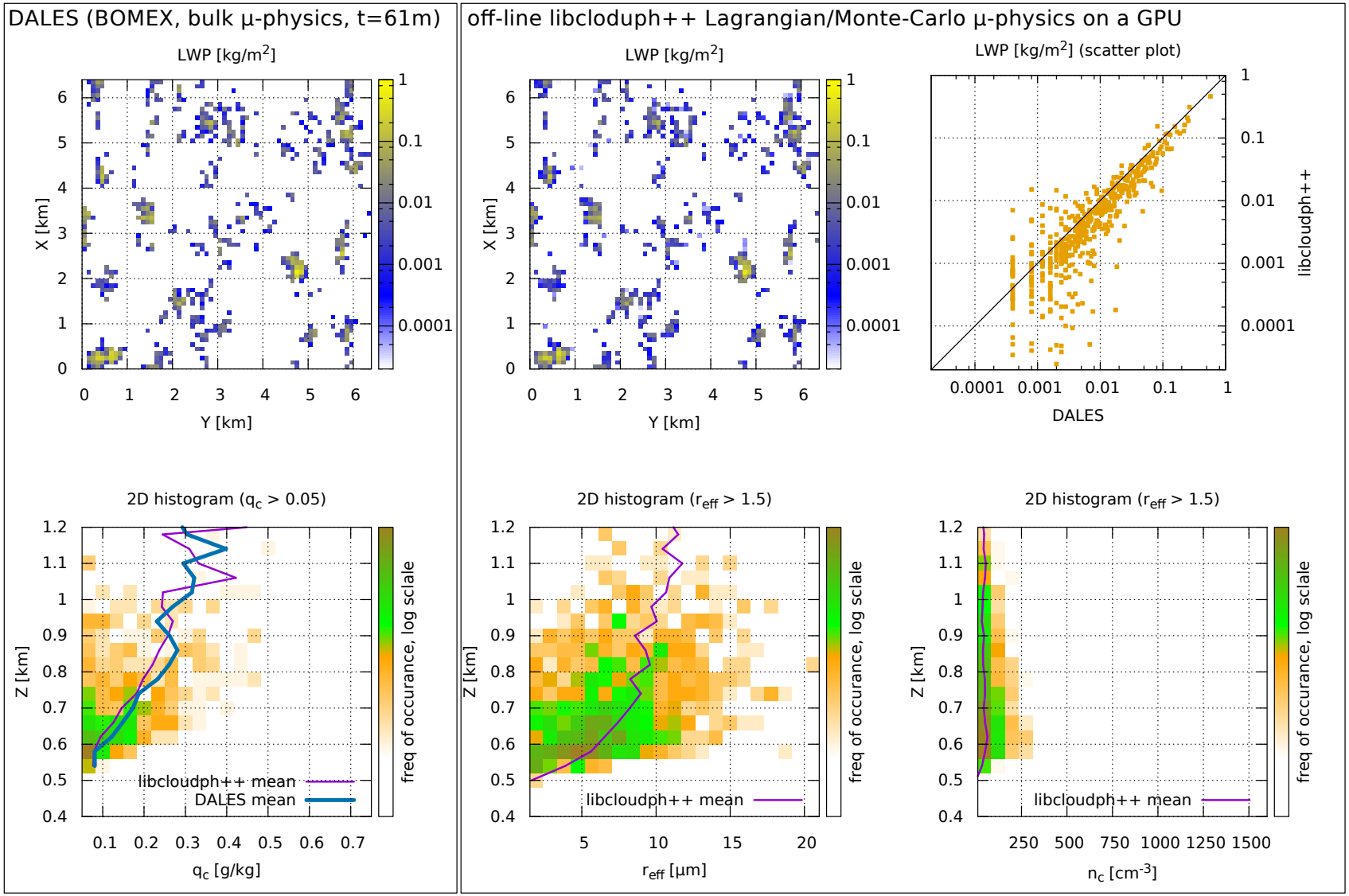
example: DALES/libclouph++ coupling



example: DALES/libclouph++ coupling



example: DALES/libclouph++ coupling



example: DALES/libclouph++ coupling

Summary:

- off-line Lagrangian microphysics for DALES on GPU
- libcloudph++:
  no modifications
- DALES:
  ca. 10 LoC changed;
  ca. 100 LoC added in a new file
- coupling code:
  ca. 300 LoC in Python

talk outline

1. introduction
2. binding to libcloudph++ library
3. bindings to WRF model
4. bindings libcloudph++ and atmospheric models
5. summary

talk outline

Summary

- Python is an efficient glue language
- bindings allow for easier access to schemes in libraries and models written in native languages: Fortran, C++,...
- avoiding changes to original code, no copy-paste
- much easier to access a scheme from a library (libclouph++) than from an monolithic code