On the design and Boost-based implementation of two new C++ libraries for atmospheric research

Sylwester Arabas

Faculty of Physics, University of Warsaw, Poland

C++Now, Aspen, Colorado, May 13 2015

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer
  - Linux user since early high-school (mid-1999)

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer
  - Linux user since early high-school (mid-1999)
  - 2 years later... programming for money

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer
  - Linux user since early high-school (mid-1999)
  - 2 years later... programming for money
  - 7 years later... programming for fun

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer
  - Linux user since early high-school (mid-1999)
  - 2 years later... programming for money
  - 7 years later... programming for fun
  - everyday C++ coder for the last 6 years

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer
  - Linux user since early high-school (mid-1999)
  - 2 years later... programming for money
  - 7 years later... programming for fun
  - everyday C++ coder for the last 6 years
- FOSS activities:

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer
  - Linux user since early high-school (mid-1999)
  - 2 years later... programming for money
  - 7 years later... programming for fun
  - everyday C++ coder for the last 6 years
- FOSS activities:
  - frequent bug reporting, few patches a year, one Boost review

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer
  - Linux user since early high-school (mid-1999)
  - 2 years later... programming for money
  - 7 years later... programming for fun
  - everyday C++ coder for the last 6 years
- FOSS activities:
  - frequent bug reporting, few patches a year, one Boost review
- recently at work (Faculty of Physics, Univ. Warsaw):

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer
  - Linux user since early high-school (mid-1999)
  - 2 years later... programming for money
  - 7 years later... programming for fun
  - everyday C++ coder for the last 6 years
- FOSS activities:
  - frequent bug reporting, few patches a year, one Boost review
- recently at work (Faculty of Physics, Univ. Warsaw):
  - co-creator of a programming team (see foss.igf.fuw.edu.pl)

few words about myself

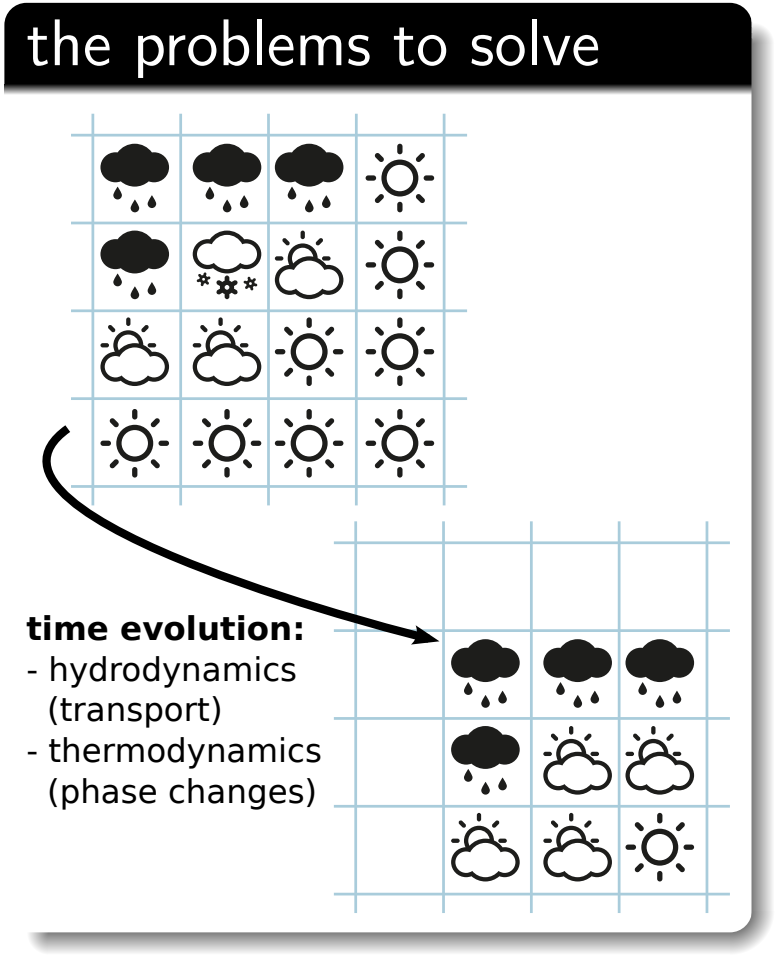
- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer
  - Linux user since early high-school (mid-1999)
  - 2 years later... programming for money
  - 7 years later... programming for fun
  - everyday C++ coder for the last 6 years
- FOSS activities:
  - frequent bug reporting, few patches a year, one Boost review
- recently at work (Faculty of Physics, Univ. Warsaw):
  - co-creator of a programming team (see foss.igf.fuw.edu.pl)
  - lecturer of C++ for undergrad. physics students

few words about myself

- atmospheric cloud physicist (MSc, PhD, postdoc)
- programmer
  - Linux user since early high-school (mid-1999)
  - 2 years later... programming for money
  - 7 years later... programming for fun
  - everyday C++ coder for the last 6 years
- FOSS activities:
  - frequent bug reporting, few patches a year, one Boost review
- recently at work (Faculty of Physics, Univ. Warsaw):
  - co-creator of a programming team (see foss.igf.fuw.edu.pl)
  - lecturer of C++ for undergrad. physics students
  - developer of open-source C++ libraries

- atmospheric cloud physicist
- programmer

- atmospheric cloud physicist
- programmer

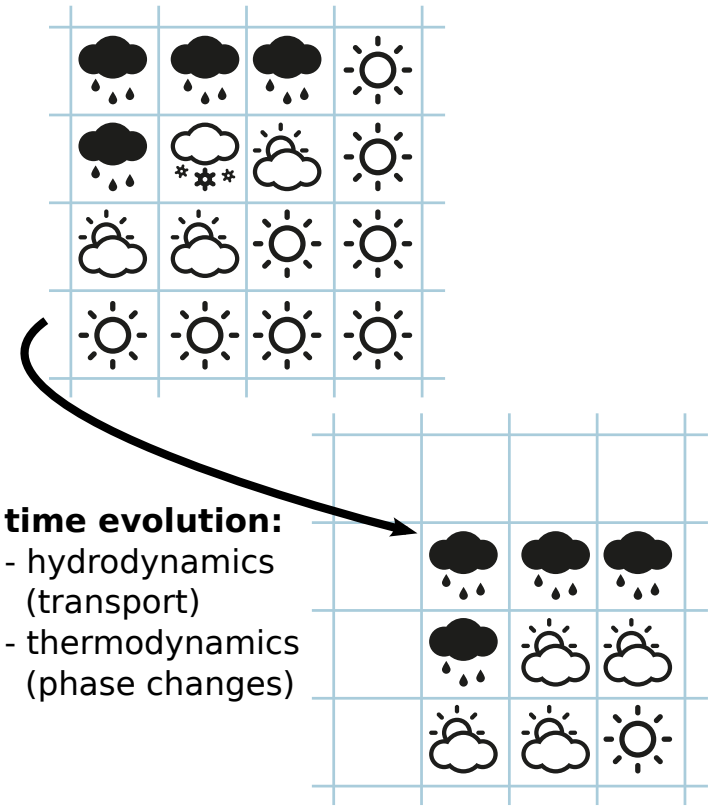


- atmospheric cloud physicist
- programmer

the problems with solving

Nature 467 (2010), doi:10.1038/467775a

the problems to solve



time evolution:
- hydrodynamics (transport)
- thermodynamics (phase changes)

scientific computing in atmospheric sciences

- we are all its users! (e.g., weather prediction)

scientific computing in atmospheric sciences

- we are all its users! (e.g., weather prediction)
- has very long tradition and notorious inertia
  - codes from 80-ties happen to be used till today,
  - it is not uncommon to hear of new codes written in Fortran 77

scientific computing in atmospheric sciences

- we are all its users! (e.g., weather prediction)
- has very long tradition and notorious inertia
  - codes from 80-ties happen to be used till today,
  - it is not uncommon to hear of new codes written in Fortran 77
- is mostly done by reseachers working in a publish-or-perish system in which it is hard to get credit for the time spent on embracing:
  - code reuse
  - unit testing
  - FOSS development model
  - modern coding techniques

scientific computing in atmospheric sciences

- in a longer perspective can benefit from the above through improved:
  - code readability and quality
  - software maintainability
  - result reproducibility

scientific computing in atmospheric sciences

- in a longer perspective can benefit from the above through improved:
  - code readability and quality
  - software maintainability
  - result reproducibility

... I don't have to convince you,
the point is we have convinced a funding agency!

scientific computing in atmospheric sciences

- in a longer perspective can benefit from the above through improved:
  - code readability and quality
  - software maintainability
  - result reproducibility

... I don't have to convince you,
the point is we have convinced a funding agency!

NATIONAL SCIENCE CENTRE
POLAND

- Title: **Aerosol processing by clouds - a multifaceted object-oriented numerical simulation framework**
- Duration: 3 years (till April 2016)
- Partners: ECMWF, Reading, UK & NCAR, Boulder, CO, US
- Budget: 1/4 M€

the team @ the University of Warsaw, Poland

Anna Zimniak | prof. Hanna Pawłowska | Anna Jaruga | Piotr Dziekan | Sylwester Arabas | Maciek Waruszewski

@ NCAR, Boulder, Colorado, USA

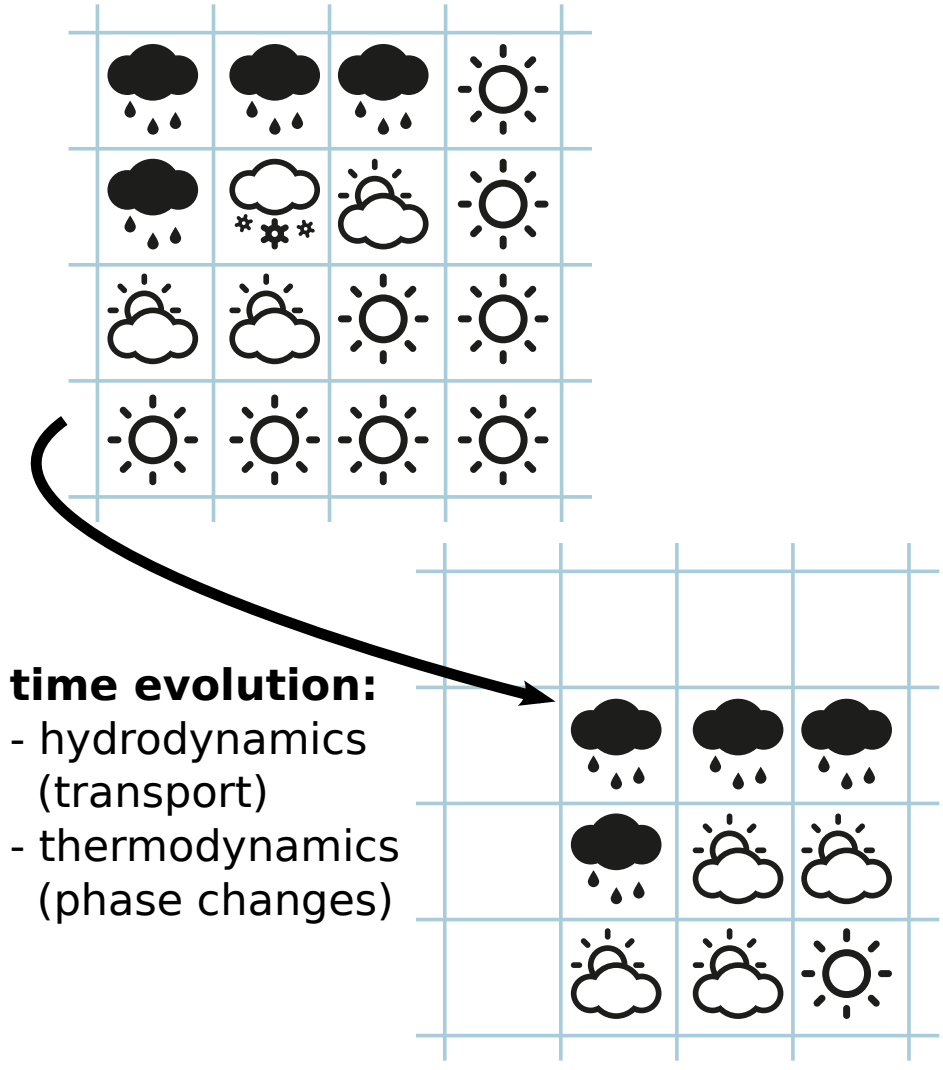
prof. Wojciech Grabowski

Dorota Jarecka

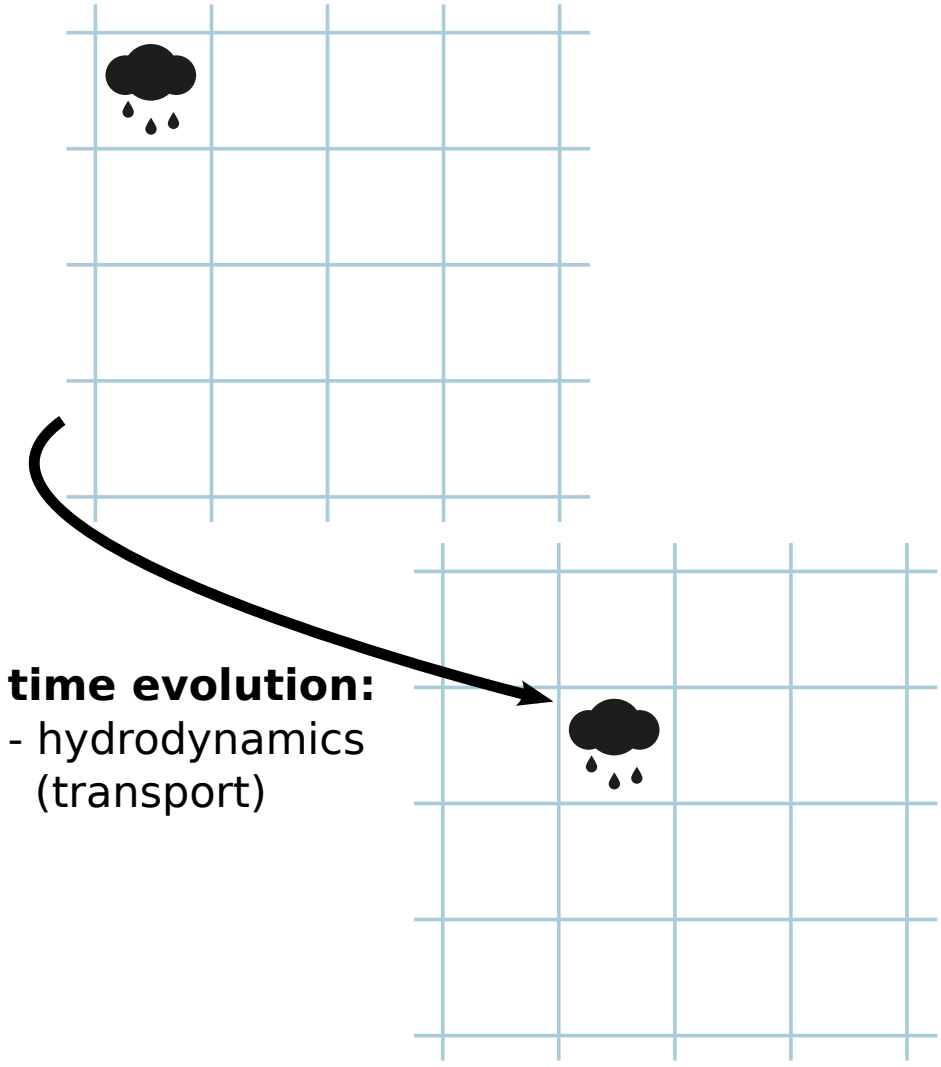
@ ECMWF, Reading, UK

prof. Piotr Smolarkiewicz

a C++ cloud-modelling hello world



a C++ cloud-modelling hello world



a C++ cloud-modelling hello world (non const-optimal, ...)

```
 1 #include <blitz/array.h>
 2
 3 const int n_dims = 2;
 4 using rng_t = blitz::Range;
 5
 6 using arr_t = blitz::Array<double, n_dims>;
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
26
27 // integration
28 int main()
29 {
30   int nx = 4, nz = 4;         // domain size
31
32
33   rng_t i(1, nx), j(1, nz);   // i,j indices
34   arr_t psi(nx+2);            // arrays (+halo)
35
36   psi = 0;                    // initial cond.
37   psi(1,1) = 1;
38   std::cout << psi(i,j) << std::endl;
39
40
41
42
43
44
45
46
47
48
49
50 }
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```cpp
#include <blitz/array.h>

const int n_dims = 2;
using rng_t = blitz::Range;

using arr_t = blitz::Array<double, n_dims>;
```

```
$ clang++ hello.cpp
$ ./a.out
(0,3) x (0,3)
[ 1 0 0 0
  0 0 0 0
  0 0 0 0
  0 0 0 0 ]
```

```cpp

// integration
int main()
{
  int nx = 4, nz = 4;         // domain size


  rng_t i(1, nx), j(1, nz);   // i,j indices
  arr_t psi(nx+2);            // arrays (+halo)

  psi = 0;                    // initial cond.
  psi(1,1) = 1;
  std::cout << psi(i,j) << std::endl;









}
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```
 1 #include <blitz/array.h>
 2
 3 const int n_dims = 2;
 4 using rng_t = blitz::Range;
 5
 6 using arr_t = blitz::Array<double, n_dims>;
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
26
27 // integration
28 int main()
29 {
30   int nx = 4, nz = 4;         // domain size
31
32
33   rng_t i(1, nx), j(1, nz);   // i,j indices
34   arr_t psi(nx+2);            // arrays (+halo)
35
36   psi = 0;                    // initial cond.
37   psi(1,1) = 1;
38   std::cout << psi(i,j) << std::endl;
39
40
41
42
43
44
45
46
47
48
49
50 }
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```
 1 #include <blitz/array.h>
 2
 3 const int n_dims = 2;
 4 using rng_t = blitz::Range;
 5
 6 using arr_t = blitz::Array<double, n_dims>;
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
26
27 // integration
28 int main()
29 {
30   int nx = 4, nz = 4;         // domain size
31
32
33   rng_t i(1, nx), j(1, nz);  // i,j indices
34   arr_t psi(nx+2), tmp(nz+2);// arrays (+halo)
35
36   psi = 0;                    // initial cond.
37   psi(1,1) = 1;
38   std::cout << psi(i,j) << std::endl;
39
40   for (int it=0; it<3; ++it) // time-stepping
41   {
42     tmp(i,j) = psi(i,j)
43       + 0                     // x term
44       + 0;                    // y term
45
46     blitz::cycleArrays(psi, tmp);
47
48     std::cout << psi(i,j) << std::endl;
49   }
50 }
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```
 1 #include <blitz/array.h>
 2
 3 const int n_dims = 2;
 4 using rng_t = blitz::R
 5
 6 using arr_t = blitz::A
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
$ clang++ hello.cpp
$ ./a.out
(0,3) x (0,3)
[ 1 0 0 0
  0 0 0 0
  0 0 0 0
  0 0 0 0 ]

(0,3) x (0,3)
[ 1 0 0 0
  0 0 0 0
  0 0 0 0
  0 0 0 0 ]

(0,3) x (0,3)
[ 1 0 0 0
  0 0 0 0
  0 0 0 0
  0 0 0 0 ]

(0,3) x (0,3)
[ 1 0 0 0
  0 0 0 0
  0 0 0 0
  0 0 0 0 ]
```

```
26
27 // integration
28 int main()
29 {
30   int nx = 4, nz = 4;         // domain size
31
32
33   rng_t i(1, nx), j(1, nz);  // i,j indices
34   arr_t psi(nx+2), tmp(nz+2);// arrays (+halo)
35
36   psi = 0;                    // initial cond.
37   psi(1,1) = 1;
38   std::cout << psi(i,j) << std::endl;
39
40   for (int it=0; it<3; ++it) // time-stepping
41   {
42     tmp(i,j) = psi(i,j)
43       + 0                     // x term
44       + 0;                    // y term
45
46     blitz::cycleArrays(psi, tmp);
47
48     std::cout << psi(i,j) << std::endl;
49   }
50 }
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```
 1 #include <blitz/array.h>
 2
 3 const int n_dims = 2;
 4 using rng_t = blitz::Range;
 5
 6 using arr_t = blitz::Array<double, n_dims>;
 7
 8
 9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

```
26
27 // integration
28 int main()
29 {
30   int nx = 4, nz = 4;         // domain size
31
32
33   rng_t i(1, nx), j(1, nz);  // i,j indices
34   arr_t psi(nx+2), tmp(nz+2);// arrays (+halo)
35
36   psi = 0;                    // initial cond.
37   psi(1,1) = 1;
38   std::cout << psi(i,j) << std::endl;
39
40   for (int it=0; it<3; ++it) // time-stepping
41   {
42     tmp(i,j) = psi(i,j)
43       + 0                     // x term
44       + 0;                    // y term
45
46     blitz::cycleArrays(psi, tmp);
47
48     std::cout << psi(i,j) << std::endl;
49   }
50 }
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```cpp
#include <blitz/array.h>

const int n_dims = 2;
using rng_t = blitz::Range;

using arr_t = blitz::Array<double, n_dims>;
```

```cpp

// integration
int main()
{
  int nx = 4, nz = 4;         // domain size
  double C[2] = {.5, .5};     // wind vector

  rng_t i(1, nx), j(1, nz);   // i,j indices
  arr_t psi(nx+2), tmp(nz+2);// arrays (+halo)

  psi = 0;                    // initial cond.
  psi(1,1) = 1;
  std::cout << psi(i,j) << std::endl;

  for (int it=0; it<3; ++it) // time-stepping
  {
    tmp(i,j) = psi(i,j)
      + f<0>(psi, C, i, j)   // x term
      + 0;                   // y term

    blitz::cycleArrays(psi, tmp);

    std::cout << psi(i,j) << std::endl;
  }
}
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```
 1 #include <blitz/array.h>
 2
 3 const int n_dims = 2;
 4 using rng_t = blitz::Range;
 5
 6 using arr_t = blitz::Array<double, n_dims>;
 7
 8
 9
10
11
12
13
14
15
16
17 // array-valued dimension-independent function
18 template <int d, class arr_t>
19 auto f(arr_t psi, double *C, rng_t i, rng_t j)
20 {
21   return blitz::safeToReturn(
22     -C[d] * psi(      i,   j ) +  // outflow
23     +C[d] * psi(      i-1, j )    // inflow
24   );
25 }
```

```
26
27 // integration
28 int main()
29 {
30   int nx = 4, nz = 4;         // domain size
31   double C[2] = {.5, .5};     // wind vector
32
33   rng_t i(1, nx), j(1, nz);  // i,j indices
34   arr_t psi(nx+2), tmp(nz+2);// arrays (+halo)
35
36   psi = 0;                    // initial cond.
37   psi(1,1) = 1;
38   std::cout << psi(i,j) << std::endl;
39
40   for (int it=0; it<3; ++it) // time-stepping
41   {
42     tmp(i,j) = psi(i,j)
43       + f<0>(psi, C, i, j)   // x term
44       + 0;                   // y term
45
46     blitz::cycleArrays(psi, tmp);
47
48     std::cout << psi(i,j) << std::endl;
49   }
50 }
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```
 1 #include <blitz/array.h>
 2
 3 const int n_dims = 2;
 4 using rng_t = blitz::Range;
 5
 6 using arr_t = blitz::Array<doub
 7
 8
 9
10
11
12
13
14
15
16
17 // array-valued dimension-indep
18 template <int d, class arr_t>
19 auto f(arr_t psi, double *C, rn
20 {
21   return blitz::safeToReturn(
22     -C[d] * psi(      i,   j )
23     +C[d] * psi(      i-1, j )
24   );
25 }
```

```
$ clang++ -std=c++14 hello.cpp
$ ./a.out
(0,3) x (0,3)
[ 1 0 0 0
  0 0 0 0
  0 0 0 0
  0 0 0 0 ]

(0,3) x (0,3)
[ 0.5 0 0 0
  0.5 0 0 0
  0 0 0 0
  0 0 0 0 ]

(0,3) x (0,3)
[ 0.25 0 0 0
  0.5 0 0 0
  0.25 0 0 0
  0 0 0 0 ]

(0,3) x (0,3)
[ 0.125 0 0 0
  0.375 0 0 0
  0.375 0 0 0
  0.125 0 0 0 ]
```

```
tion


4, nz = 4;          // domain size
2] = {.5, .5};     // wind vector

, nx), j(1, nz);   // i,j indices
(nx+2), tmp(nz+2);// arrays (+halo)

                    // initial cond.
= 1;
 << psi(i,j) << std::endl;

it=0; it<3; ++it) // time-stepping

) = psi(i,j)
>(psi, C, i, j)    // x term
                    // y term

cycleArrays(psi, tmp);

ut << psi(i,j) << std::endl;
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```
 1 #include <blitz/array.h>
 2
 3 const int n_dims = 2;
 4 using rng_t = blitz::Range;
 5
 6 using arr_t = blitz::Array<double, n_dims>;
 7
 8
 9
10
11
12
13
14
15
16
17 // array-valued dimension-independent function
18 template <int d, class arr_t>
19 auto f(arr_t psi, double *C, rng_t i, rng_t j)
20 {
21   return blitz::safeToReturn(
22     -C[d] * psi(      i,   j ) +  // outflow
23     +C[d] * psi(      i-1, j )    // inflow
24   );
25 }
```

```
26
27 // integration
28 int main()
29 {
30   int nx = 4, nz = 4;         // domain size
31   double C[2] = {.5, .5};     // wind vector
32
33   rng_t i(1, nx), j(1, nz);  // i,j indices
34   arr_t psi(nx+2), tmp(nz+2);// arrays (+halo)
35
36   psi = 0;                    // initial cond.
37   psi(1,1) = 1;
38   std::cout << psi(i,j) << std::endl;
39
40   for (int it=0; it<3; ++it) // time-stepping
41   {
42     tmp(i,j) = psi(i,j)
43       + f<0>(psi, C, i, j)   // x term
44       + 0;                   // y term
45
46     blitz::cycleArrays(psi, tmp);
47
48     std::cout << psi(i,j) << std::endl;
49   }
50 }
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```cpp
#include <blitz/array.h>

const int n_dims = 2;
using rng_t = blitz::Range;

using arr_t = blitz::Array<double, n_dims>;









// array-valued dimension-independent function
template <int d, class arr_t>
auto f(arr_t psi, double *C, rng_t i, rng_t j)
{
  return blitz::safeToReturn(
    -C[d] * psi(pi<d>(i,   j)) +  // outflow
    +C[d] * psi(pi<d>(i-1, j))    // inflow
  );
}
```

```cpp

// integration
int main()
{
  int nx = 4, nz = 4;         // domain size
  double C[2] = {.5, .5};     // wind vector

  rng_t i(1, nx), j(1, nz);  // i,j indices
  arr_t psi(nx+2), tmp(nz+2);// arrays (+halo)

  psi = 0;                    // initial cond.
  psi(1,1) = 1;
  std::cout << psi(i,j) << std::endl;

  for (int it=0; it<3; ++it) // time-stepping
  {
    tmp(i,j) = psi(i,j)
      + f<0>(psi, C, i, j)   // x term
      + f<1>(psi, C, j, i);  // y term

    blitz::cycleArrays(psi, tmp);

    std::cout << psi(i,j) << std::endl;
  }
}
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```cpp
#include <blitz/array.h>

const int n_dims = 2;
using rng_t = blitz::Range;
using idx_t = blitz::RectDomain<n_dims>;
using arr_t = blitz::Array<double, n_dims>;

// index permutations
template<int d> idx_t pi(rng_t i, rng_t j);

template<> idx_t pi<0> (rng_t i, rng_t j)
{ return idx_t({ i, j}); } // ~\_____ /~   !!!
                           // ./      \.
template<> idx_t pi<1> (rng_t j, rng_t i)
{ return idx_t({ i, j}); }

// array-valued dimension-independent function
template <int d, class arr_t>
auto f(arr_t psi, double *C, rng_t i, rng_t j)
{
  return blitz::safeToReturn(
    -C[d] * psi(pi<d>(i,   j)) +  // outflow
    +C[d] * psi(pi<d>(i-1, j))    // inflow
  );
}

// integration
int main()
{
  int nx = 4, nz = 4;         // domain size
  double C[2] = {.5, .5};     // wind vector

  rng_t i(1, nx), j(1, nz);  // i,j indices
  arr_t psi(nx+2), tmp(nz+2);// arrays (+halo)

  psi = 0;                    // initial cond.
  psi(1,1) = 1;
  std::cout << psi(i,j) << std::endl;

  for (int it=0; it<3; ++it) // time-stepping
  {
    tmp(i,j) = psi(i,j)
      + f<0>(psi, C, i, j)   // x term
      + f<1>(psi, C, j, i);  // y term

    blitz::cycleArrays(psi, tmp);

    std::cout << psi(i,j) << std::endl;
  }
}
```

a C++ cloud-modelling hello world (non const-optimal, ...)

```
 1 #include <blitz/array.h>
 2
 3 const int n_dims = 2;
 4 using rng_t = blitz::Range;
 5 using idx_t = blitz::RectDomain
 6 using arr_t = blitz::Array<dou
 7
 8 // index permutations
 9 template<int d> idx_t pi(rng_t
10
11 template<> idx_t pi<0> (rng_t
12 { return idx_t({ i, j}); } //
13                             //
14 template<> idx_t pi<1> (rng_t
15 { return idx_t({ i, j}); }
16
17 // array-valued dimension-indep
18 template <int d, class arr_t>
19 auto f(arr_t psi, double *C, r
20 {
21   return blitz::safeToReturn(
22     -C[d] * psi(pi<d>(i,   j))
23     +C[d] * psi(pi<d>(i-1, j))
24   );
25 }
```

```
$ clang++ -std=c++14 hello.cpp
$ ./a.out
(0,3) x (0,3)
[ 1 0 0 0
  0 0 0 0
  0 0 0 0
  0 0 0 0 ]

(0,3) x (0,3)
[ 0 0.5 0 0
  0.5 0 0 0
  0 0 0 0
  0 0 0 0 ]

(0,3) x (0,3)
[ 0 0 0.25 0
  0 0.5 0 0
  0.25 0 0 0
  0 0 0 0 ]

(0,3) x (0,3)
[ 0 0 0 0.125
  0 0 0.375 0
  0 0.375 0 0
  0.125 0 0 0 ]
```

```
tion


4, nz = 4;          // domain size
2] = {.5, .5};     // wind vector

, nx), j(1, nz);   // i,j indices
(nx+2), tmp(nz+2);// arrays (+halo)

                    // initial cond.
= 1;
 << psi(i,j) << std::endl;

it=0; it<3; ++it) // time-stepping

) = psi(i,j)
>(psi, C, i, j)    // x term
>(psi, C, j, i);   // y term

cycleArrays(psi, tmp);

ut << psi(i,j) << std::endl;
```

a cloud-modelling hello world: summary

OOP for "blackboard abstractions"

- multidimensional array containers

a cloud-modelling hello world: summary

OOP for "blackboard abstractions"

- multidimensional array containers
- expression-template-based loop-free aritmetics (e.g., Blitz++)

a cloud-modelling hello world: summary

OOP for "blackboard abstractions"

- multidimensional array containers
- expression-template-based loop-free aritmetics (e.g., Blitz++)
- array-valued functions (C++11/14, blitz::safeToReturn)

a cloud-modelling hello world: summary

OOP for "blackboard abstractions"

- multidimensional array containers
- expression-template-based loop-free aritmetics (e.g., Blitz++)
- array-valued functions (C++11/14, blitz::safeToReturn)
- multidimensional OO array indexing (blitz::RectDomain)

a cloud-modelling hello world: summary

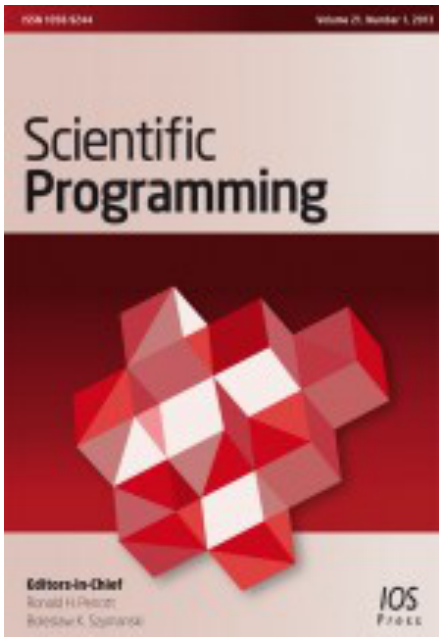
OOP for "blackboard abstractions"

- multidimensional array containers
- expression-template-based loop-free aritmetics (e.g., Blitz++)
- array-valued functions (C++11/14, blitz::safeToReturn)
- multidimensional OO array indexing (blitz::RectDomain)
- permuted-dimension formulæ

a cloud-modelling hello world: summary

OOP for "blackboard abstractions"

- multidimensional array containers
- expression-template-based loop-free aritmetics (e.g., Blitz++)
- array-valued functions (C++11/14, blitz::safeToReturn)
- multidimensional OO array indexing (blitz::RectDomain)
- permuted-dimension formulæ



Formula translation in Blitz++, NumPy and modern Fortran: A case study of the language choice tradeoffs

Sylwester Arabas[1], Dorota Jarecka[1], Anna Jaruga[1], Maciej Fijałkowski[2]

[1]Institute of Geophysics, Faculty of Physics, University of Warsaw
[2]PyPy Team

Journal	Scientific Programming
DOI	10.3233/SPR-140379
Online Date	Monday, March 24, 2014

plan of the talk

introduction

libmpdata++

libcloudph++

n-dim array containers

plan of the talk

introduction

libmpdata++

libcloudph++

n-dim array containers

libmpdata++

http://libmpdataxx.igf.fuw.edu.pl/

going from hello-world to real-world

going from hello-world to real-world

- multiple dimensions

going from hello-world to real-world

- multiple dimensions
- $\mathrm{f}()$ grows to few kLOC (per dimension)

going from hello-world to real-world

- multiple dimensions
- $\mathrm{f}()$ grows to few kLOC (per dimension)
- multiple transported fields

going from hello-world to real-world

- multiple dimensions
- $\mathrm{f}()$ grows to few kLOC (per dimension)
- multiple transported fields
- boundary conditions

going from hello-world to real-world

- multiple dimensions
- $\mathrm{f}()$ grows to few kLOC (per dimension)
- multiple transported fields
- boundary conditions
- concurrency logic

going from hello-world to real-world

- multiple dimensions
- $\mathrm{f}()$ grows to few kLOC (per dimension)
- multiple transported fields
- boundary conditions
- concurrency logic
- input/output

going from hello-world to real-world

- multiple dimensions
- $\mathrm{f}()$ grows to few kLOC (per dimension)
- multiple transported fields
- boundary conditions
- concurrency logic
- input/output
- more physics $\rightsquigarrow$ fluid dynamics, phase changes

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): $>$600 citations

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): $>$600 citations
  - Google Scholar: $\sim$ 700 research papers

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): $>$600 citations
  - Google Scholar: $\sim$ 700 research papers
  - Google Books: $\sim$ 200 mentions in books

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): >600 citations
  - Google Scholar: $\sim$ 700 research papers
  - Google Books: $\sim$ 200 mentions in books

- original single-file Fortran 77 implementation used till today:

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): >600 citations
  - Google Scholar: $\sim$ 700 research papers
  - Google Books: $\sim$ 200 mentions in books

- original single-file Fortran 77 implementation used till today:
  - unspecified license, no versioning

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): >600 citations
  - Google Scholar: $\sim$ 700 research papers
  - Google Books: $\sim$ 200 mentions in books

- original single-file Fortran 77 implementation used till today:
  - unspecified license, no versioning
  - e-mail distribution

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): >600 citations
  - Google Scholar: $\sim$ 700 research papers
  - Google Books: $\sim$ 200 mentions in books

- original single-file Fortran 77 implementation used till today:
  - unspecified license, no versioning
  - e-mail distribution
  - copy-paste-modify reuse

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): >600 citations
  - Google Scholar: $\sim$ 700 research papers
  - Google Books: $\sim$ 200 mentions in books
- original single-file Fortran 77 implementation used till today:
  - unspecified license, no versioning
  - e-mail distribution
  - copy-paste-modify reuse
- **libmpdata++**: a new Blitz++ based implementation:

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): >600 citations
  - Google Scholar: $\sim$ 700 research papers
  - Google Books: $\sim$ 200 mentions in books
- original single-file Fortran 77 implementation used till today:
  - unspecified license, no versioning
  - e-mail distribution
  - copy-paste-modify reuse
- **libmpdata++**: a new Blitz++ based implementation:
  - an over order-of-magnitude lower number of lines of code,

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): >600 citations
  - Google Scholar: $\sim$ 700 research papers
  - Google Books: $\sim$ 200 mentions in books
- original single-file Fortran 77 implementation used till today:
  - unspecified license, no versioning
  - e-mail distribution
  - copy-paste-modify reuse
- **libmpdata++**: a new Blitz++ based implementation:
  - an over order-of-magnitude lower number of lines of code,
  - comparable performance,

MPDATA algorithm family (Piotr Smolarkiewicz et al.)

One of the established methods for solving transport problems

- some statistics from Google:
  - first MPDATA article (Smolarkiewicz, 1984): >600 citations
  - Google Scholar: $\sim$ 700 research papers
  - Google Books: $\sim$ 200 mentions in books
- original single-file Fortran 77 implementation used till today:
  - unspecified license, no versioning
  - e-mail distribution
  - copy-paste-modify reuse
- **libmpdata++**: a new Blitz++ based implementation:
  - an over order-of-magnitude lower number of lines of code,
  - comparable performance,
  - major improvement in reusability and maintainability

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

dependencies

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

dependencies

- C++11

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

dependencies

- C++11
- Blitz++

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

dependencies

- C++11
- Blitz++
- Boost (ptr_container, timer, thread, preprocessor, filesystem, format, property_tree)

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

dependencies

- C++11
- Blitz++
- Boost (ptr_container, timer, thread, preprocessor, filesystem, format, property_tree)
- CMake, CTest

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

dependencies

- C++11
- Blitz++
- Boost (ptr_container, timer, thread, preprocessor, filesystem, format, property_tree)
- CMake, CTest

API

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

dependencies

- C++11
- Blitz++
- Boost (ptr_container, timer, thread, preprocessor, filesystem, format, property_tree)
- CMake, CTest

API

- header-only library

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

dependencies

- C++11
- Blitz++
- Boost (ptr_container, timer, thread, preprocessor, filesystem, format, property_tree)
- CMake, CTest

API

- header-only library
- template-based component selection

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

dependencies

- C++11
- Blitz++
- Boost (ptr_container, timer, thread, preprocessor, filesystem, format, property_tree)
- CMake, CTest

API

- header-only library
- template-based component selection
- inheritance-based component extensions

libmpdata++: some design choices

- license: GPL
- repo: github.com/igfuw/

library components

- solvers/algorithms:
  - ...
- boundary conditions:
  - ...
- output handlers:
  - HDF5
  - HDF5 + XDMF
  - gnuplot
- concurrency handlers:
  - OpenMP
  - Boost.Thread
  - C++11 threads

dependencies

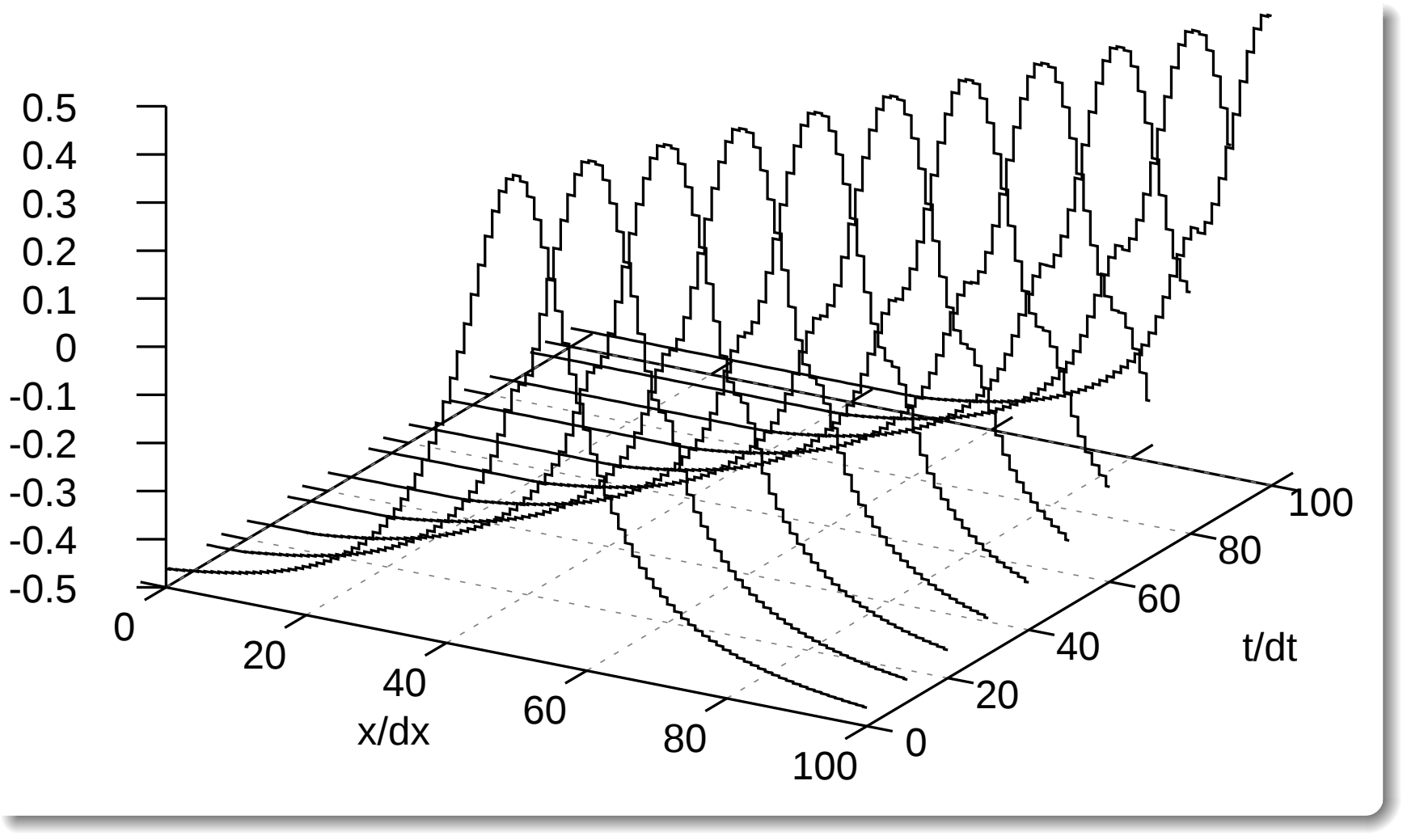
- C++11
- Blitz++
- Boost (ptr_container, timer, thread, preprocessor, filesystem, format, property_tree)
- CMake, CTest

API

- header-only library
- template-based component selection
- inheritance-based component extensions
- user exposed to Blitz++ API

libmpdata++: hello world

libmpdata++: hello world



libmpdata++: hello world

hello.cpp 1/2

```
 1 #include <libmpdata++/solvers/mpdata.hpp>
 2 #include <libmpdata++/concurr/serial.hpp>
 3 #include <libmpdata++/output/gnuplot.hpp>
 4 using namespace libmpdataxx;
 5
 6 int main()
 7 {
 8   struct ct_params_t : ct_params_default_t // compile-time parameters
 9   {
10     using real_t = double;
11     enum { n_dims = 1 };
12     enum { n_eqns = 1 };
13     enum { opts = opts::fct };
14   };
15
16   using solver_t = output::gnuplot<        // solver & ouput choice
17     solvers::mpdata<ct_params_t>
18   >;
```

. . .

libmpdata++: hello world

hello.cpp 2/2

. . .

```
19
20    typename solver_t::rt_params_t p;          // run-time parameters
21    p.grid_size = { 101 };
22    p.outfreq = 10;
23
24    concurr::serial<                           // concurrency choice
25      solver_t,                                // solver defined above
26      bcond::open, bcond::open                 // left, right boundary cond.
27    > run(p);
28
29    blitz::firstIndex i;                       // initial condition
30    run.advectee() = -.5 + 1 / (
31      pow(0.1 * (i - (p.grid_size[0]-1)/2.), 2) + 1
32    );
33
34    run.advector() = .5;                       // Courant number
35    run.advance(100);                          // integration (100 timesteps)
36  }
```

libmpdata++: hello world

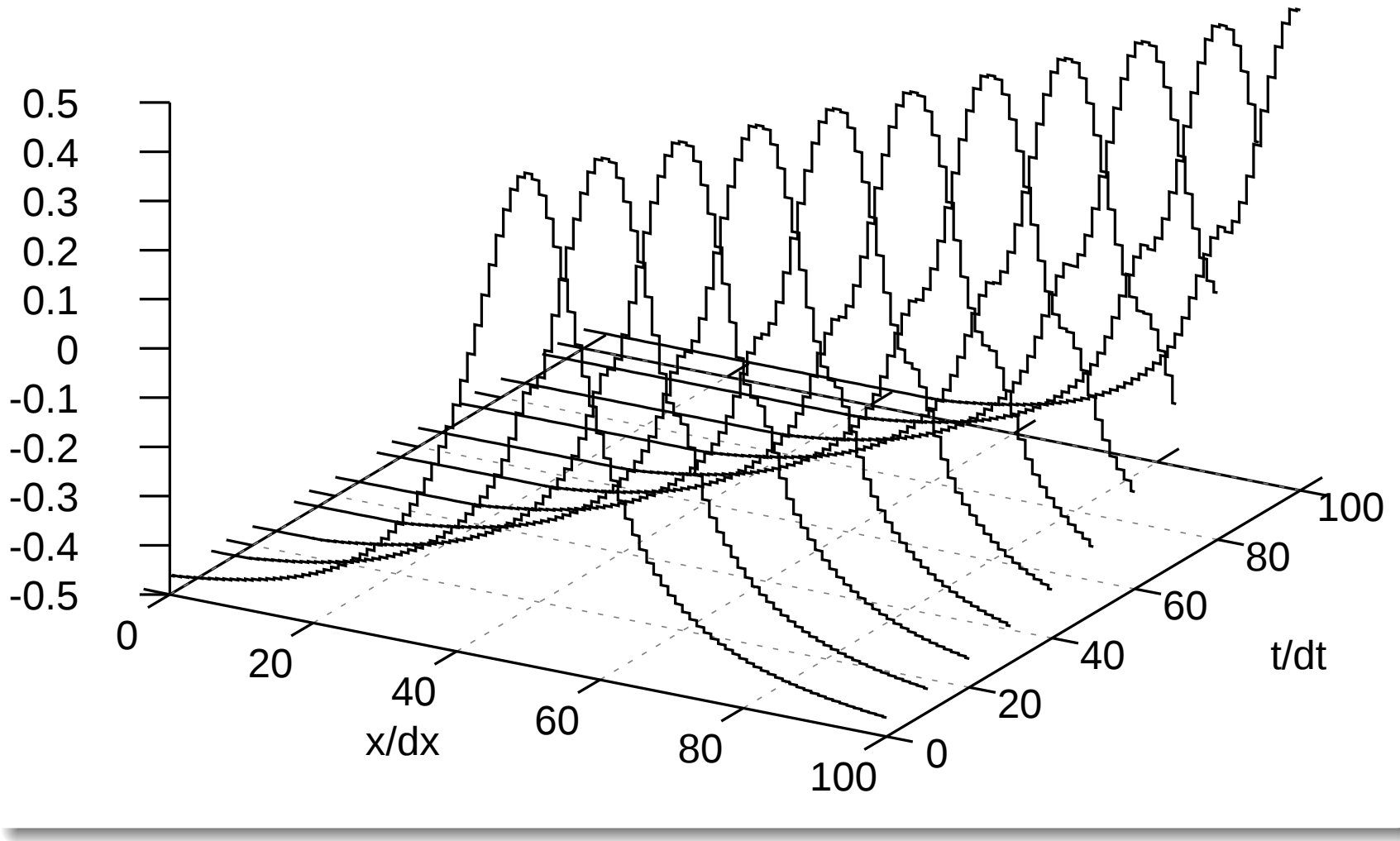
CMakeLists.txt

```
1 cmake_minimum_required(VERSION 3.0)
2 project(hello CXX)
3 find_package(libmpdata++)
4 set(CMAKE_CXX_FLAGS ${libmpdataxx_CXX_FLAGS_RELEASE})
5 add_executable(hello hello.cpp)
6 target_link_libraries(hello ${libmpdataxx_LIBRARIES})
```

```
$ cmake .
-- Configuring done
-- Generating done
-- Build files have been written to: ...
$ make
[100%] Building CXX object CMakeFiles/hello.dir/hello.cpp.o
Linking CXX executable hello
[100%] Built target hello
$ ./hello
 wall time: 0.0177881s user time: 0.02s system time: 0s
```

libmpdata++: hello world

out.svg



libmpdata++: solver/algorithm hierarchy

libmpdata++: solver/algorithm hierarchy

$$\partial_t(G\psi) + \nabla \cdot (G\vec{u}\psi) = 0$$

homogeneous advection

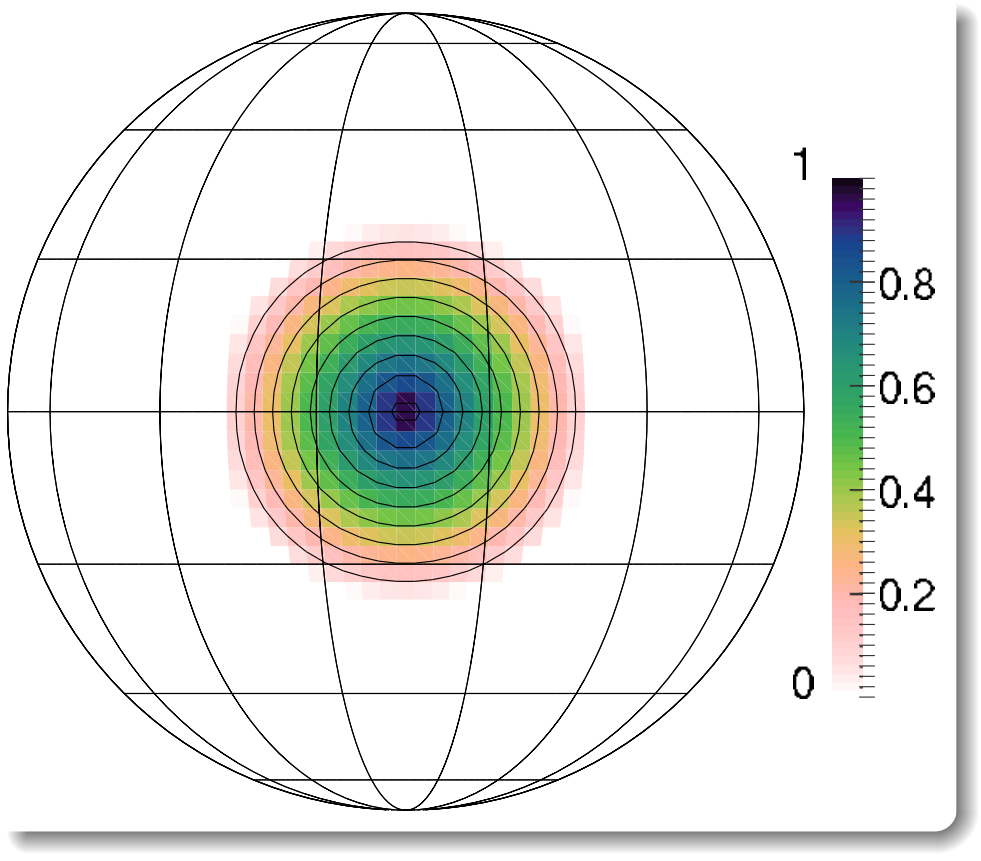
libmpdata++: solver/algorithm hierarchy

$$\partial_t(G\psi) + \nabla \cdot (G\vec{u}\psi) = 0$$

homogeneous advection

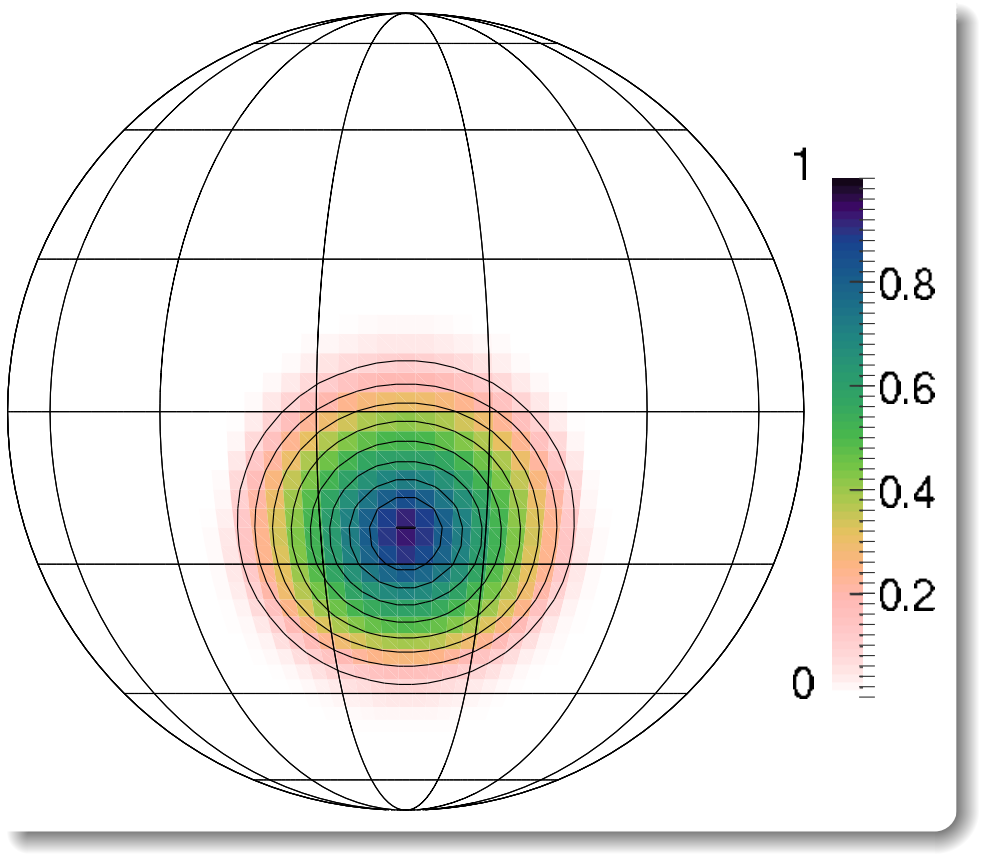
user/test code

libmpdata++: 2D advection on a sphere example



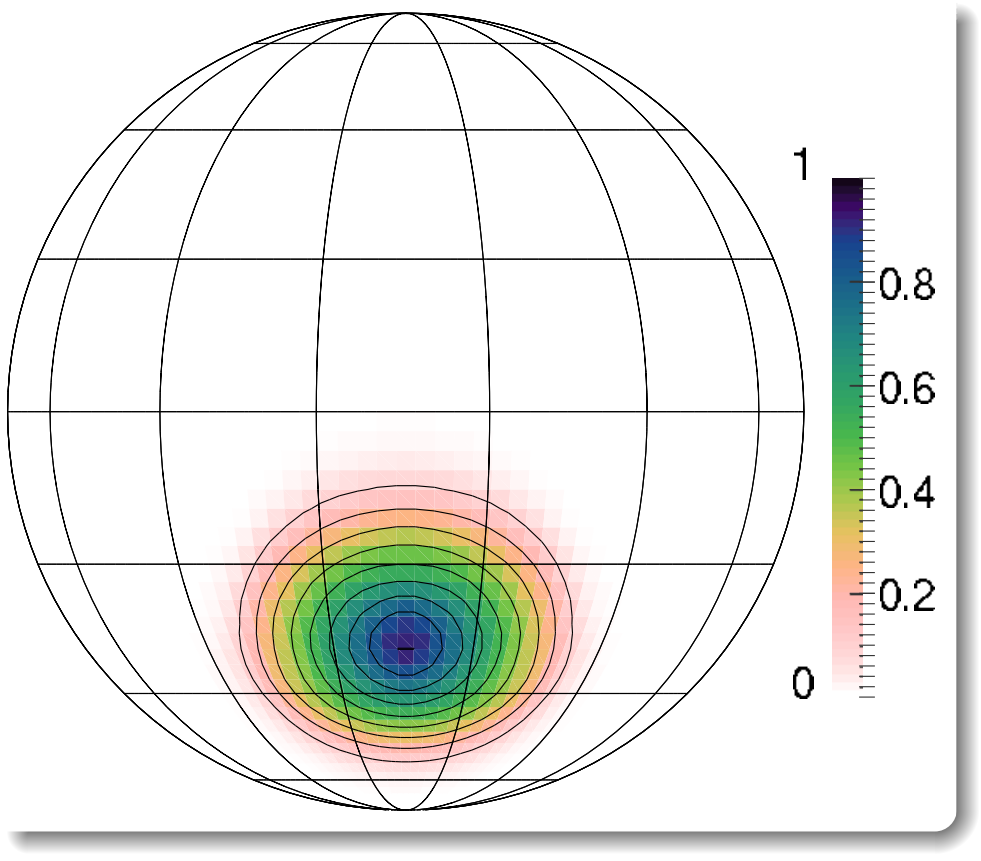
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



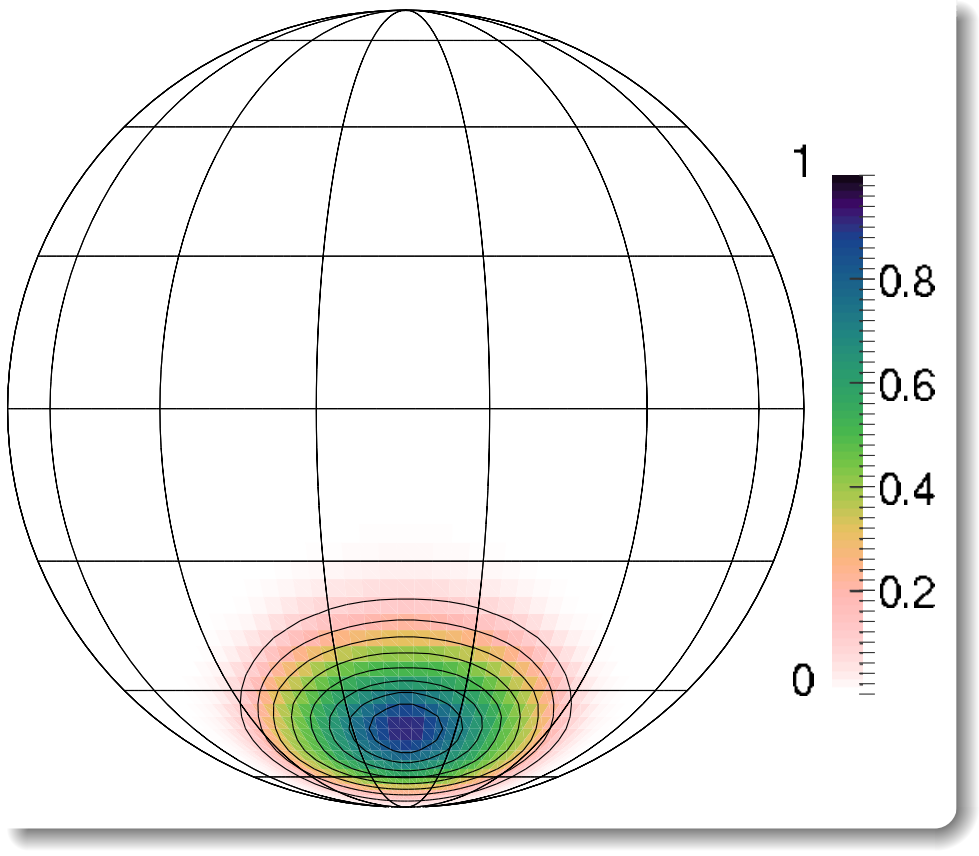
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



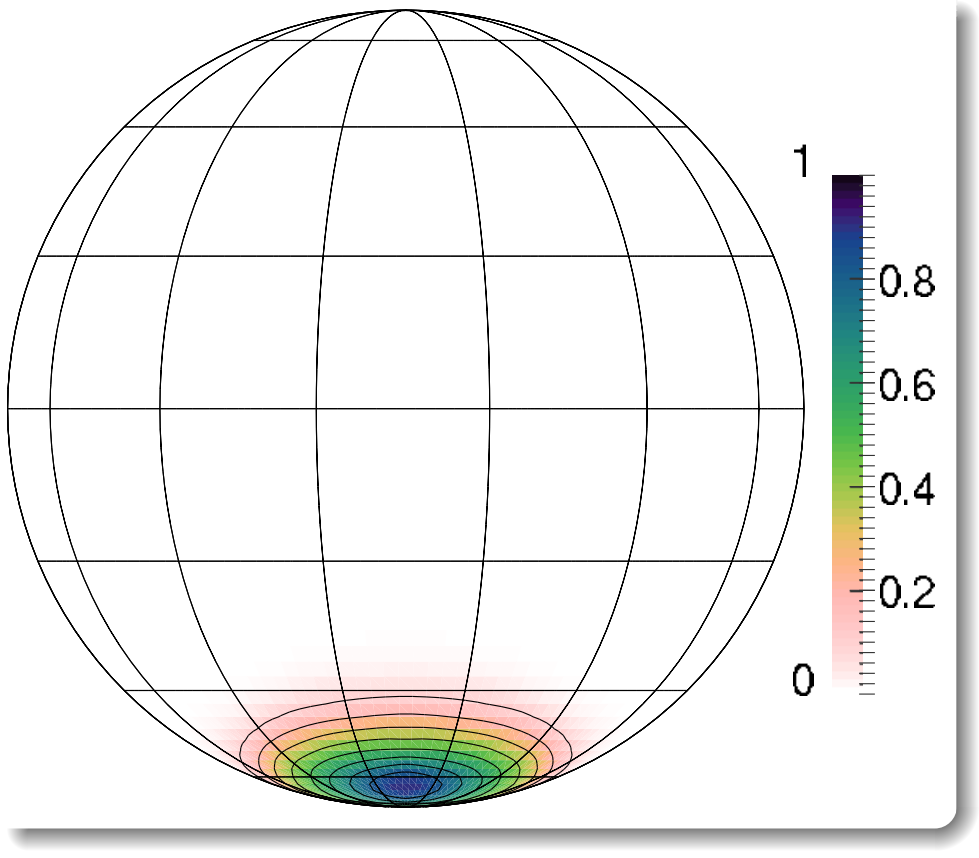
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



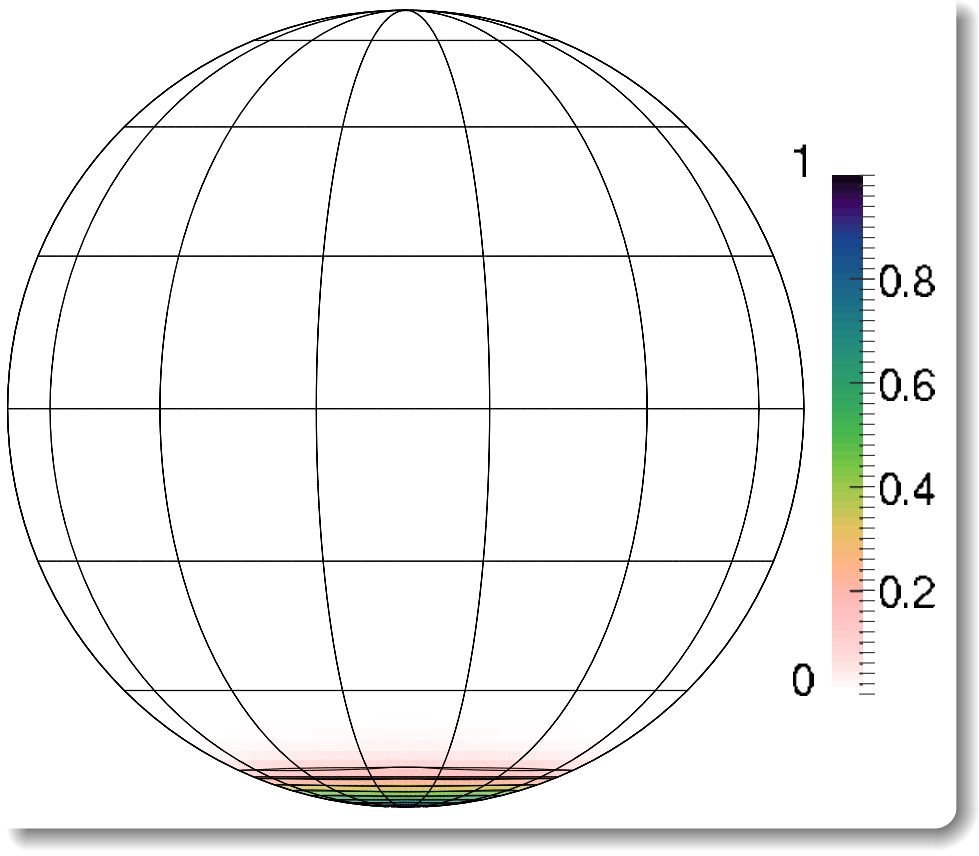
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



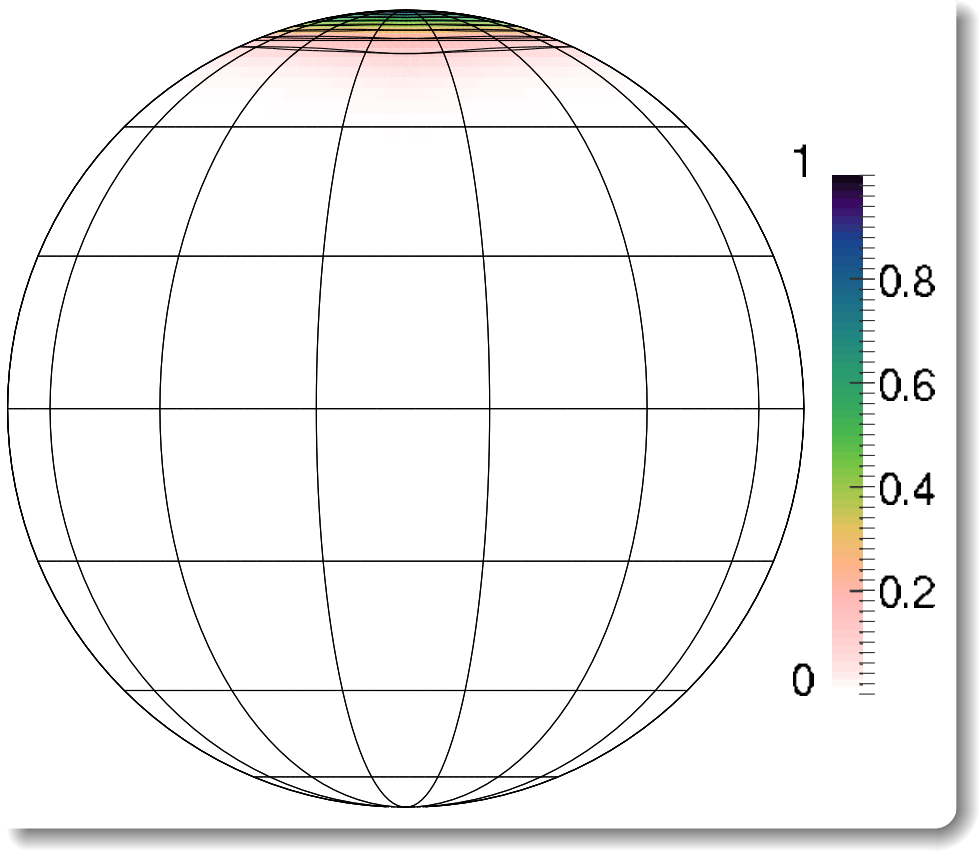
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



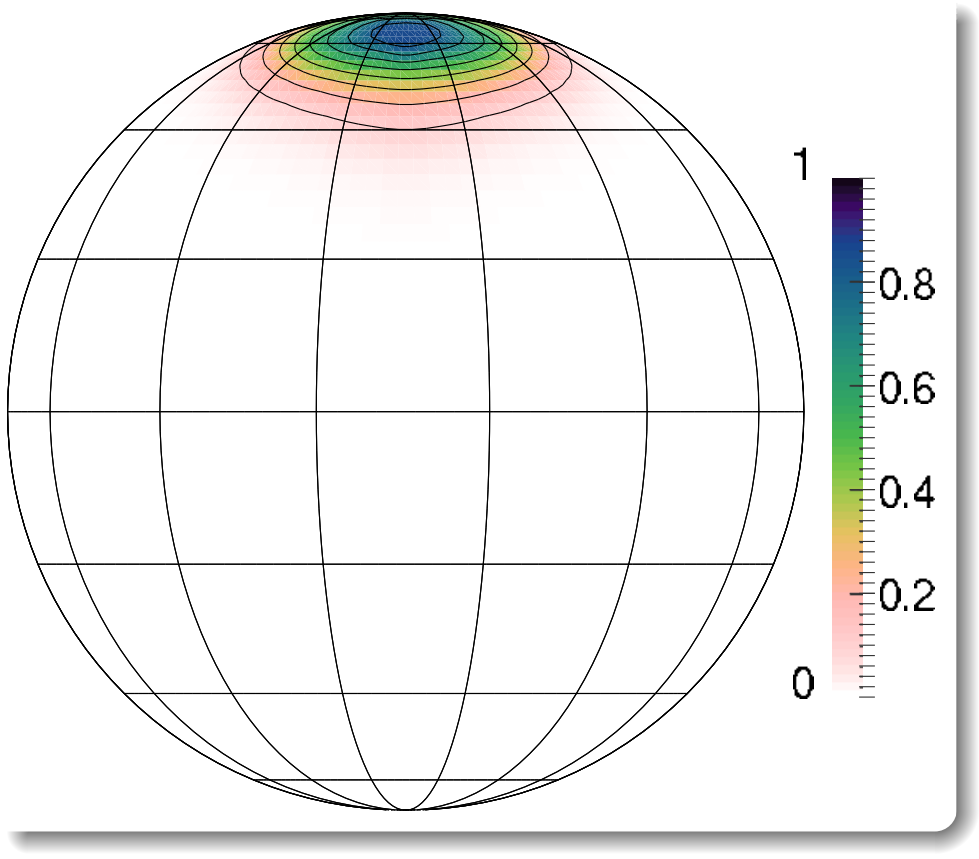
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



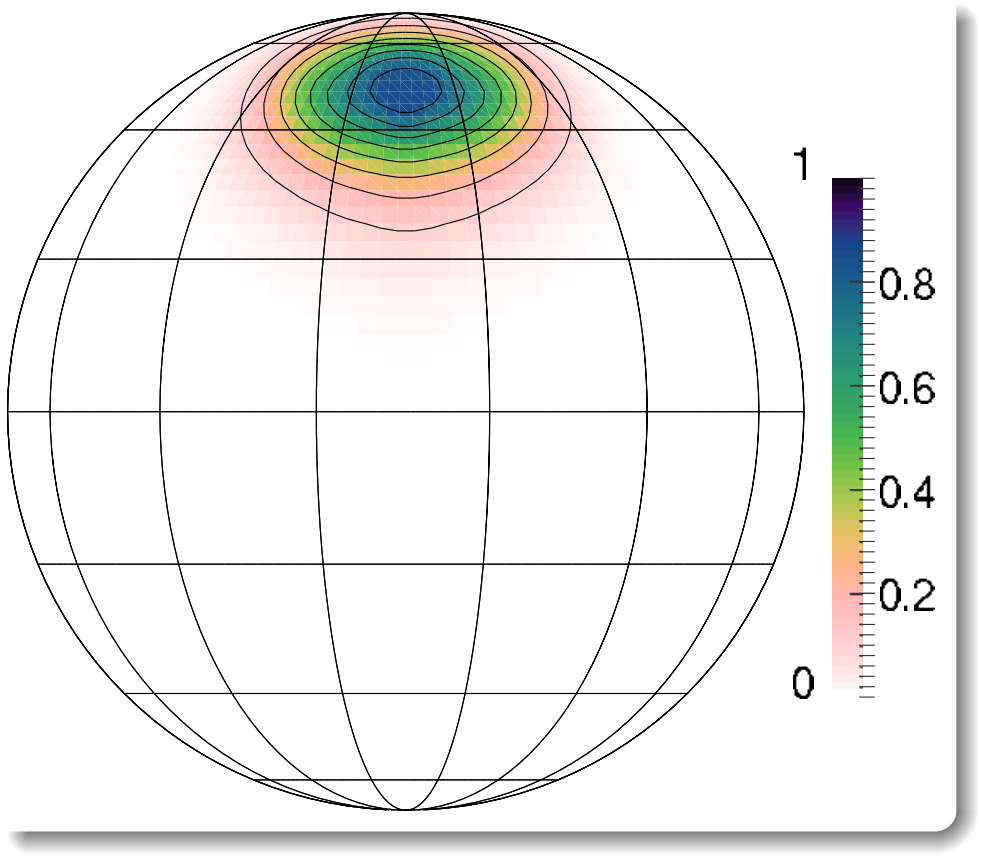
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



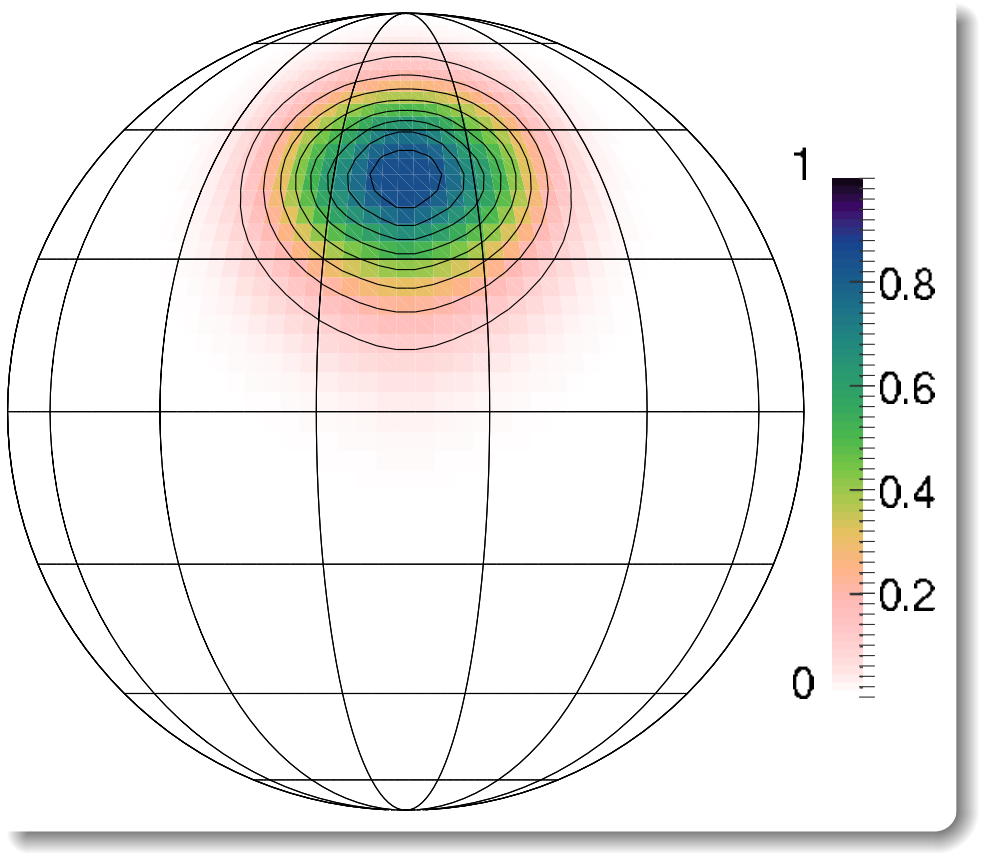
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



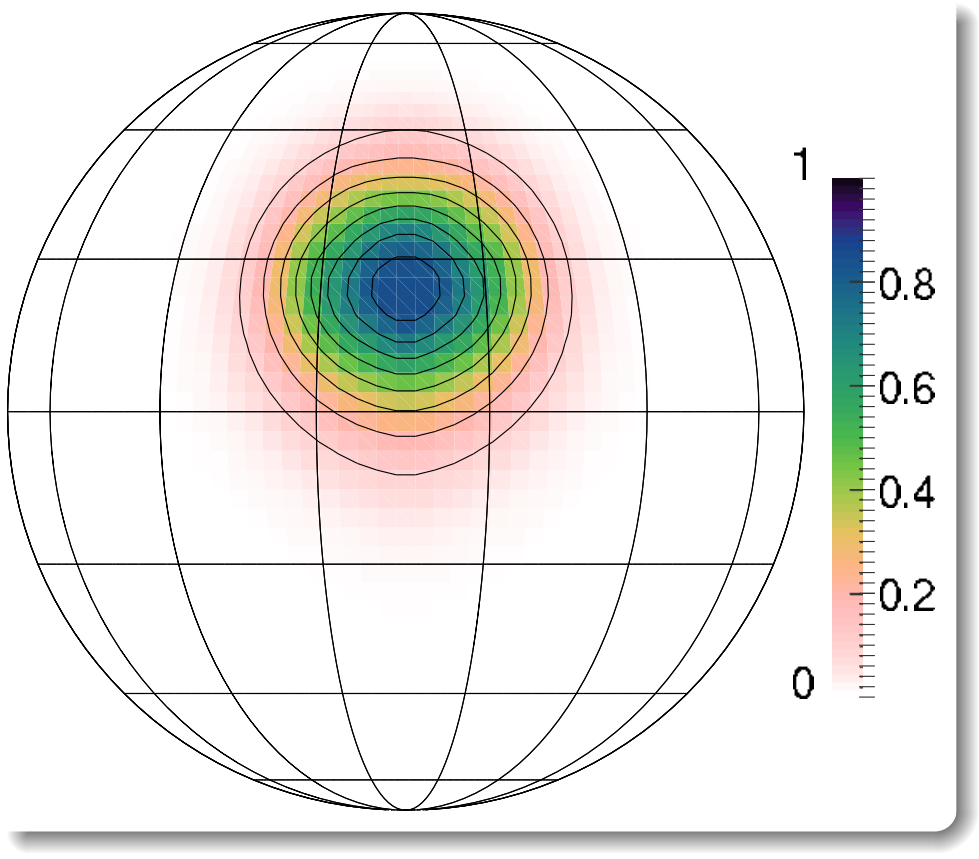
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



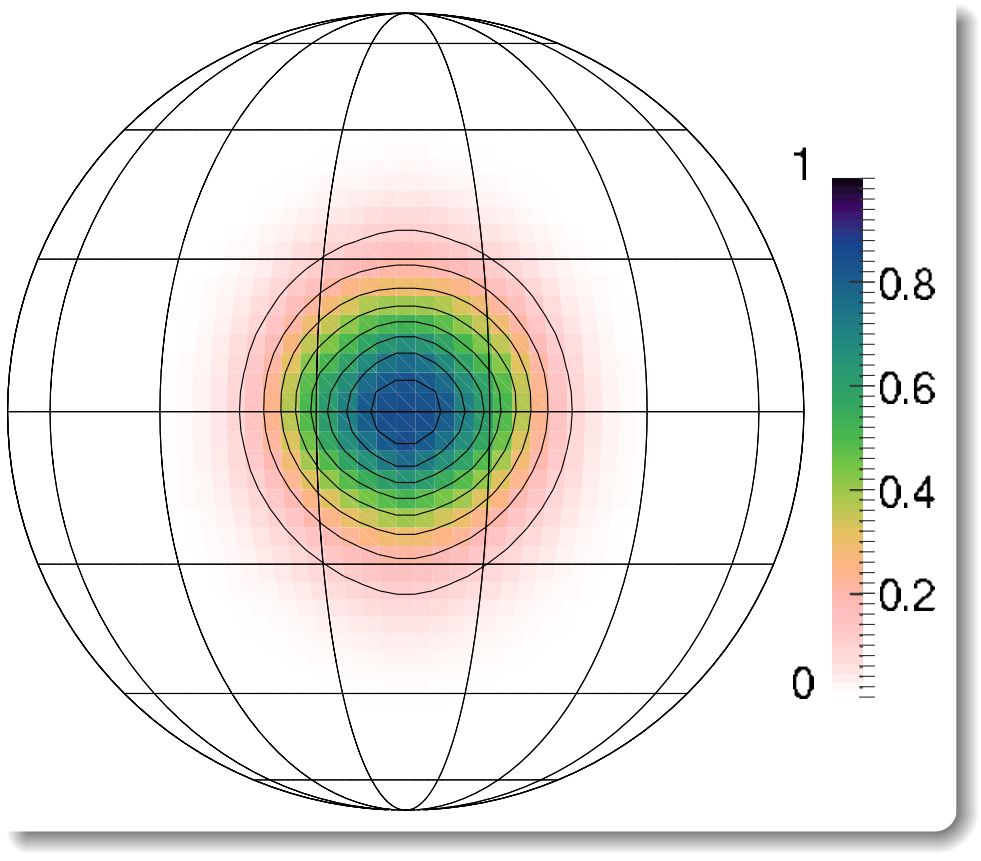
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



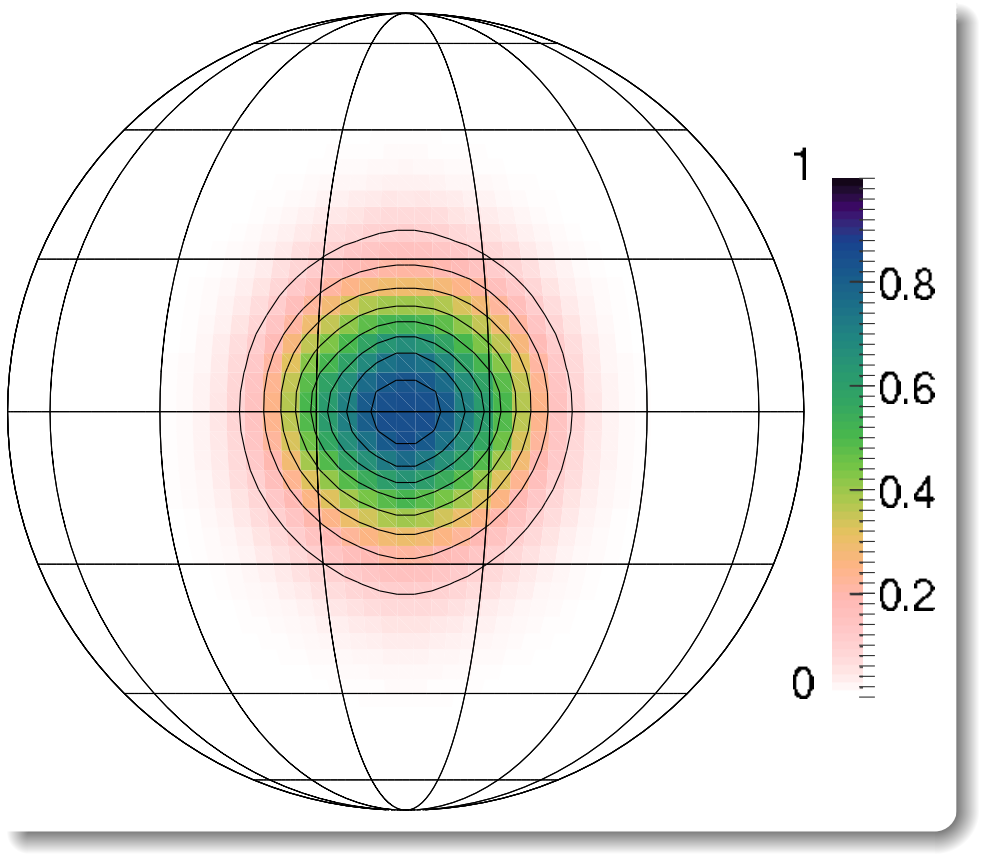
- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



- reproduced experiment of Williamson and Rasch, 1989

libmpdata++: 2D advection on a sphere example



- reproduced experiment of Williamson and Rasch, 1989
- <100 lines of code with libmpdata++

`https://github.com/igfuw/libmpdataxx/tree/master/tests/paper_2015_GMD/5_over_the_pole_2d`

libmpdata++: solver/algorithm hierarchy

$$\partial_t(G\psi) + \nabla \cdot (G\vec{u}\psi) = 0$$

homogeneous advection

user/test code

libmpdata++: solver/algorithm hierarchy

$$\partial_t(G\psi) + \nabla \cdot (G\vec{u}\psi) = 0$$

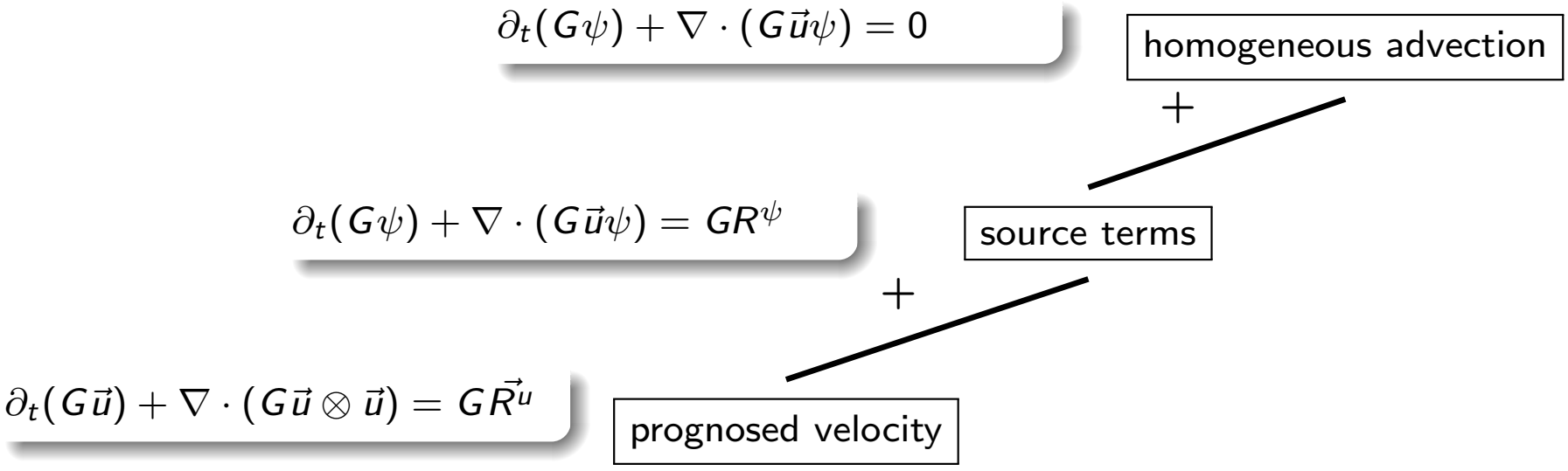
homogeneous advection

+

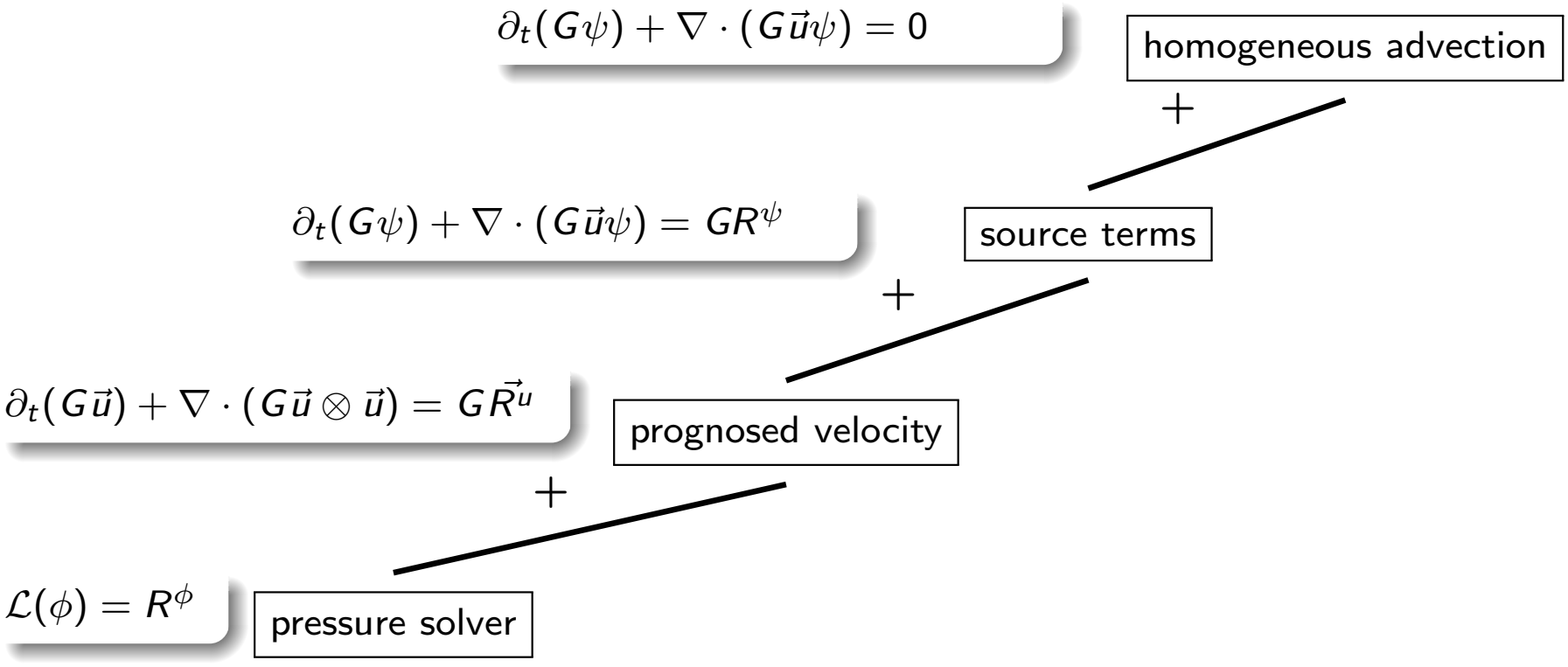
$$\partial_t(G\psi) + \nabla \cdot (G\vec{u}\psi) = GR^\psi$$

source terms

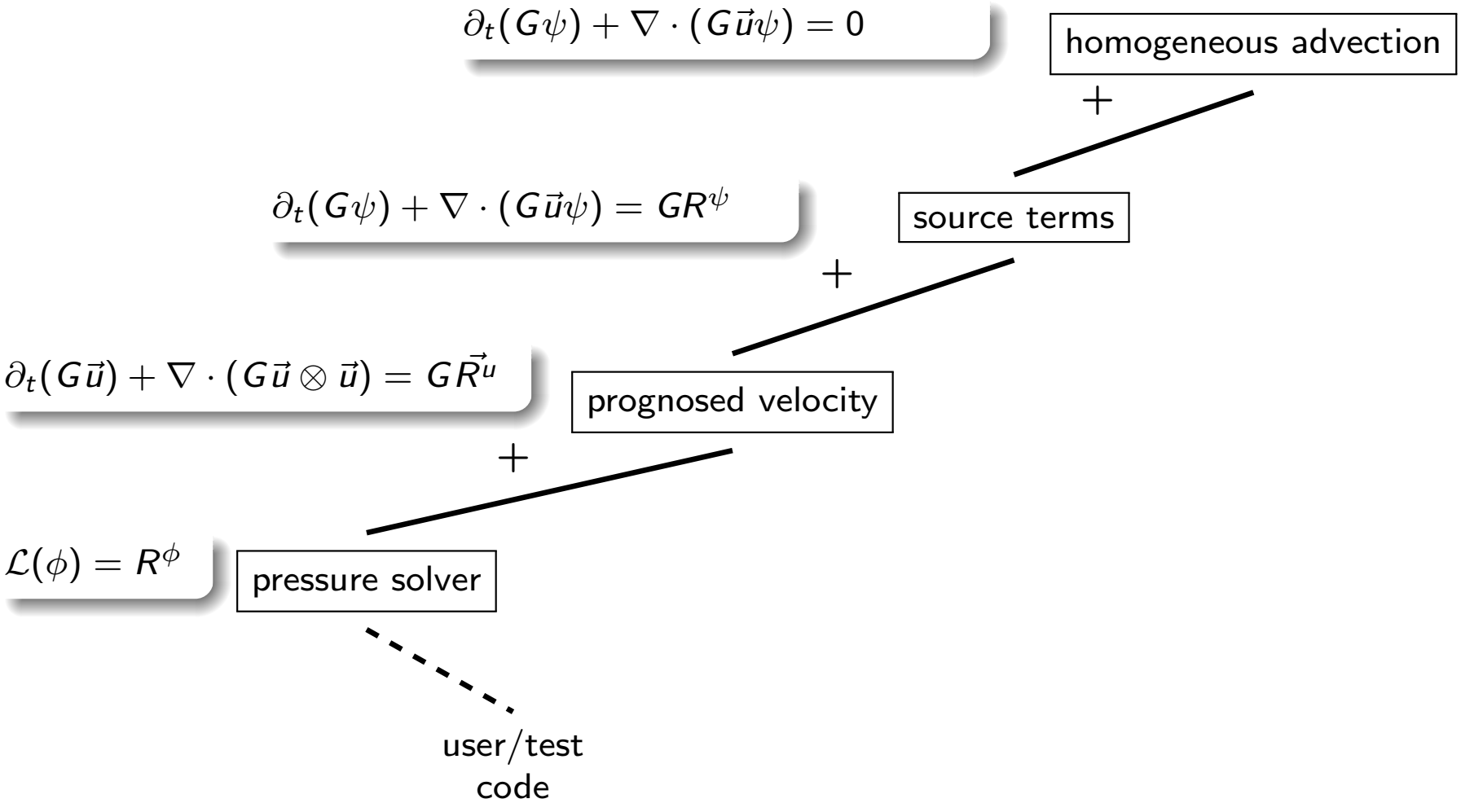
libmpdata++: solver/algorithm hierarchy



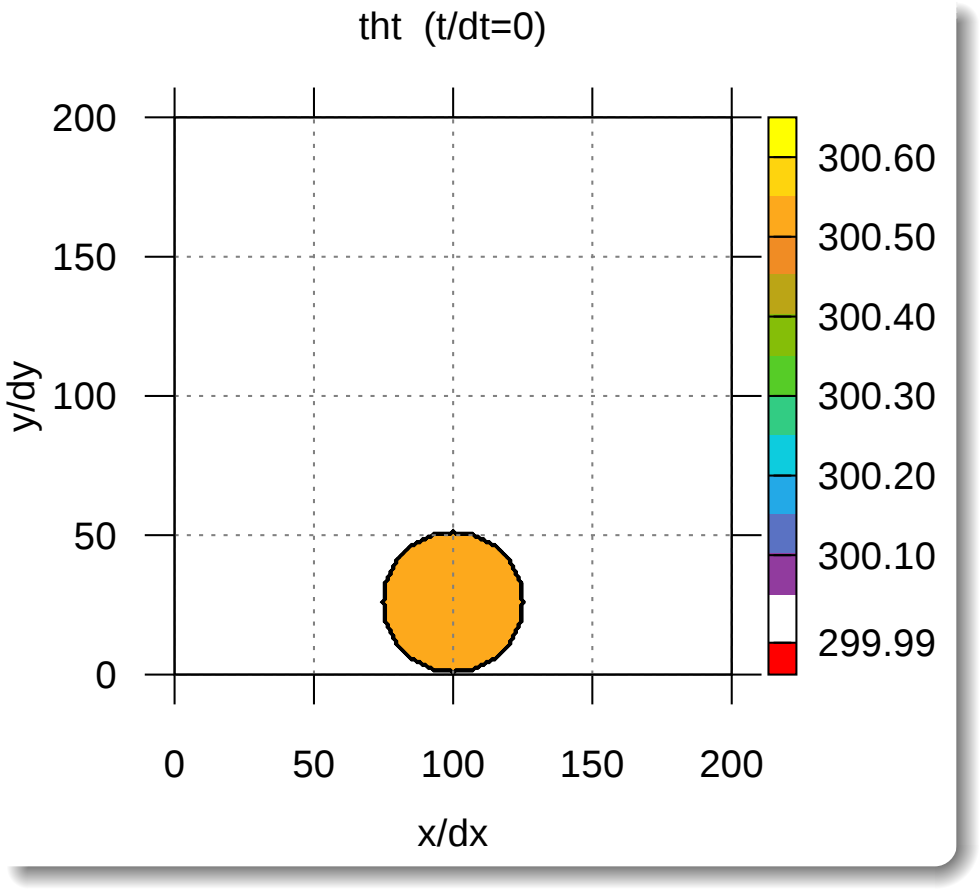
libmpdata++: solver/algorithm hierarchy



libmpdata++: solver/algorithm hierarchy

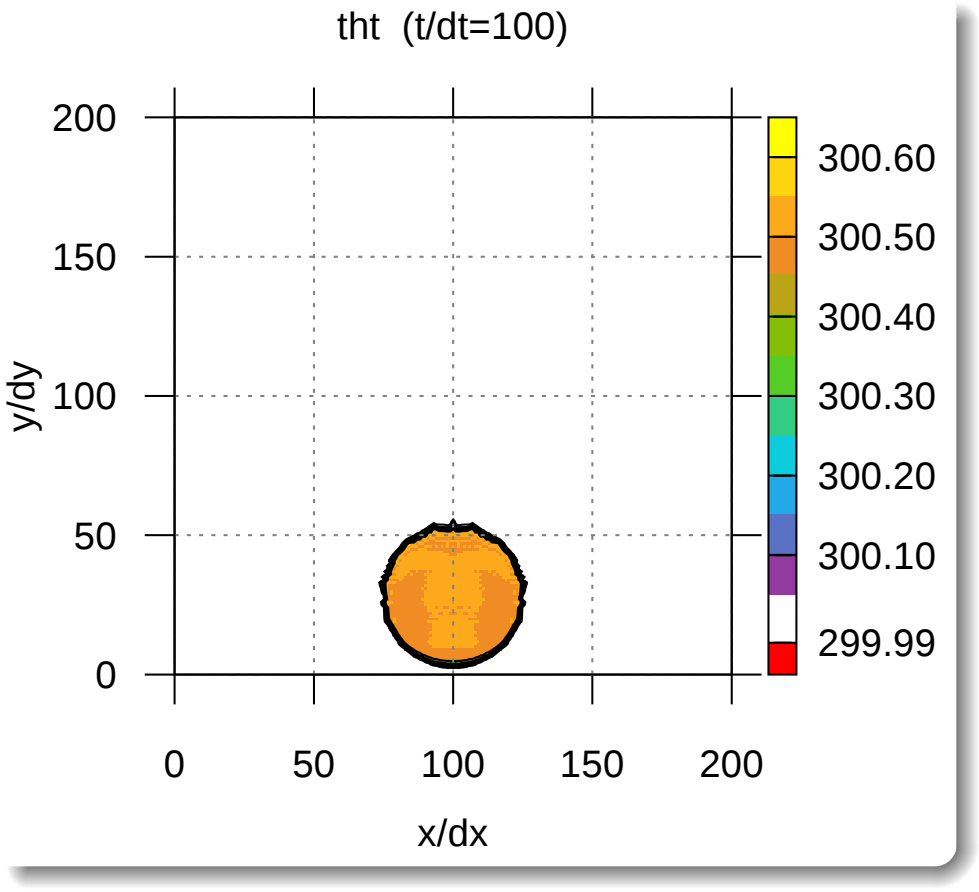


libmpdata++: 2D Boussinesq convection example



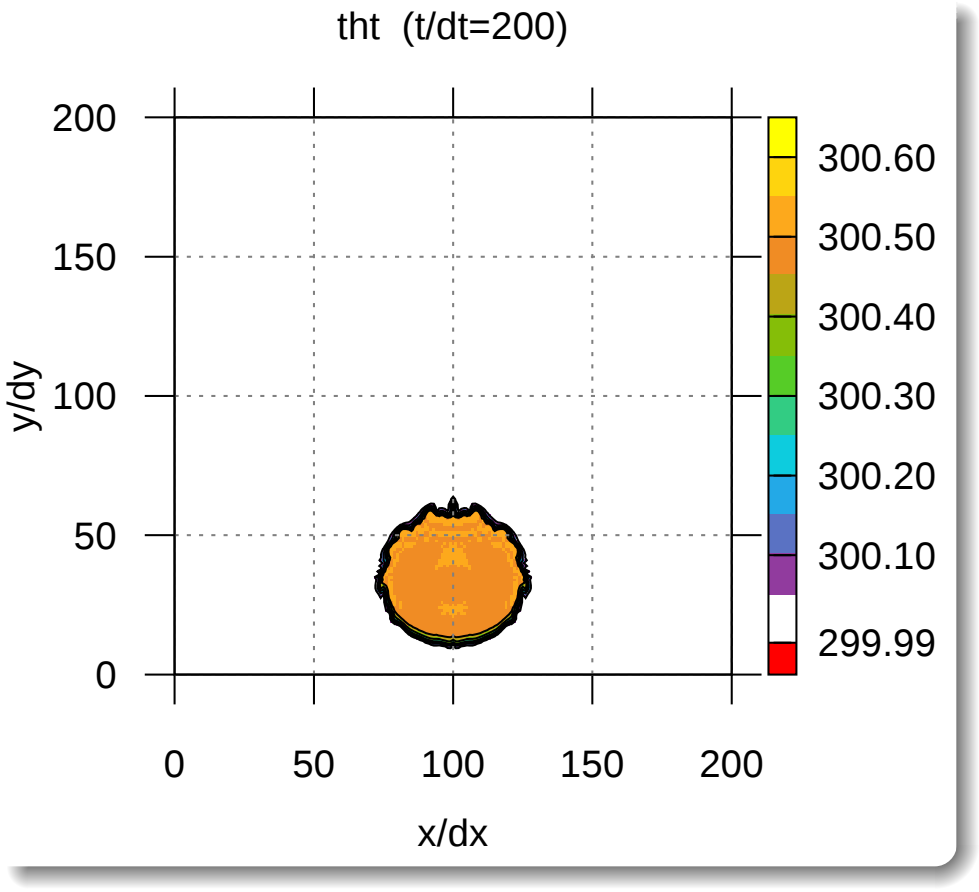
- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992

libmpdata++: 2D Boussinesq convection example



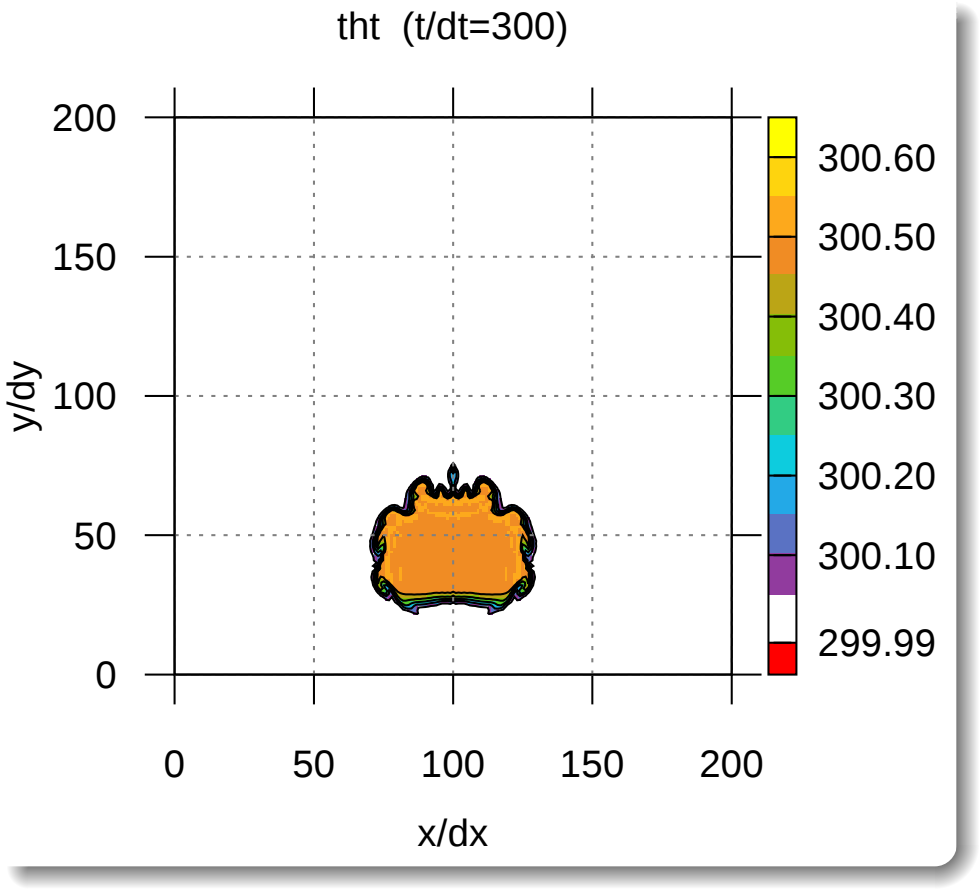
- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992

libmpdata++: 2D Boussinesq convection example



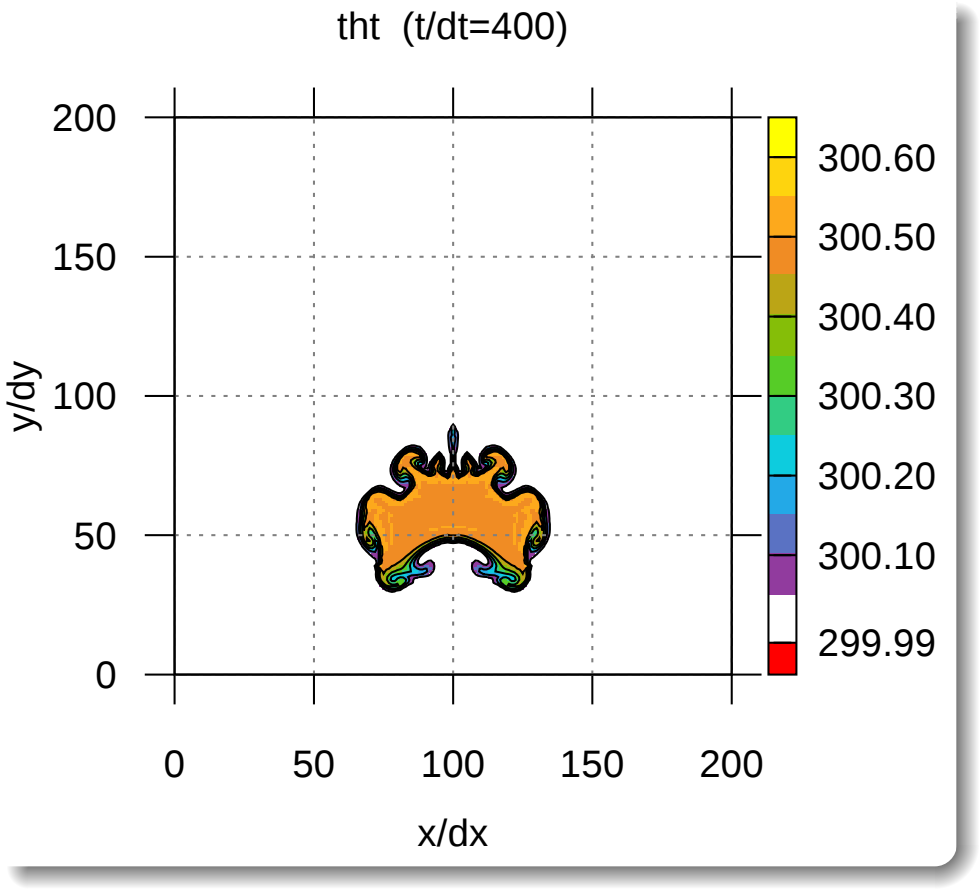
- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992

libmpdata++: 2D Boussinesq convection example



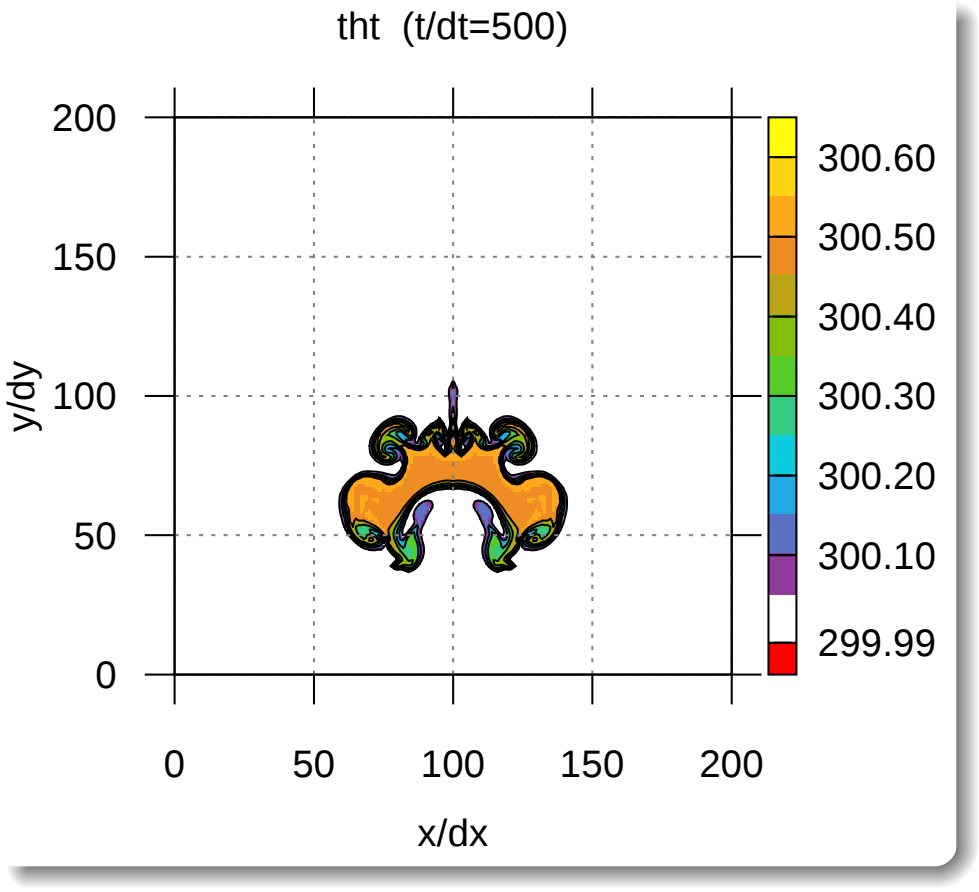
- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992

libmpdata++: 2D Boussinesq convection example



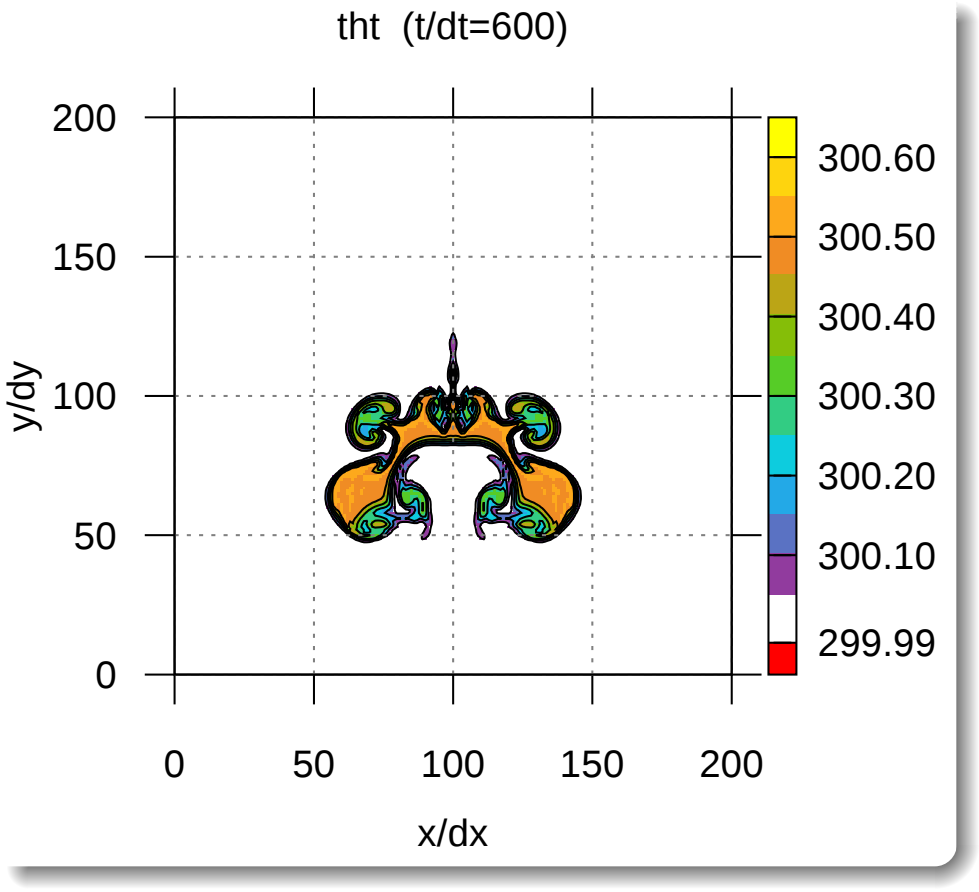
- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992

libmpdata++: 2D Boussinesq convection example



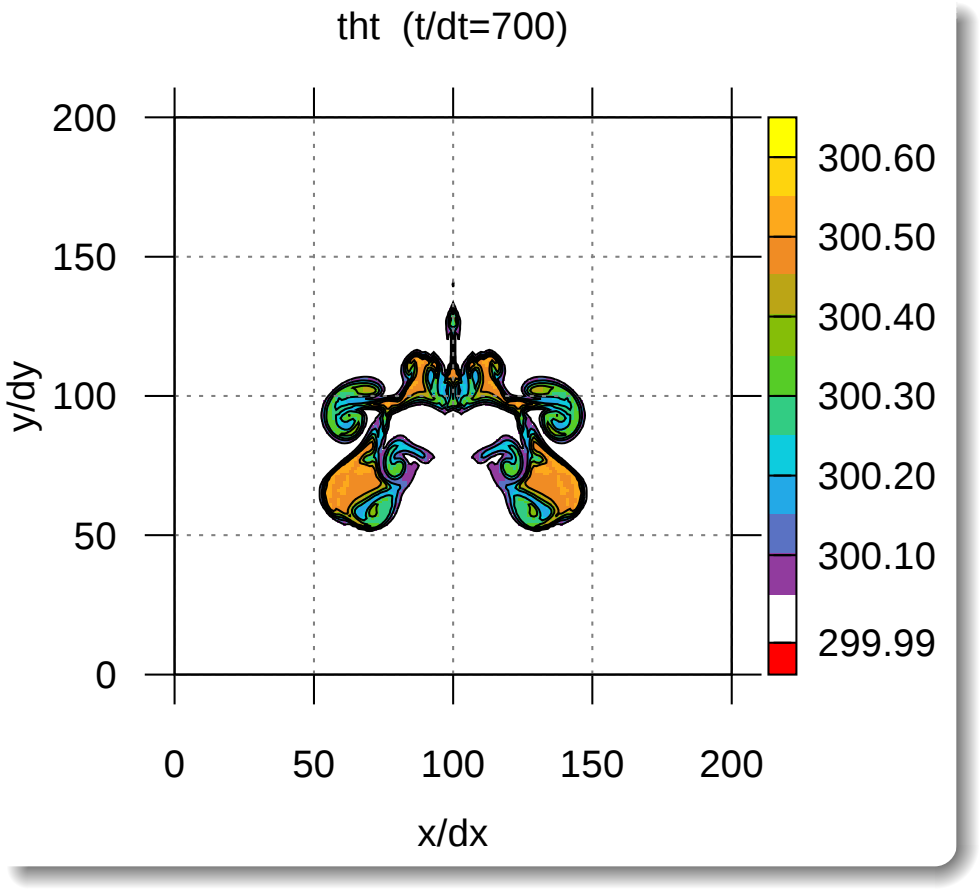
- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992

libmpdata++: 2D Boussinesq convection example



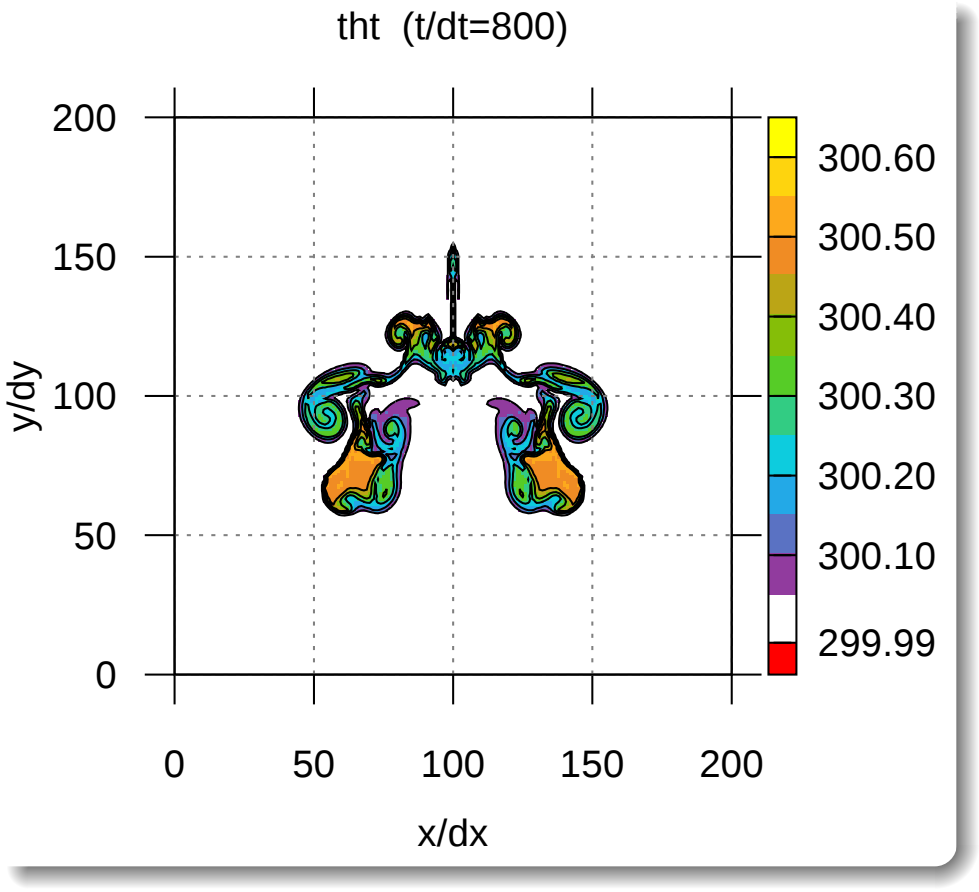
- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992

libmpdata++: 2D Boussinesq convection example



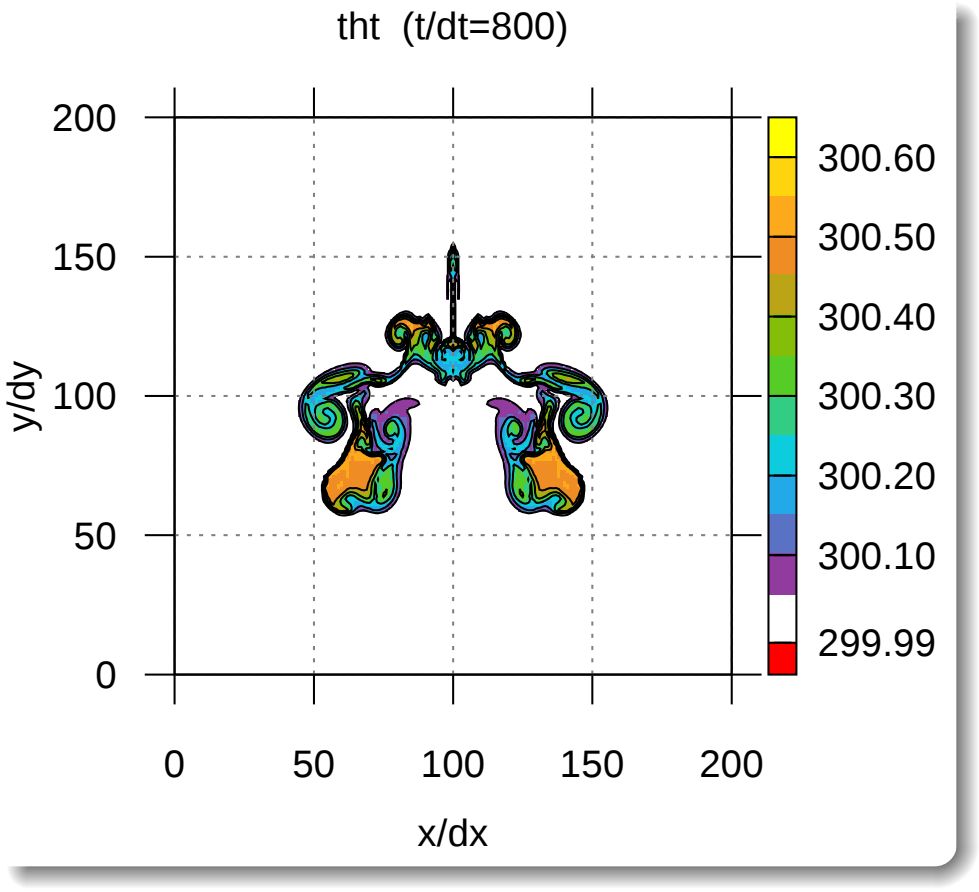
- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992

libmpdata++: 2D Boussinesq convection example



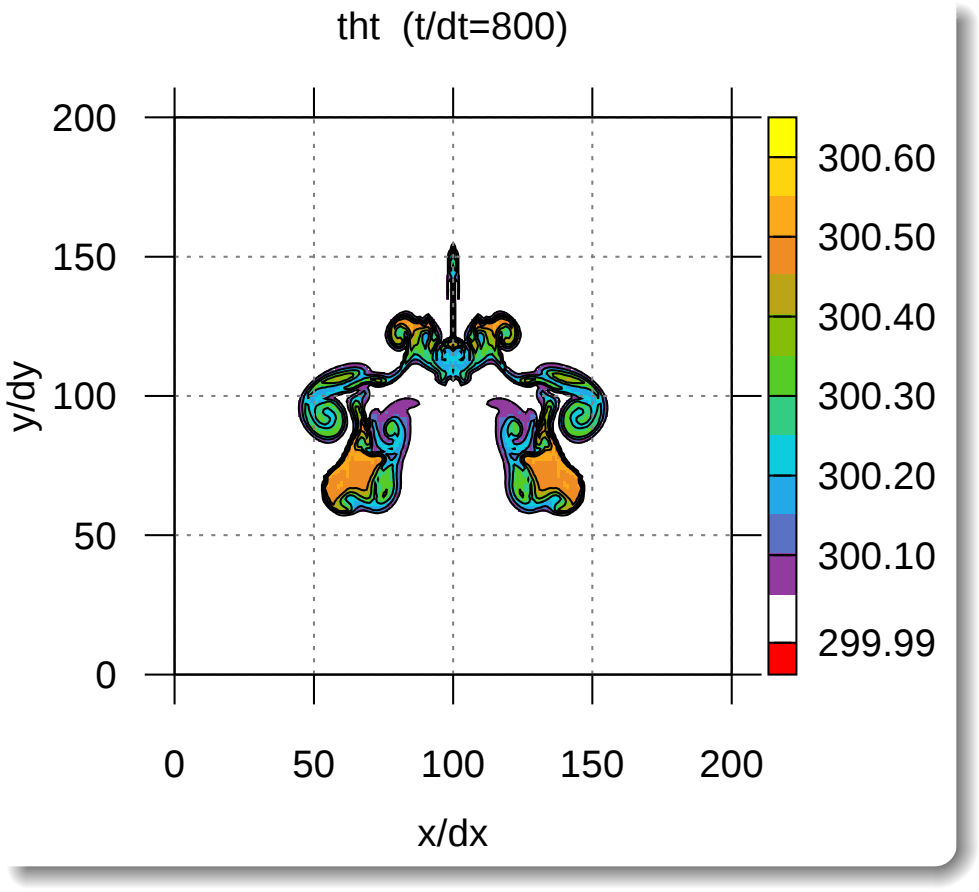
- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992

libmpdata++: 2D Boussinesq convection example



- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992

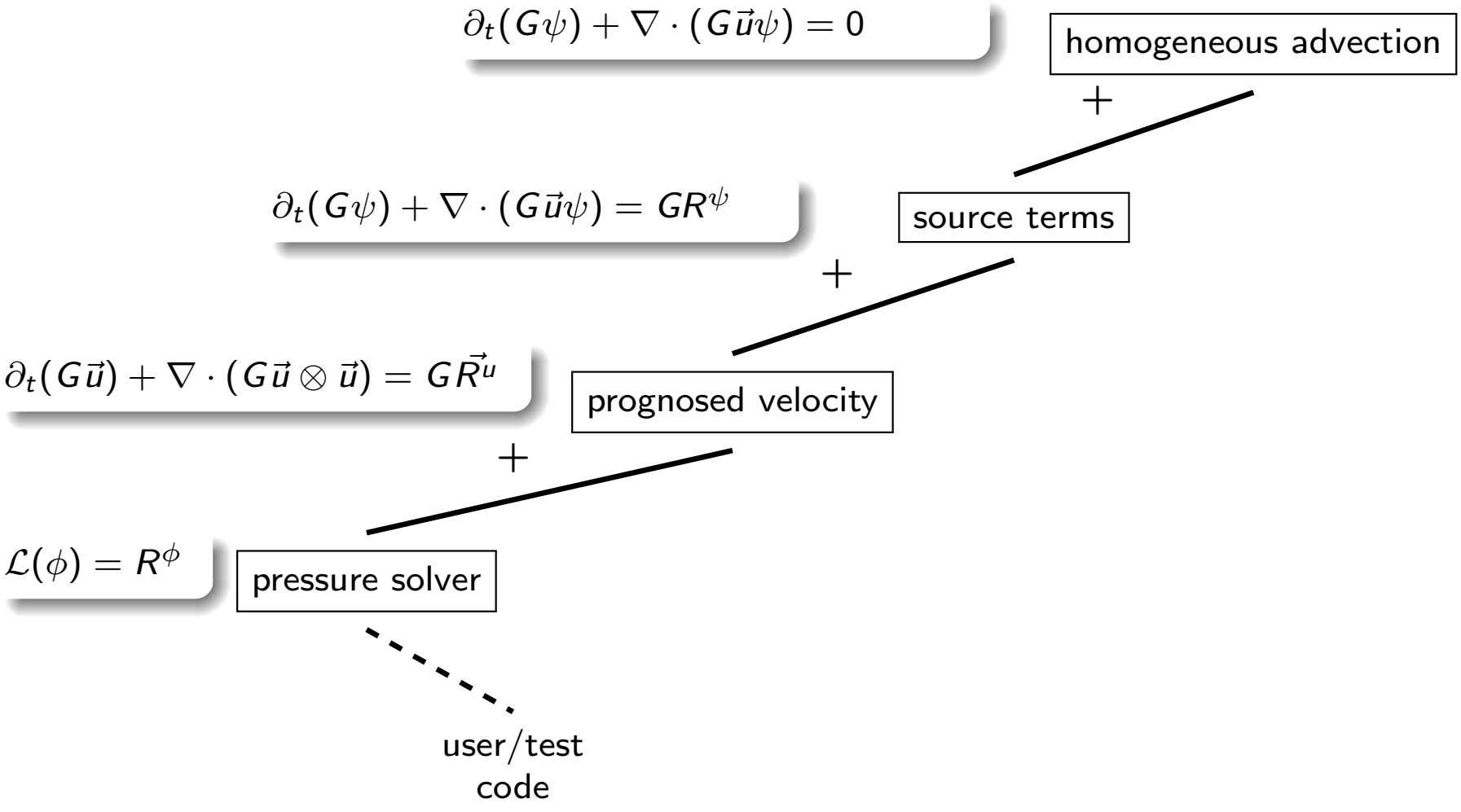
libmpdata++: 2D Boussinesq convection example



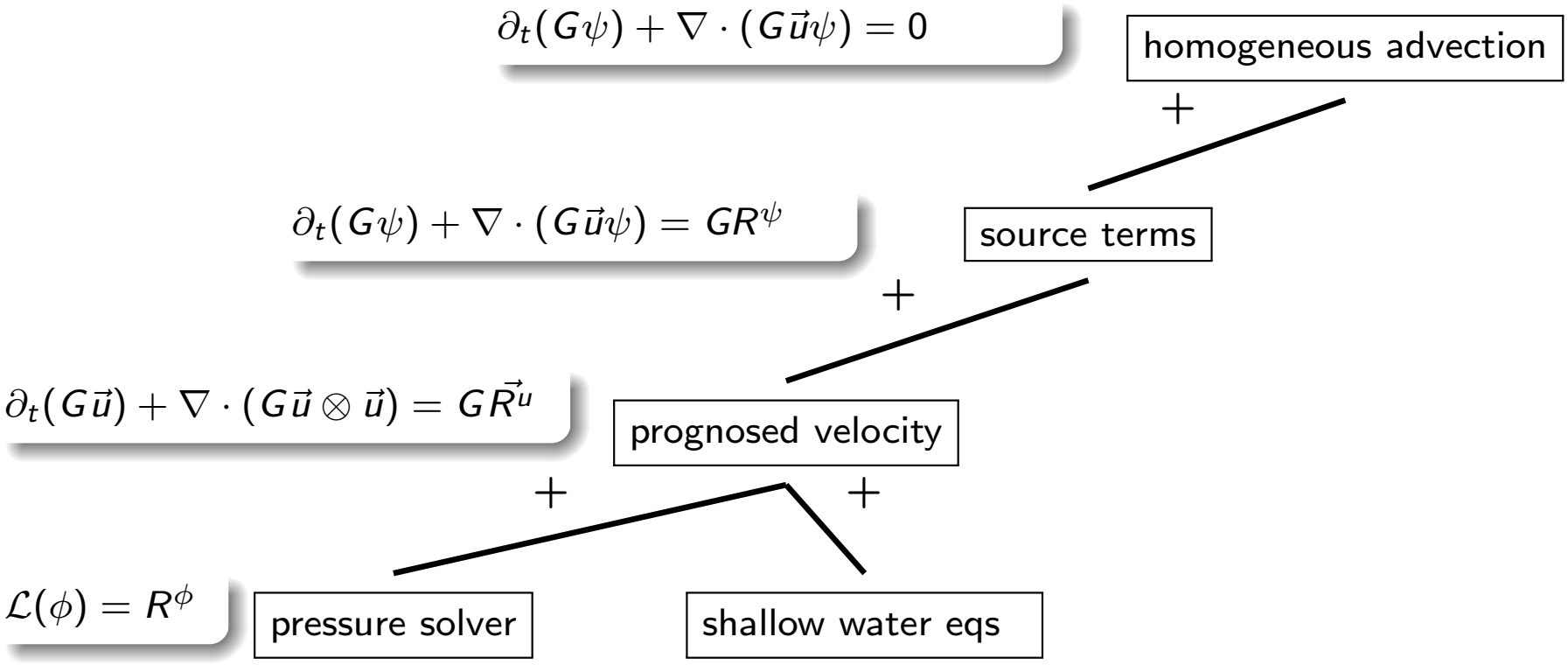
- reproduced experiment of Smolarkiewicz and Pudykiewicz, 1992
- <200 lines of code with libmpdata++

`https://github.com/igfuw/libmpdataxx/tree/master/tests/paper_2015_GMD/8_boussinesq_2d`

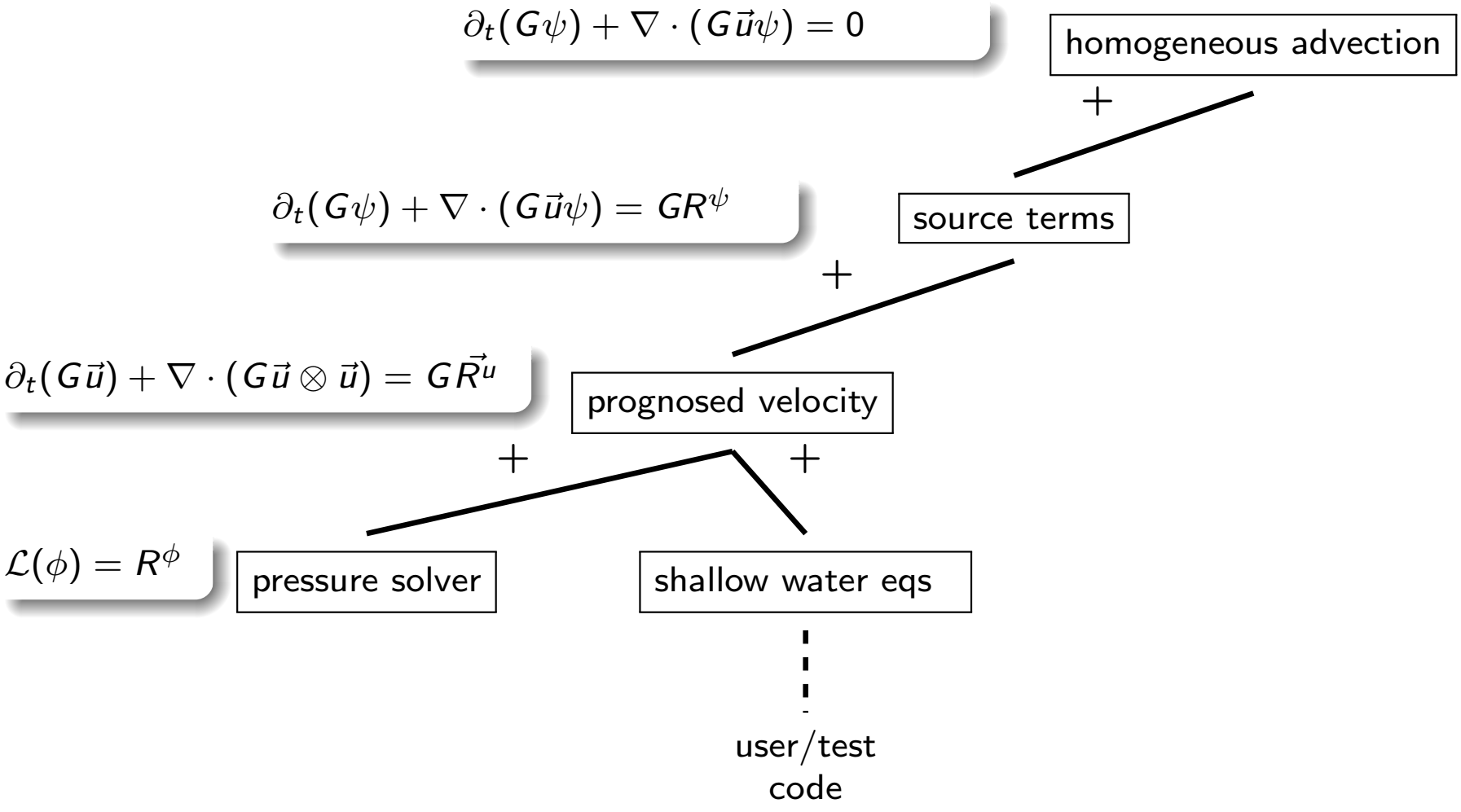
libmpdata++: solver/algorithm hierarchy



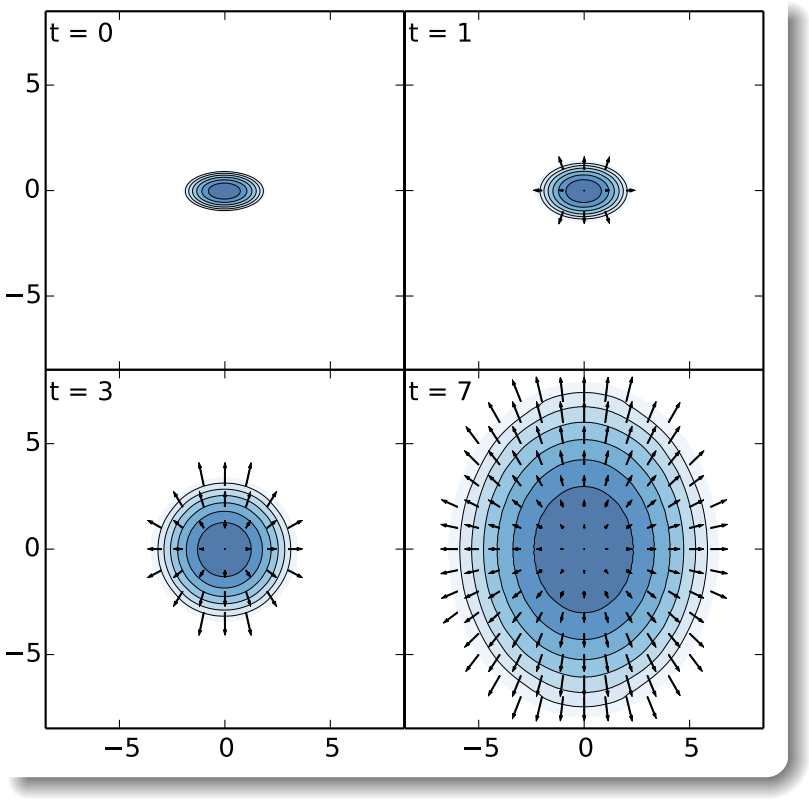
libmpdata++: solver/algorithm hierarchy



libmpdata++: solver/algorithm hierarchy

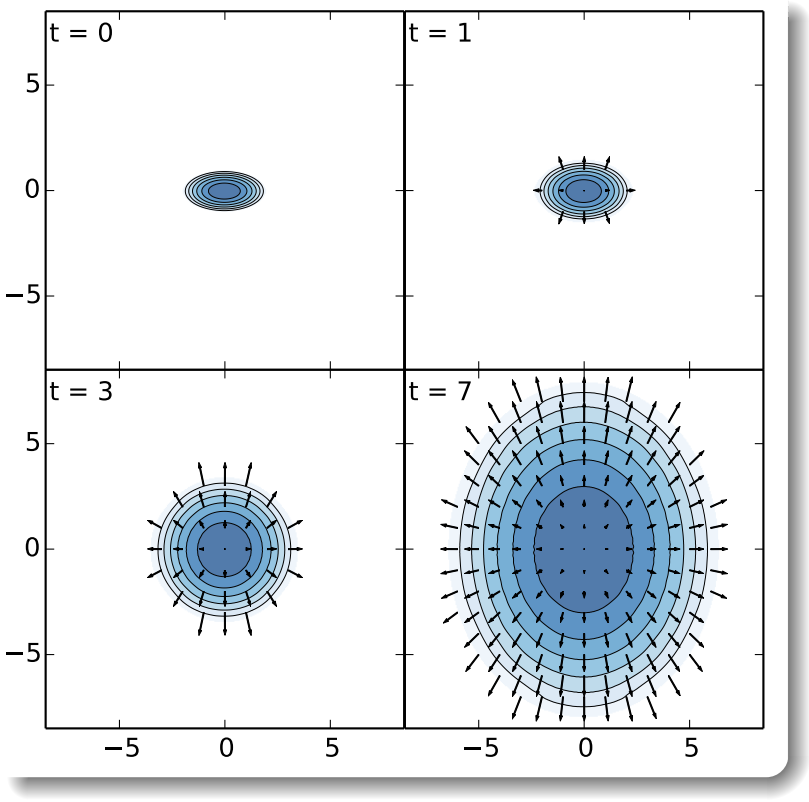


libmpdata++: 3D shallow-water system example



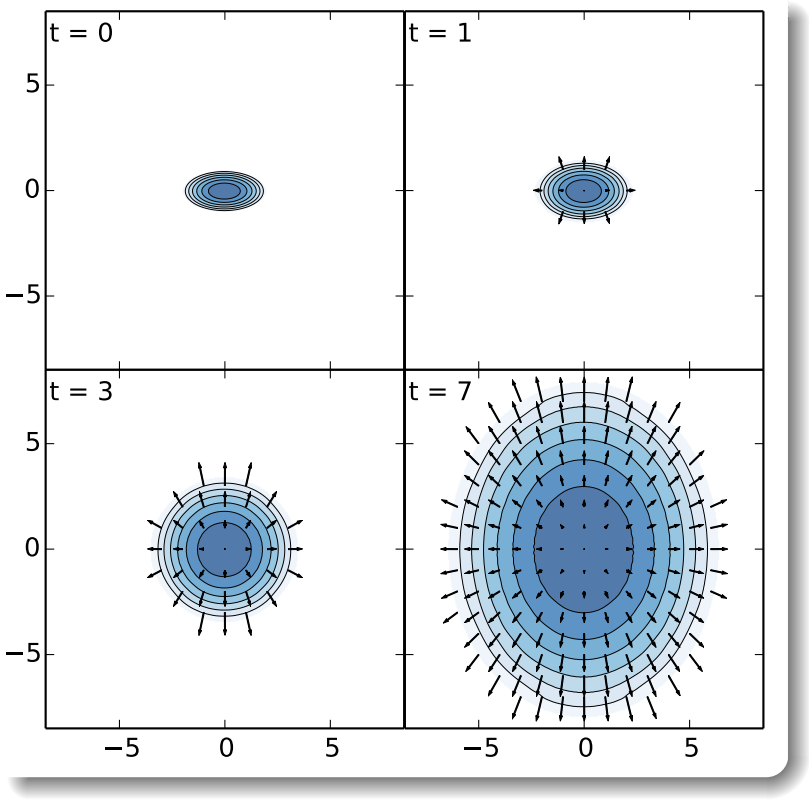
- inspired by 2D experiment of Schär and Smolarkiewicz, 1996

libmpdata++: 3D shallow-water system example



- inspired by 2D experiment of Schär and Smolarkiewicz, 1996
- example and original analytic solution by Dorota Jarecka / NCAR (Jarecka, Jaruga & Smolarkiewicz 2015, J. Comp. Phys. 289)

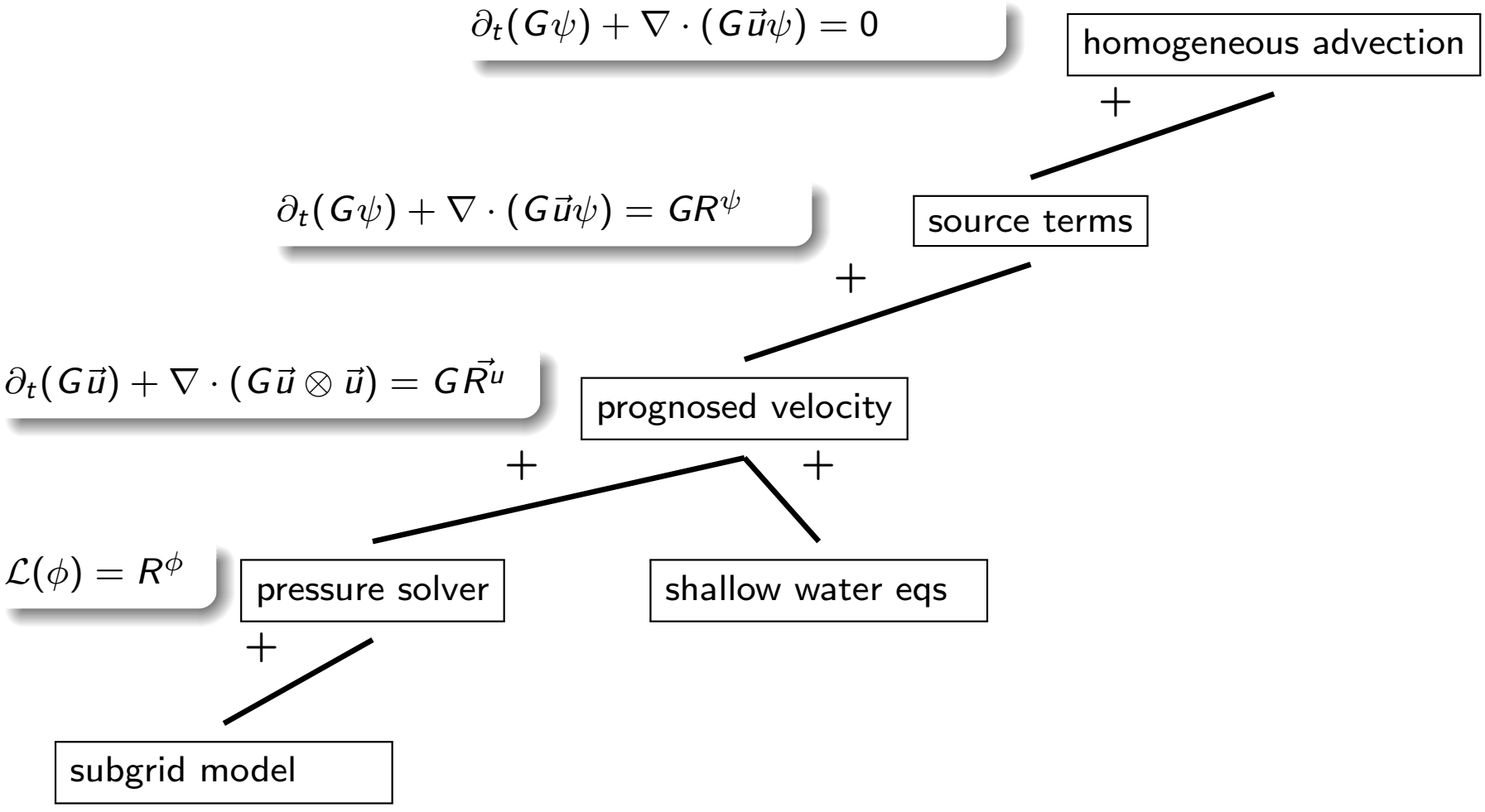
libmpdata++: 3D shallow-water system example



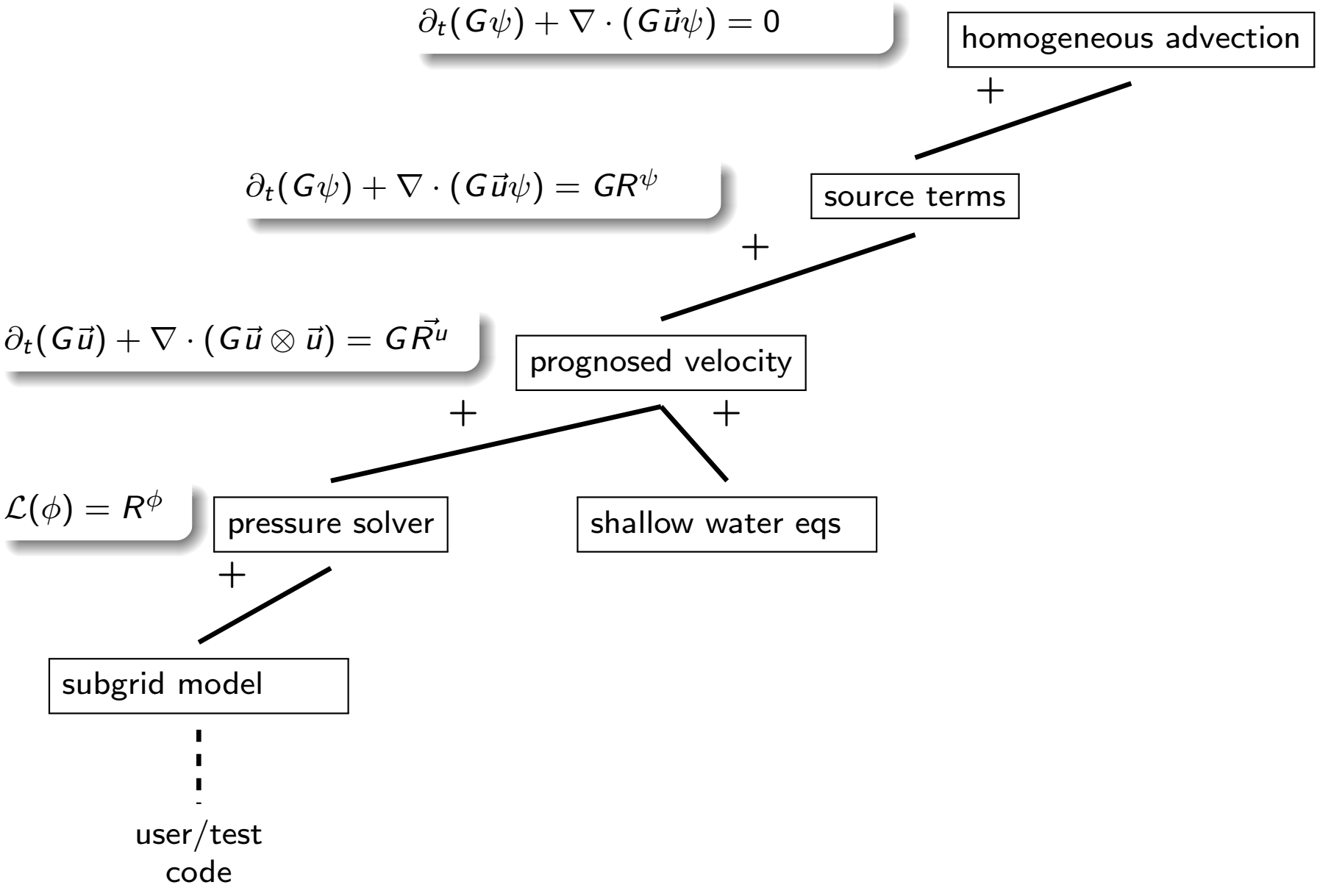
- inspired by 2D experiment of Schär and Smolarkiewicz, 1996
- example and original analytic solution by Dorota Jarecka / NCAR (Jarecka, Jaruga & Smolarkiewicz 2015, J. Comp. Phys. 289)
- <120 lines of code with libmpdata++

`https://github.com/igfuw/shallow-water-elliptic-drop/tree/master/numerical`

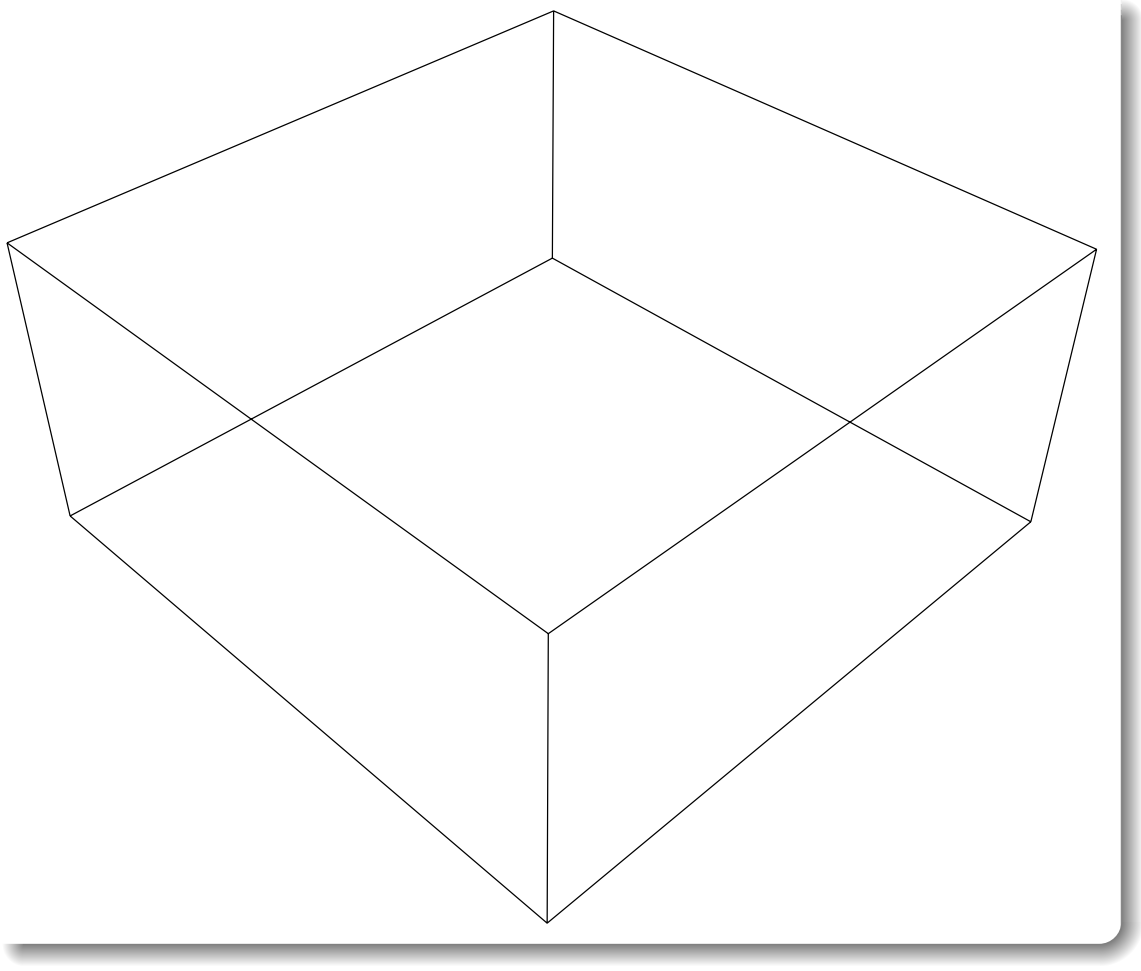
libmpdata++: solver/algorithm hierarchy



libmpdata++: solver/algorithm hierarchy

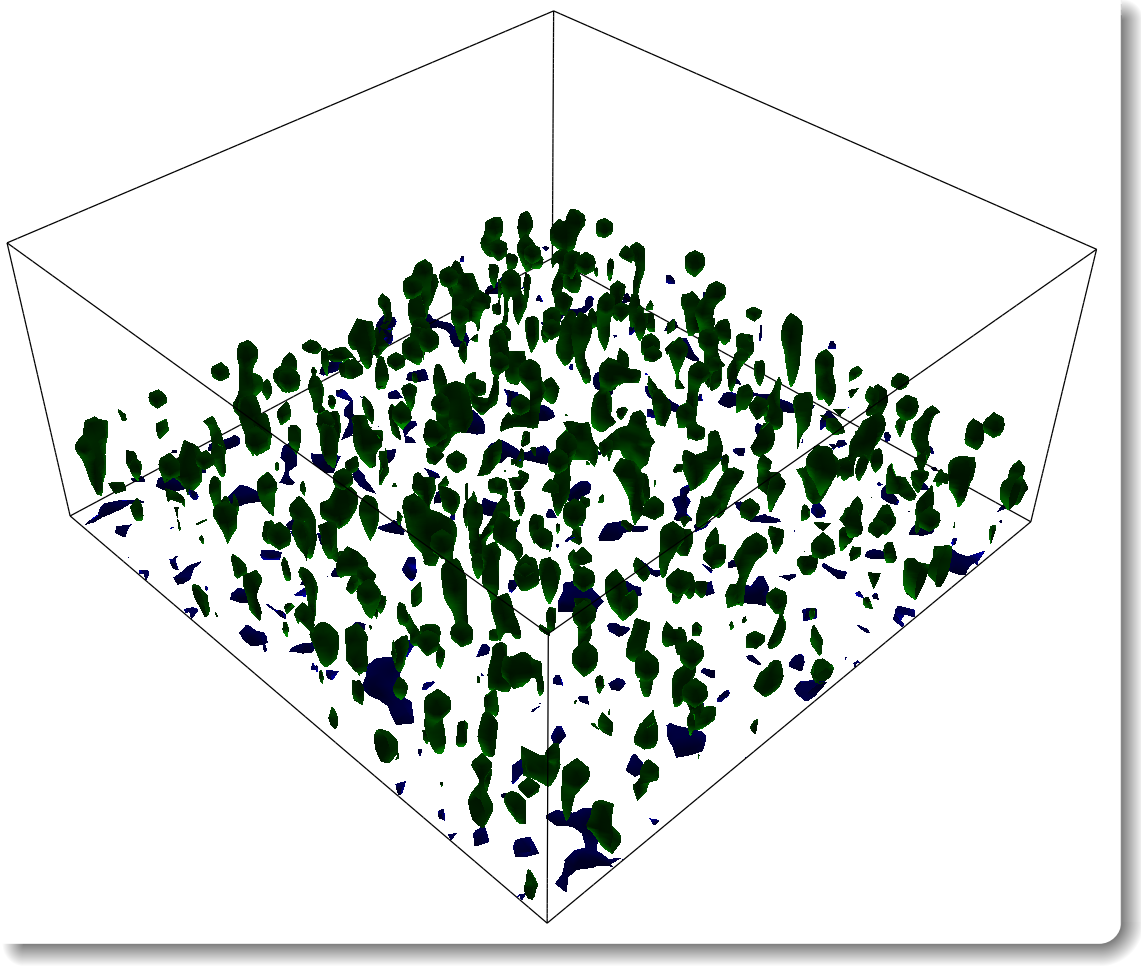


libmpdata++: convective boundary layer example



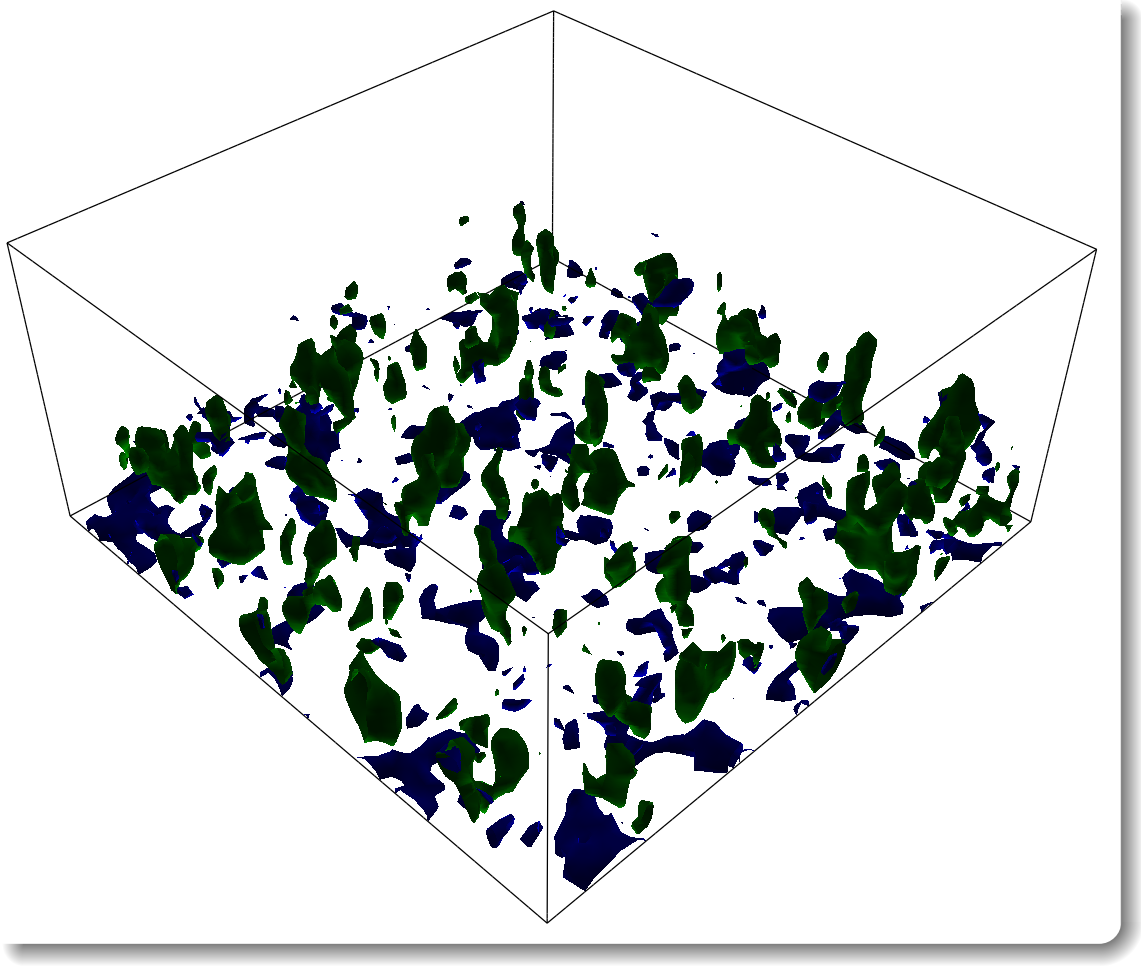
- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



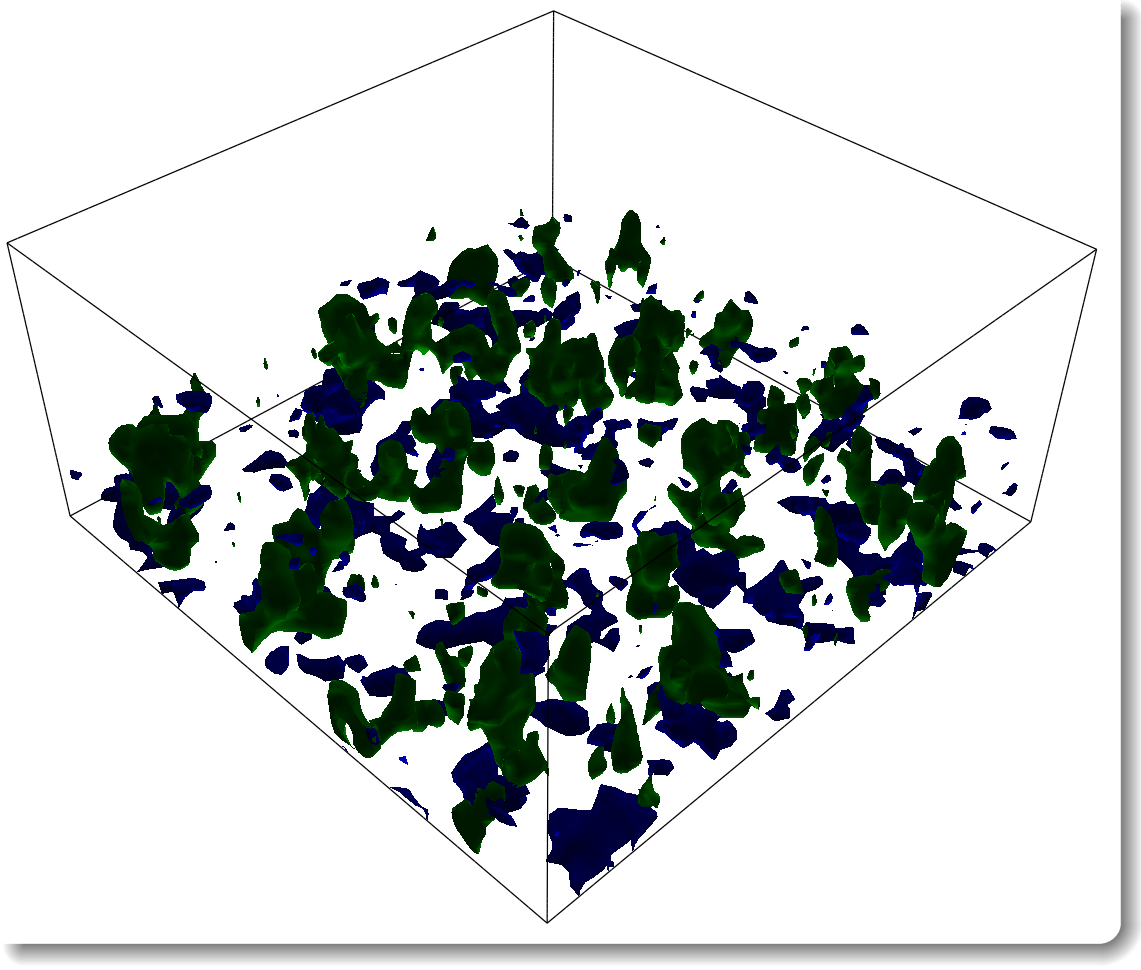
- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



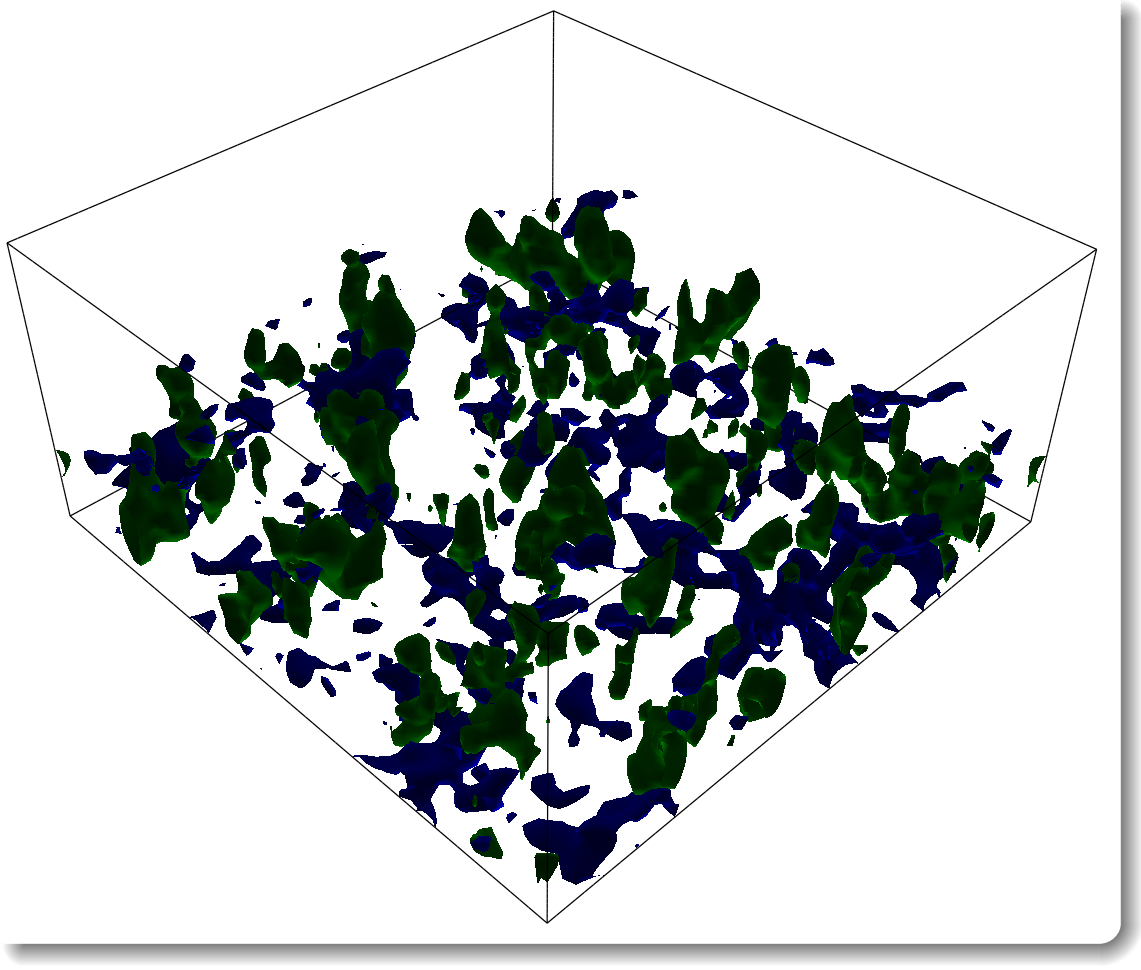
- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



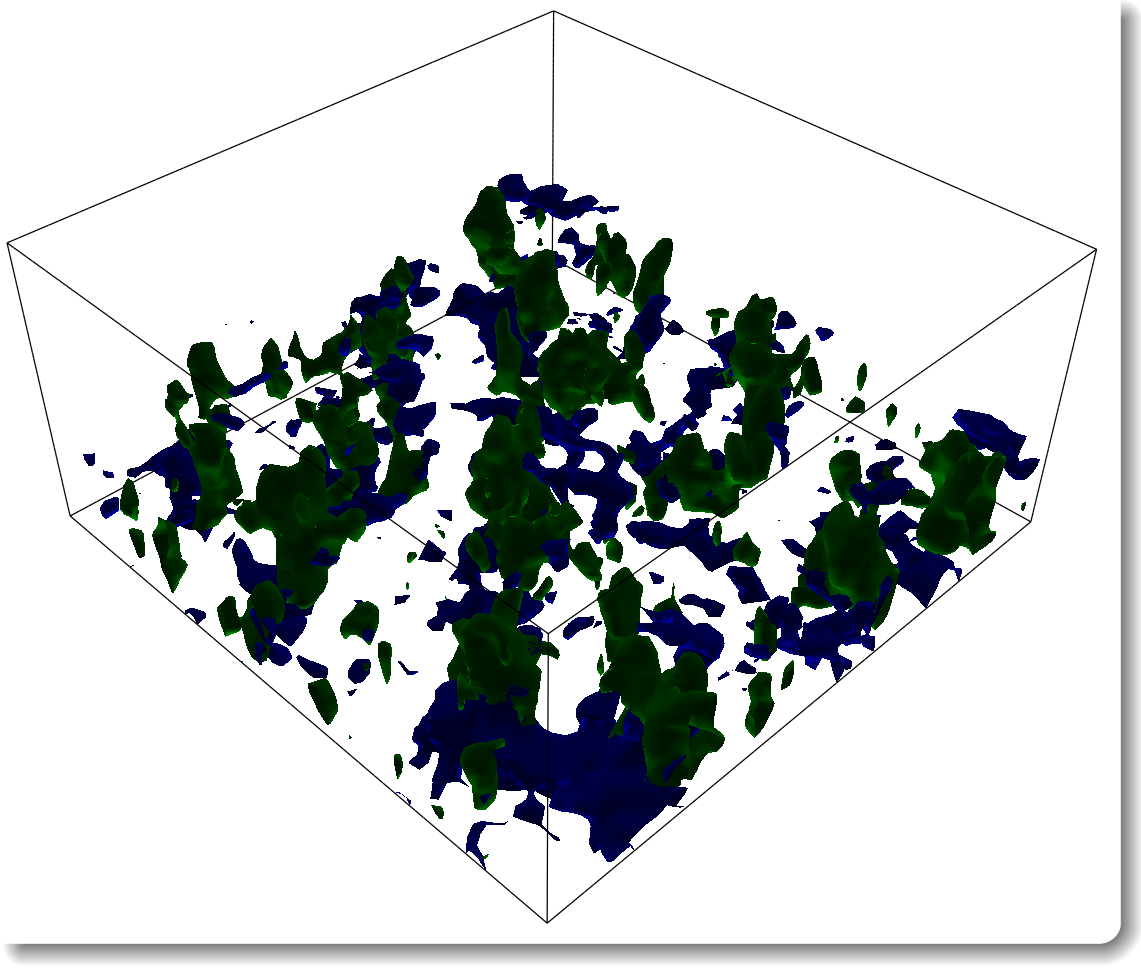
- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



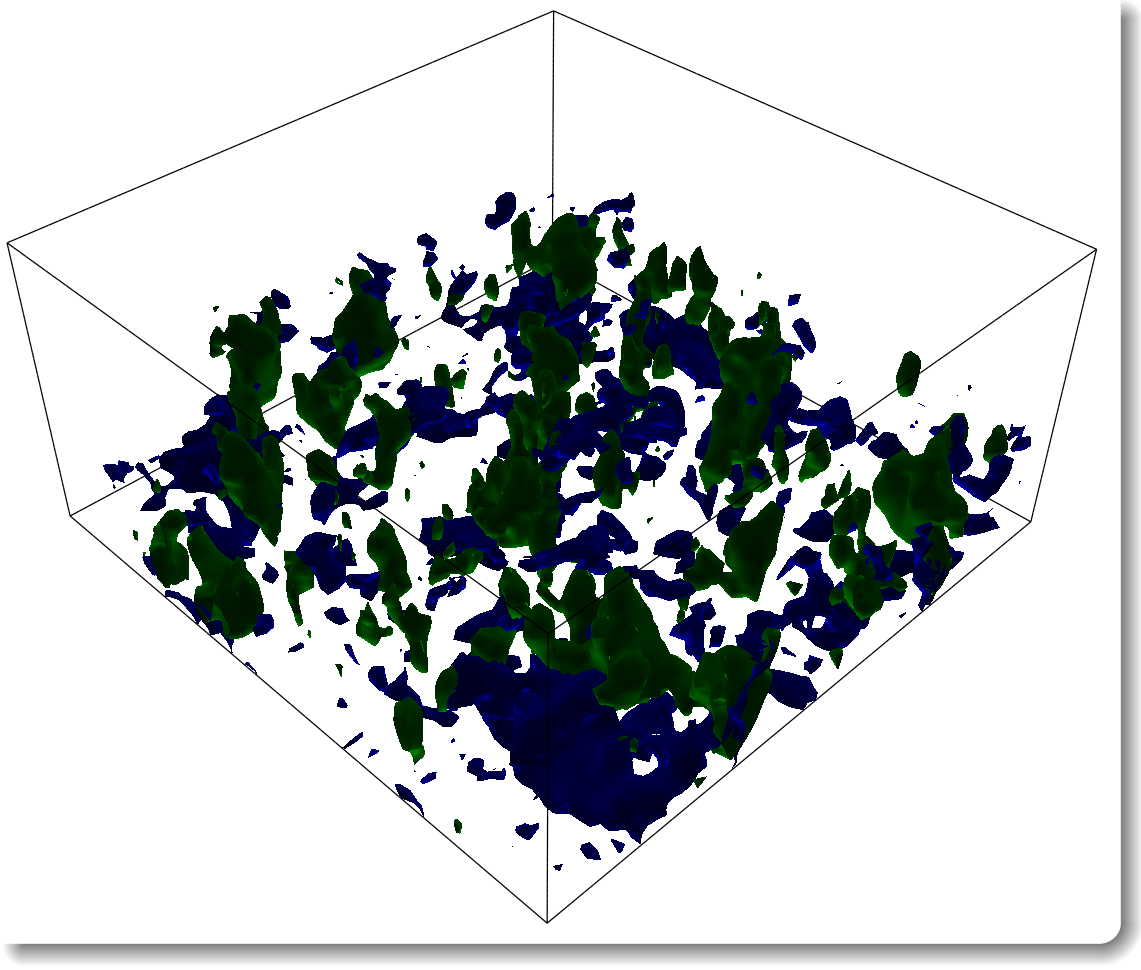
- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



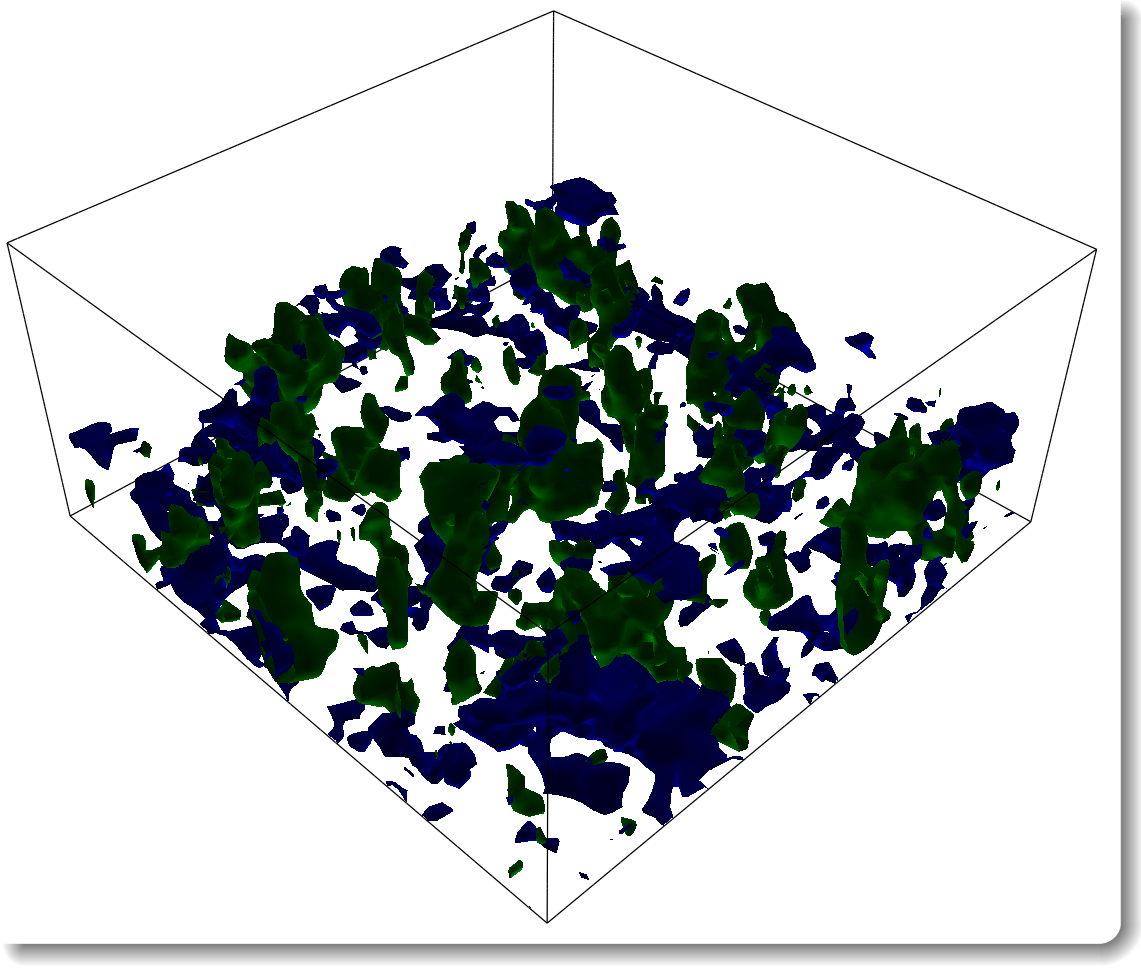
- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



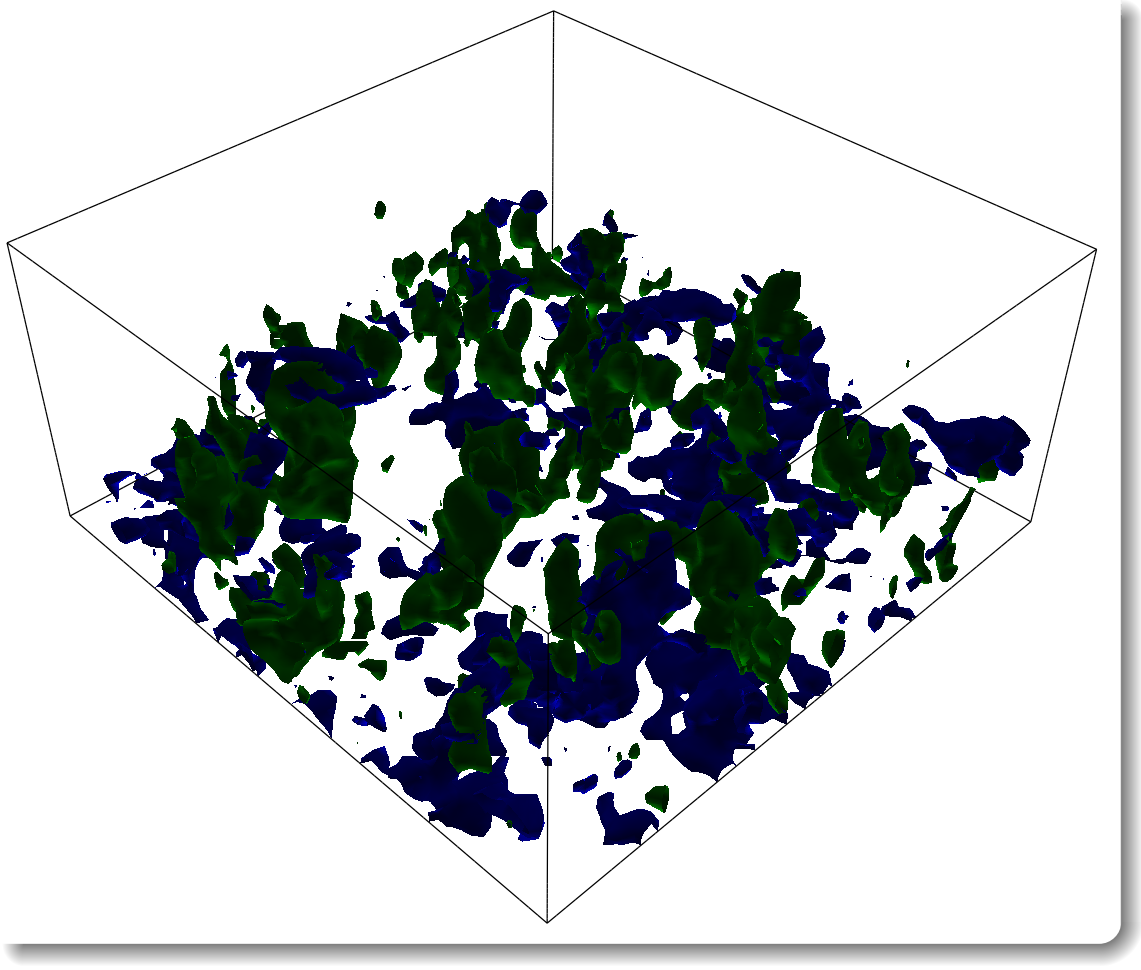
- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



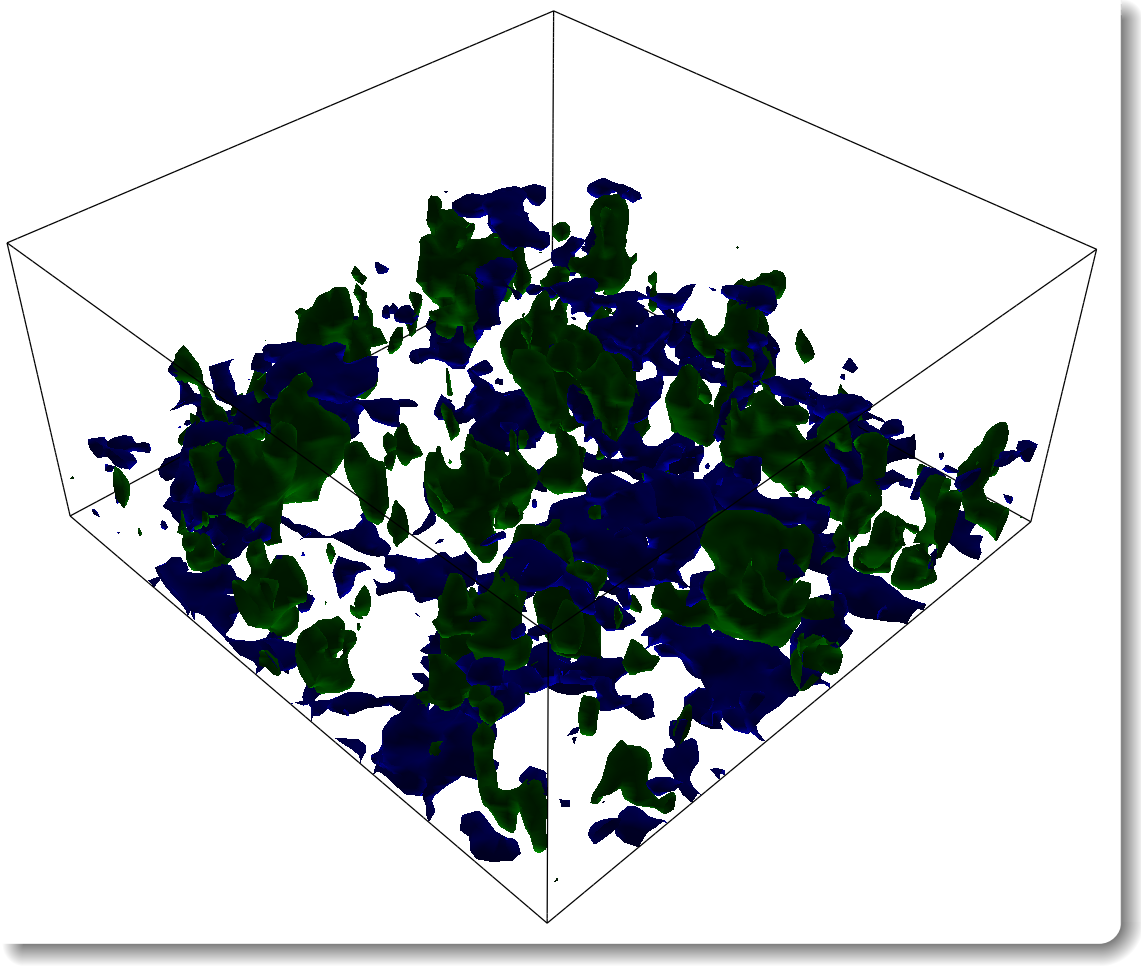
- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



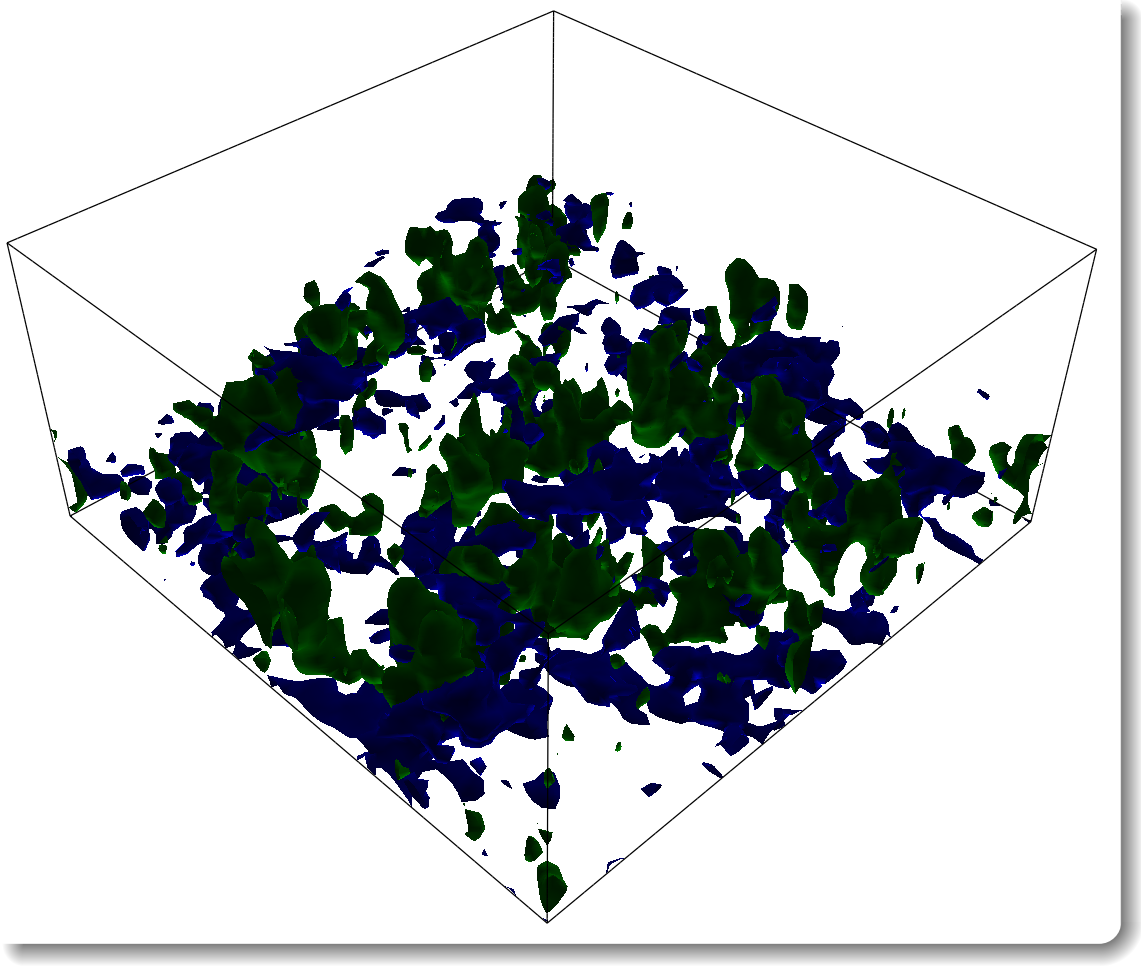
- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



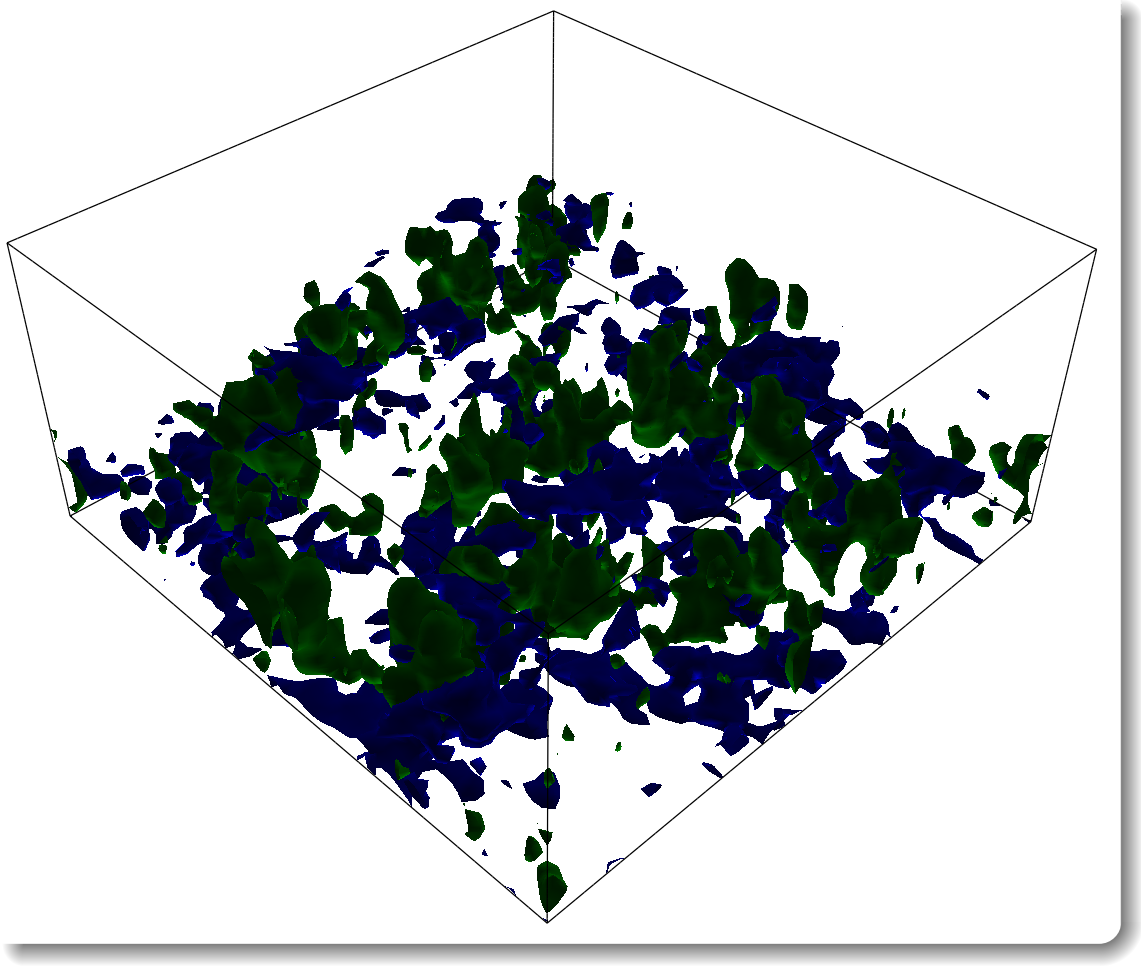
- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



- setup following Margolin et al., 1999

libmpdata++: convective boundary layer example



- setup following Margolin et al., 1999
- <250 lines of code with libmpdata++

Jaruga et al 2015

Geosci. Model Dev., 8, 1005–1032, 2015
www.geosci-model-dev.net/8/1005/2015/
doi:10.5194/gmd-8-1005-2015

Geoscientific Model Development
Open Access

libmpdata++ 1.0: a library of parallel MPDATA solvers for systems of generalised transport equations

A. Jaruga[1], S. Arabas[1], D. Jarecka[1,2], H. Pawlowska[1], P. K. Smolarkiewicz[3], and M. Waruszewski[1]

[1]Institute of Geophysics, Faculty of Physics, University of Warsaw, Warsaw, Poland
[2]National Center for Atmospheric Research, Boulder, CO, USA
[3]European Centre for Medium-Range Weather Forecasts, Reading, UK

libmpdata++: documentation

Jaruga et al 2015

Geosci. Model Dev., 8, 1005–1032, 2015
www.geosci-model-dev.net/8/1005/2015/
doi:10.5194/gmd-8-1005-2015

Geoscientific Model Development
Open Access

libmpdata++ 1.0: a library of parallel MPDATA solvers for systems of generalised transport equations

A. Jaruga[1], S. Arabas[1], D. Jarecka[1,2], H. Pawlowska[1], P. K. Smolarkiewicz[3], and M. Waruszewski[1]

[1]Institute of Geophysics, Faculty of Physics, University of Warsaw, Warsaw, Poland
[2]National Center for Atmospheric Research, Boulder, CO, USA
[3]European Centre for Medium-Range Weather Forecasts, Reading, UK

Geosci. Model Dev. policy (doi: 10.5194/gmd-6-1233-2013)

- *"paper must be accompanied by the code, or means of accessing the code, for the purpose of peer-review"*
- *"we strongly encourage referees to compile the code, and run test cases supplied by the authors"*

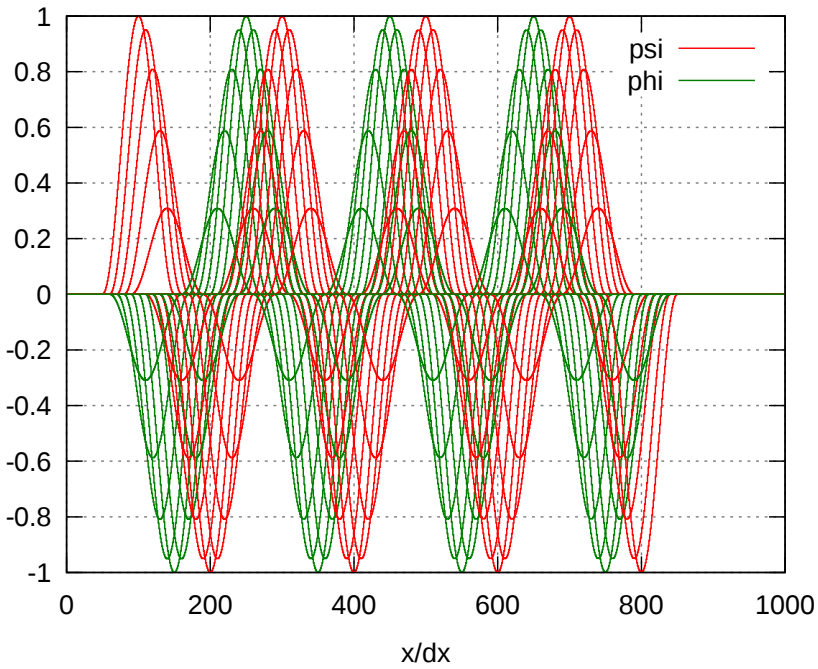


Figure 15. Simulation results of the example presented in Sect. 4.3. Abscissa marks the spatial dimension and ordinate represents the oscillator amplitude. The oscillator state is plotted every 20 time steps.

(partial differential equation) system (16) leads to the following system of coupled implicit algebraic equations:

$$\psi_i^{n+1} = \psi_i^* + 0.5\ \Delta t\ \omega\ \phi_i^{n+1},$$
$$\phi_i^{n+1} = \phi_i^* - 0.5\ \Delta t\ \omega\ \psi_i^{n+1}, \tag{17}$$

where ψ_i^* and ϕ_i^* stand for

```
#include <libmpdata++/solvers/mpdata_rhs.hpp>

template <class ct_params_t>
struct coupled_harmosc : public
  libmpdataxx::solvers::mpdata_rhs<ct_params_t>
{ // aliases
  using parent_t =
    libmpdataxx::solvers::mpdata_rhs<ct_params_t>;
  using ix = typename ct_params_t::ix;
  // member fields
  typename ct_params_t::real_t omega;

  // method called by mpdata_rhs
  void update_rhs(
    libmpdataxx::arrvec_t<
      typename parent_t::arr_t
    > &rhs,
    const typename parent_t::real_t &dt,
    const int &at
  ) {
    parent_t::update_rhs(rhs, dt, at);

    // just to shorten code
    const auto &psi = this->state(ix::psi);
    const auto &phi = this->state(ix::phi);
    const auto &i = this->i;

    switch (at)
    { // explicit solution for R^{n}
      // (note: with trapez used only at t=0)
      case (0):
      rhs.at(ix::psi)(i) += omega * phi(i);
      rhs.at(ix::phi)(i) -= omega * psi(i);
      break;
```

plan of the talk

introduction

libmpdata++

libcloudph++

n-dim array containers

plan of the talk

introduction

libmpdata++

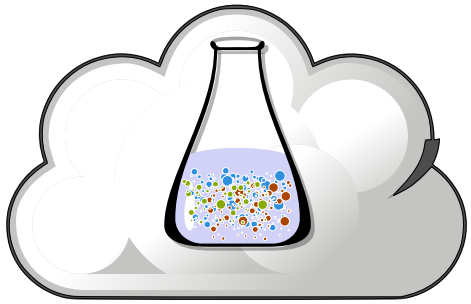
libcloudph++

n-dim array containers

libcloudph++

http://libcloudphxx.igf.fuw.edu.pl/

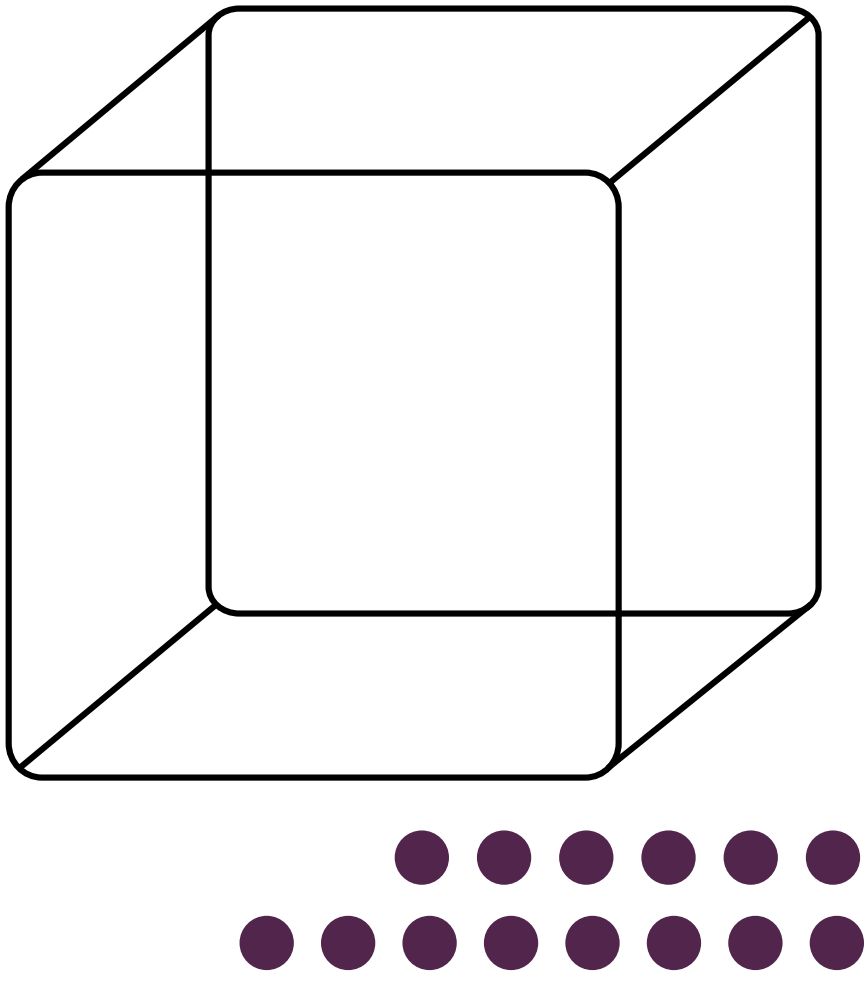
- droplet formation
- condensation



- collisional growth
- aqueous chemistry

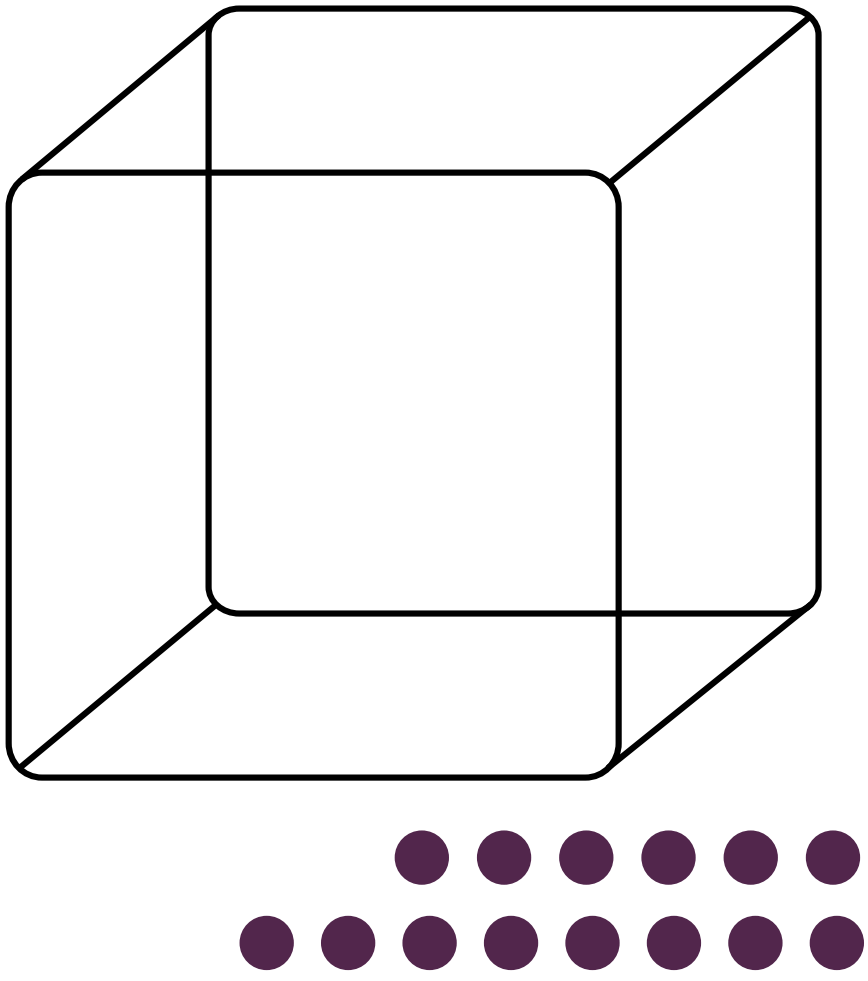
- precipitation
- evaporation

libcloudph++: particle-based Monte-Carlo algorithm



The domain is populated with "information carriers"
(alias computational particles, super droplets)

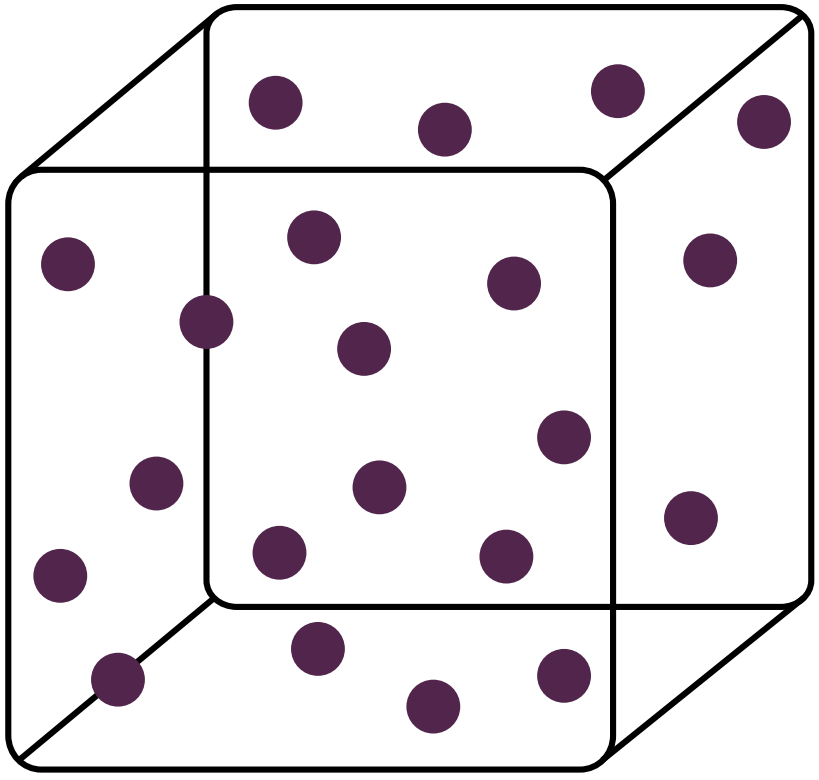
libcloudph++: particle-based Monte-Carlo algorithm



The domain is populated with "information carriers"
(alias computational particles, super droplets)

attributes:

libcloudph++: particle-based Monte-Carlo algorithm

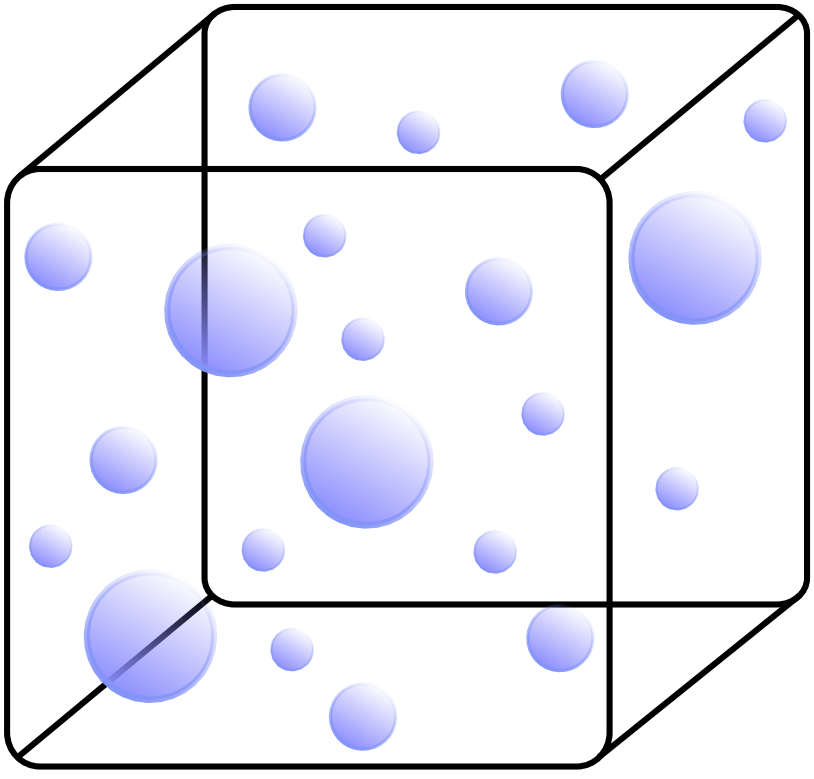


The domain is populated with "information carriers" (alias computational particles, super droplets)

attributes:

- spatial coords

libcloudph++: particle-based Monte-Carlo algorithm

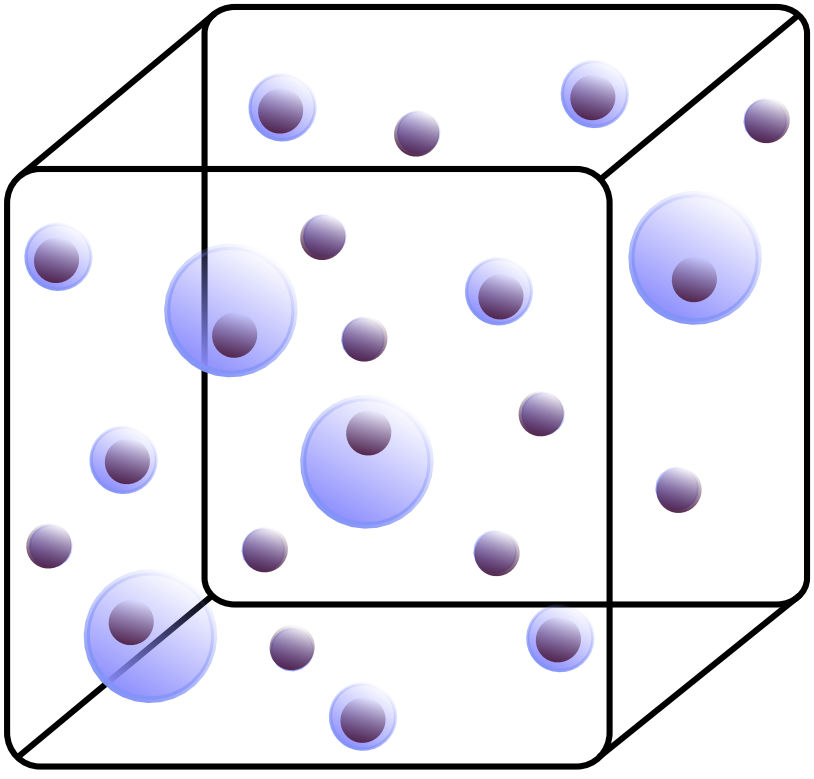


The domain is populated with "information carriers" (alias computational particles, super droplets)

attributes:

- spatial coords
- wet radius

libcloudph++: particle-based Monte-Carlo algorithm

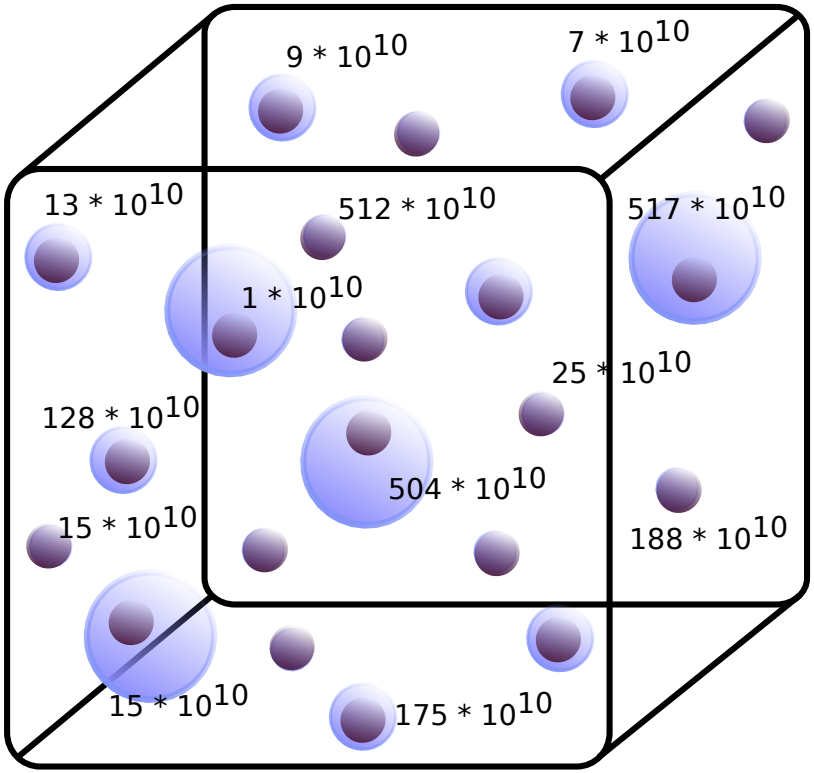


The domain is populated with "information carriers" (alias computational particles, super droplets)

attributes:

- spatial coords
- wet radius
- dry radius

libcloudph++: particle-based Monte-Carlo algorithm

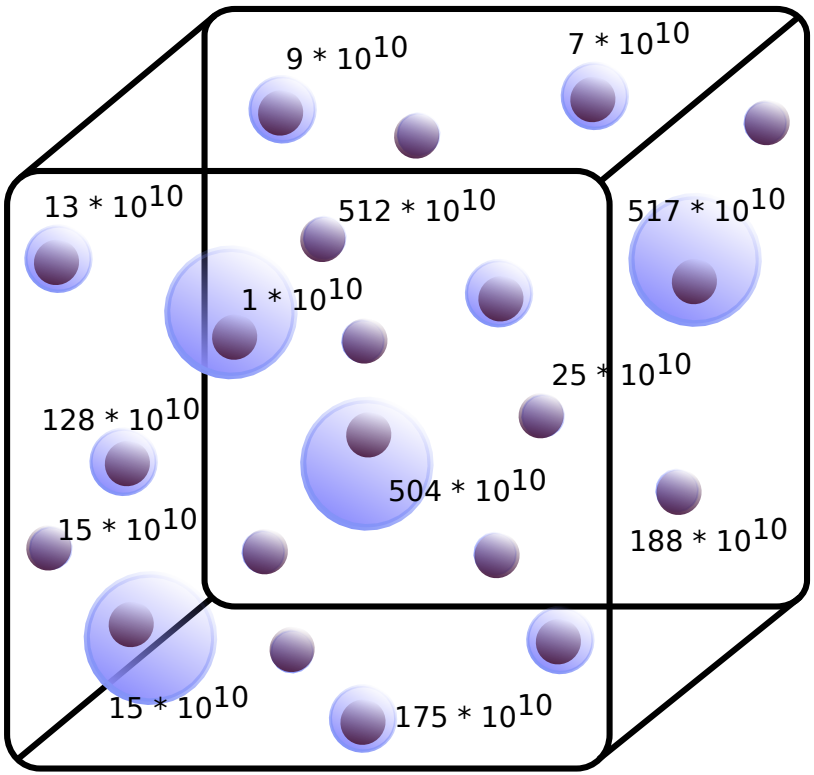


The domain is populated with "information carriers" (alias computational particles, super droplets)

attributes:

- spatial coords
- wet radius
- dry radius
- multiplicity

libcloudph++: particle-based Monte-Carlo algorithm

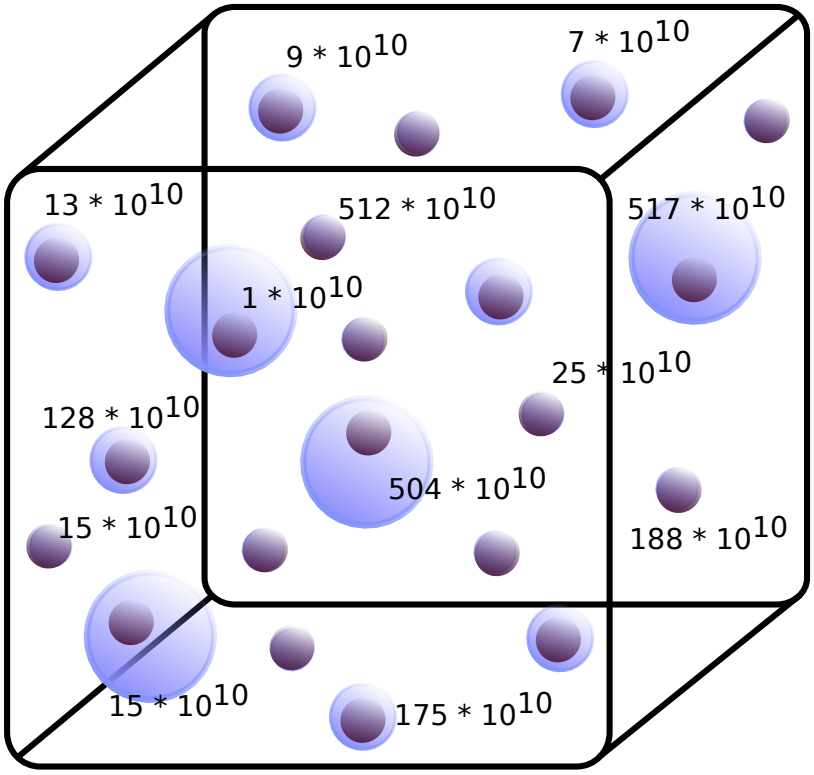


The domain is populated with "information carriers" (alias computational particles, super droplets)

attributes:

- spatial coords
- wet radius
- dry radius
- multiplicity
- ...

libcloudph++: particle-based Monte-Carlo algorithm



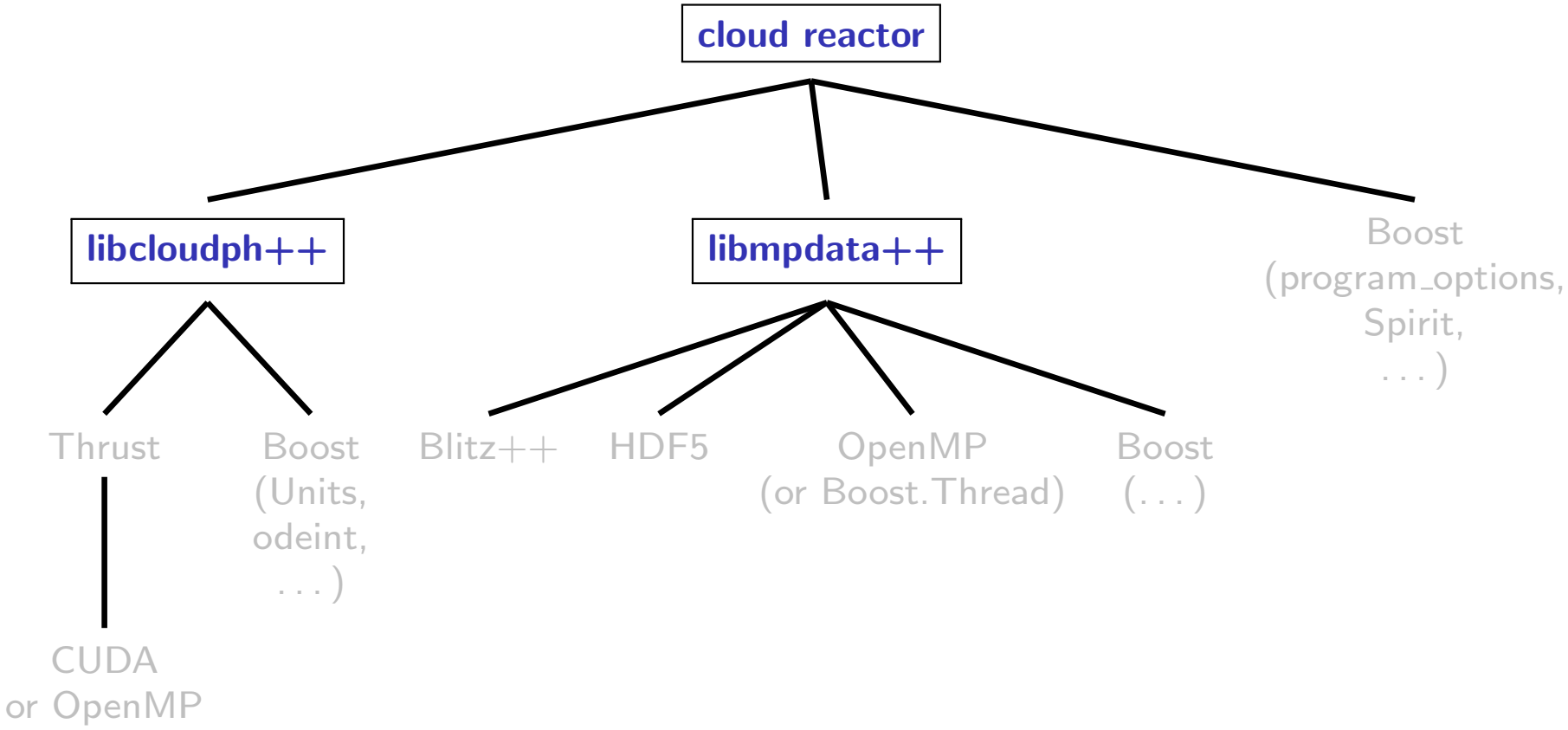
The domain is populated with "information carriers" (alias computational particles, super droplets)

attributes:

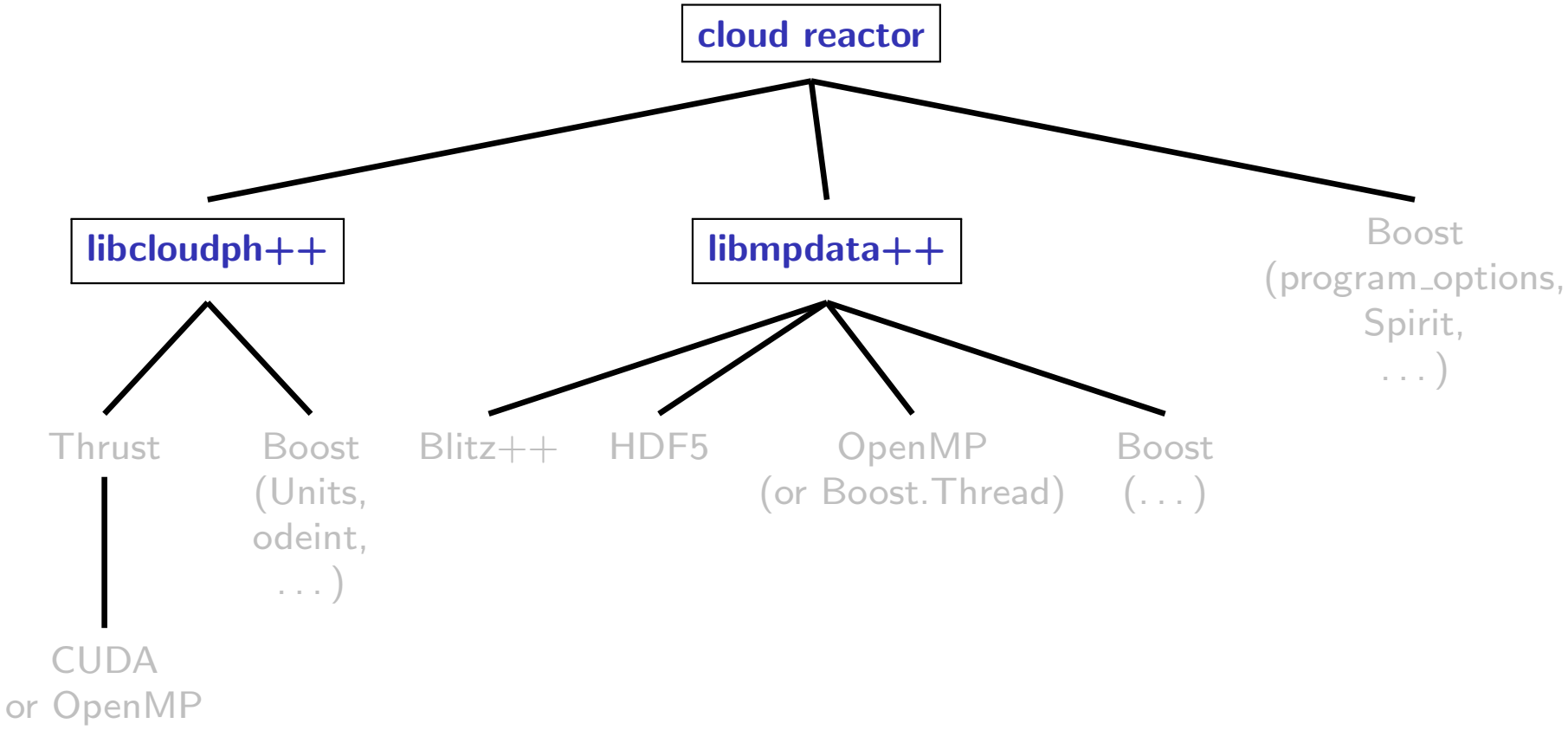
- spatial coords
- wet radius
- dry radius
- multiplicity
- ...

transport does not incur numerical diffusion!

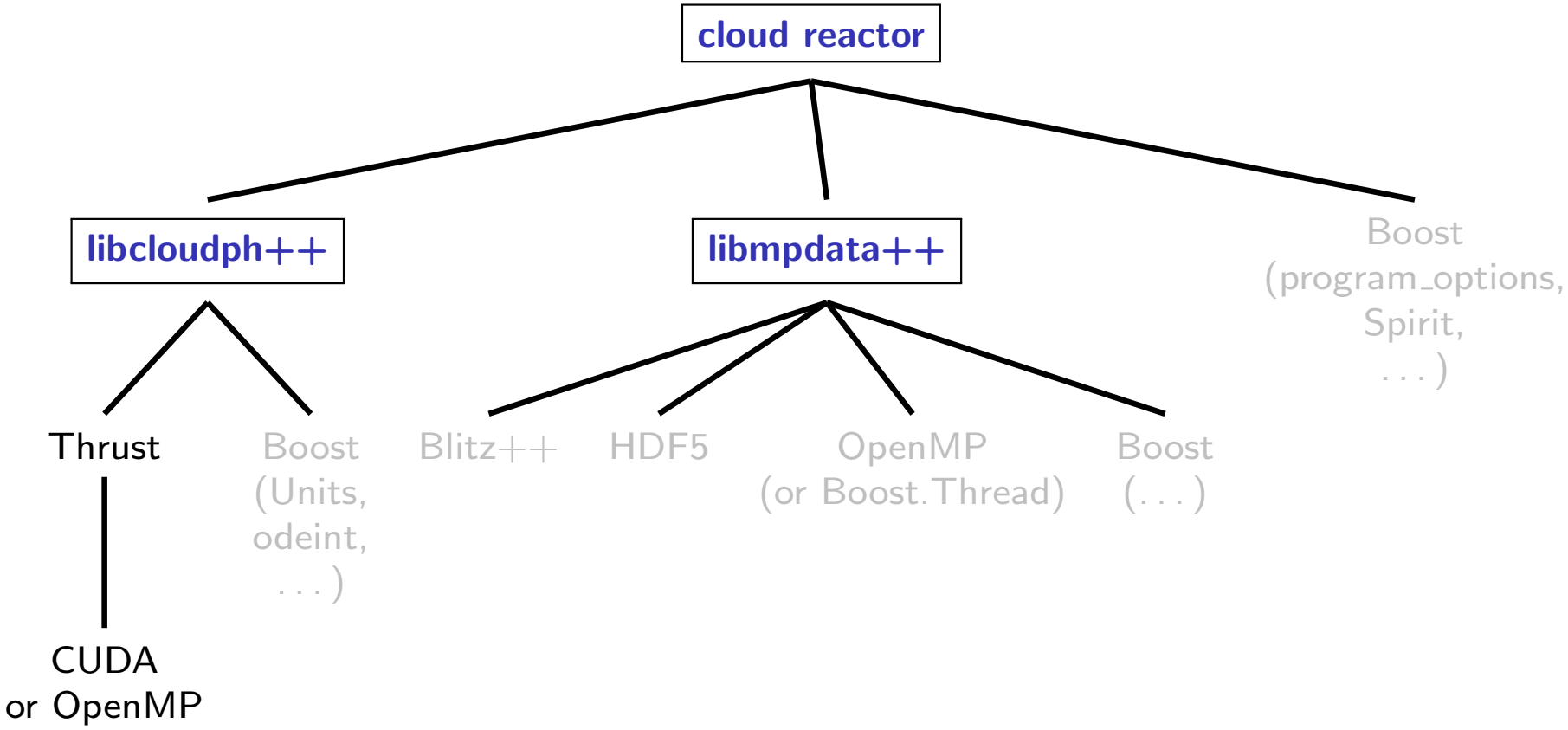
libcloudph++: some design choices



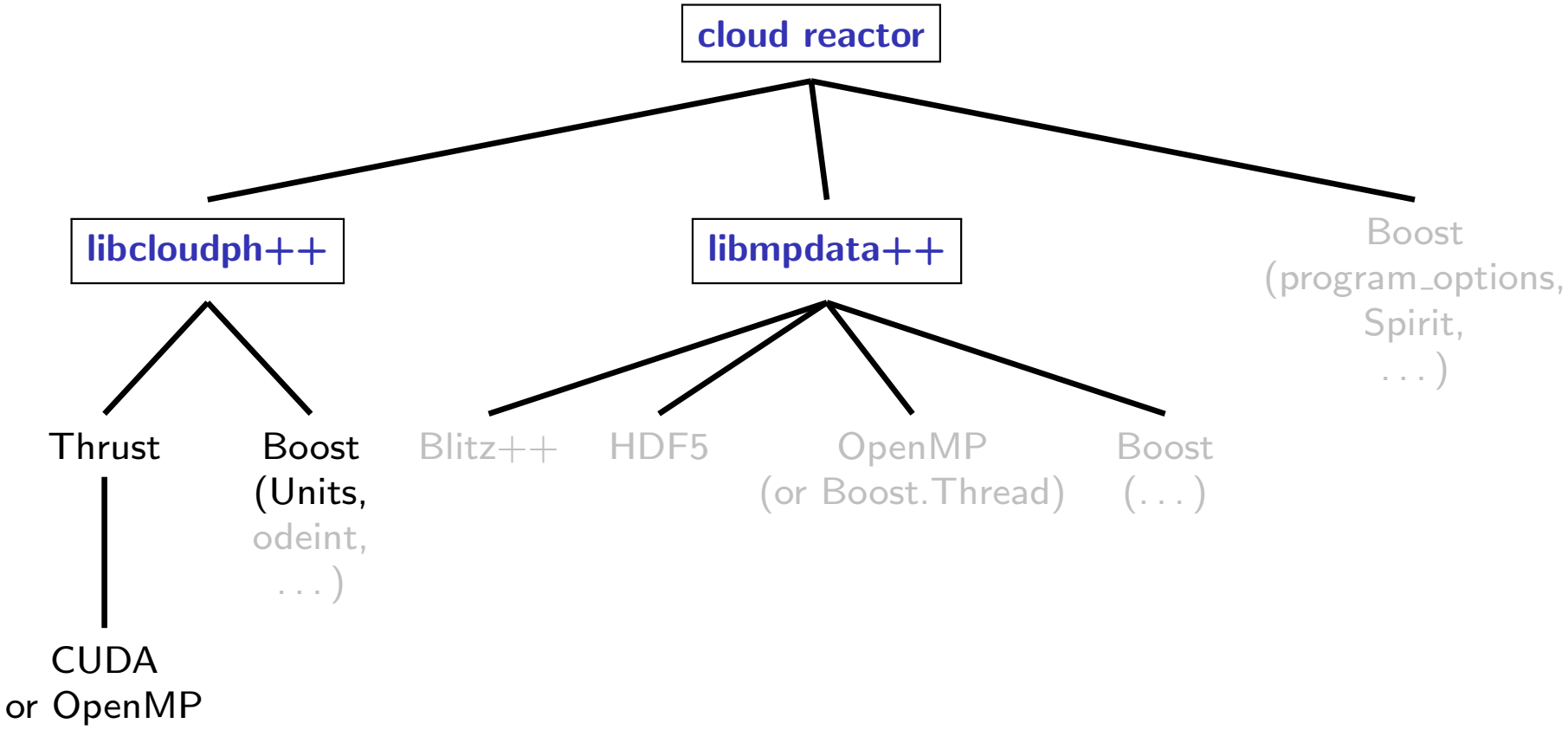
libcloudph++: some design choices



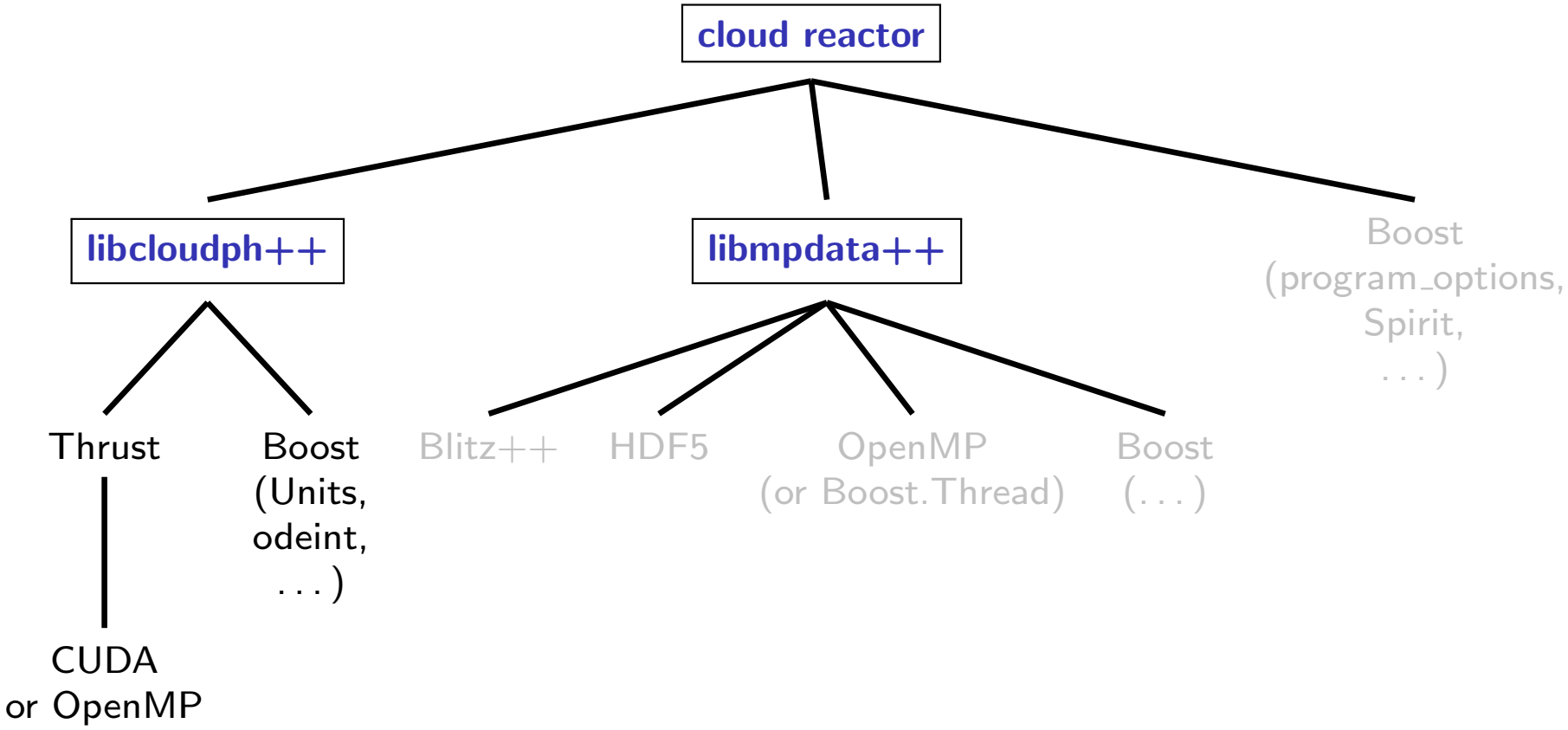
libcloudph++: some design choices



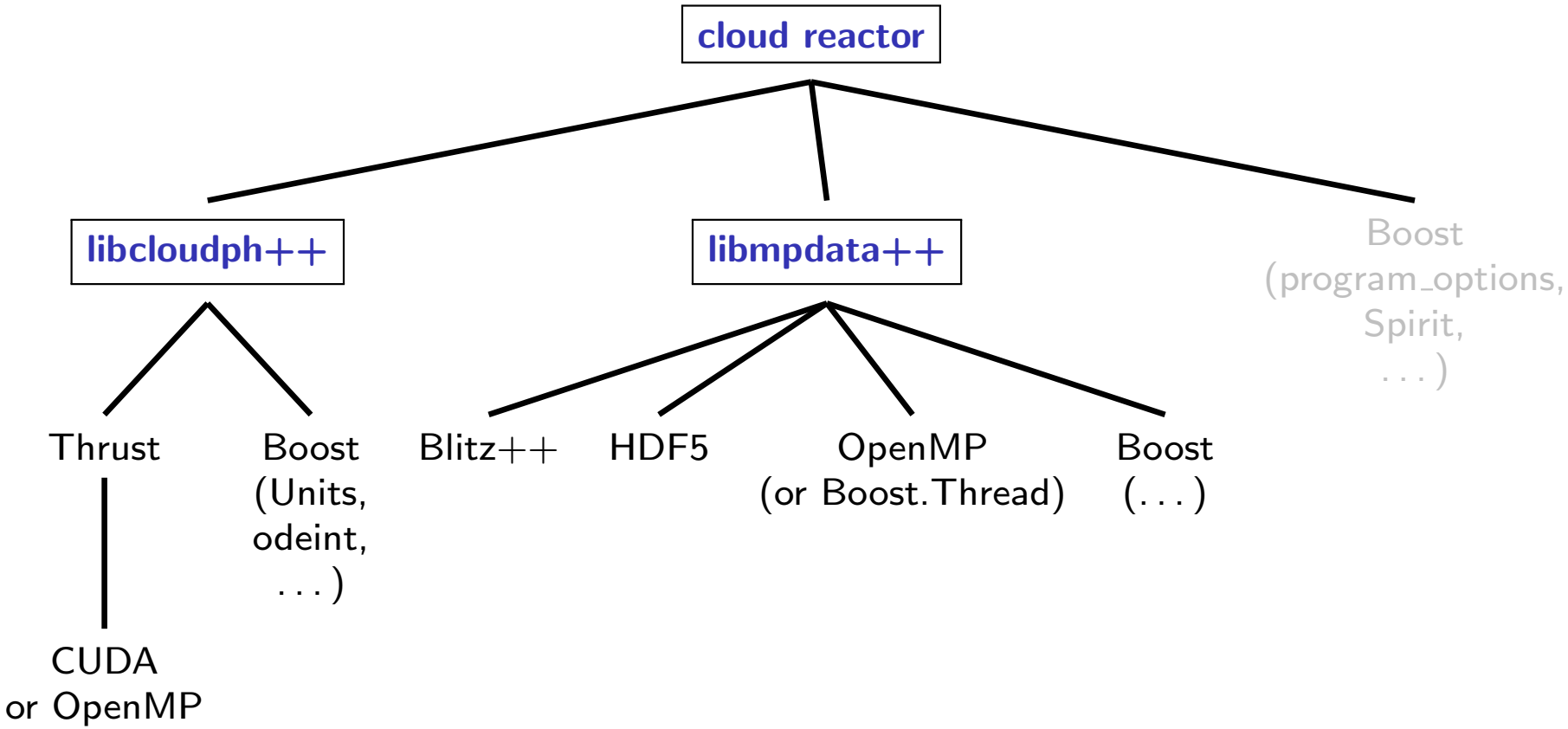
libcloudph++: some design choices



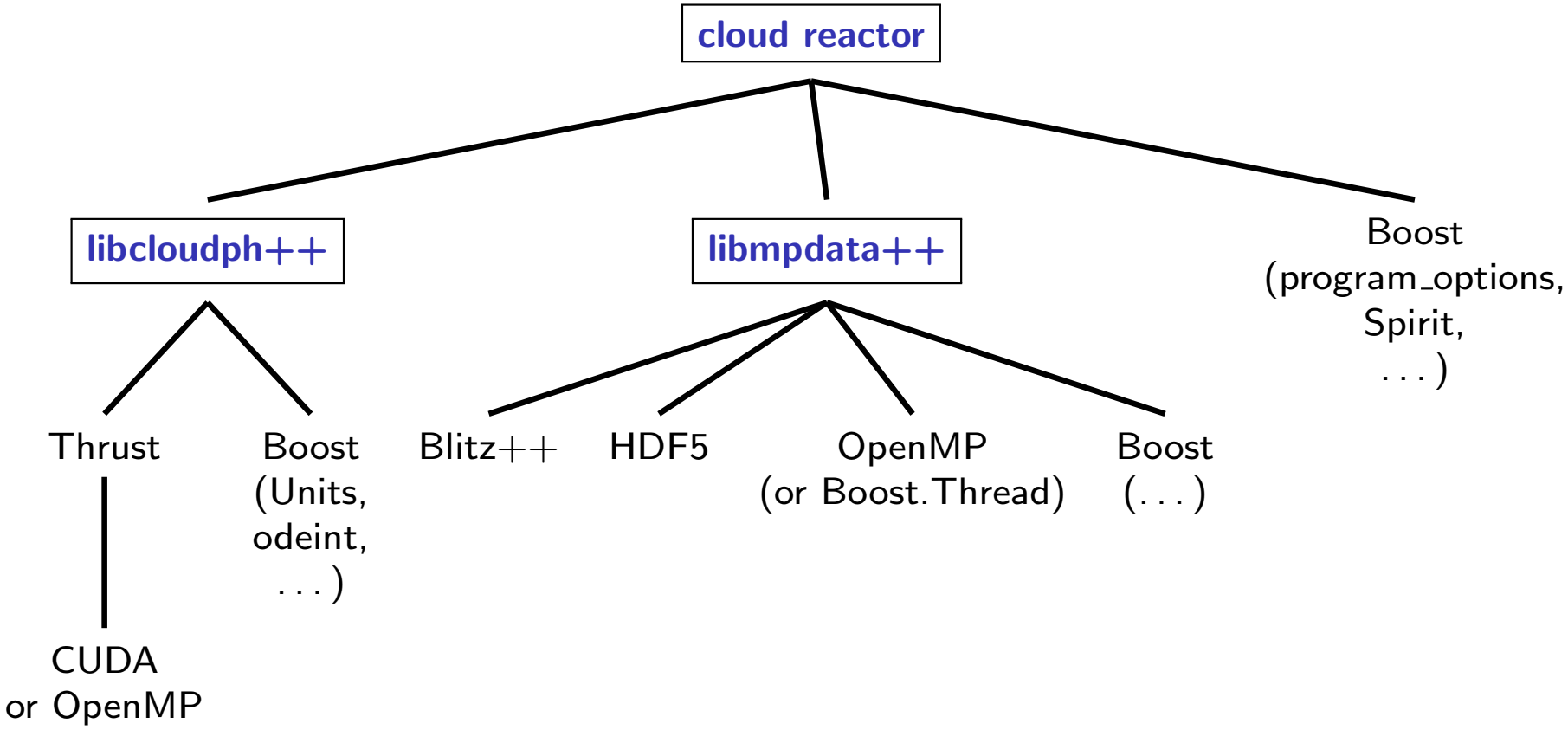
libcloudph++: some design choices



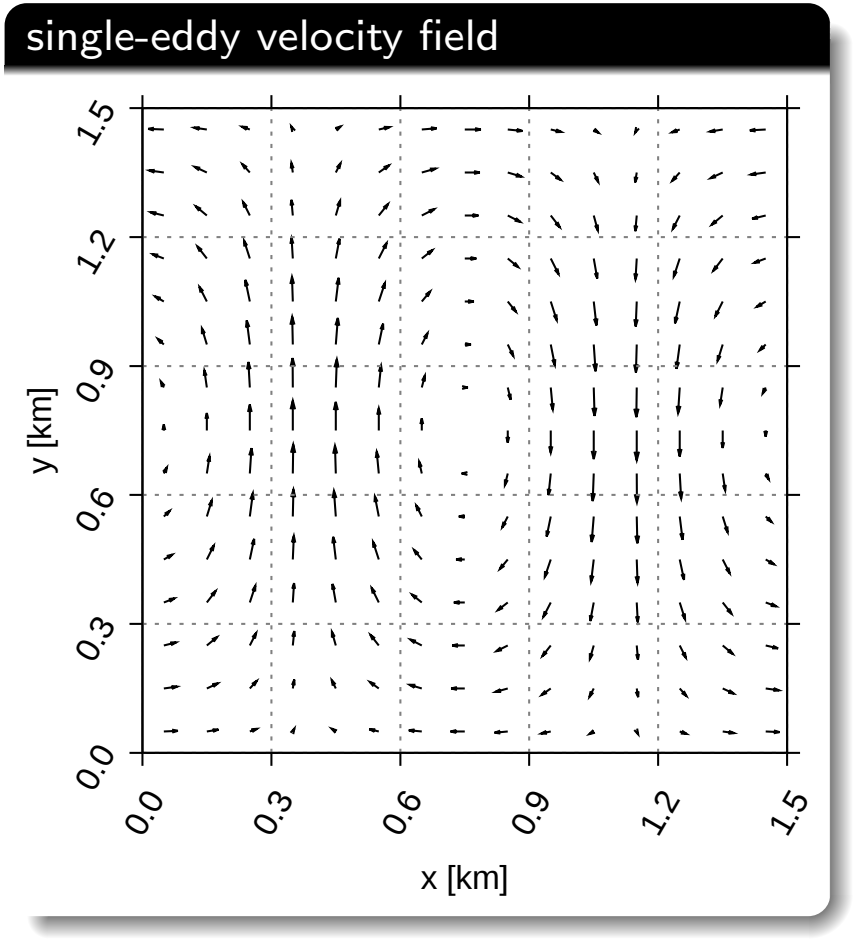
libcloudph++: some design choices



libcloudph++: some design choices

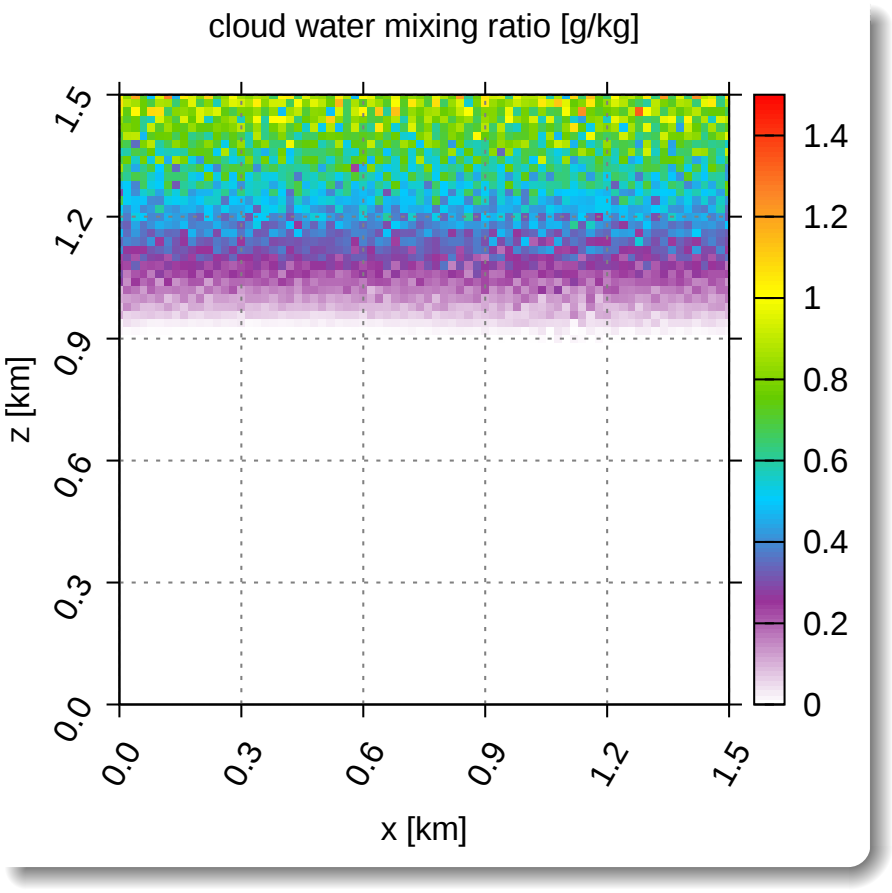


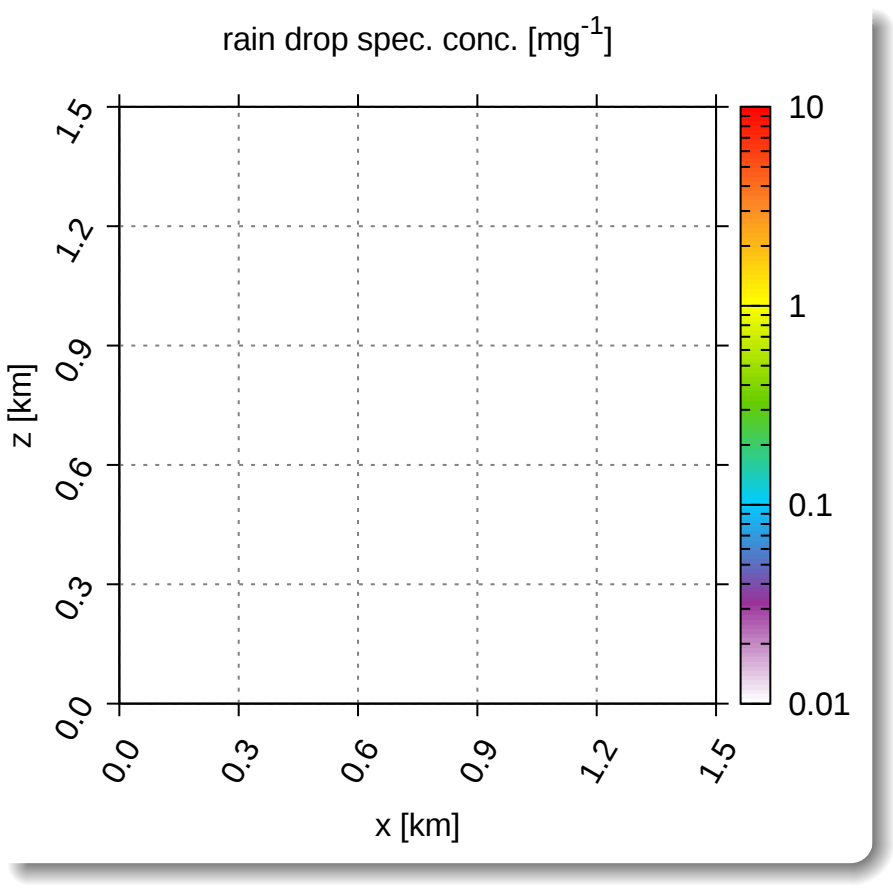
libcloudph++: example 2D prescribed-flow simulation



libcloudph++: example 2D prescribed-flow simulation

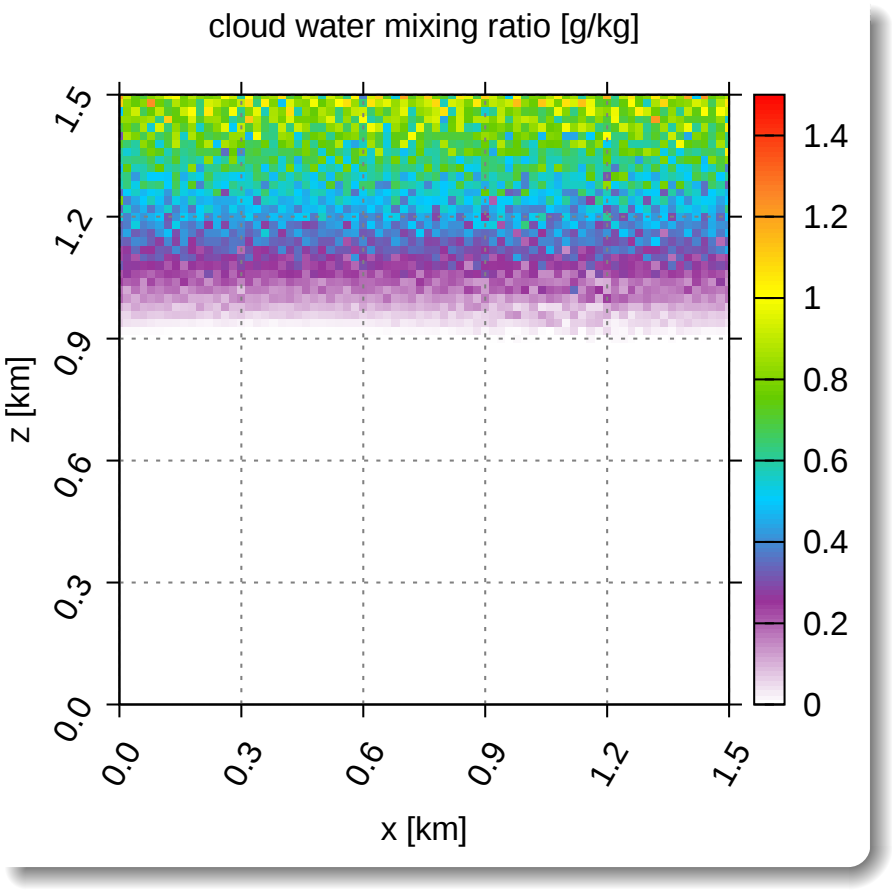
x

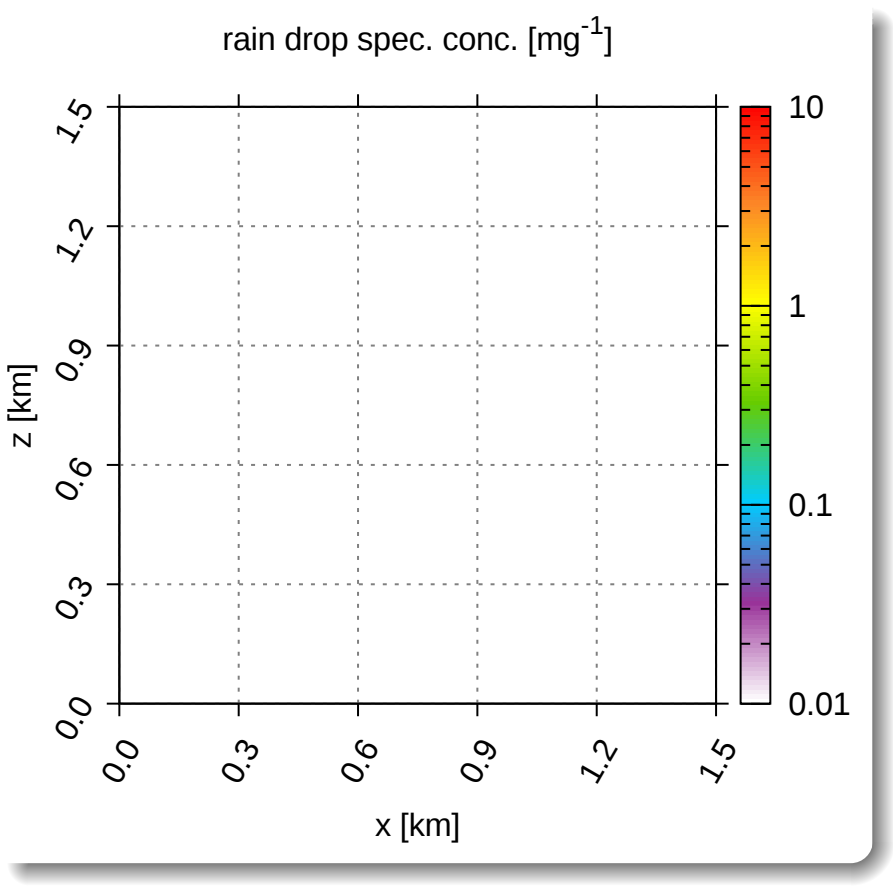




libcloudph++: example 2D prescribed-flow simulation

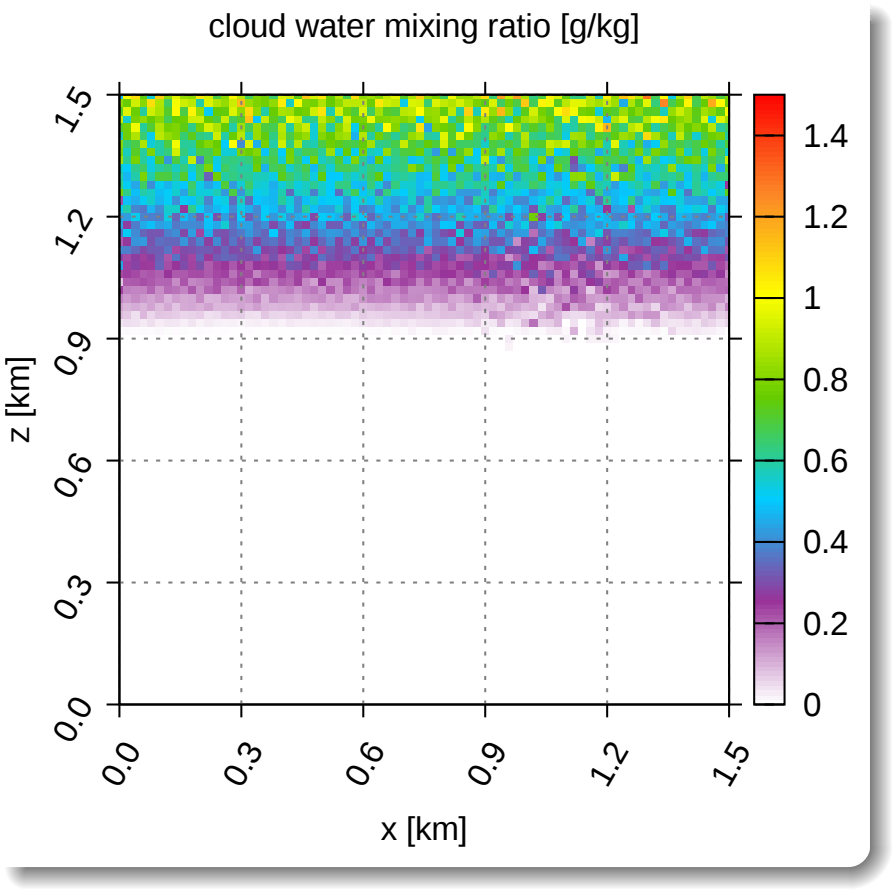
xx

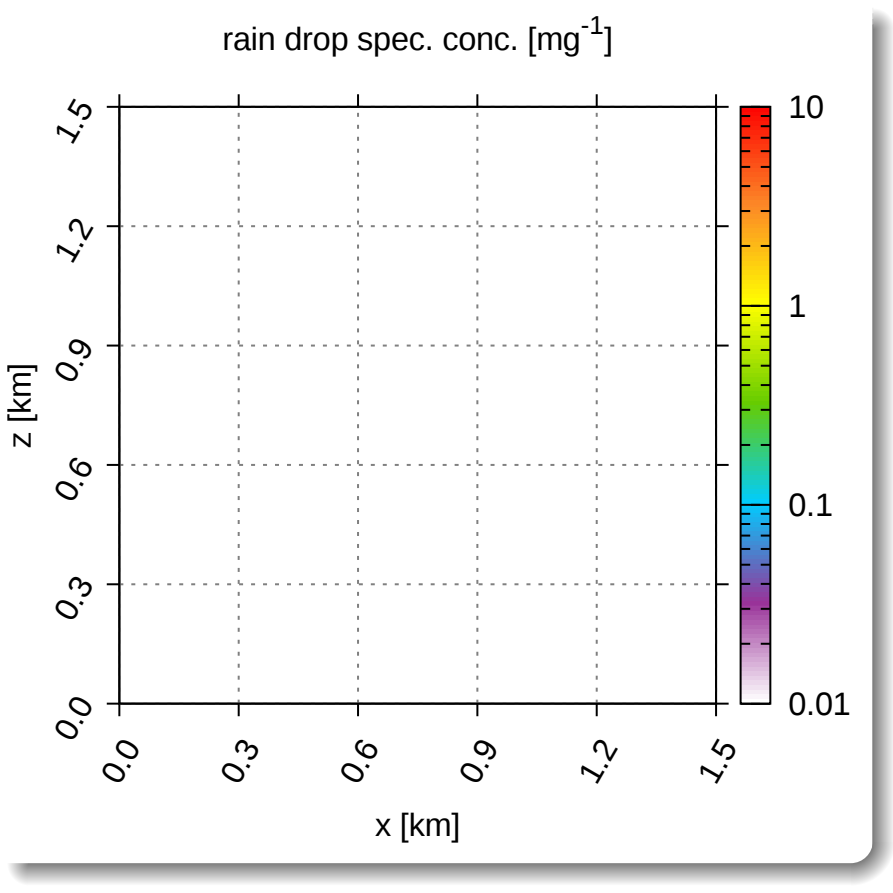




libcloudph++: example 2D prescribed-flow simulation

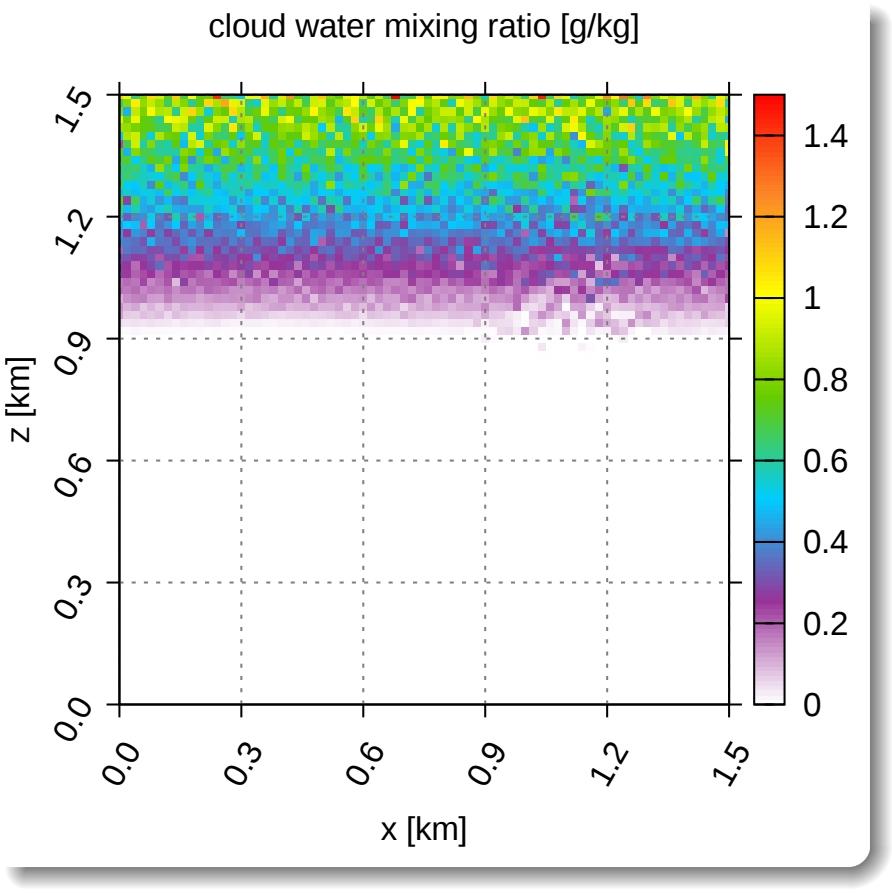
xxx

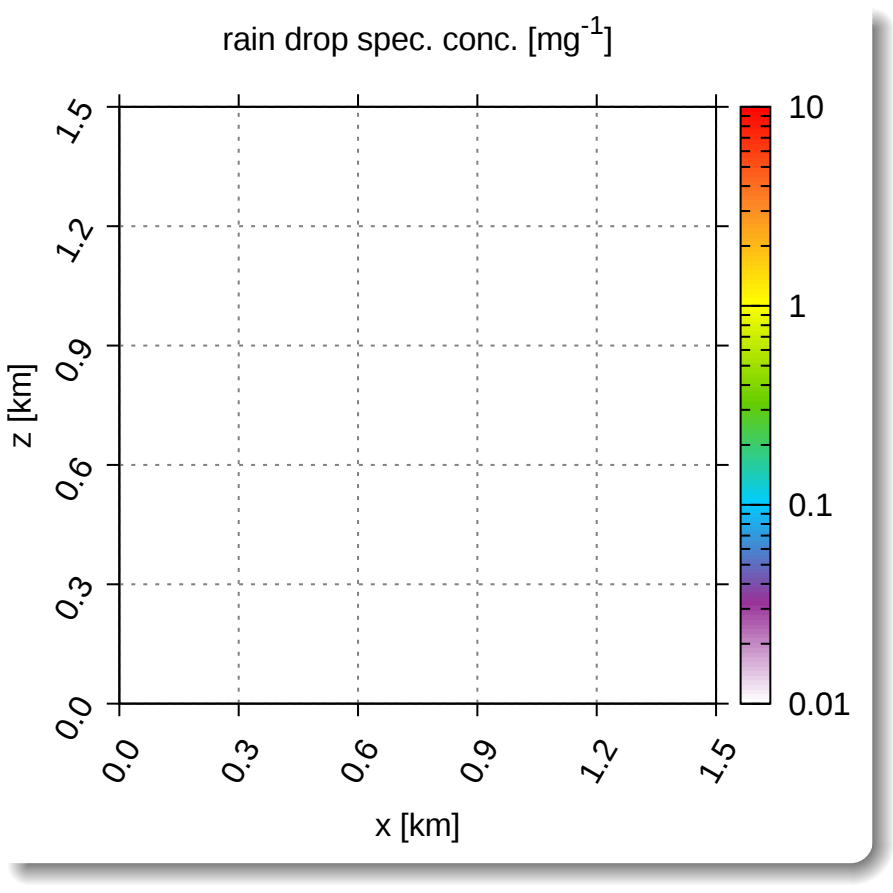




libcloudph++: example 2D prescribed-flow simulation

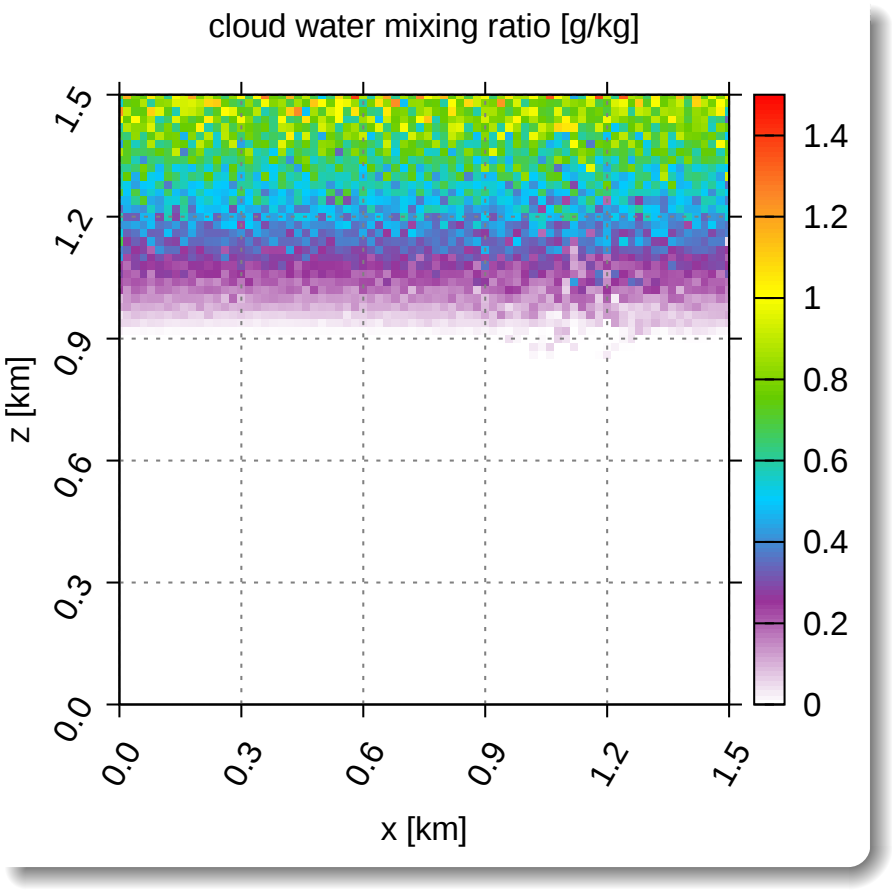
xxxx

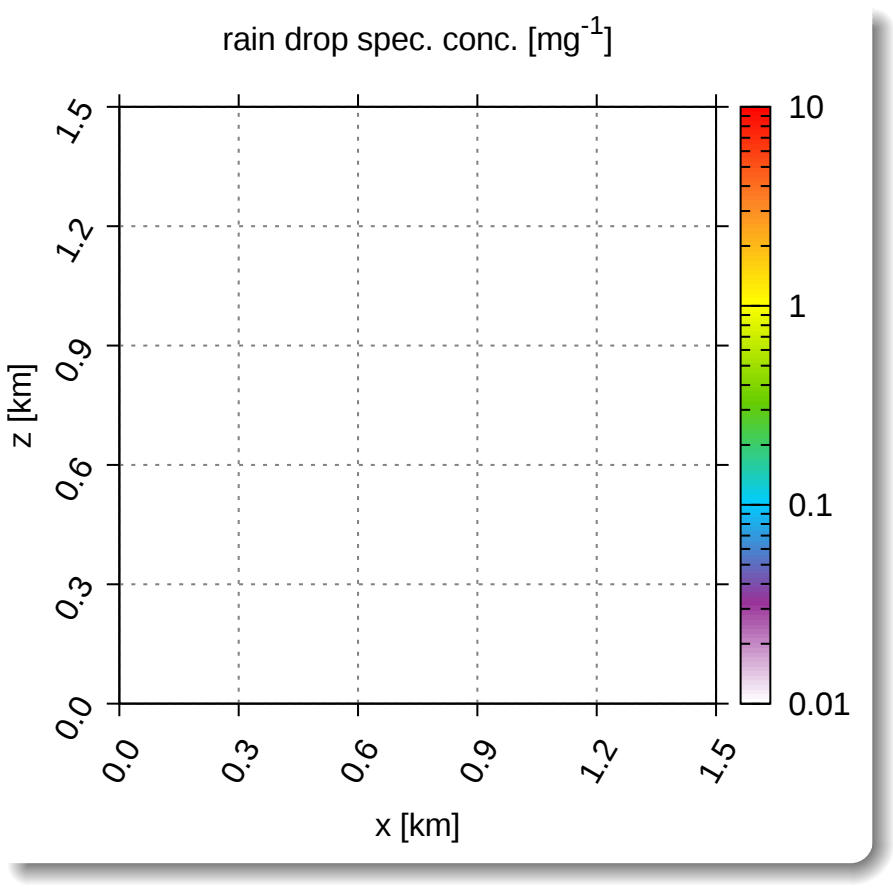




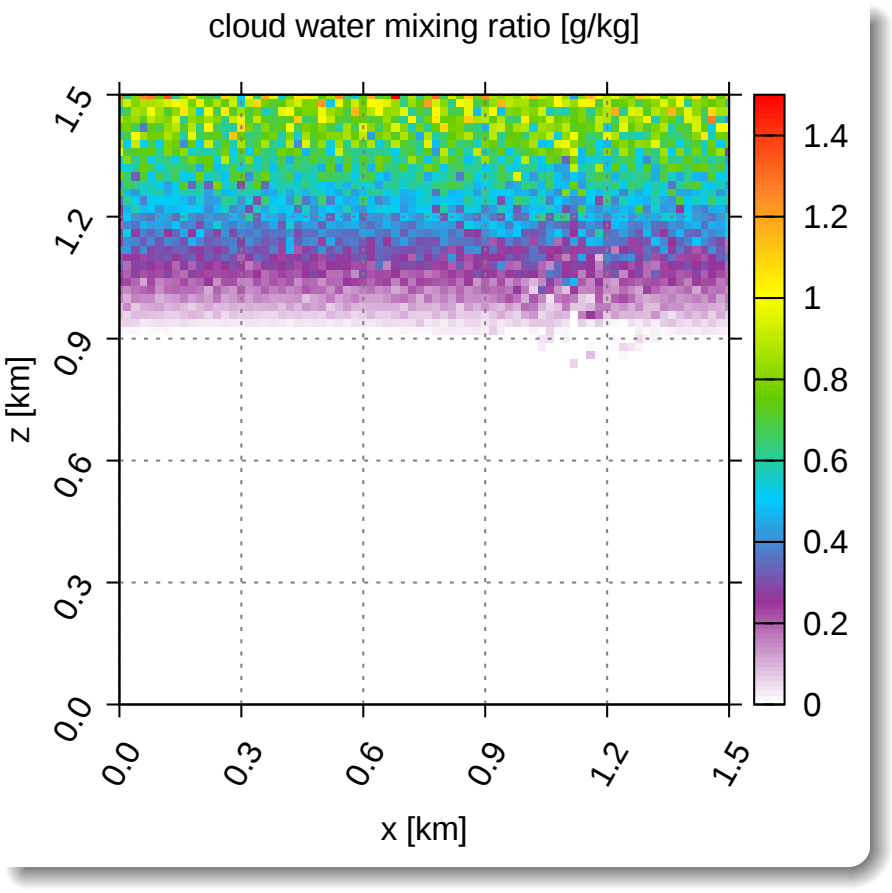
libcloudph++: example 2D prescribed-flow simulation

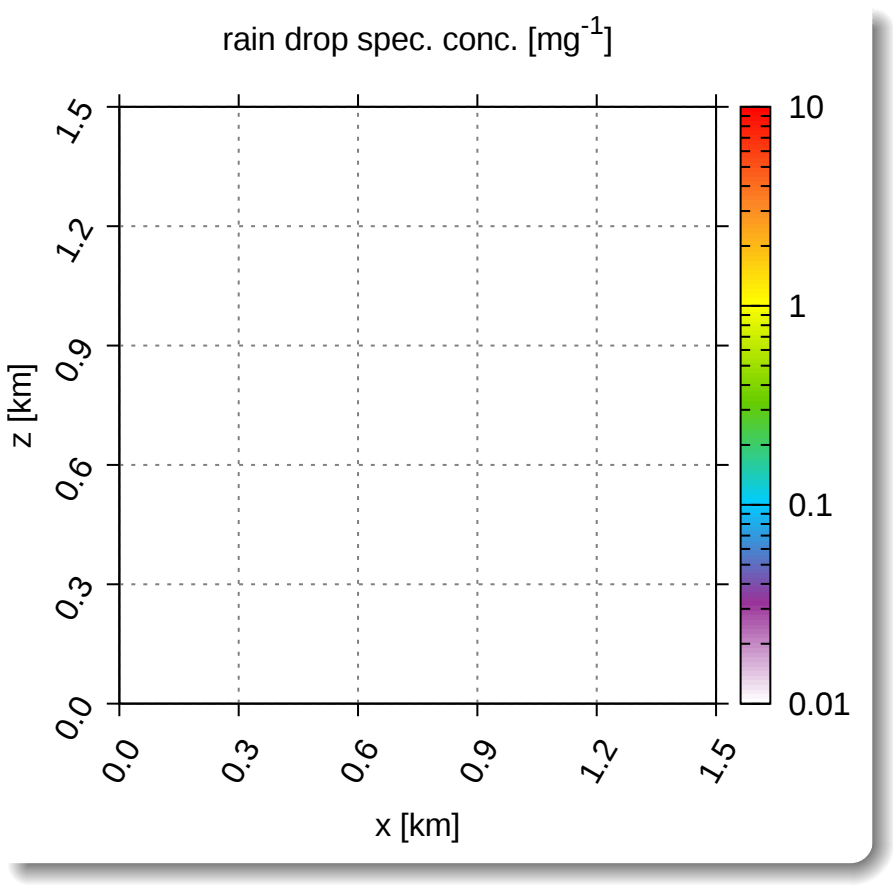
xxxxx



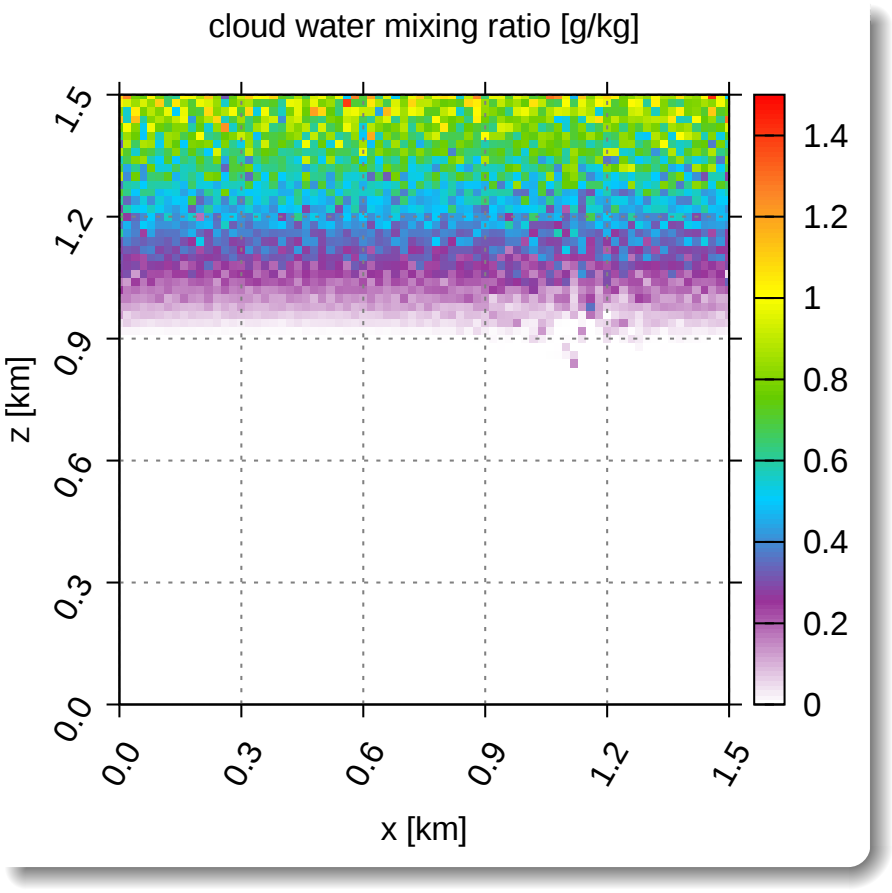


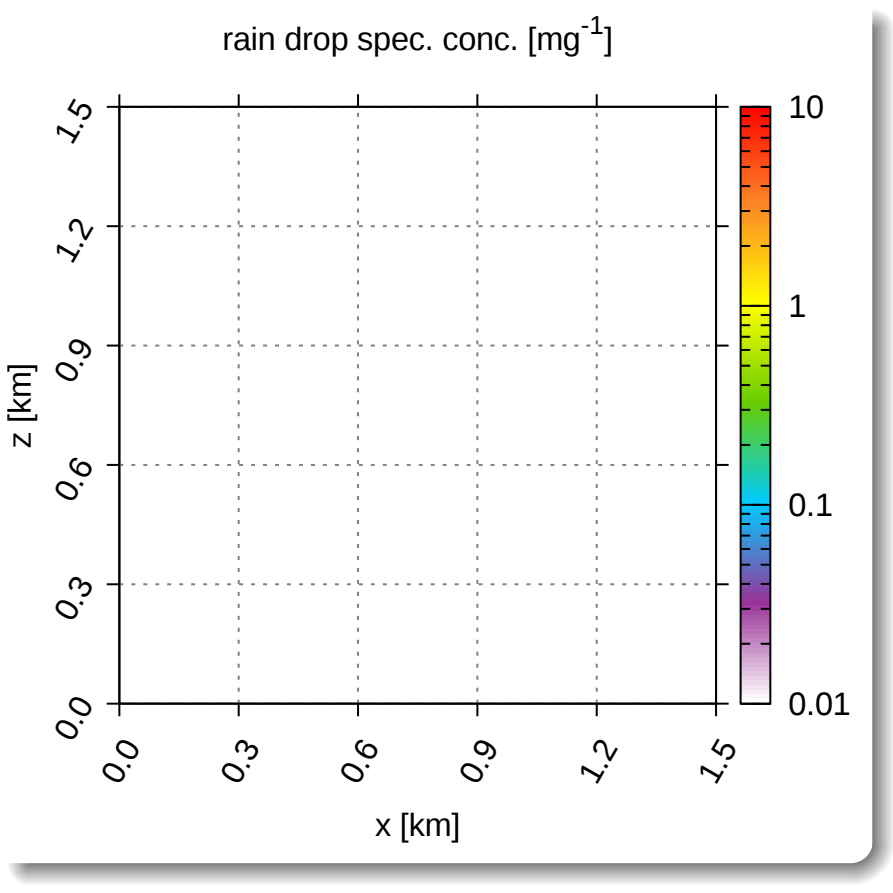
libcloudph++: example 2D prescribed-flow simulation
xxxxxx



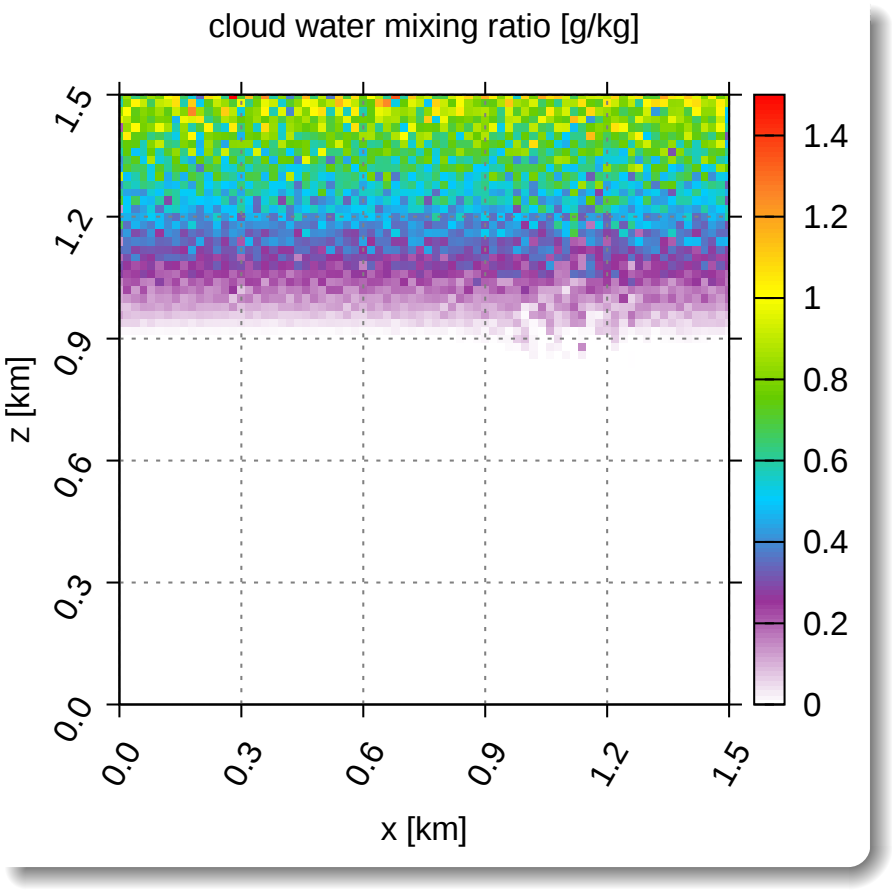


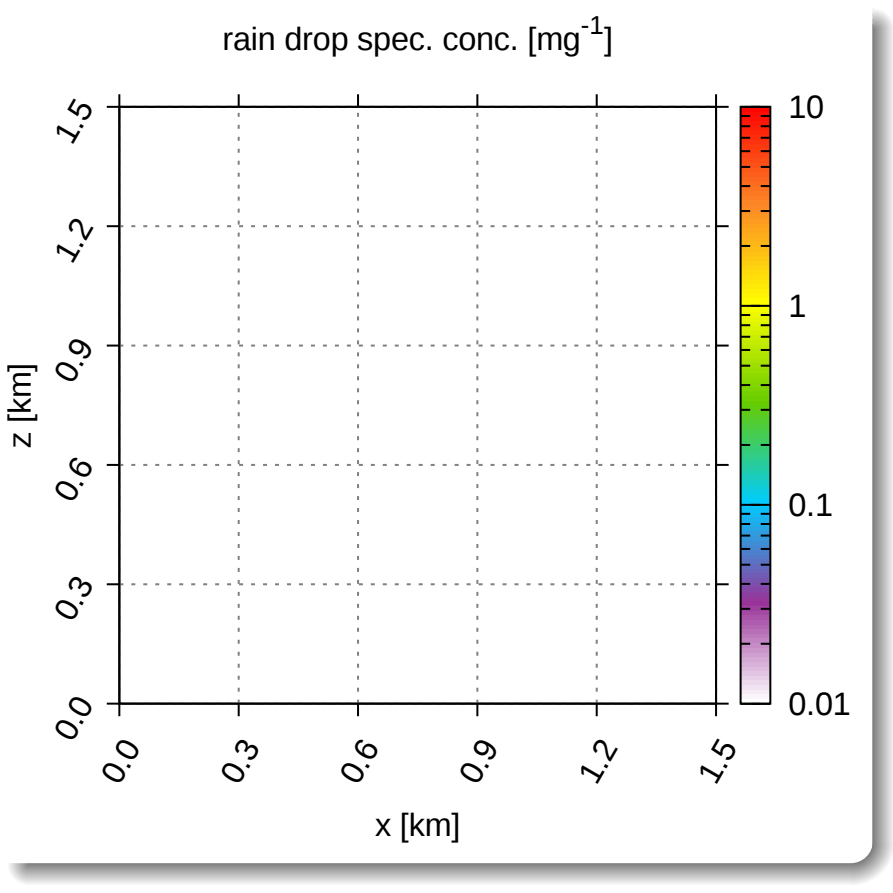
libcloudph++: example 2D prescribed-flow simulation
xxxxxxx





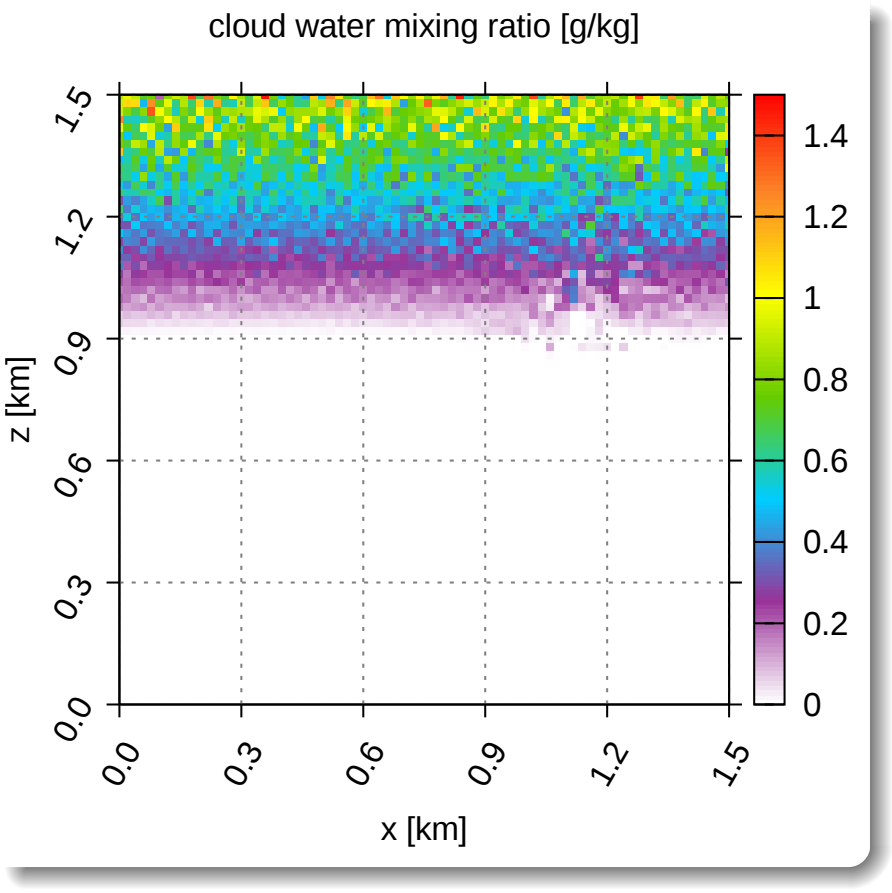
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxx

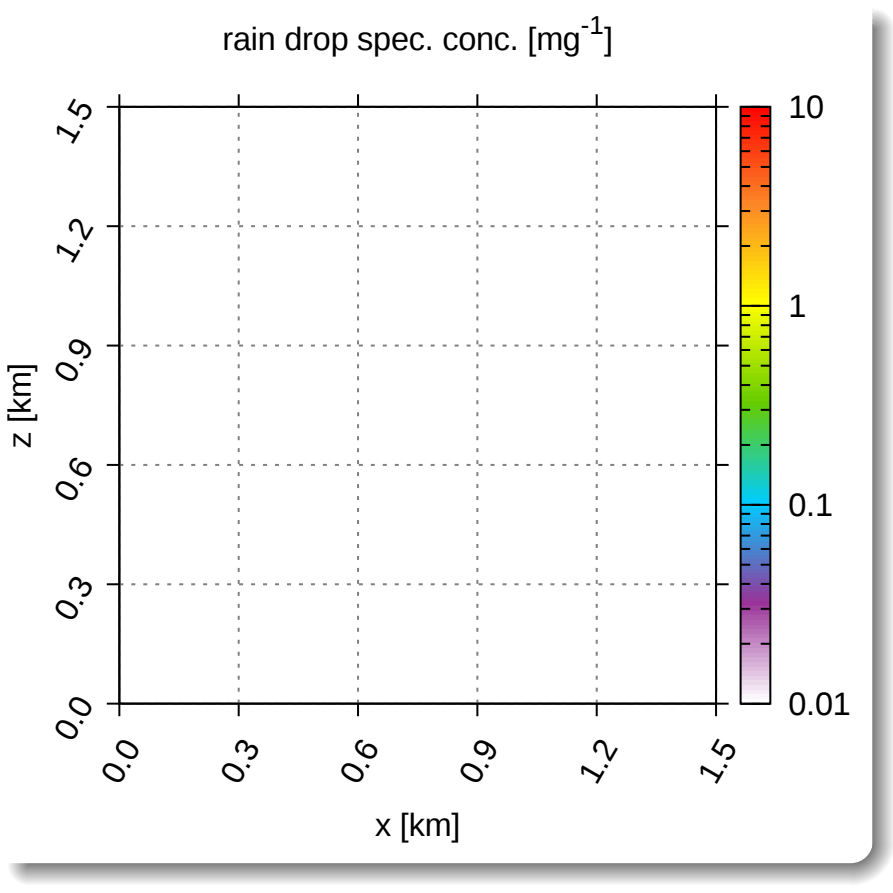




libcloudph++: example 2D prescribed-flow simulation

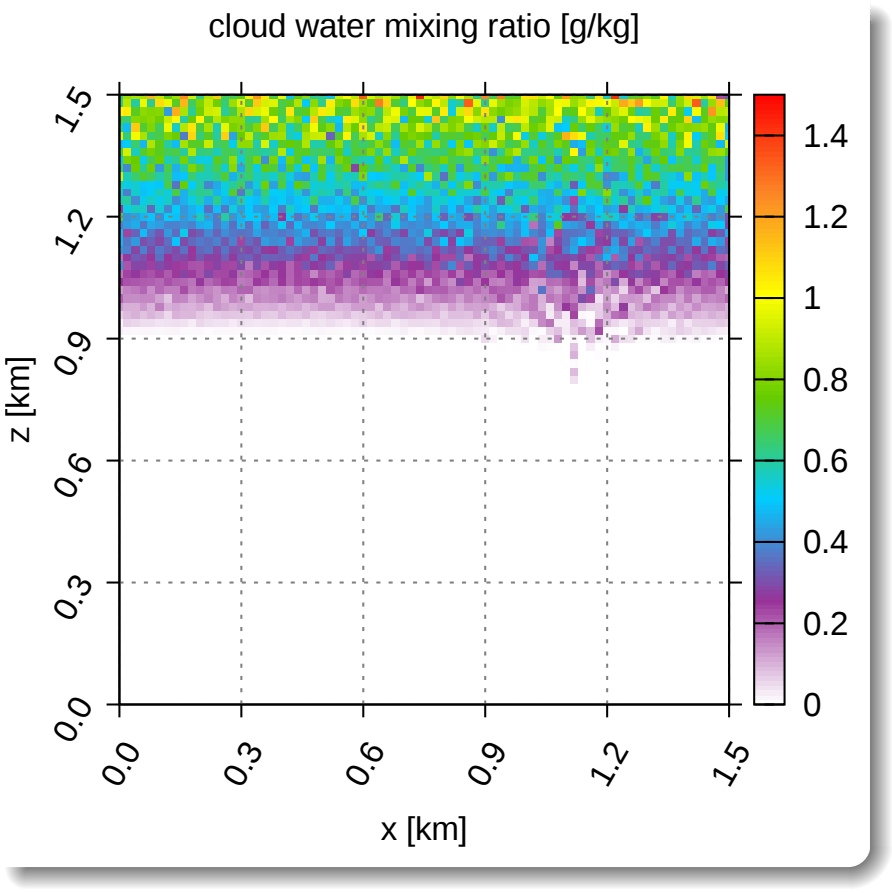
xxxxxxxxxx

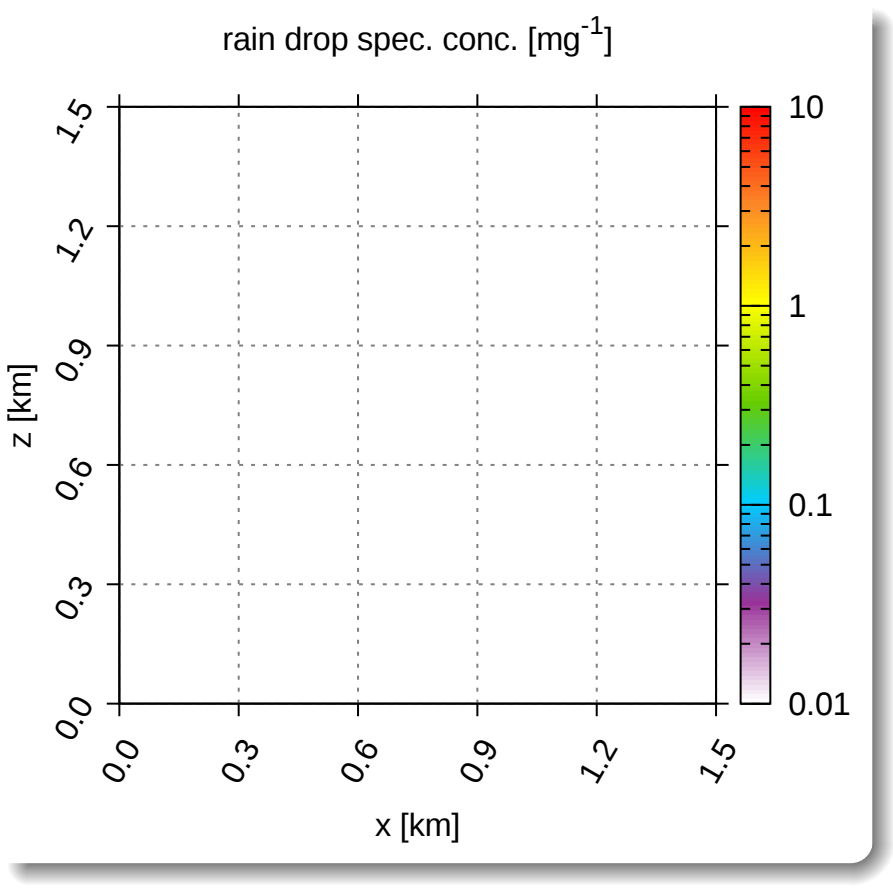




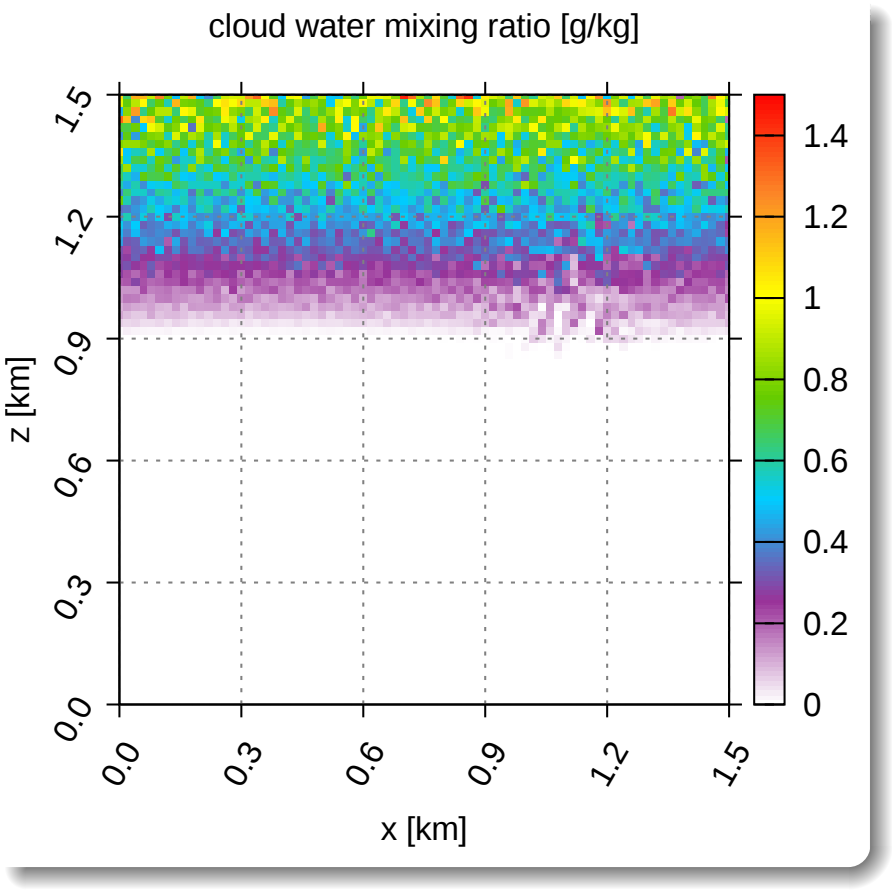
libcloudph++: example 2D prescribed-flow simulation

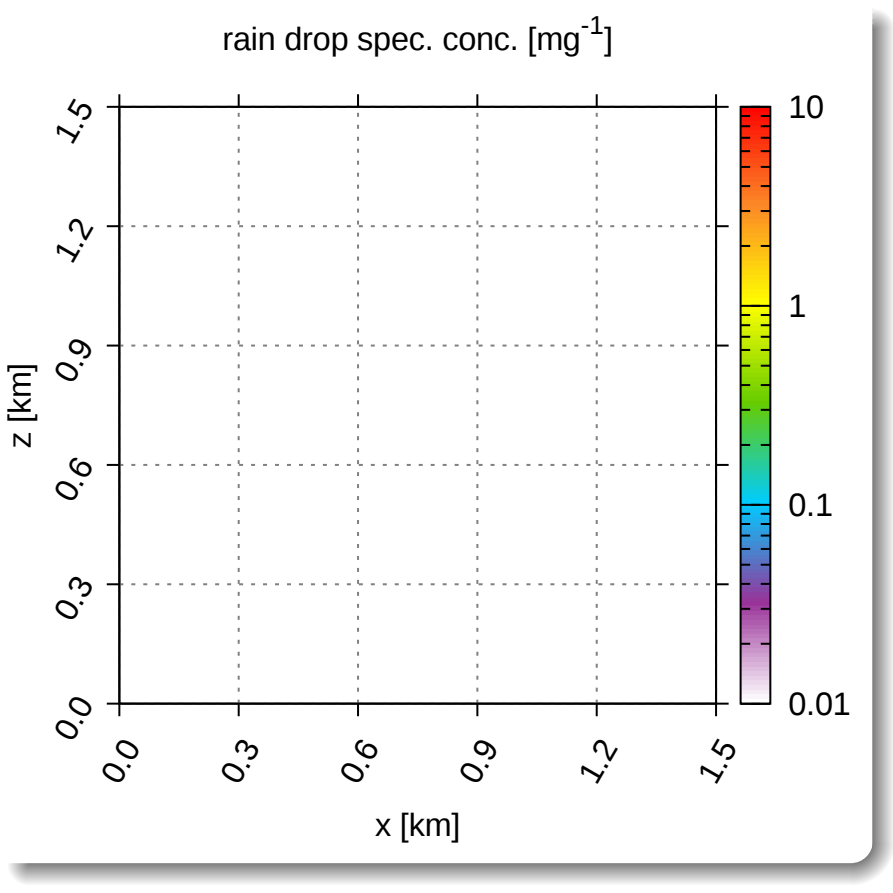
xxxxxxxxxxx



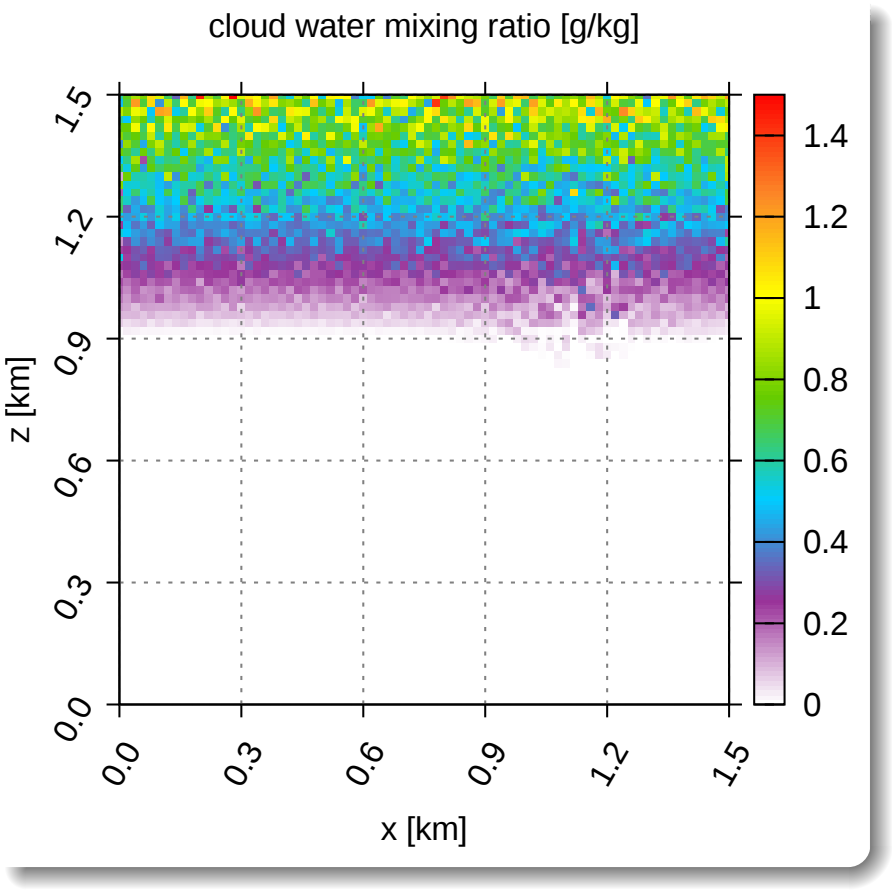


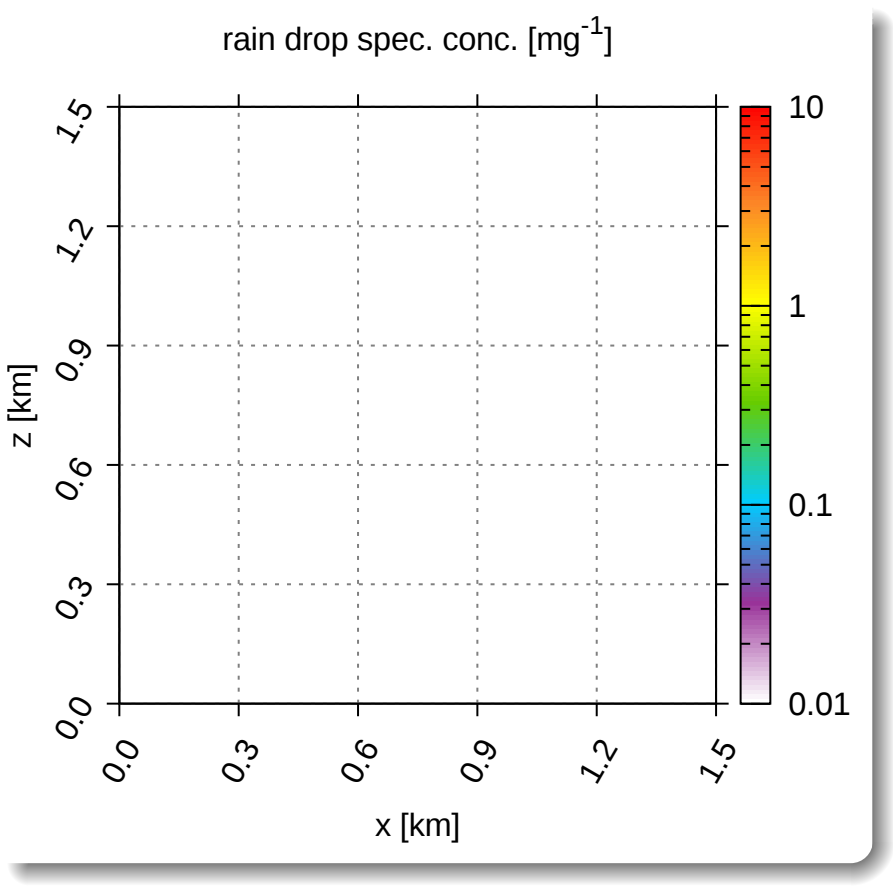
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxx



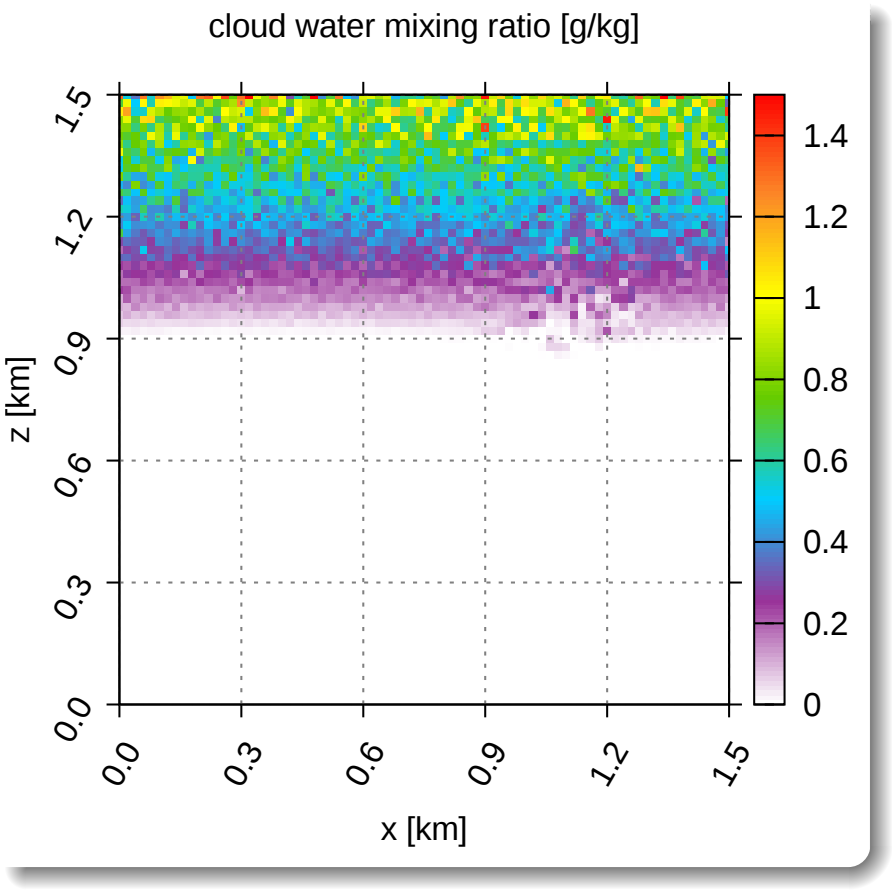


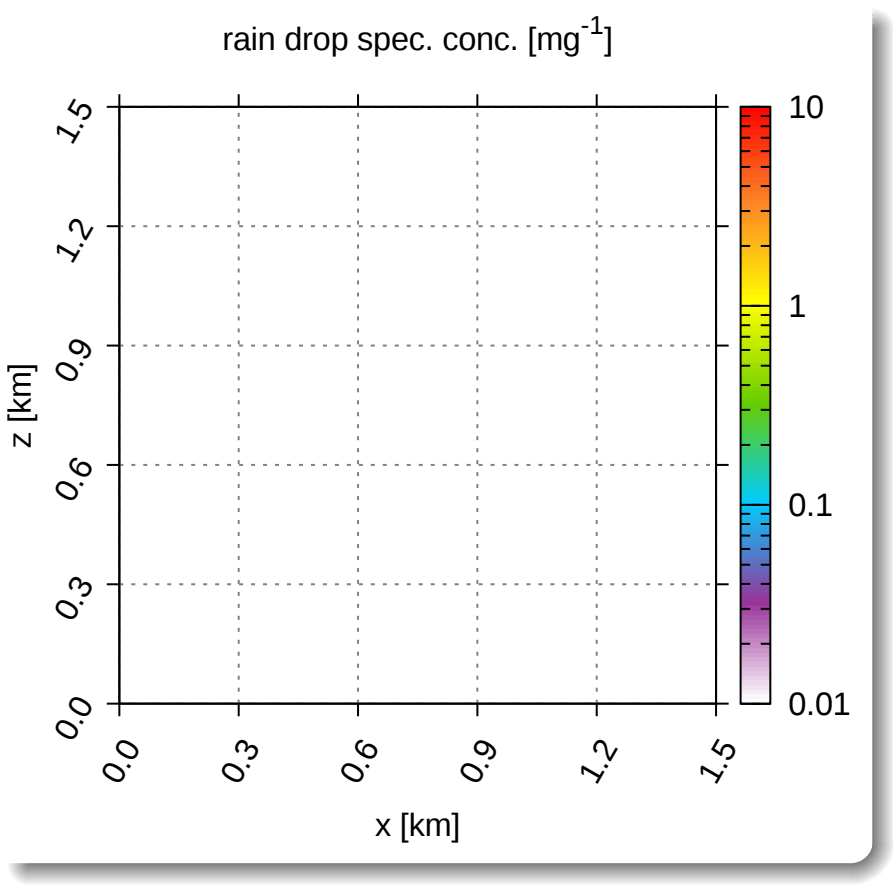
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxxx



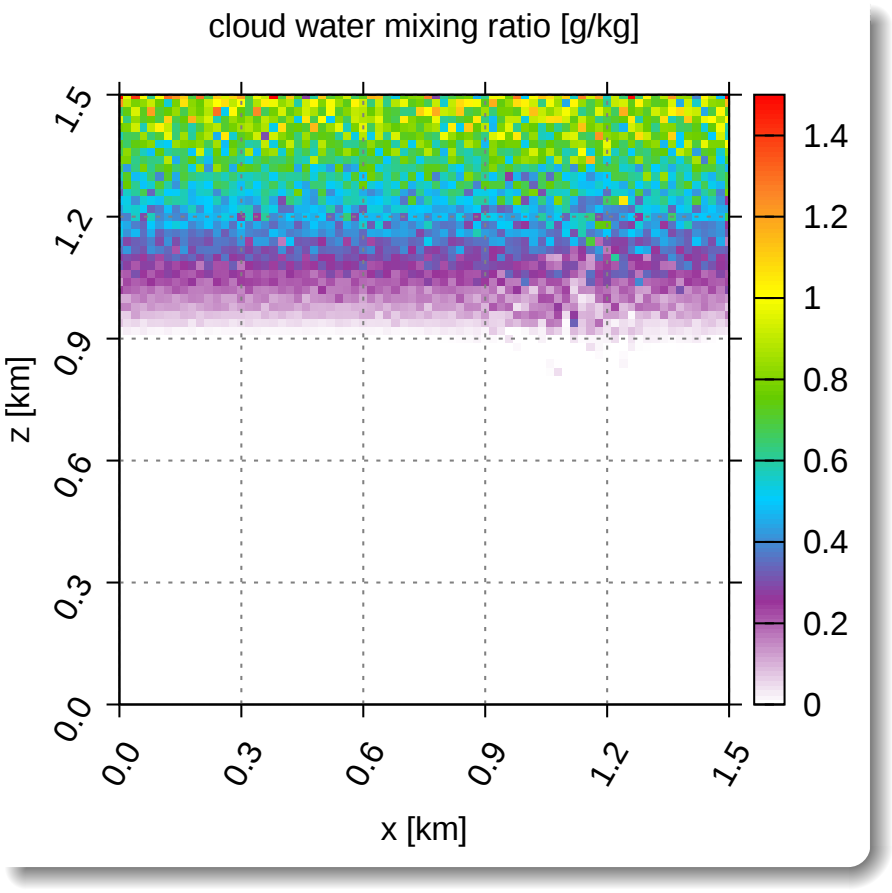


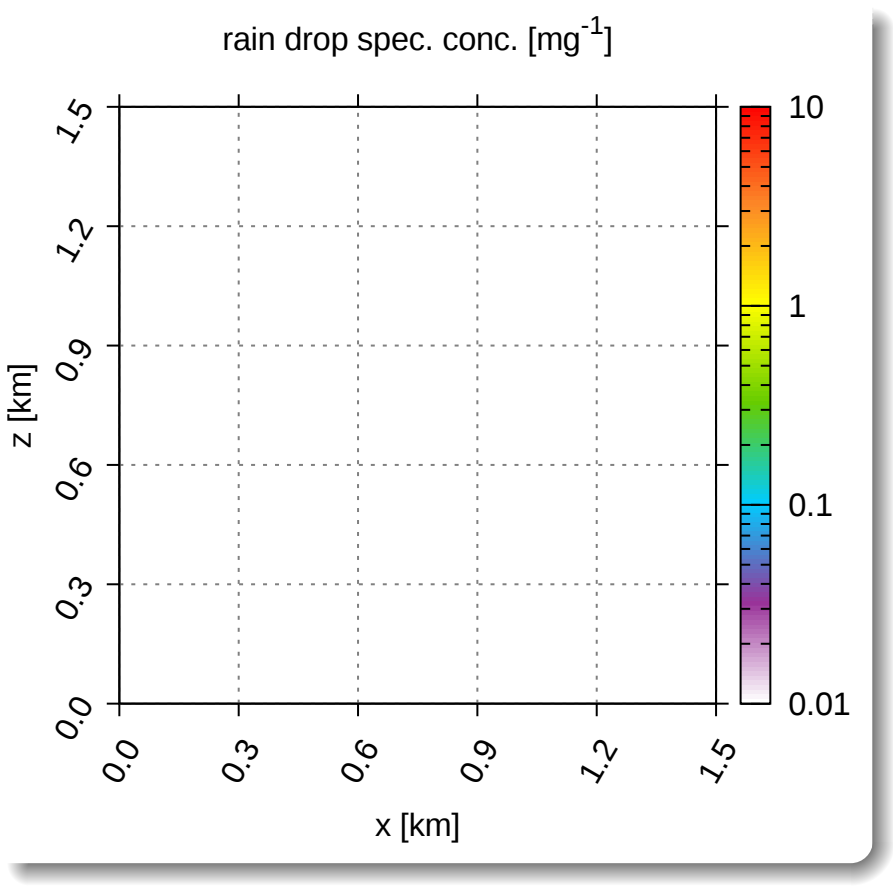
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxxxx





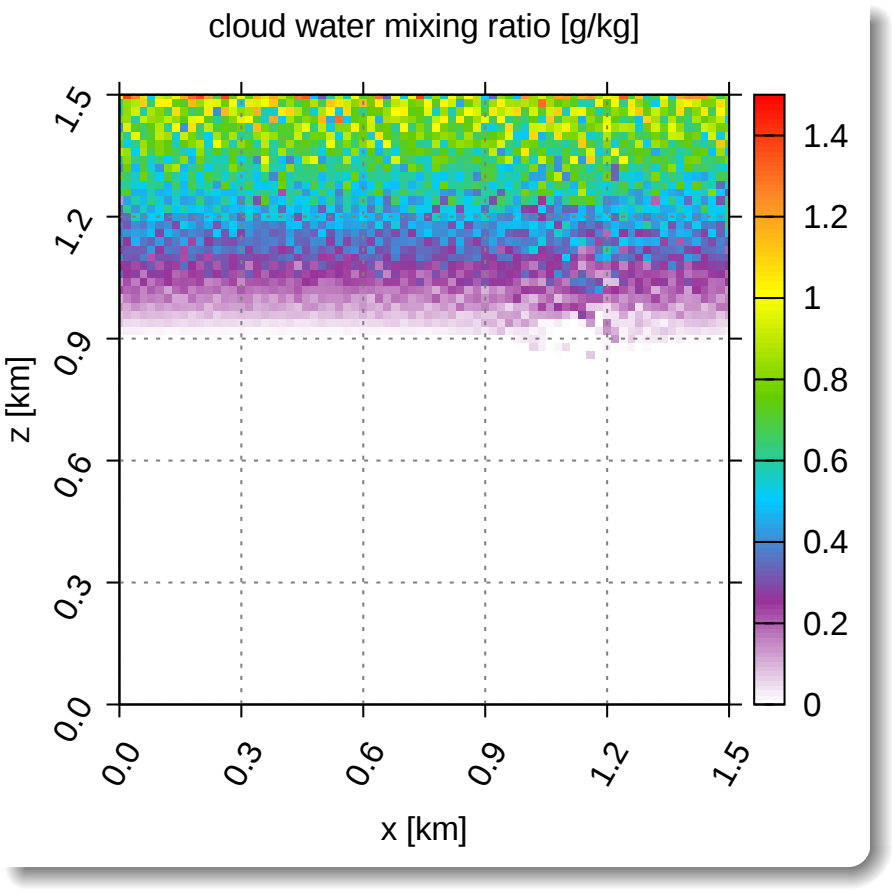
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxxxxx

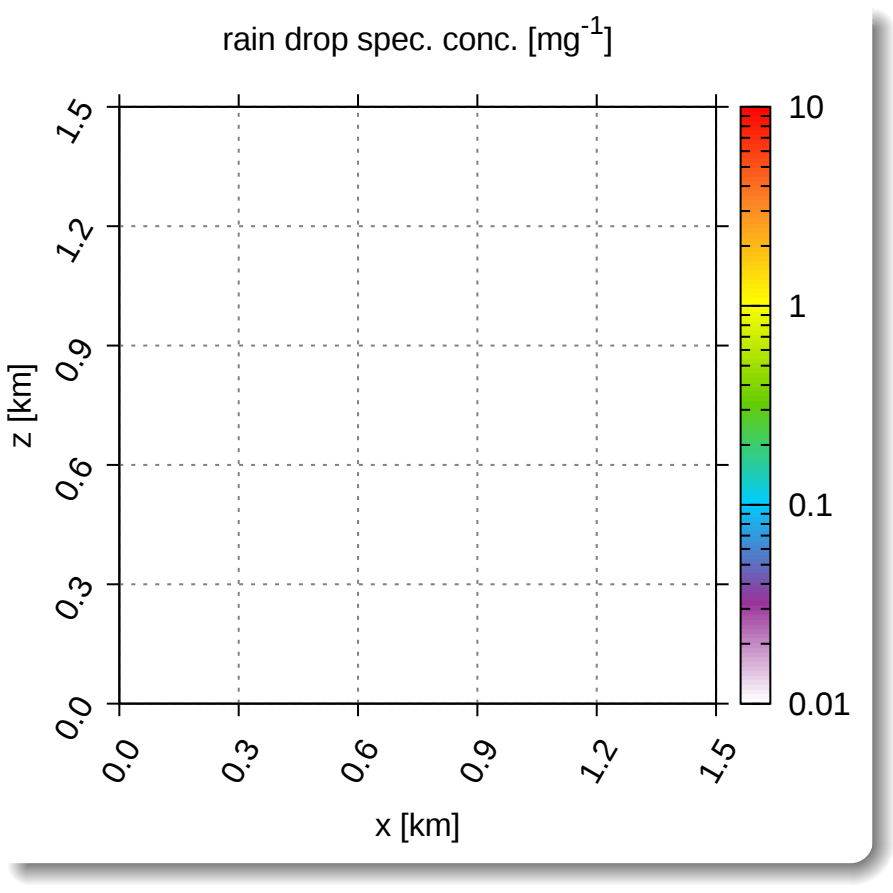




libcloudph++: example 2D prescribed-flow simulation

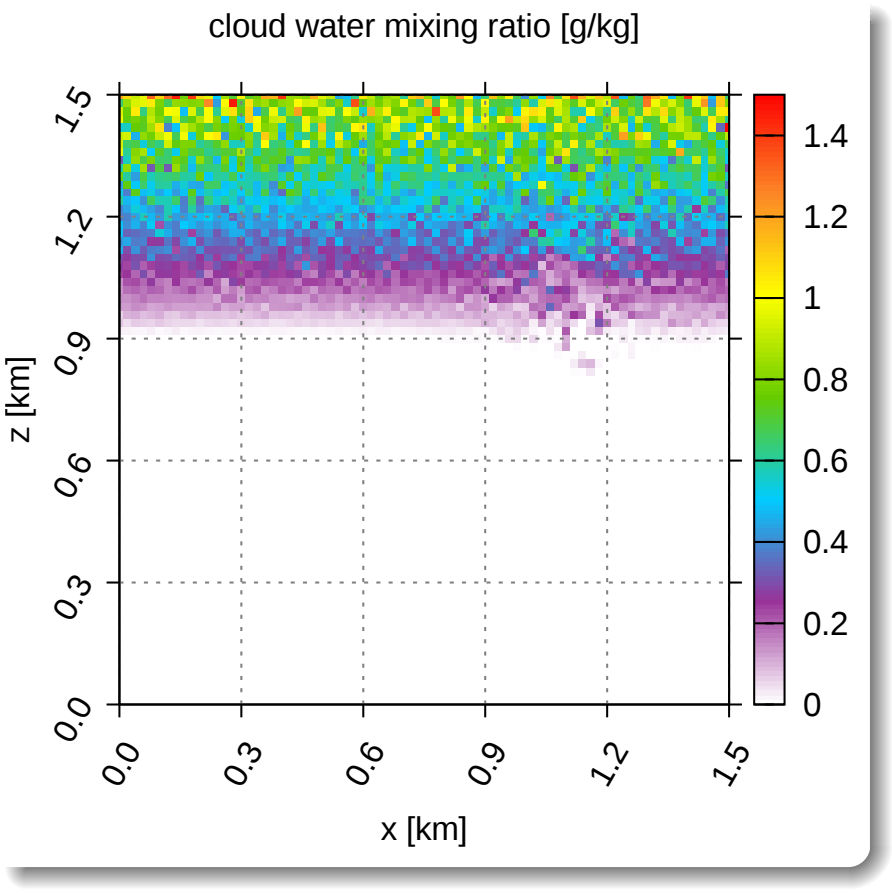
xxxxxxxxxxxxxxxx

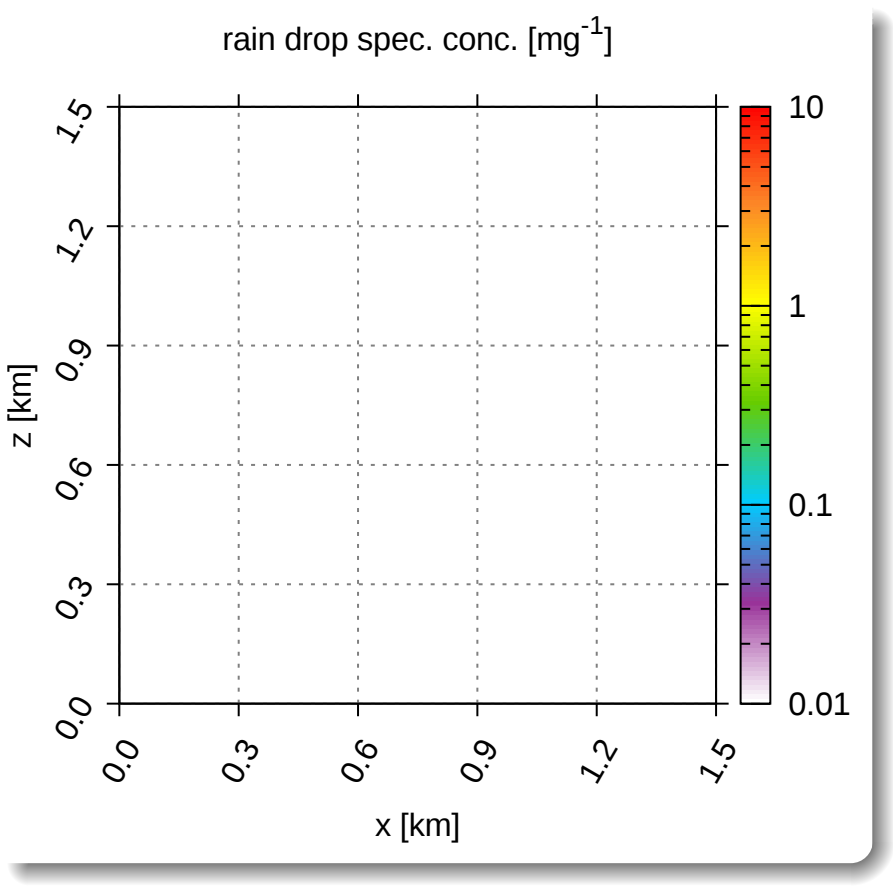




libcloudph++: example 2D prescribed-flow simulation

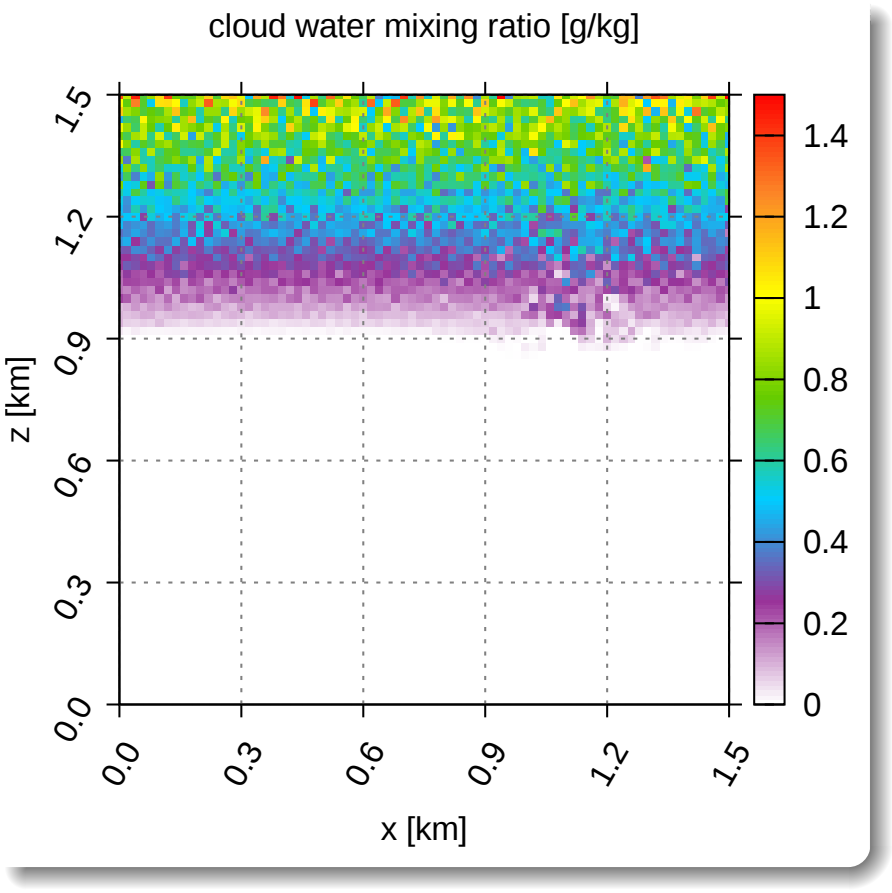
xxxxxxxxxxxxxxxxxx

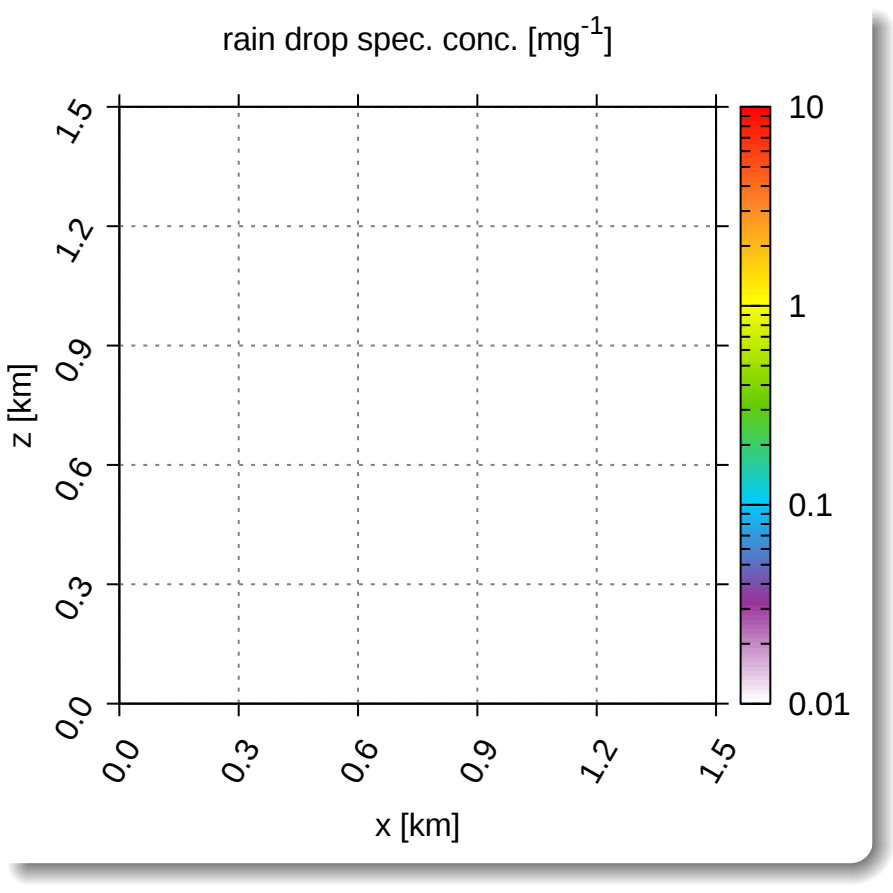




libcloudph++: example 2D prescribed-flow simulation

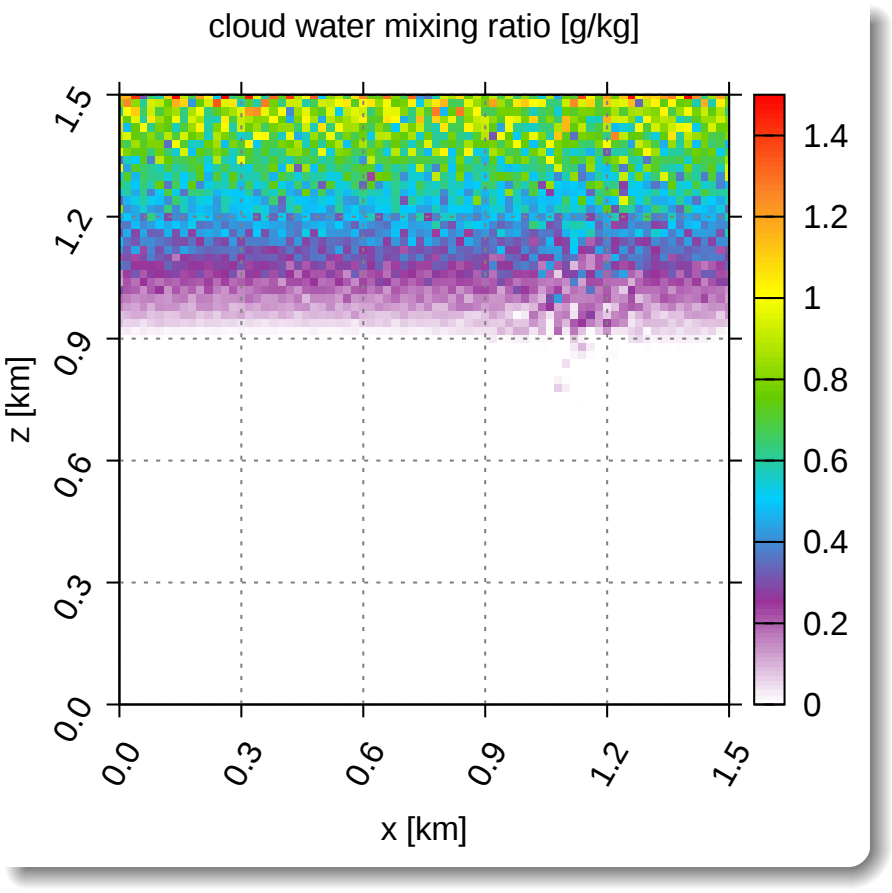
xxxxxxxxxxxxxxxxxxx

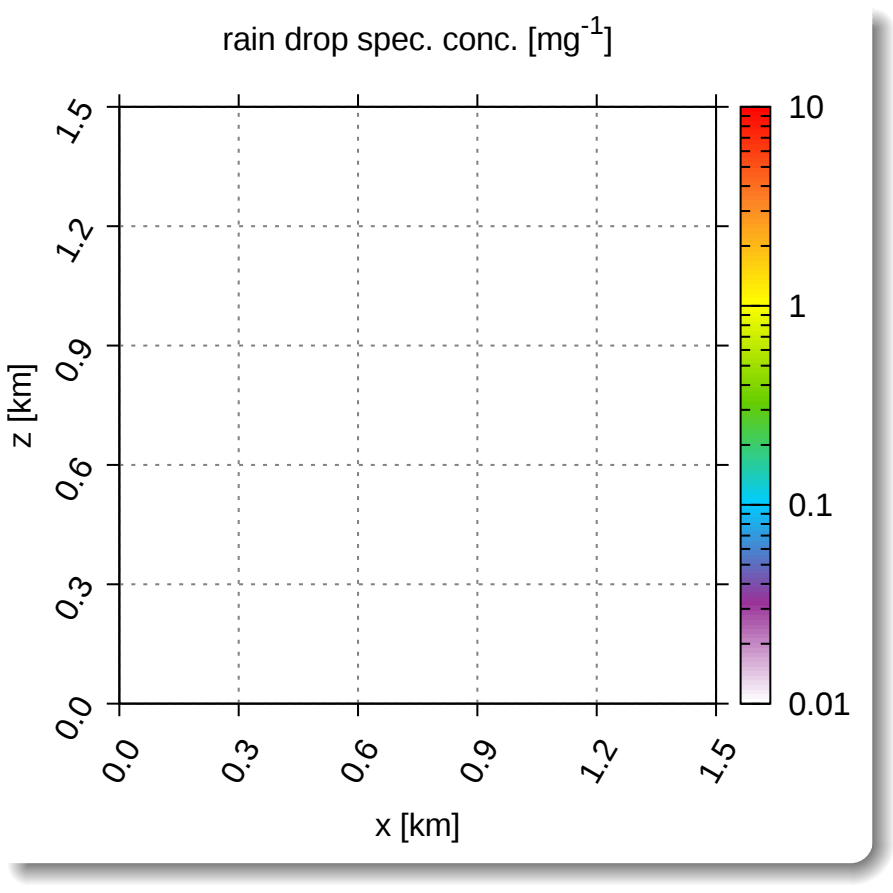




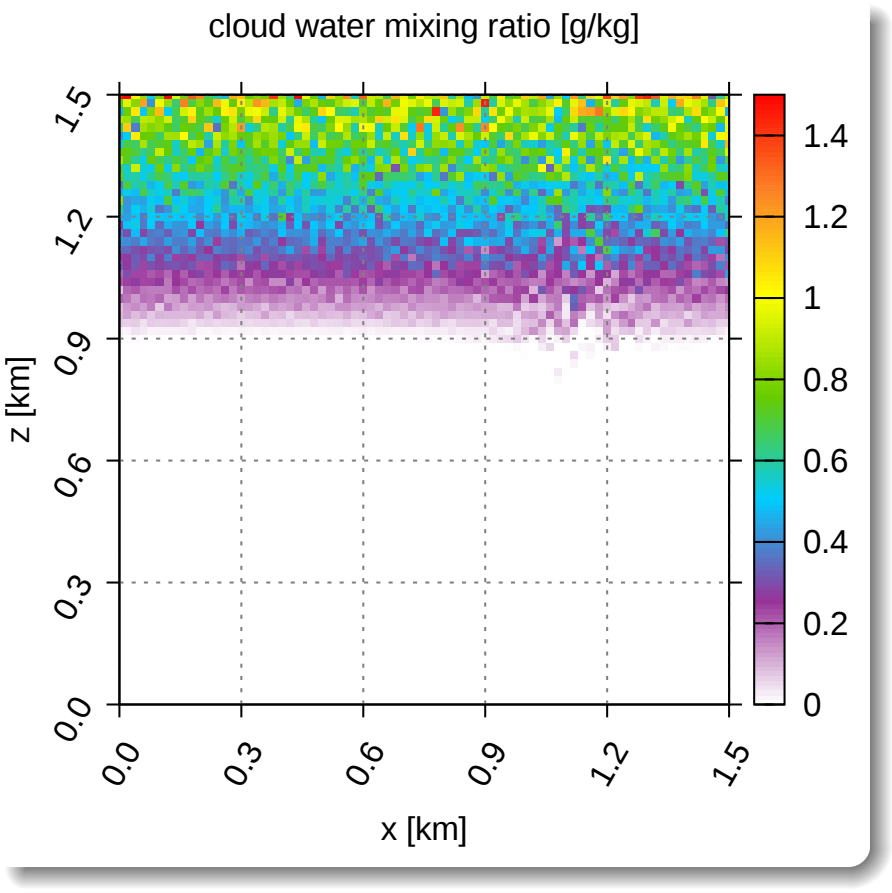
libcloudph++: example 2D prescribed-flow simulation

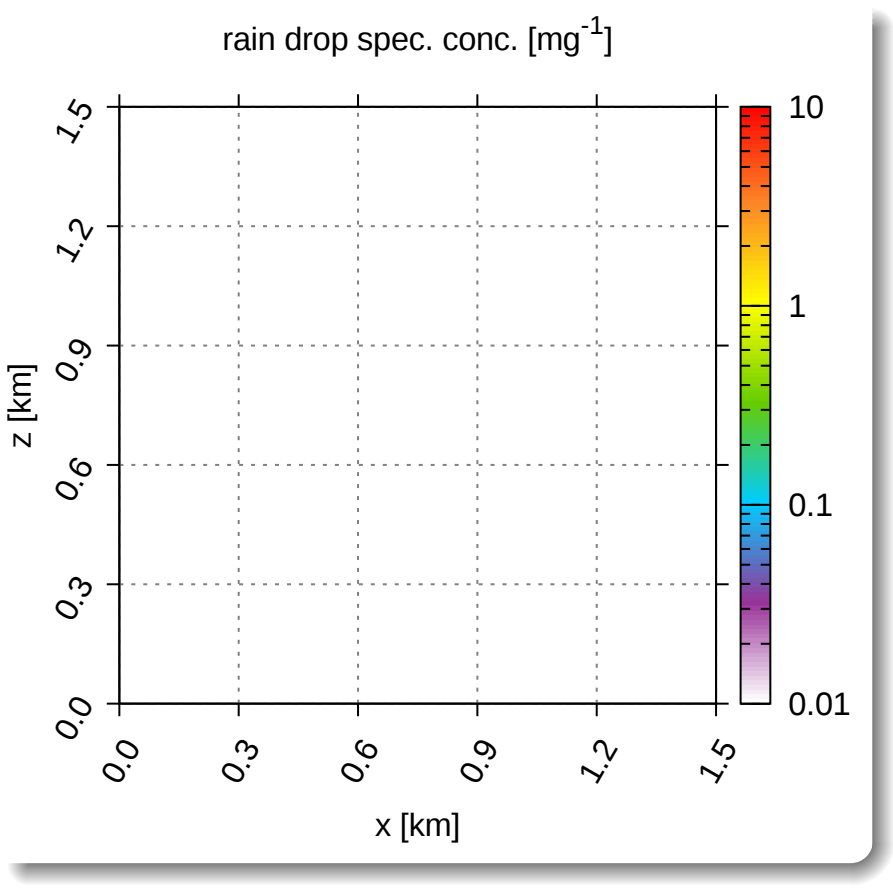
xxxxxxxxxxxxxxxxxxxx



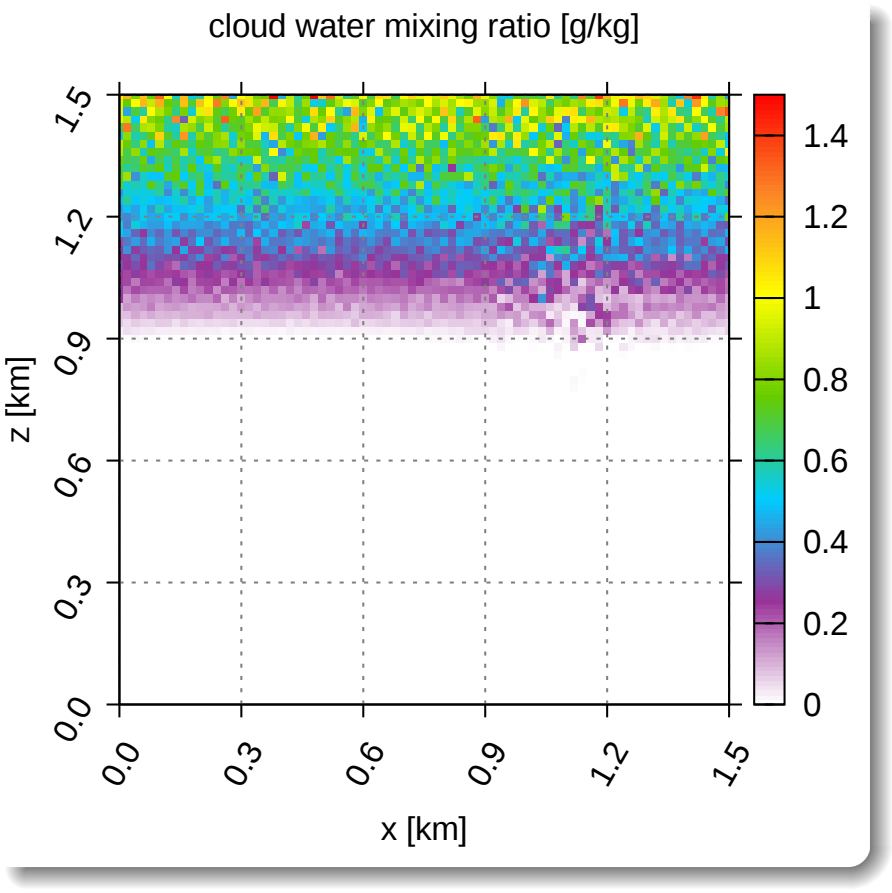


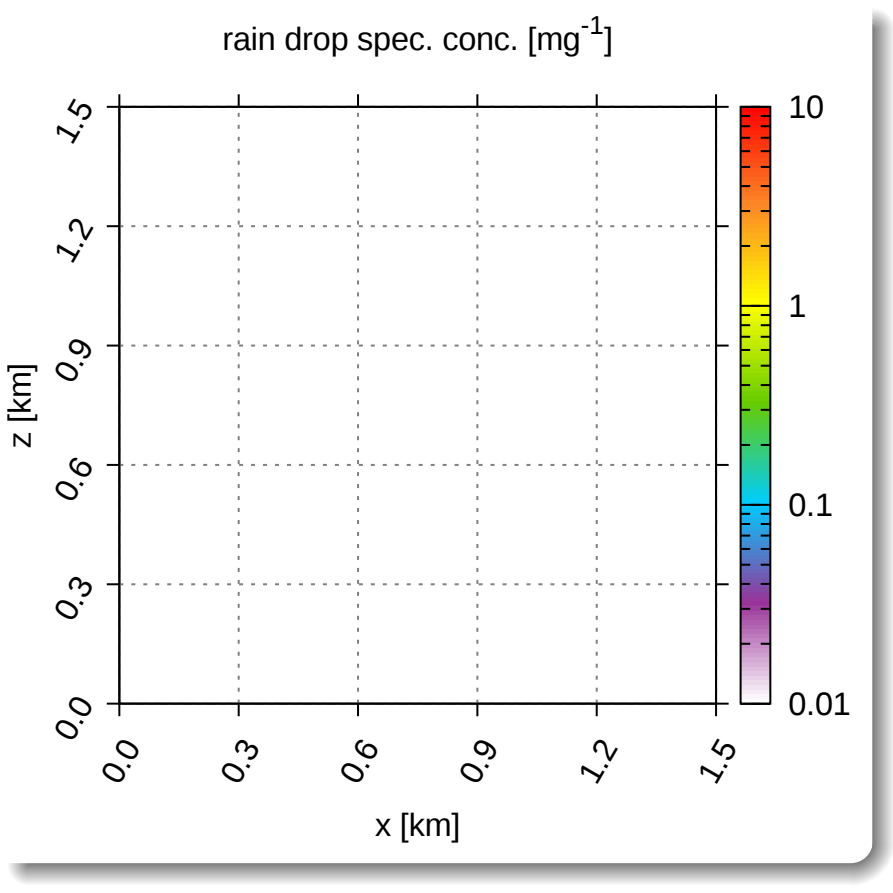
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxxxxxxxxxxx





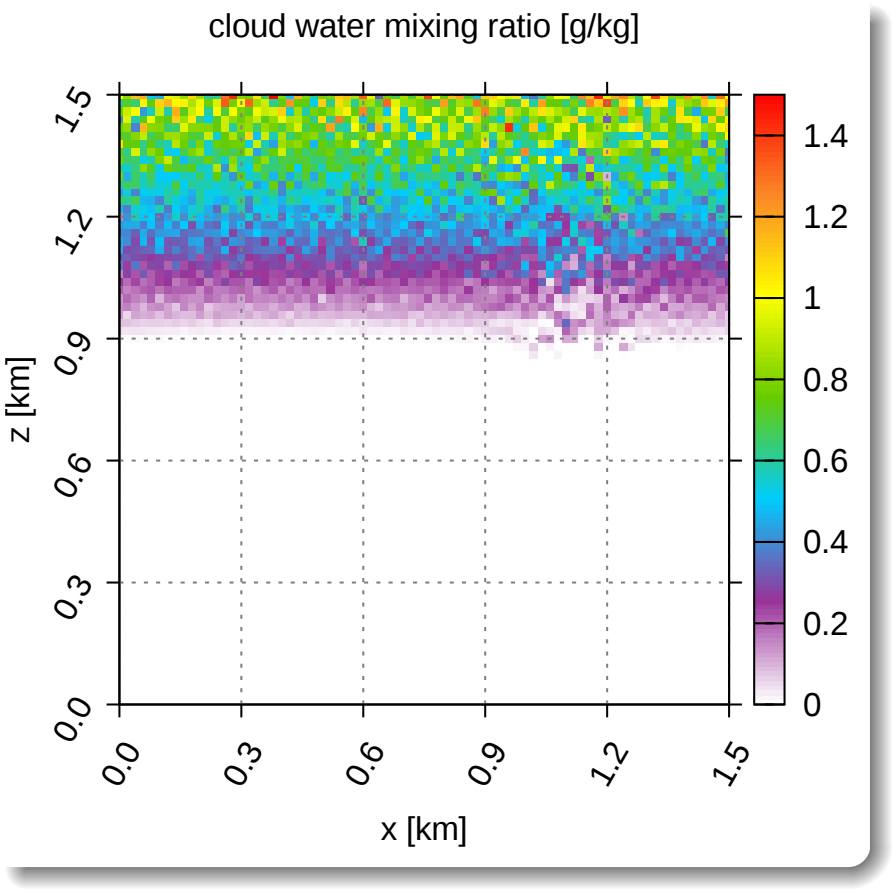
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxxxxxxxxxxxxx

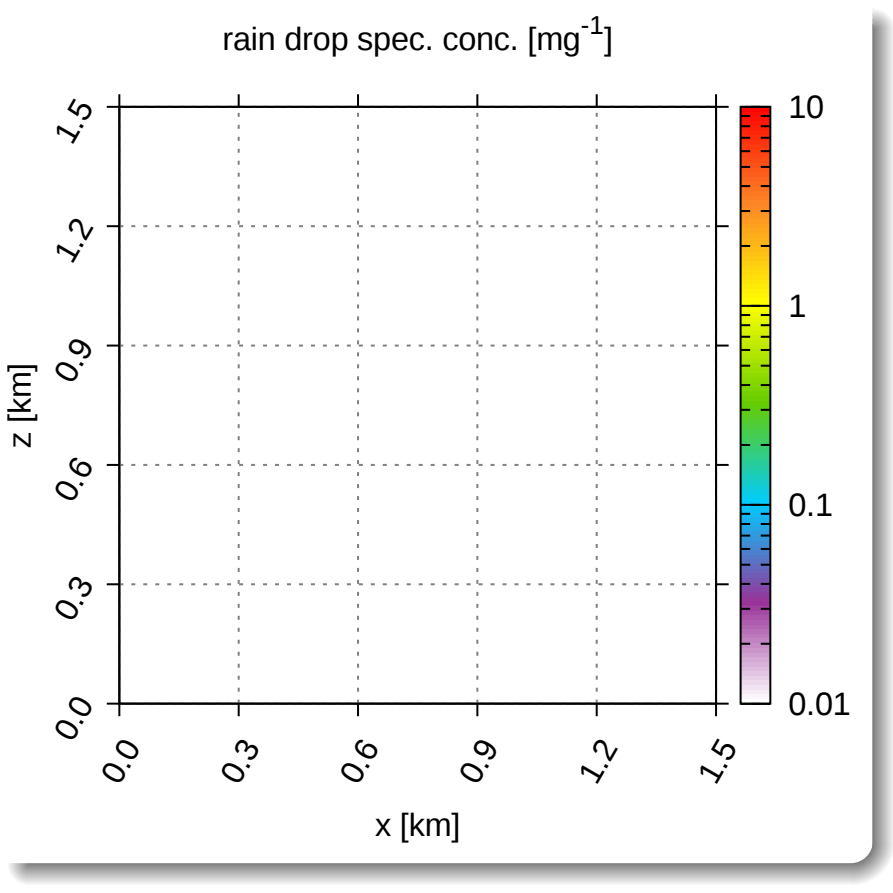




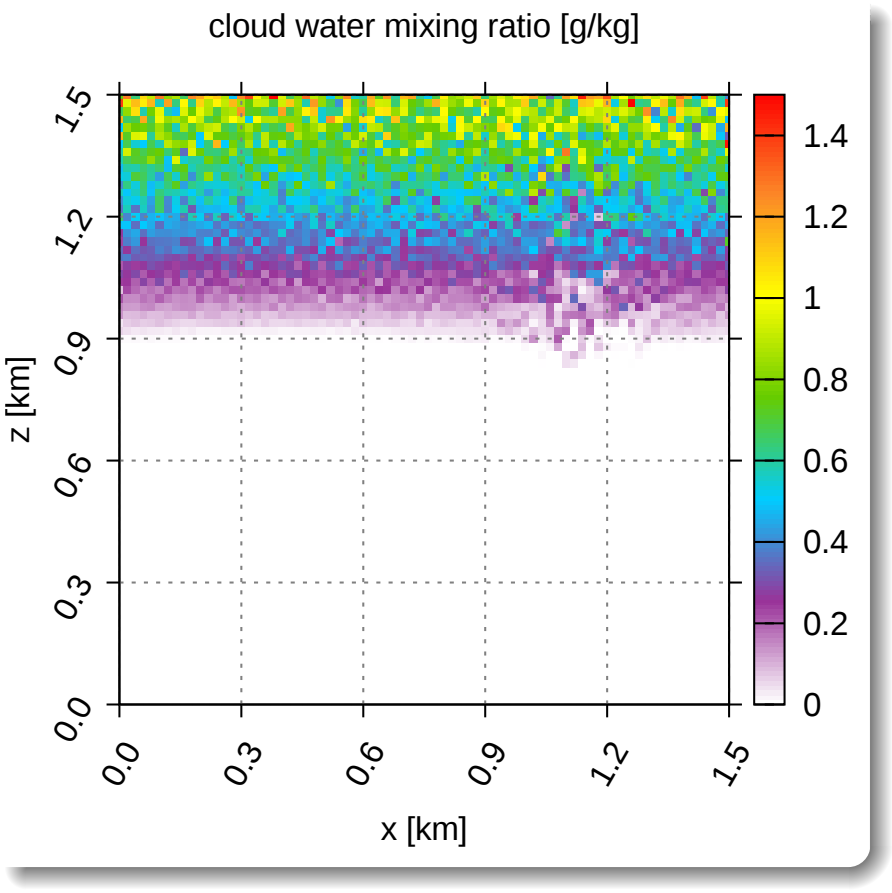
libcloudph++: example 2D prescribed-flow simulation

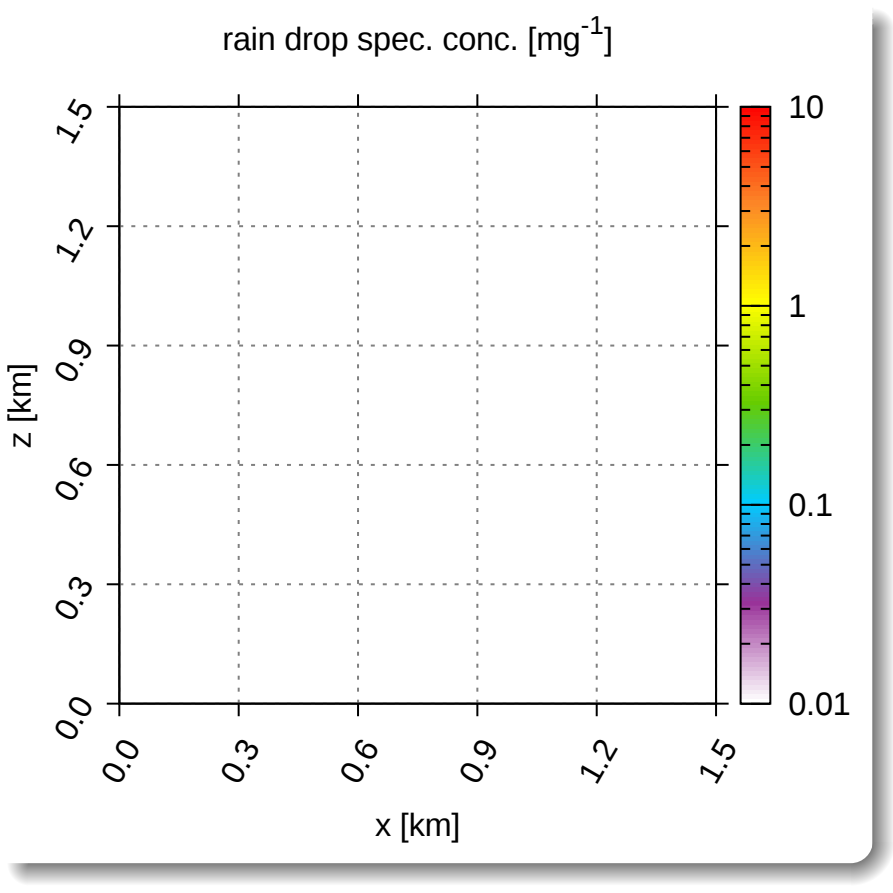
xxxxxxxxxxxxxxxxxxxxxxxx



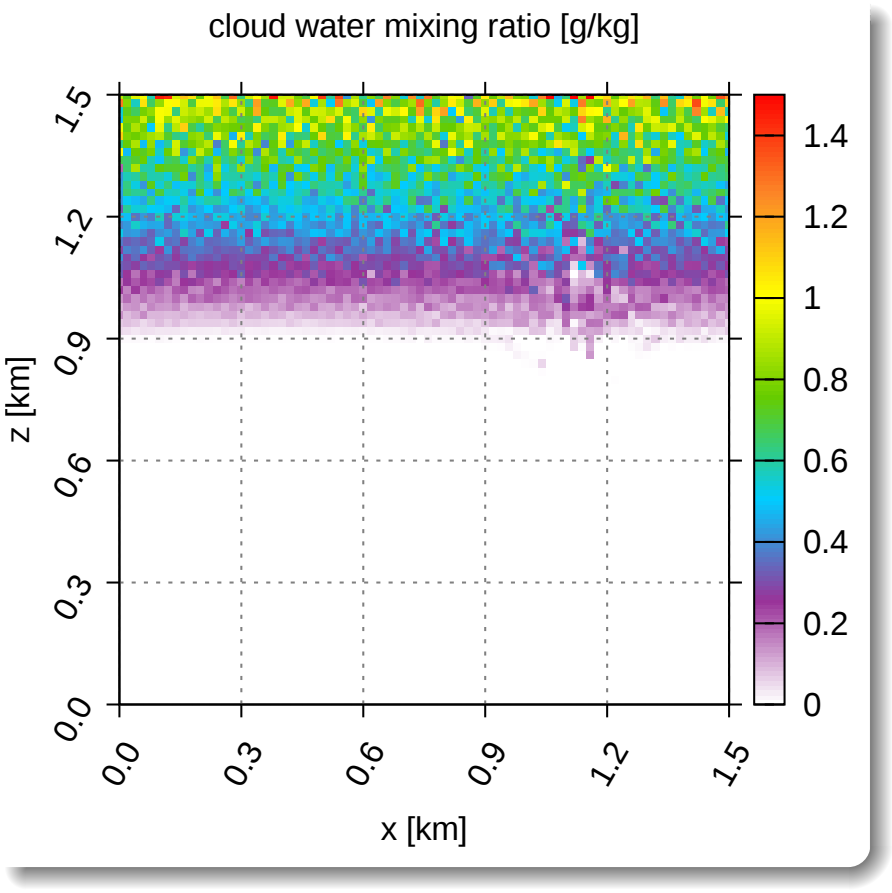


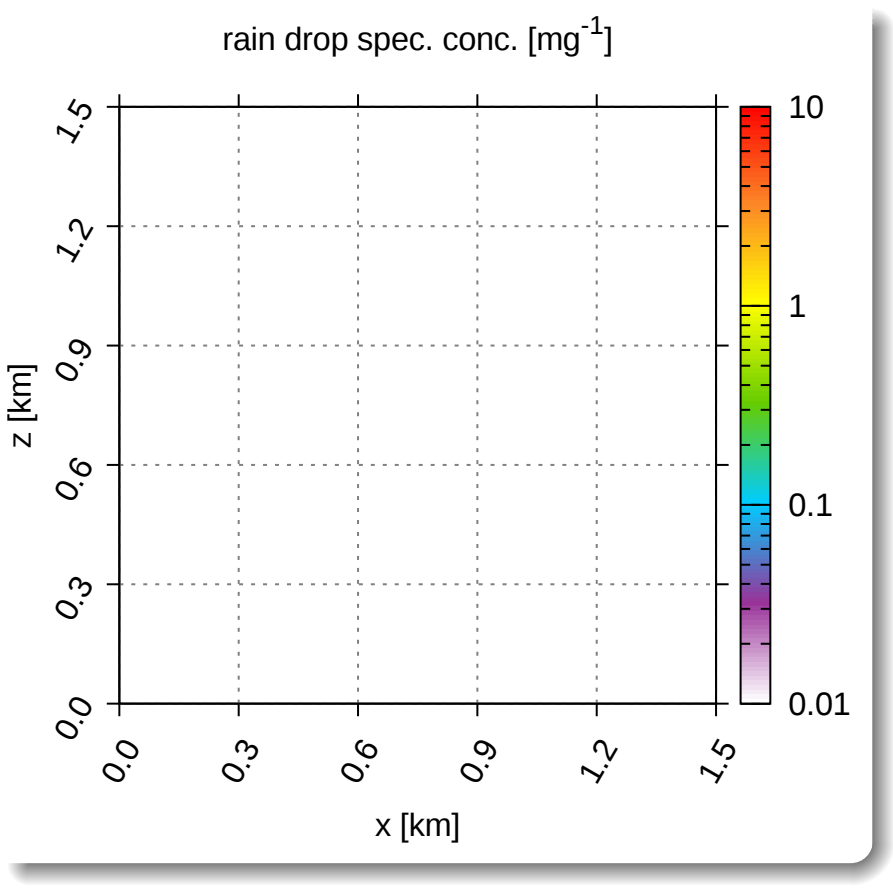
libcloudph++: example 2D prescribed-flow simulation





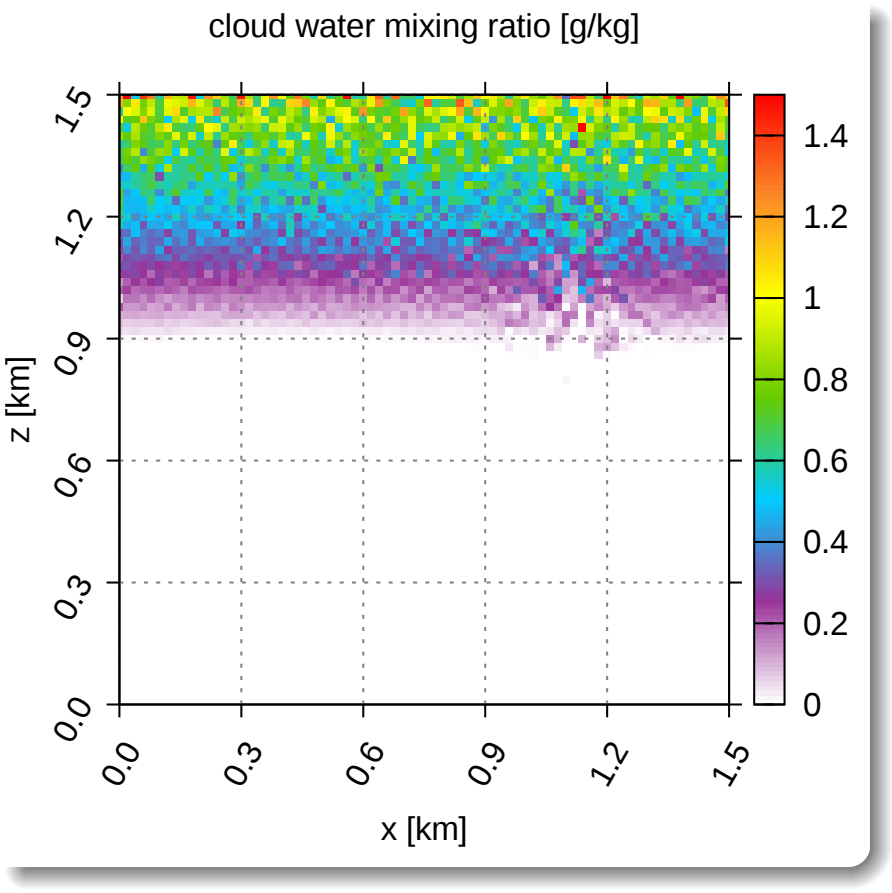
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxxxxxxxxxxxxxxxx

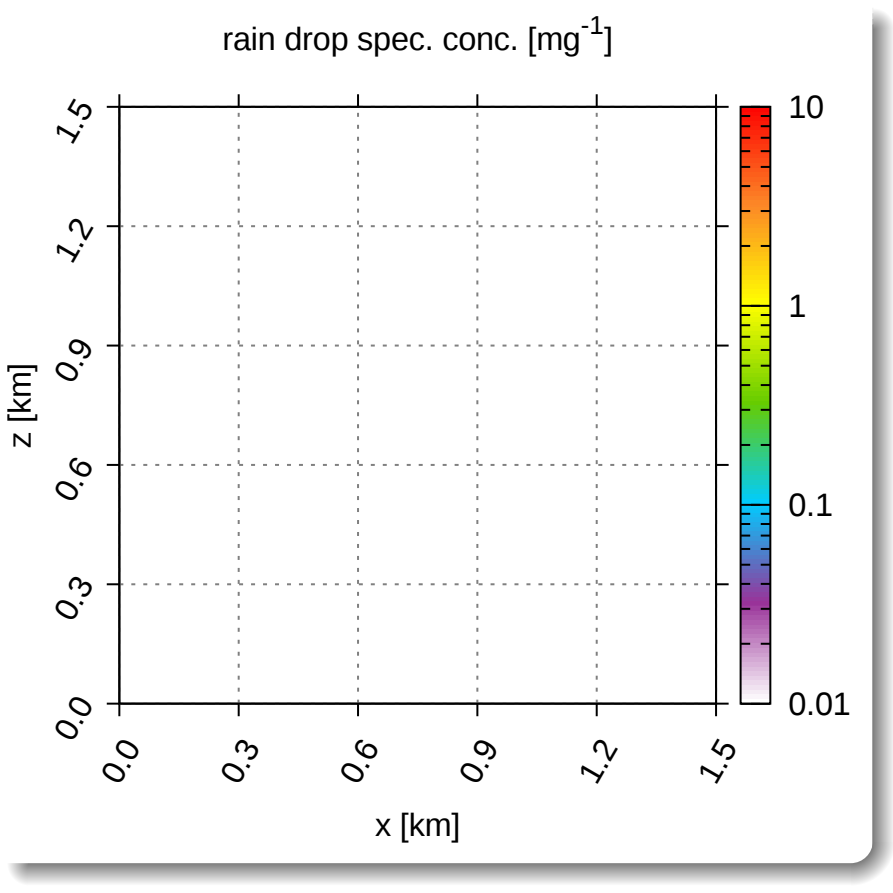




libcloudph++: example 2D prescribed-flow simulation

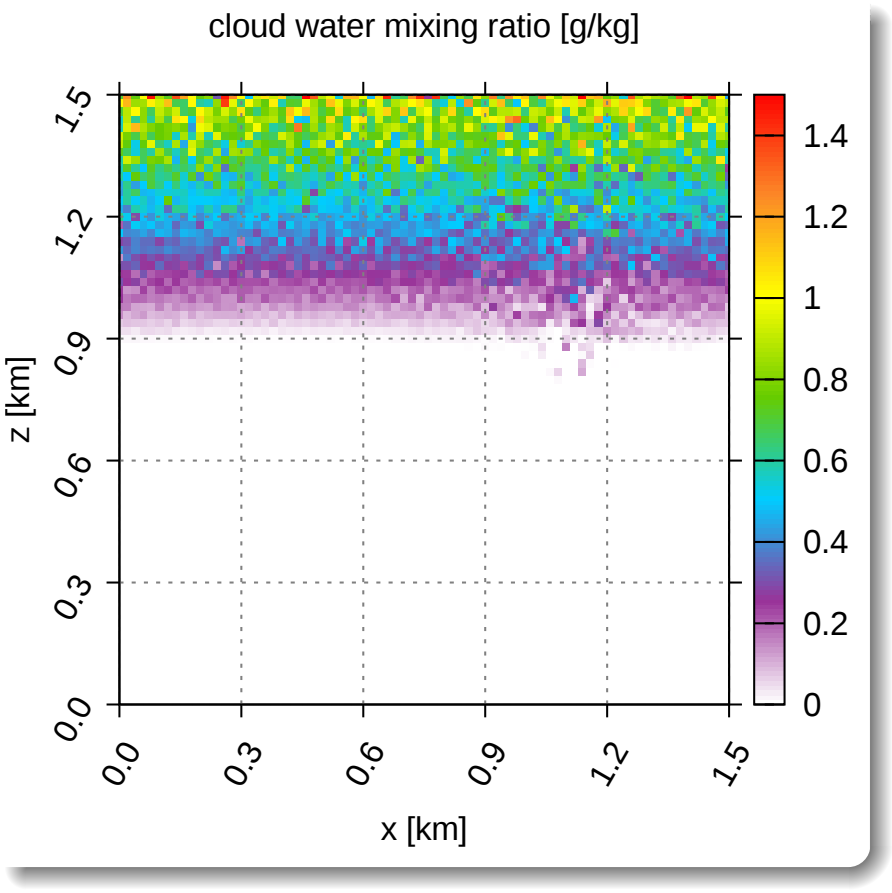
xxxxxxxxxxxxxxxxxxxxxxxxxxx

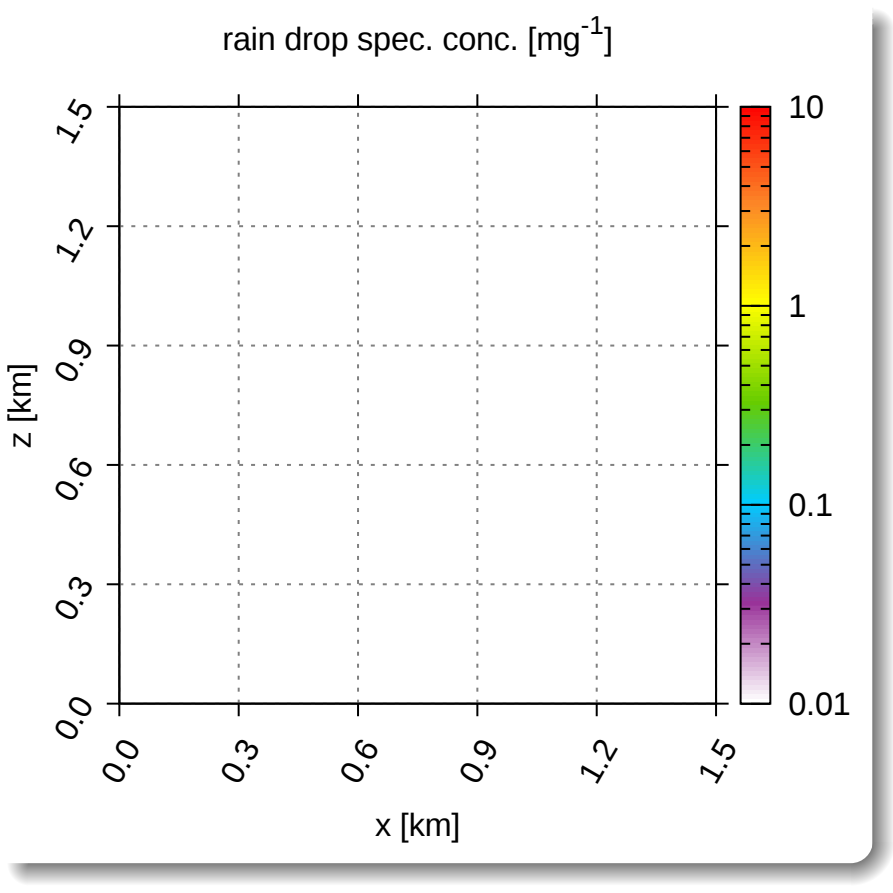




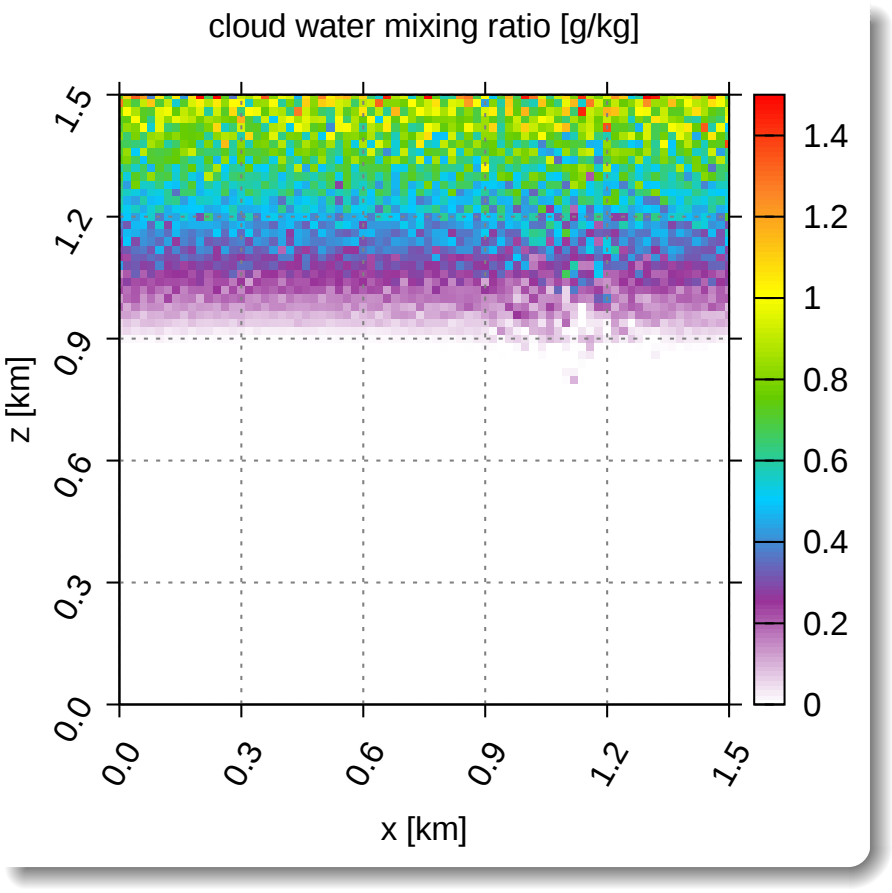
libcloudph++: example 2D prescribed-flow simulation

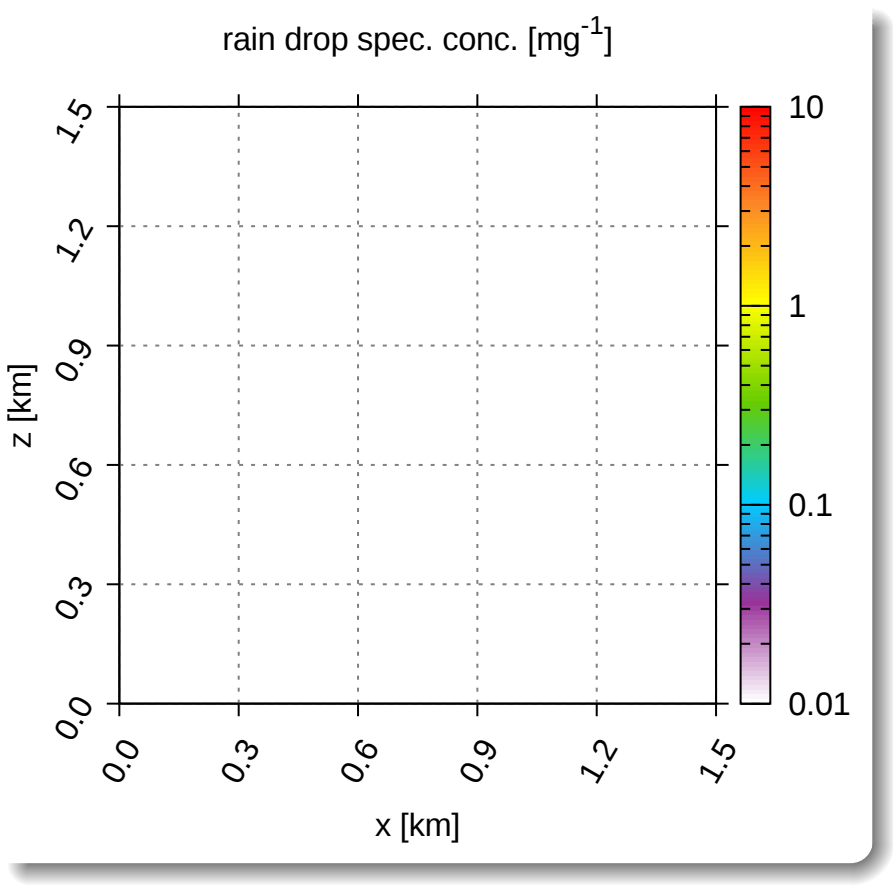
xxxxxxxxxxxxxxxxxxxxxxxxxxxx





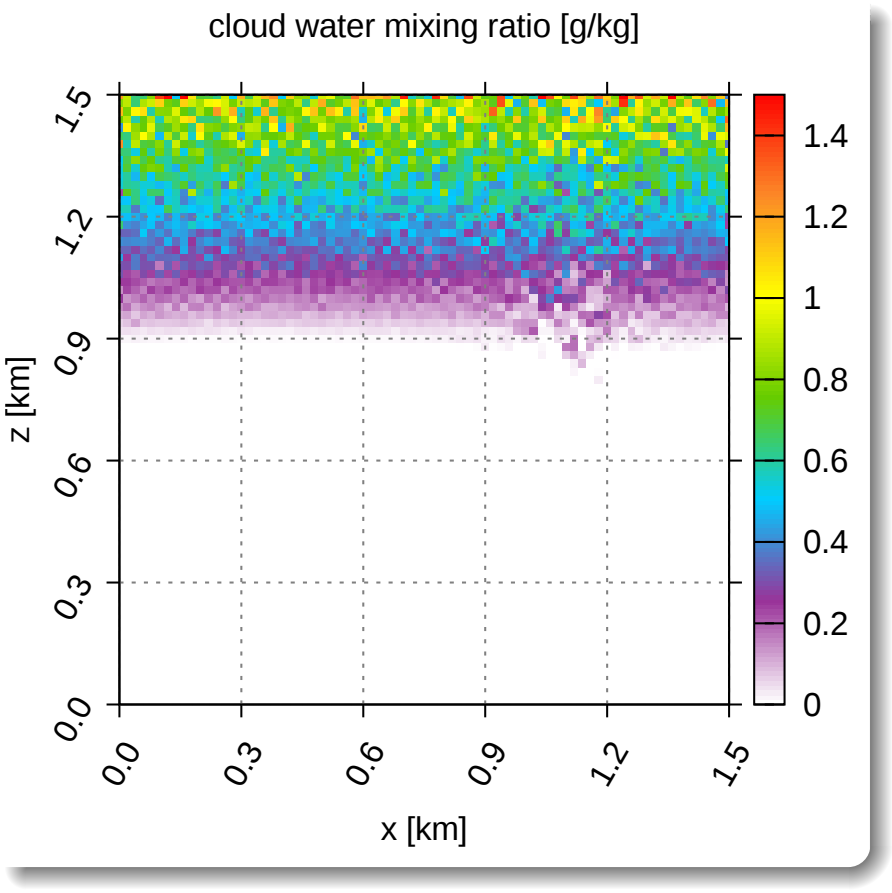
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxxxxxxxxxxxxxxxxxxx

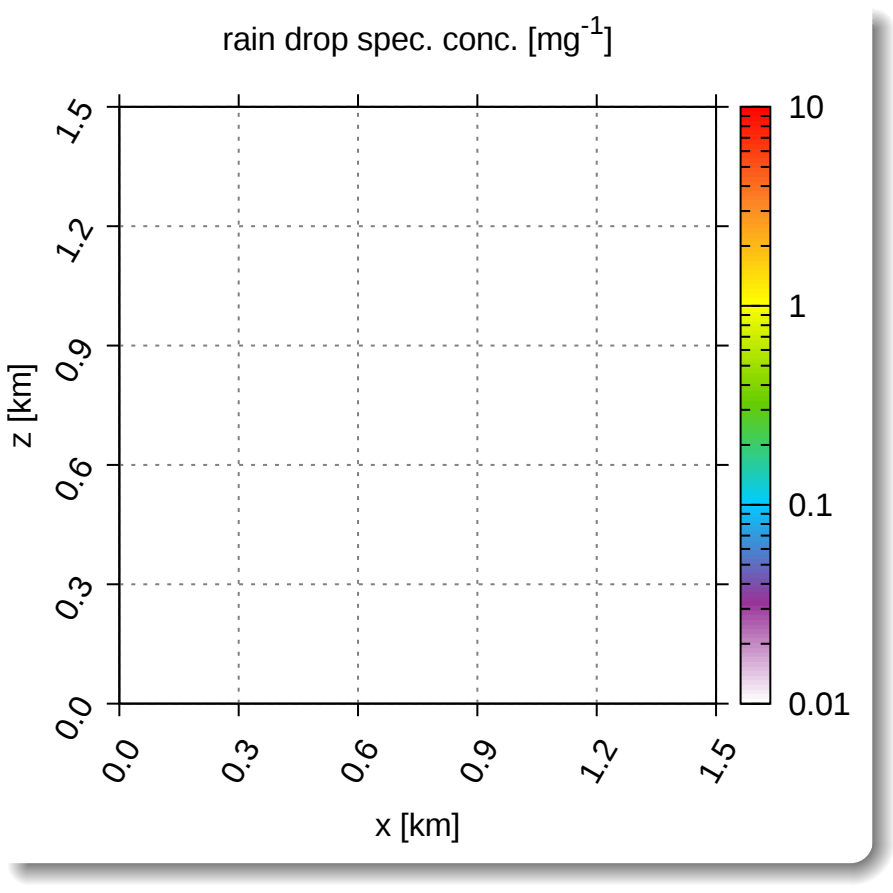




libcloudph++: example 2D prescribed-flow simulation

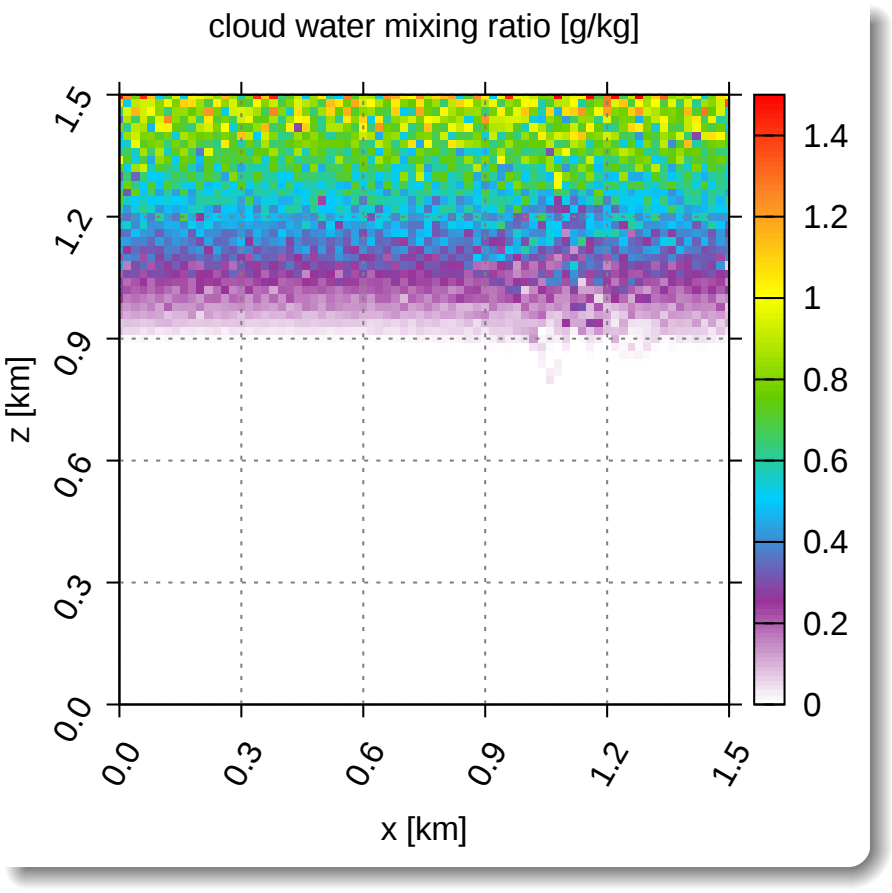
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

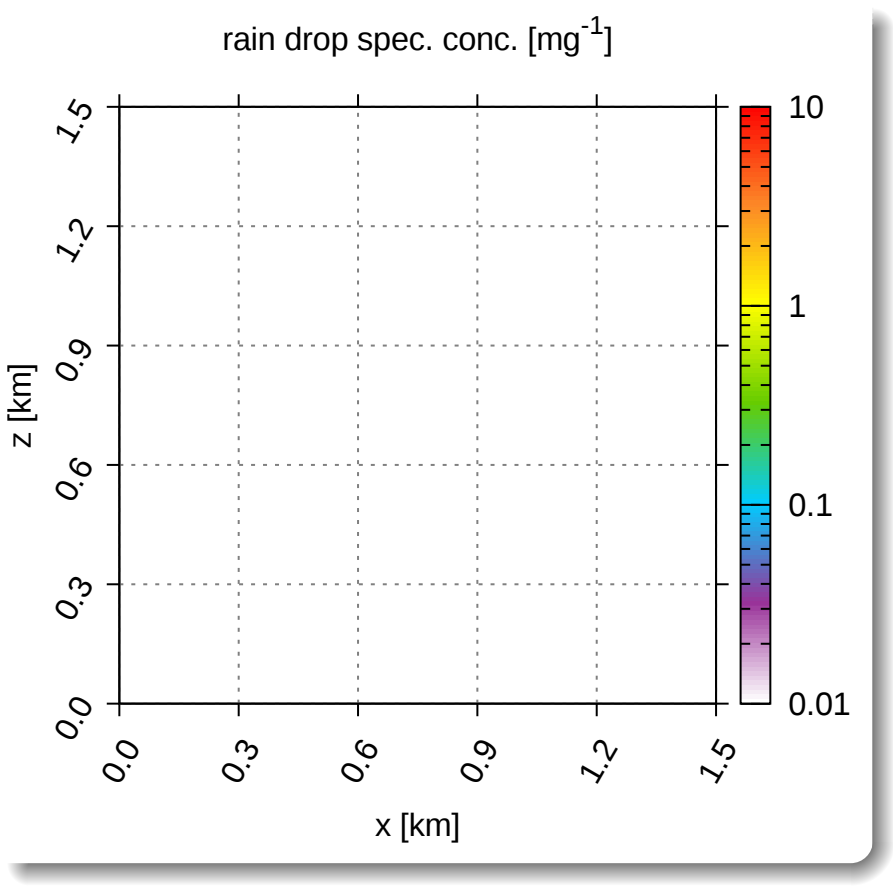




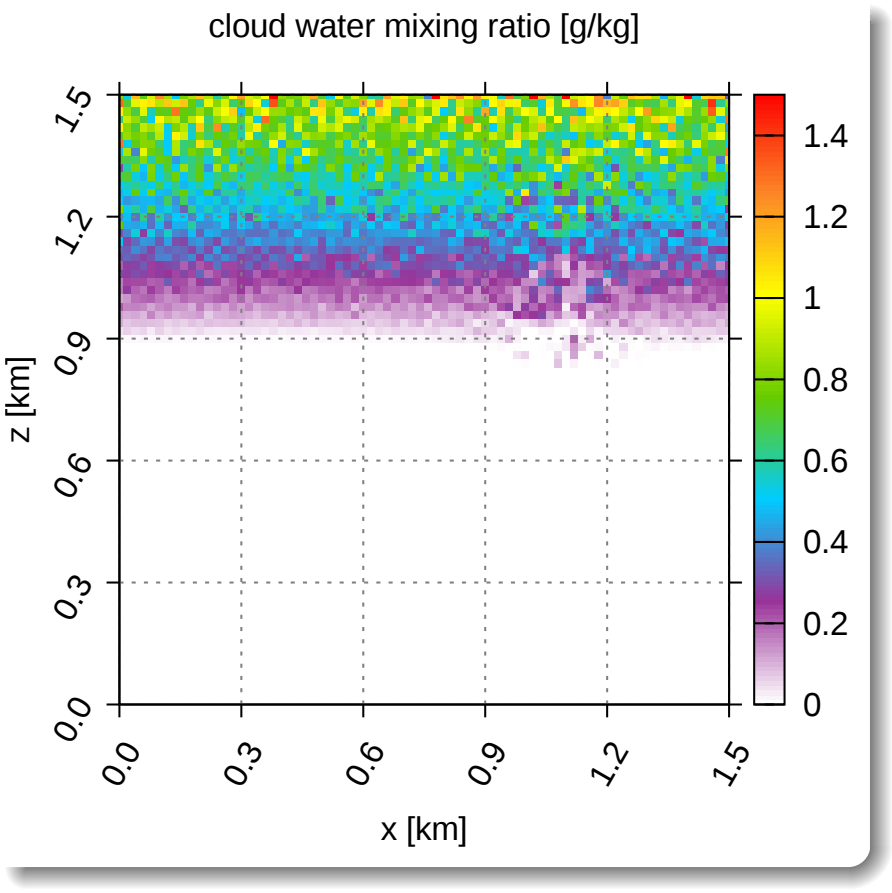
libcloudph++: example 2D prescribed-flow simulation

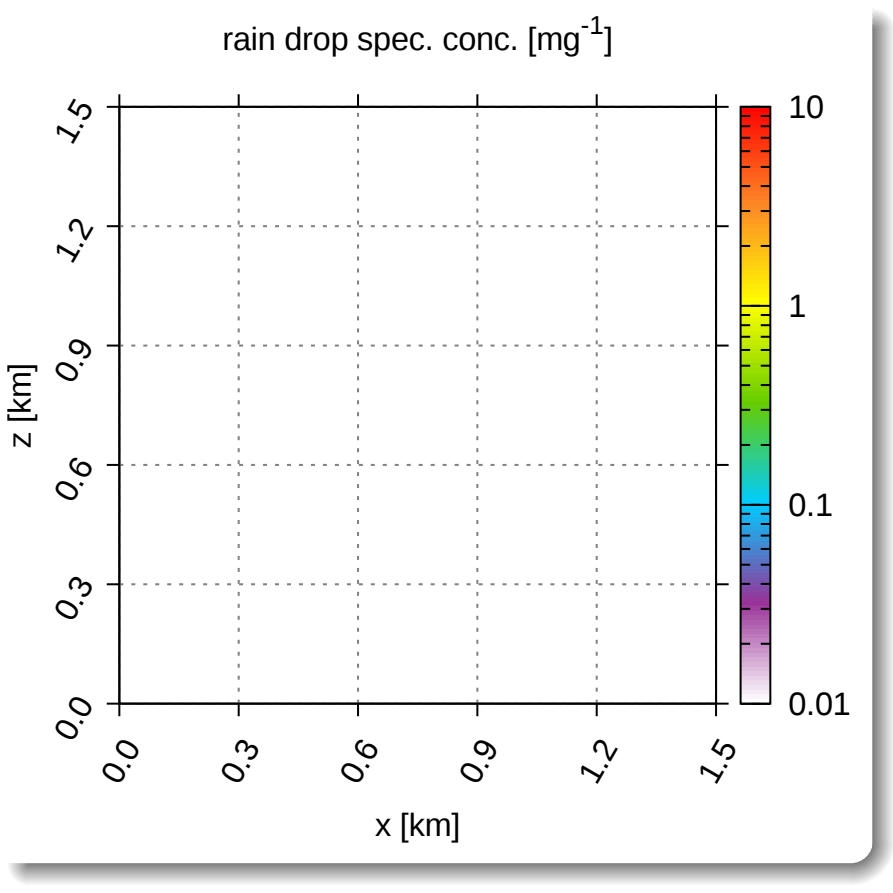
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx



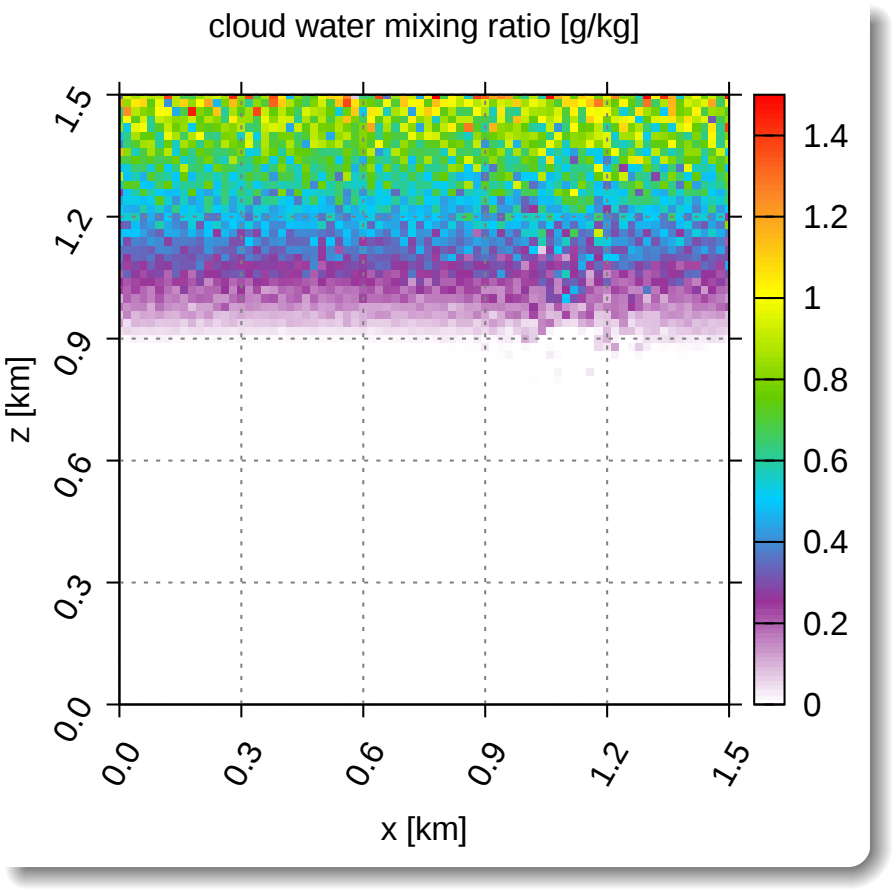


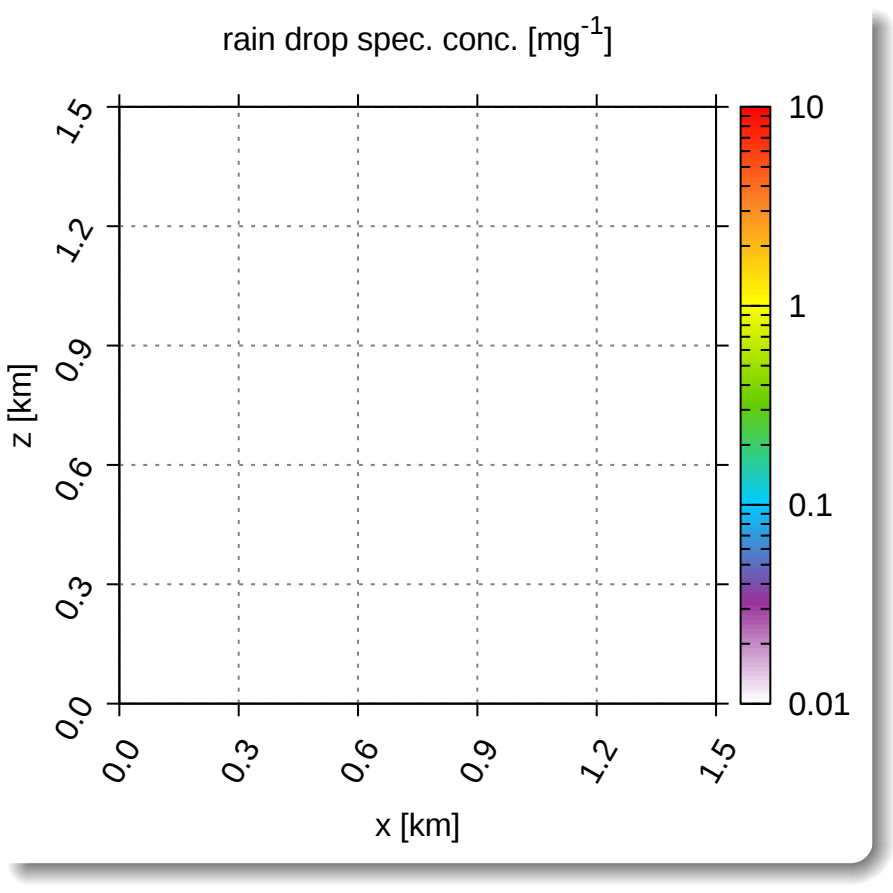
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx





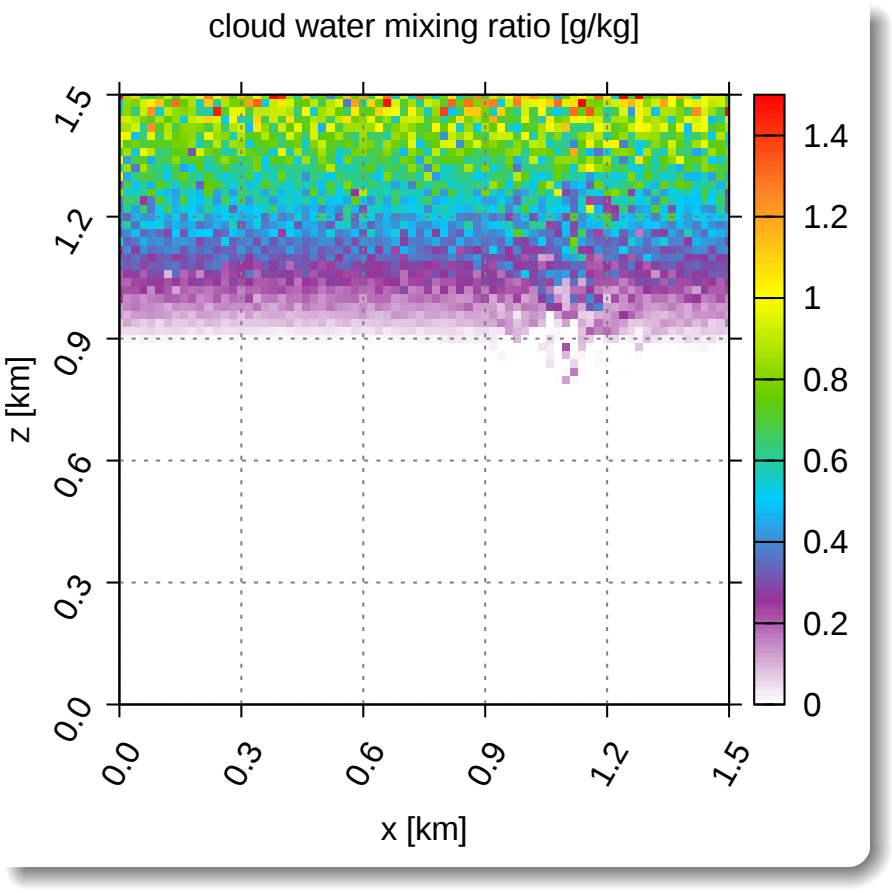
libcloudph++: example 2D prescribed-flow simulation

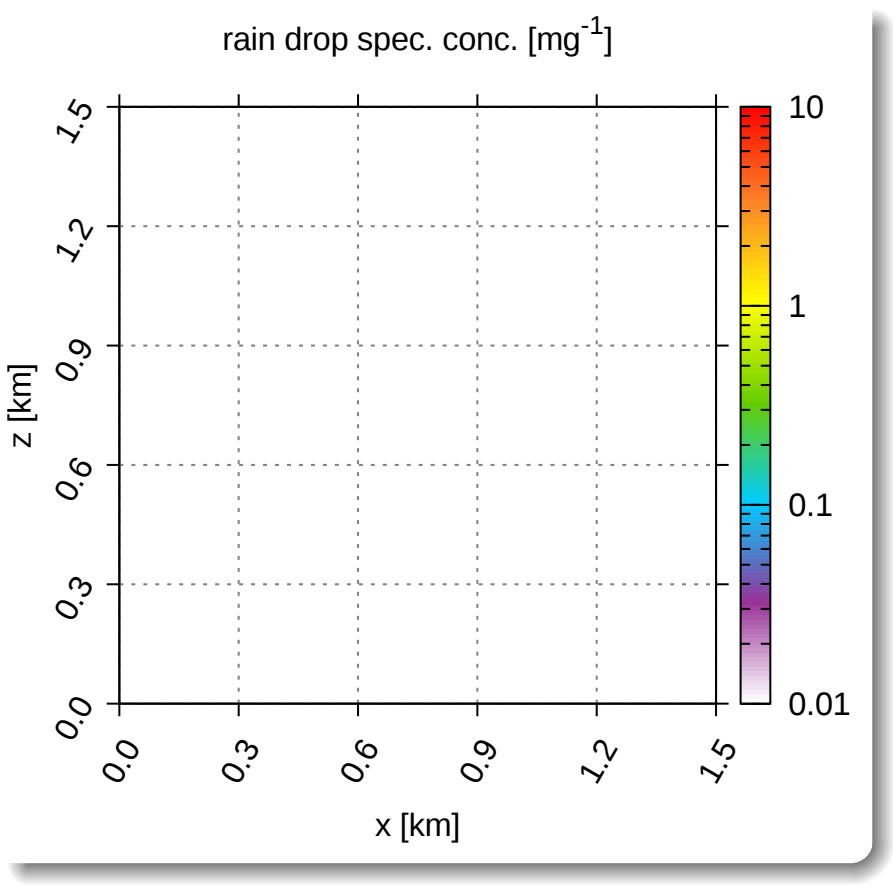




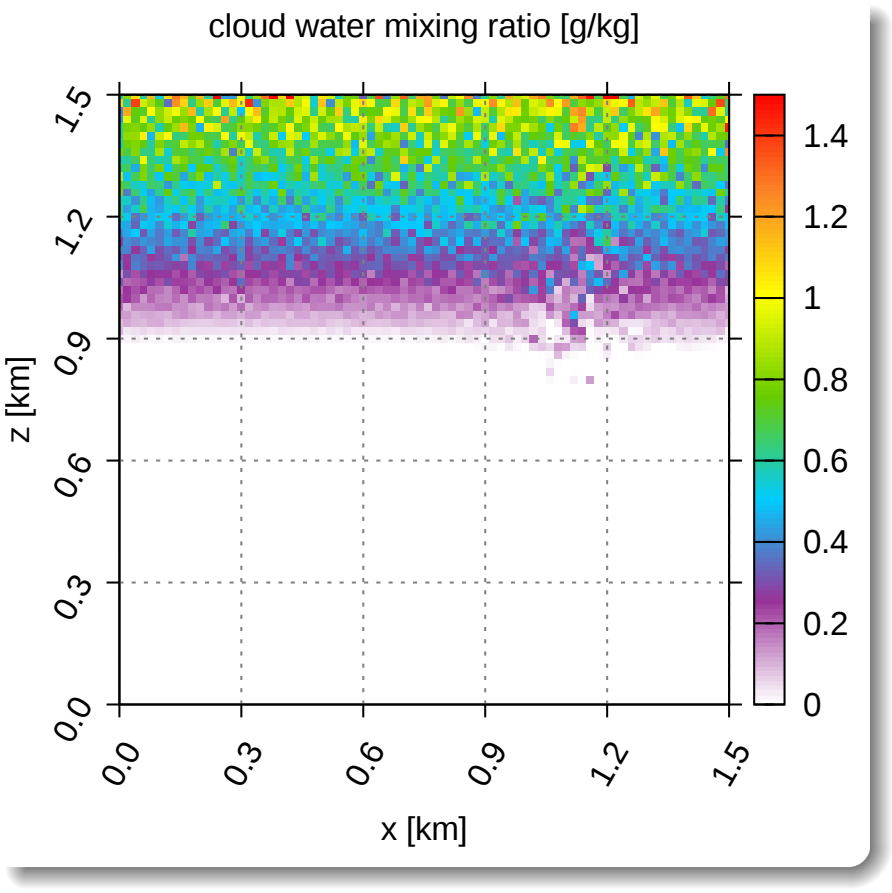
libcloudph++: example 2D prescribed-flow simulation

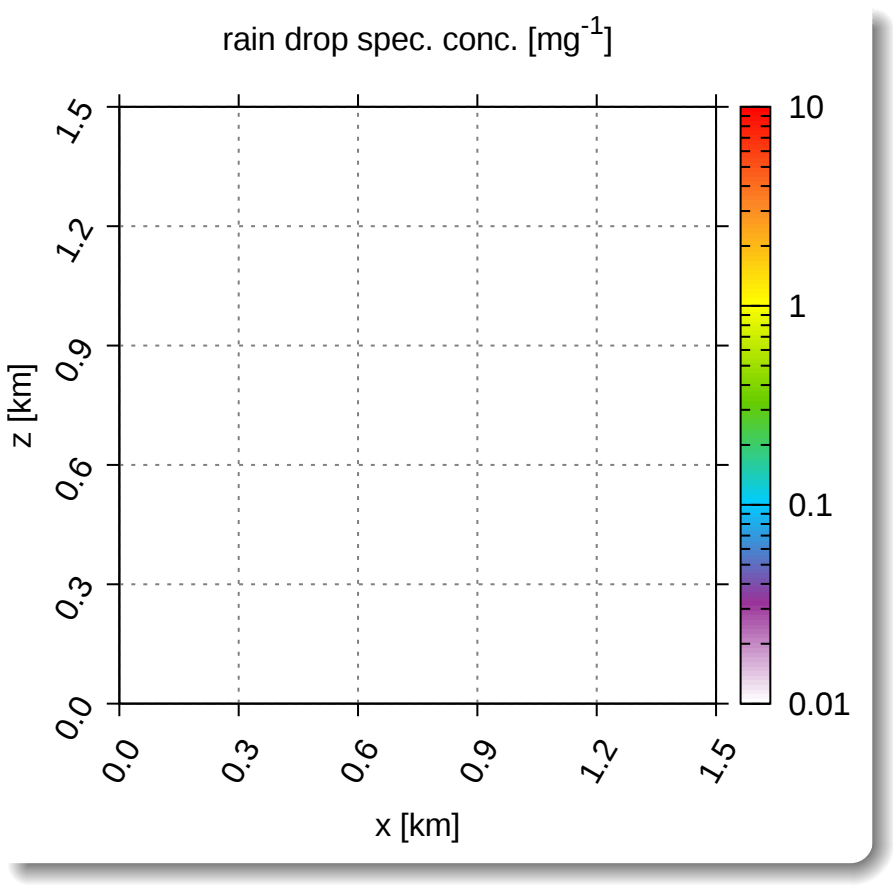
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx



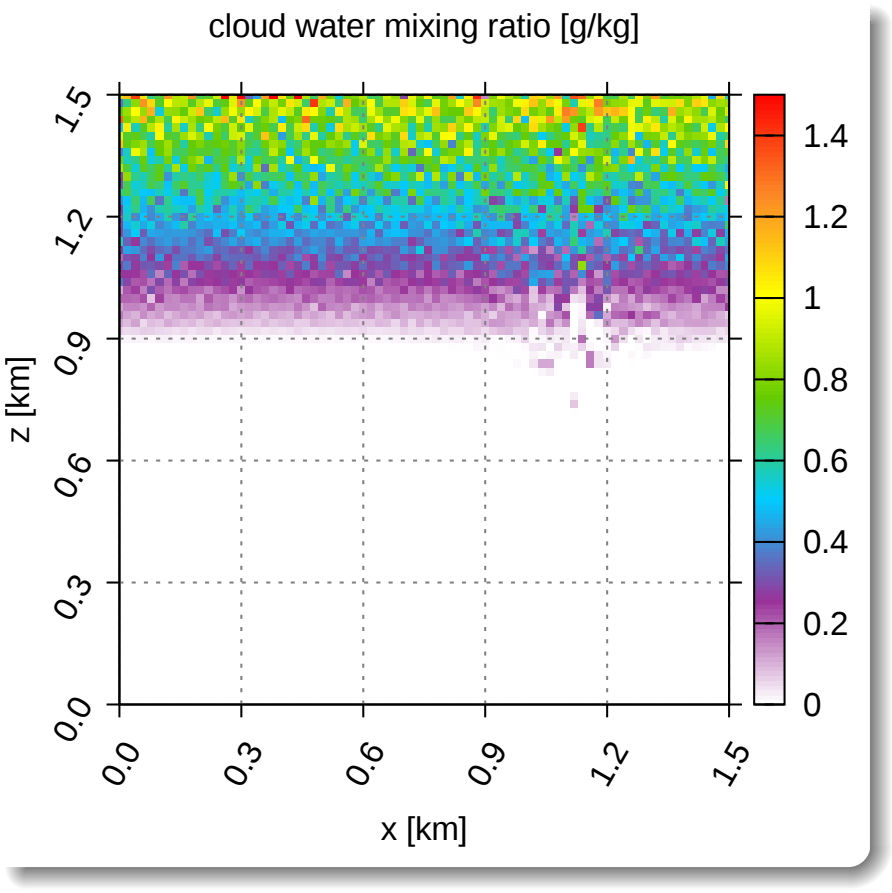


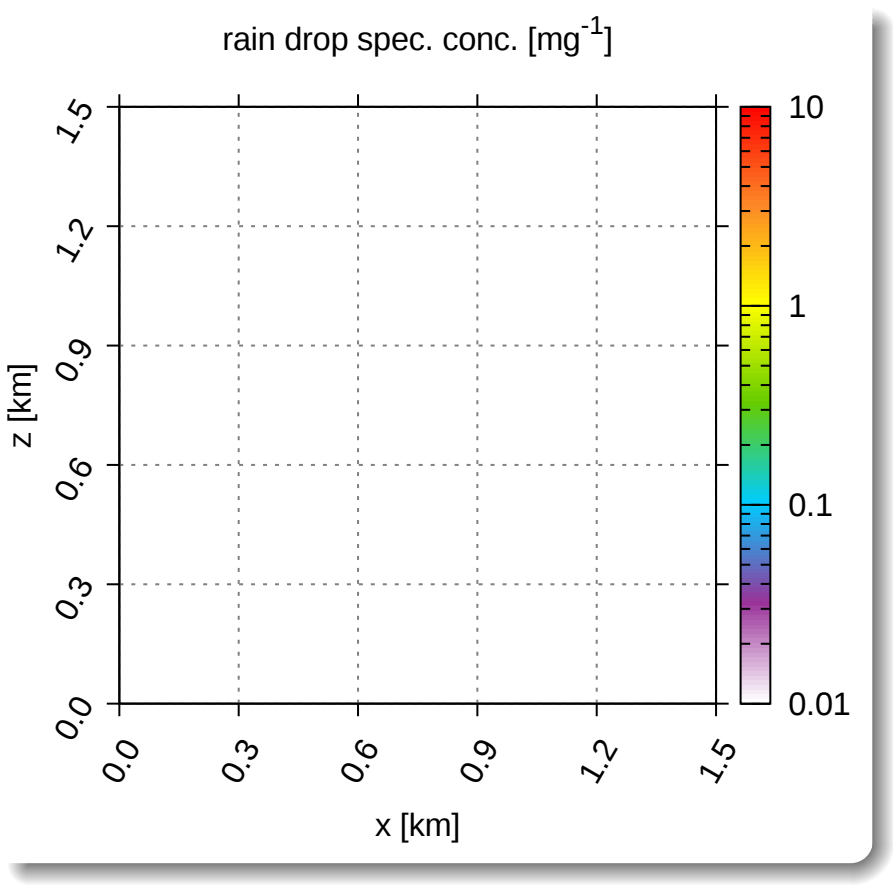
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx



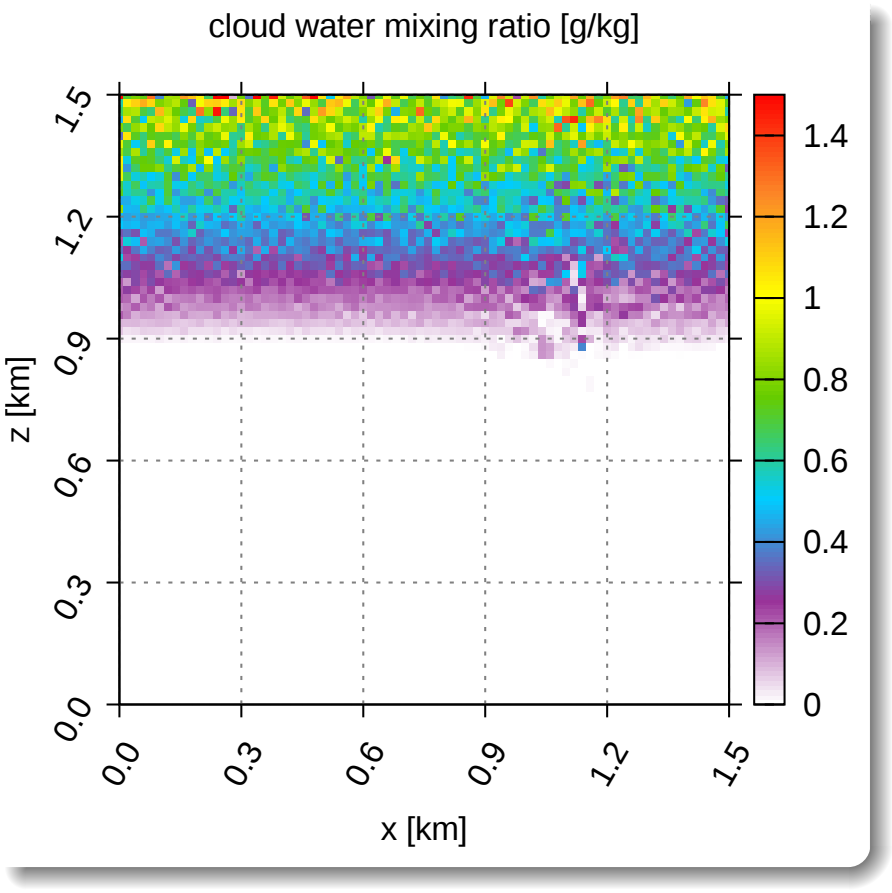


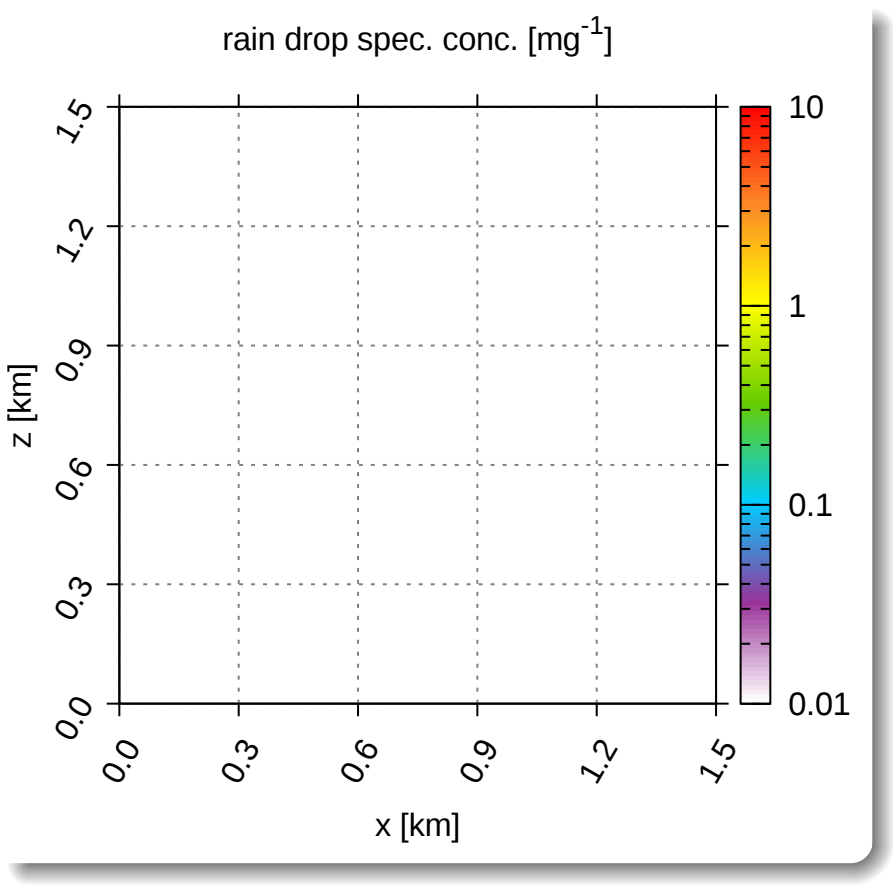
libcloudph++: example 2D prescribed-flow simulation



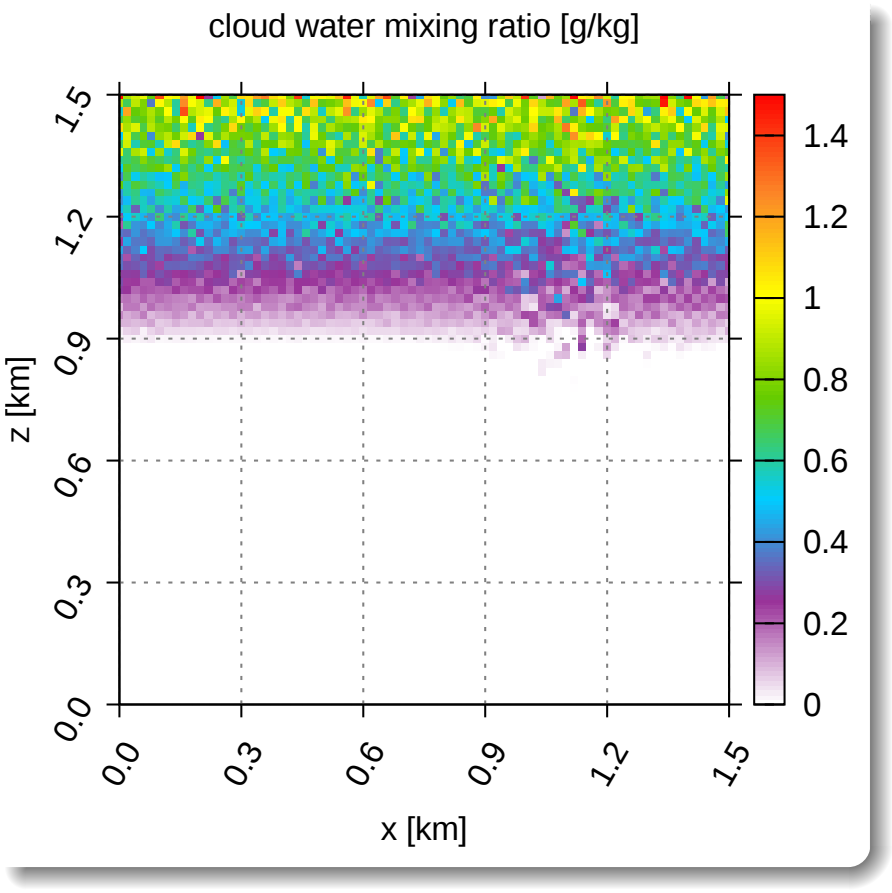


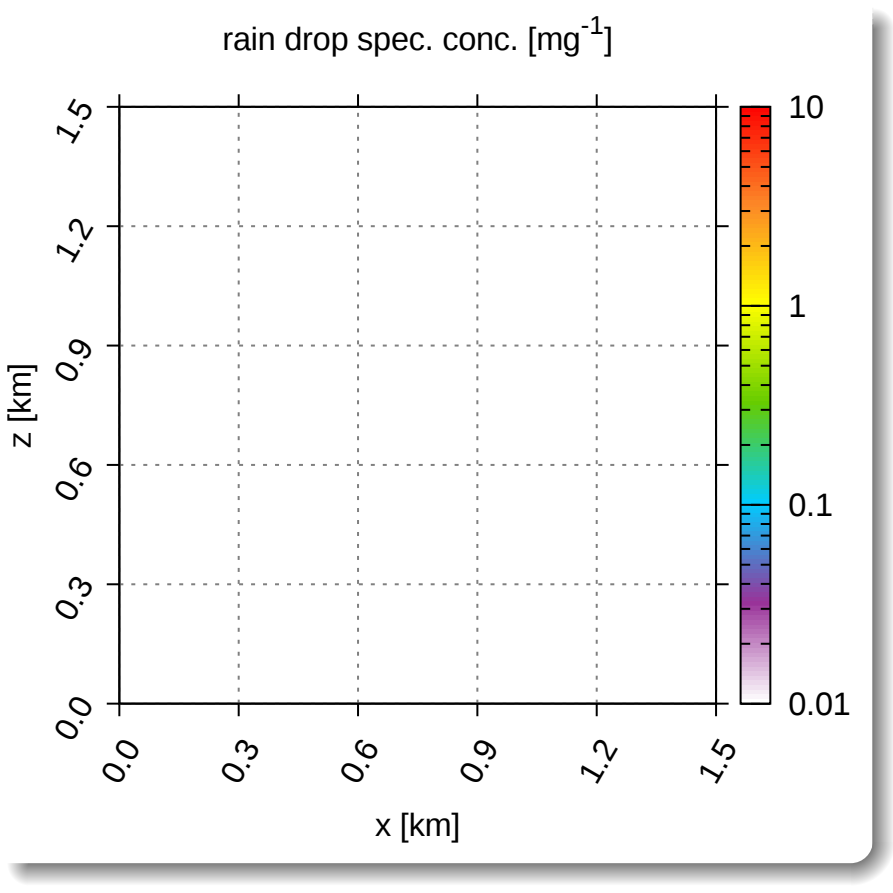
libcloudph++: example 2D prescribed-flow simulation



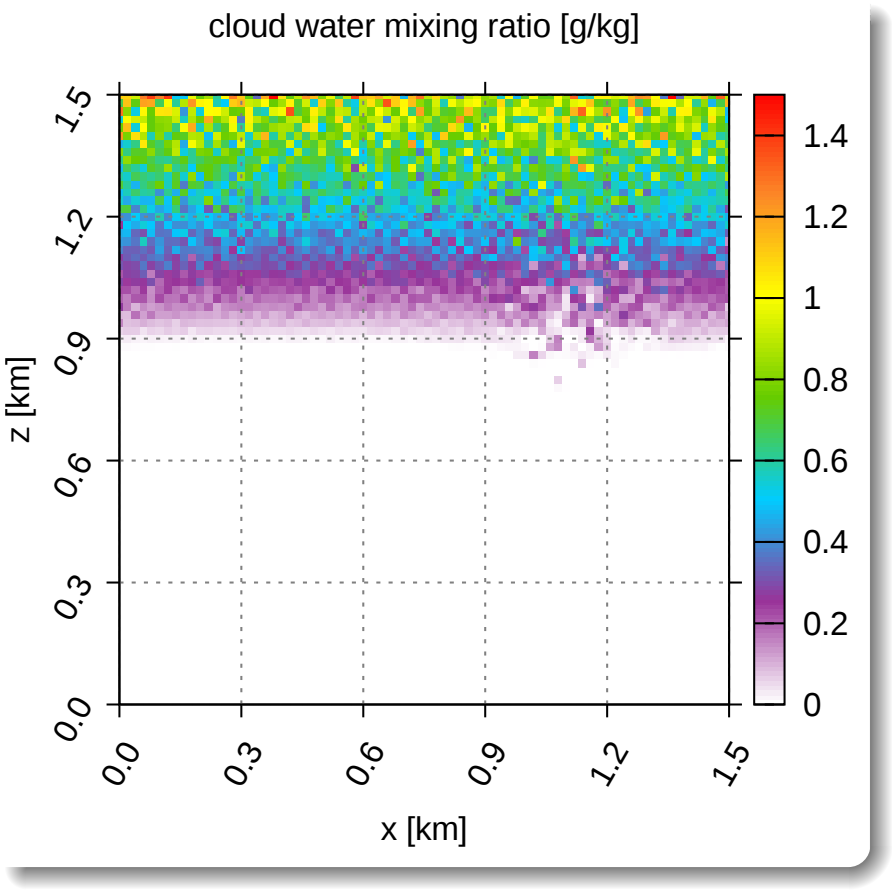


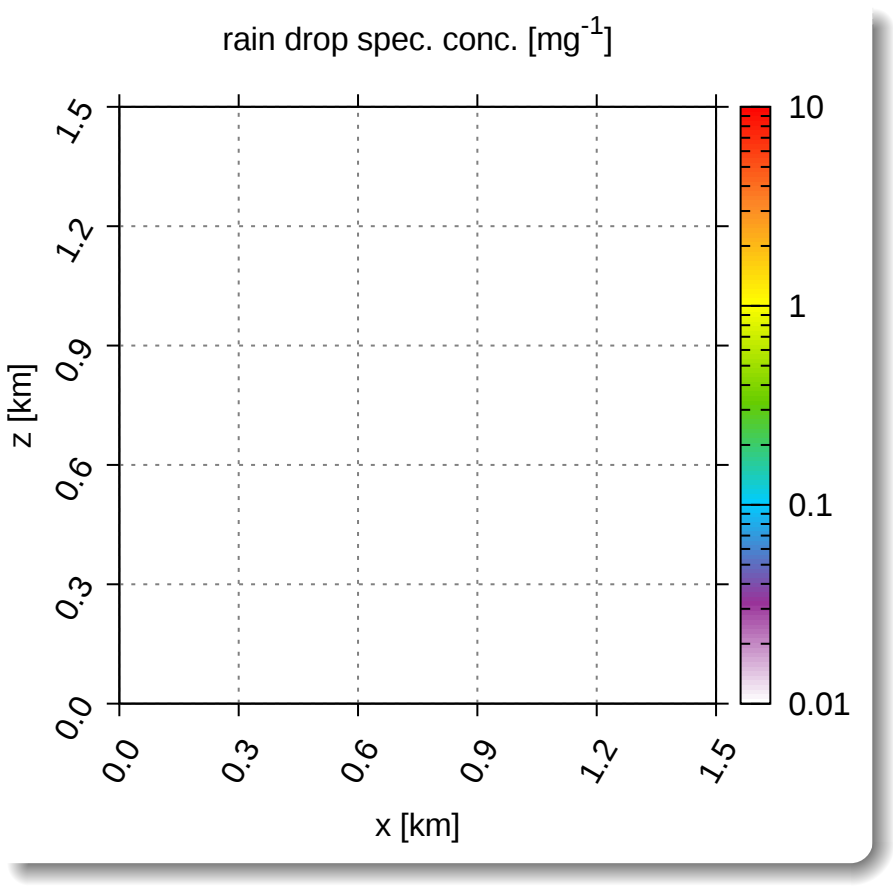
libcloudph++: example 2D prescribed-flow simulation
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx





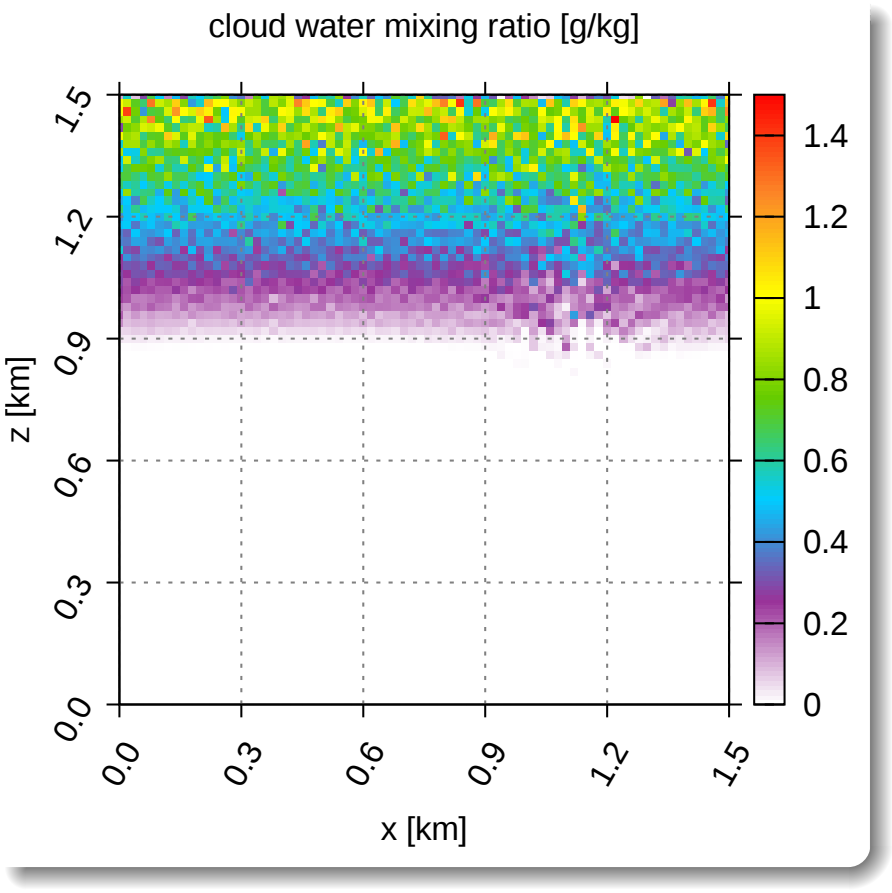
libcloudph++: example 2D prescribed-flow simulation

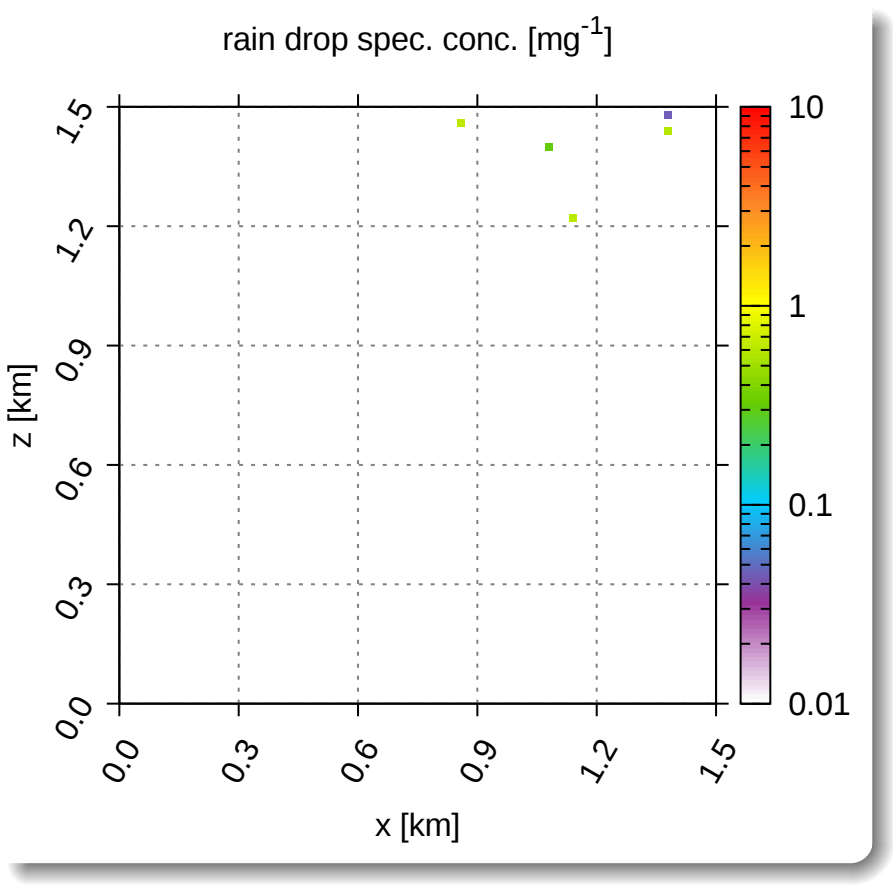




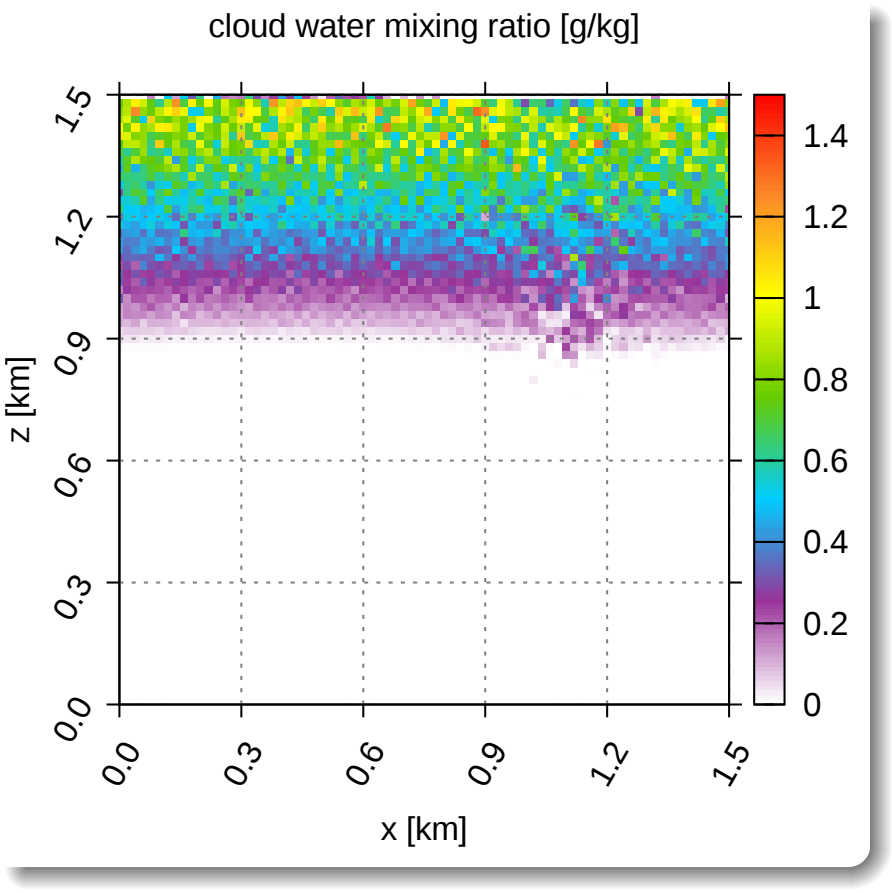
libcloudph++: example 2D prescribed-flow simulation

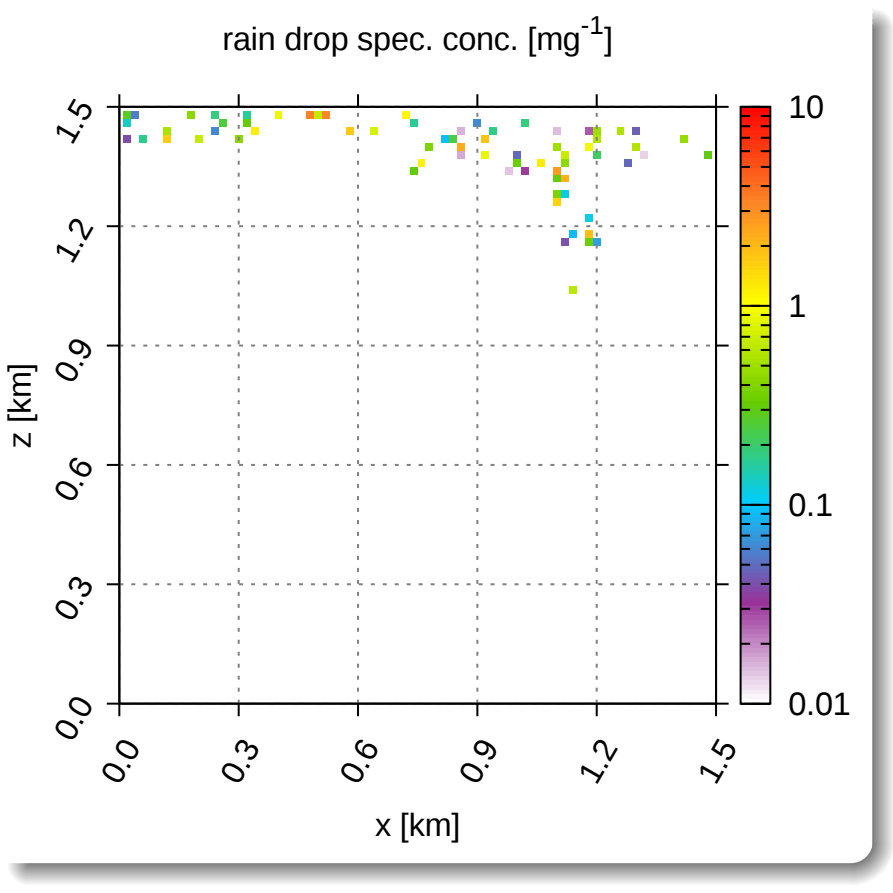
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxo



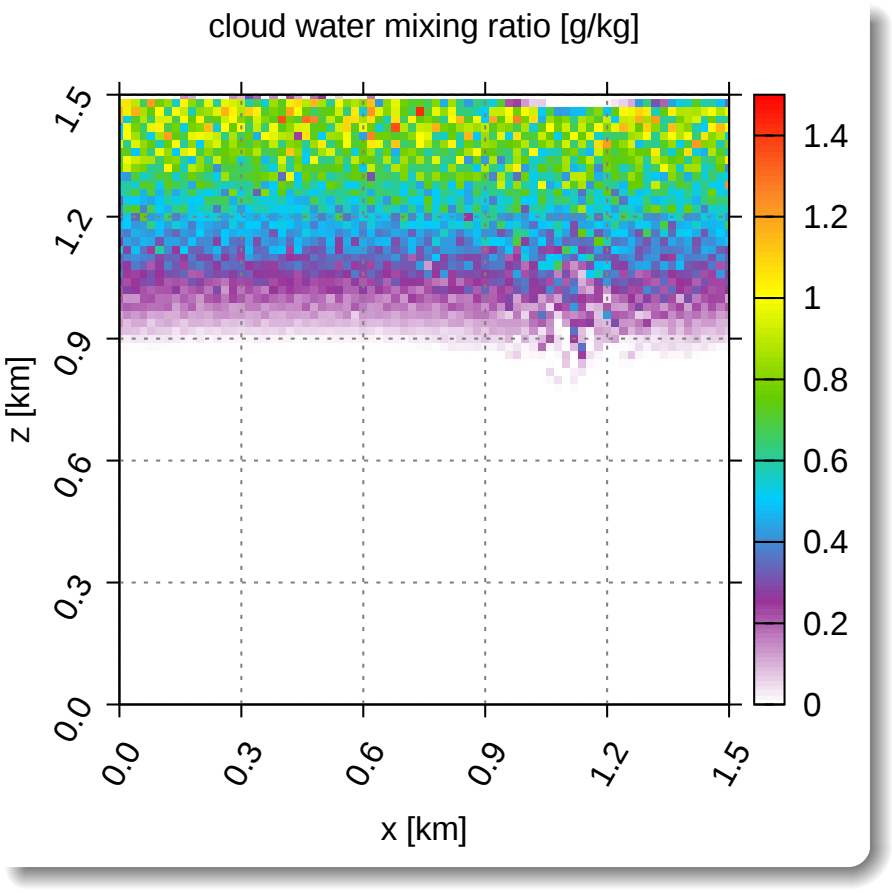


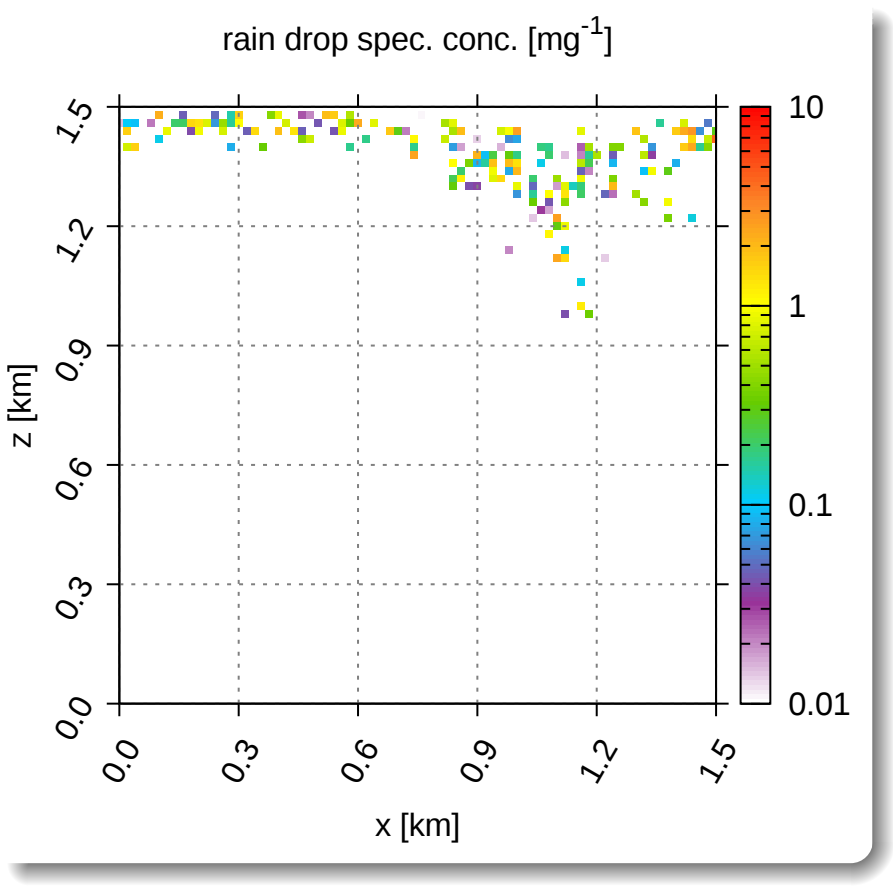
libcloudph++: example 2D prescribed-flow simulation



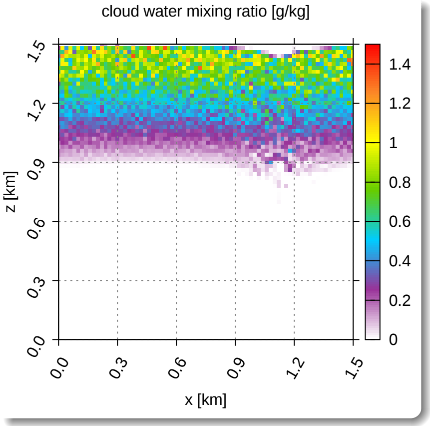


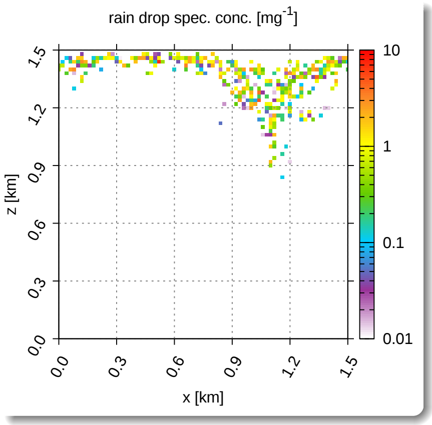
libcloudph++: example 2D prescribed-flow simulation



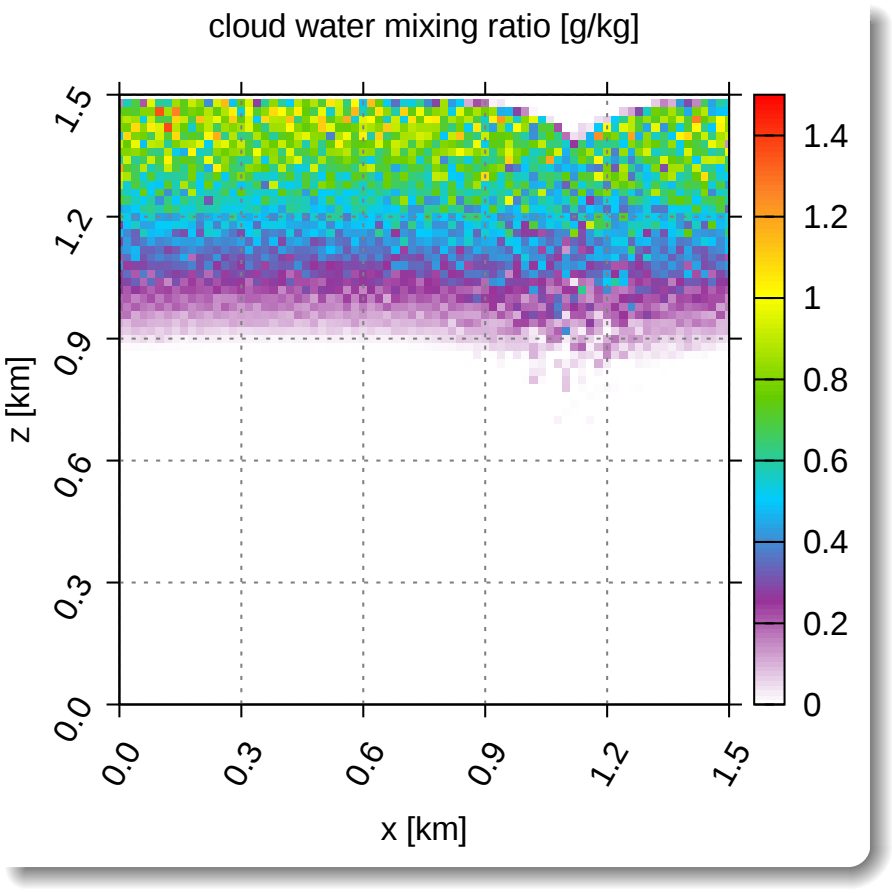


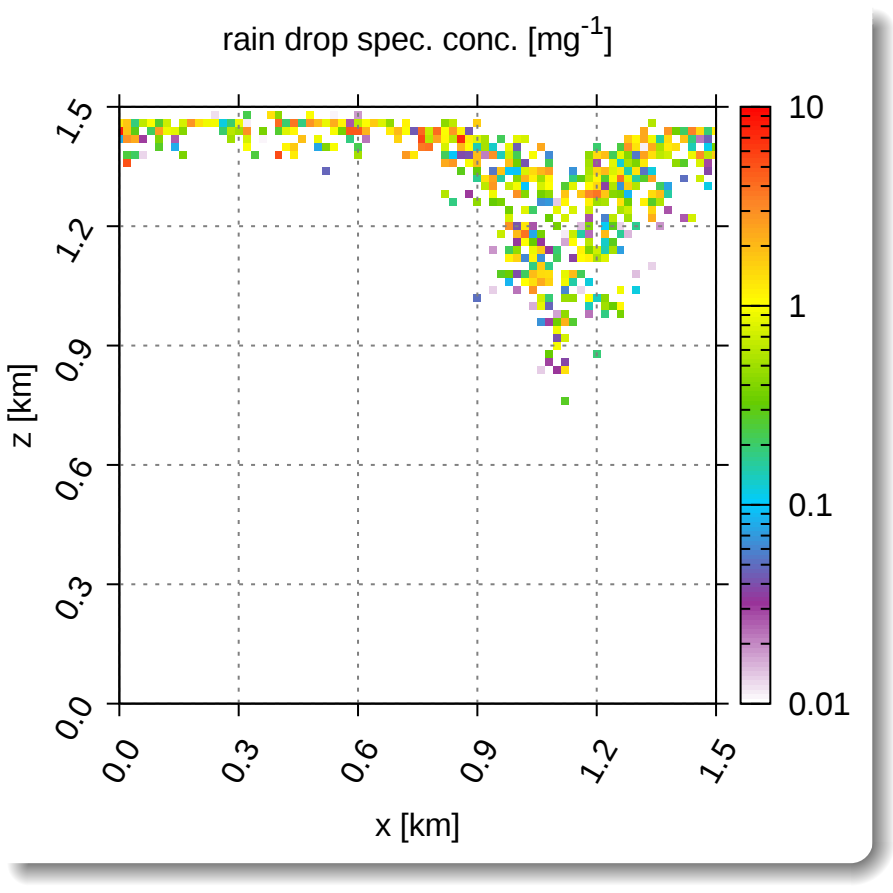
libcloudph++: example 2D prescribed-flow simulation



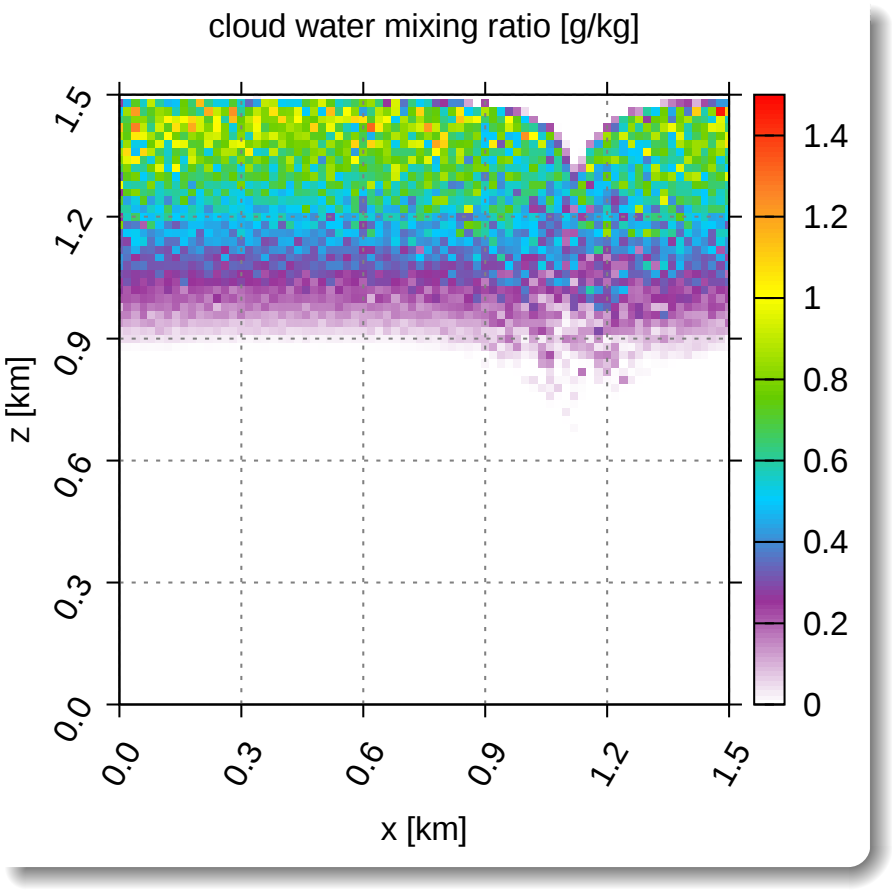


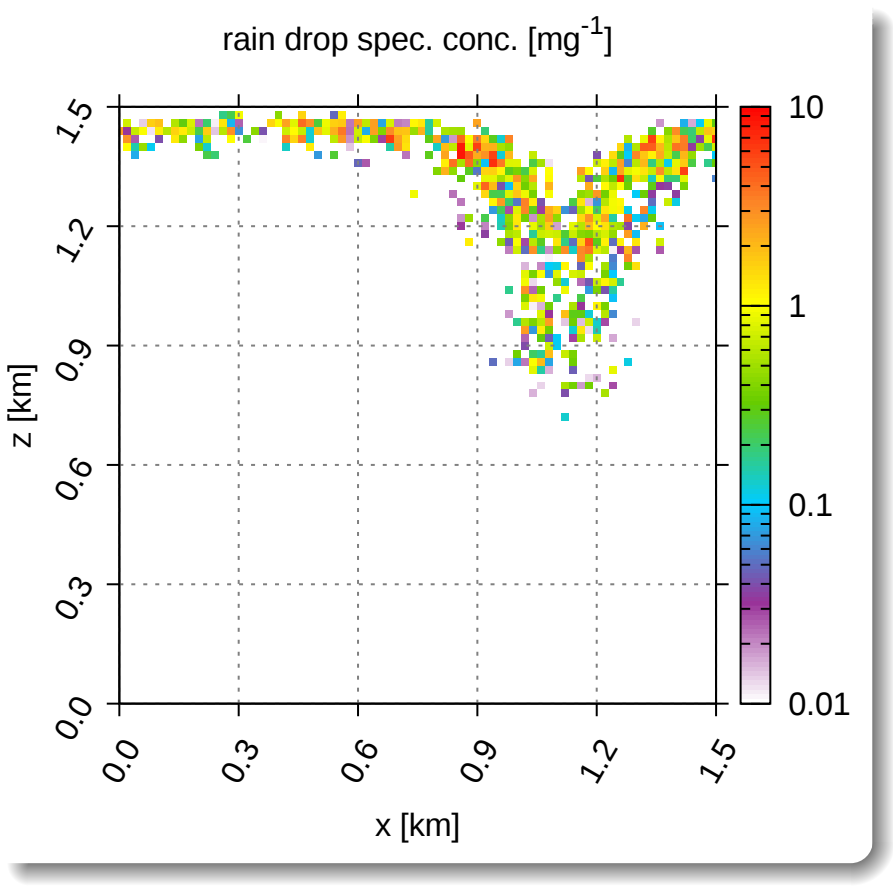
libcloudph++: example 2D prescribed-flow simulation



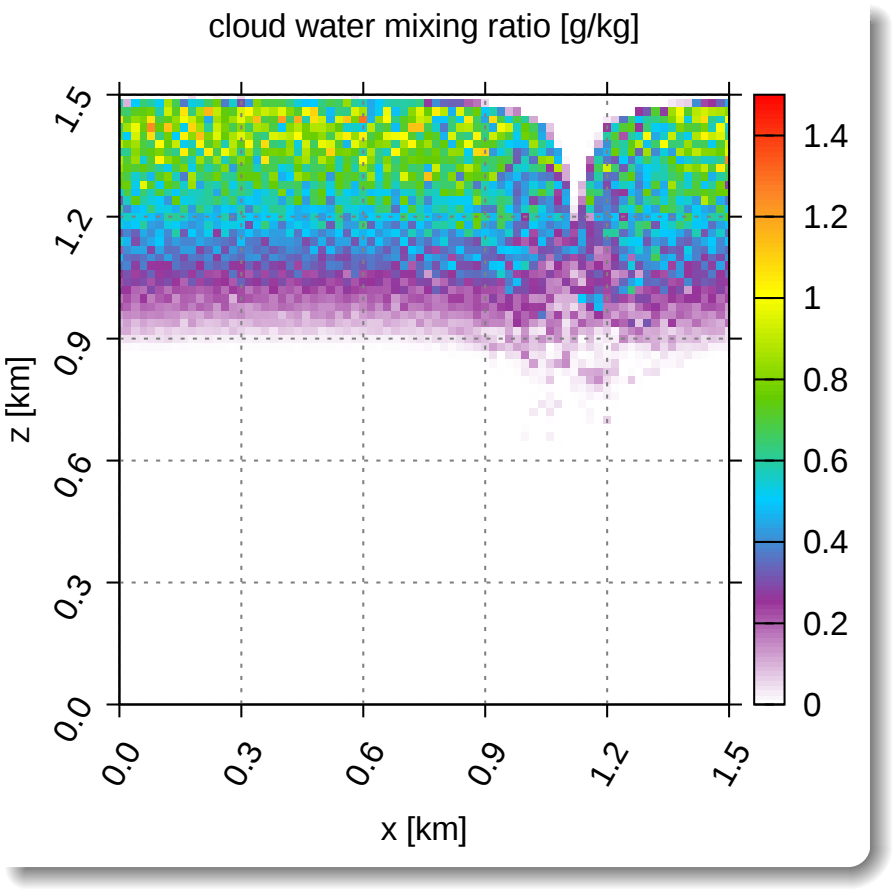


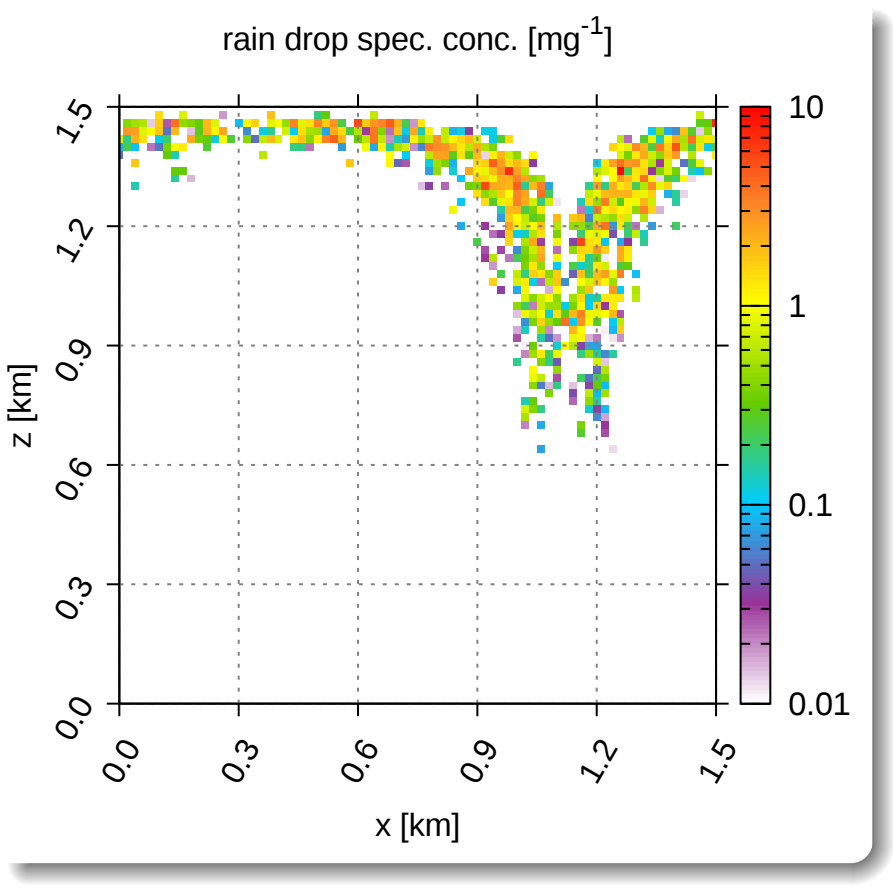
libcloudph++: example 2D prescribed-flow simulation



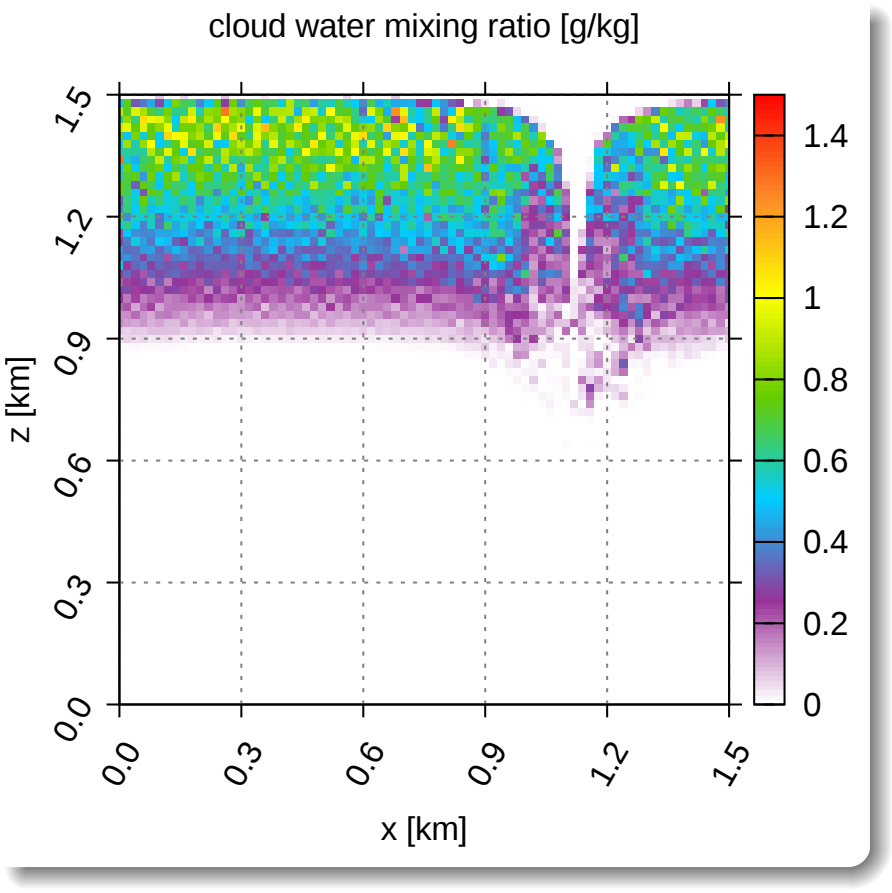


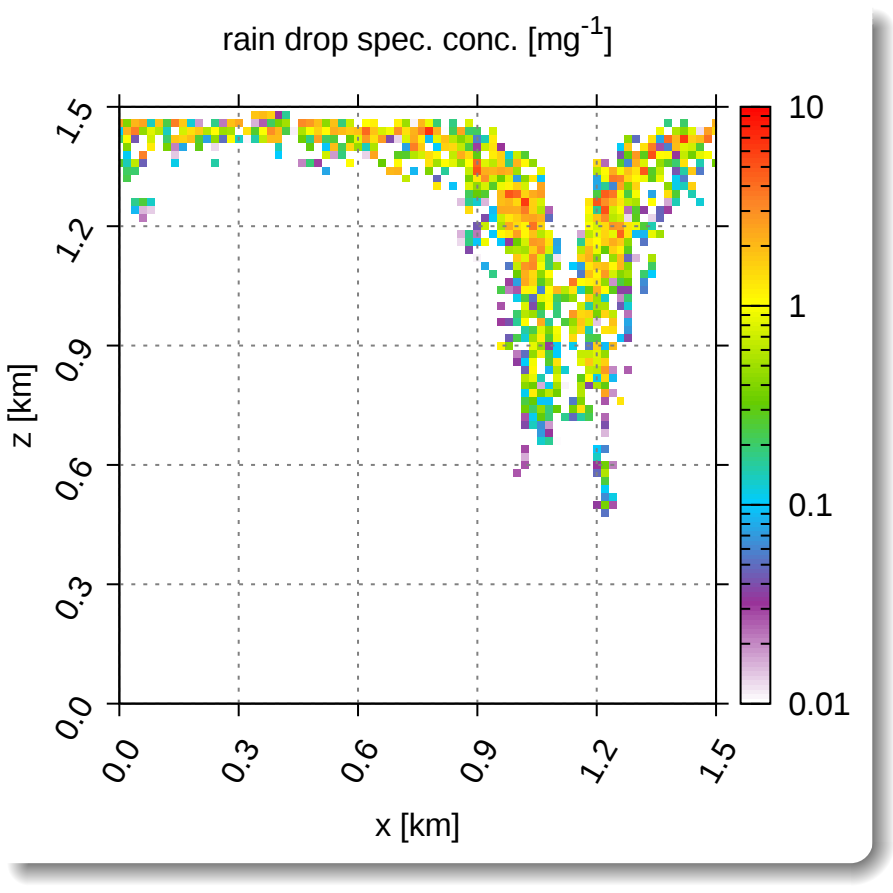
libcloudph++: example 2D prescribed-flow simulation



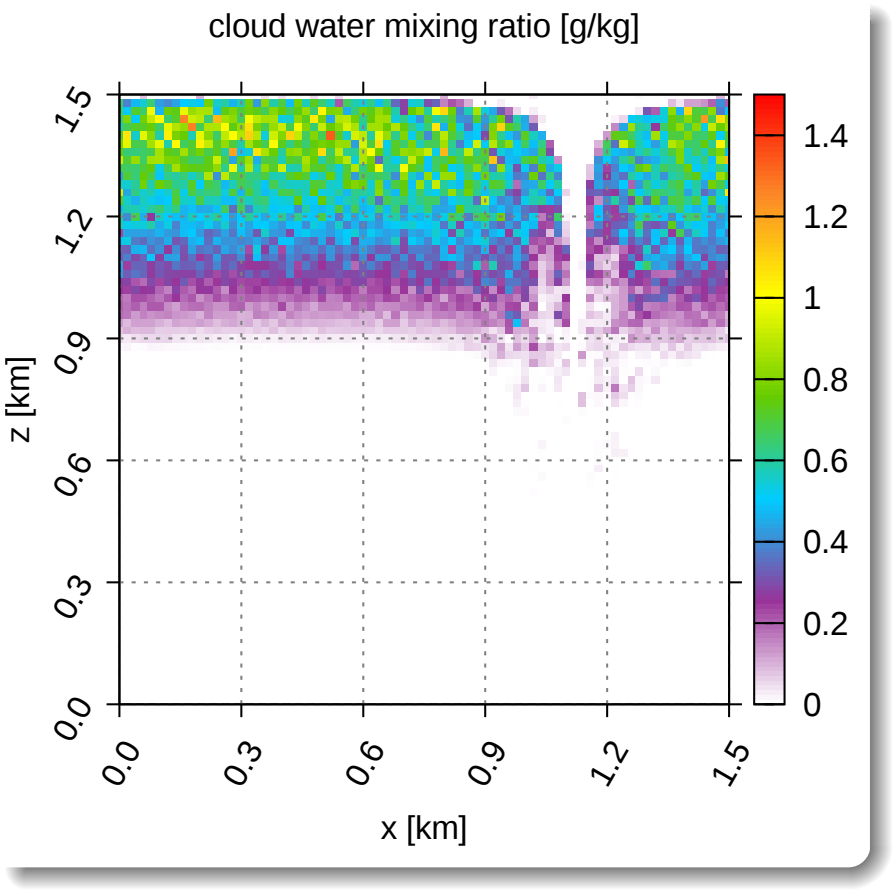


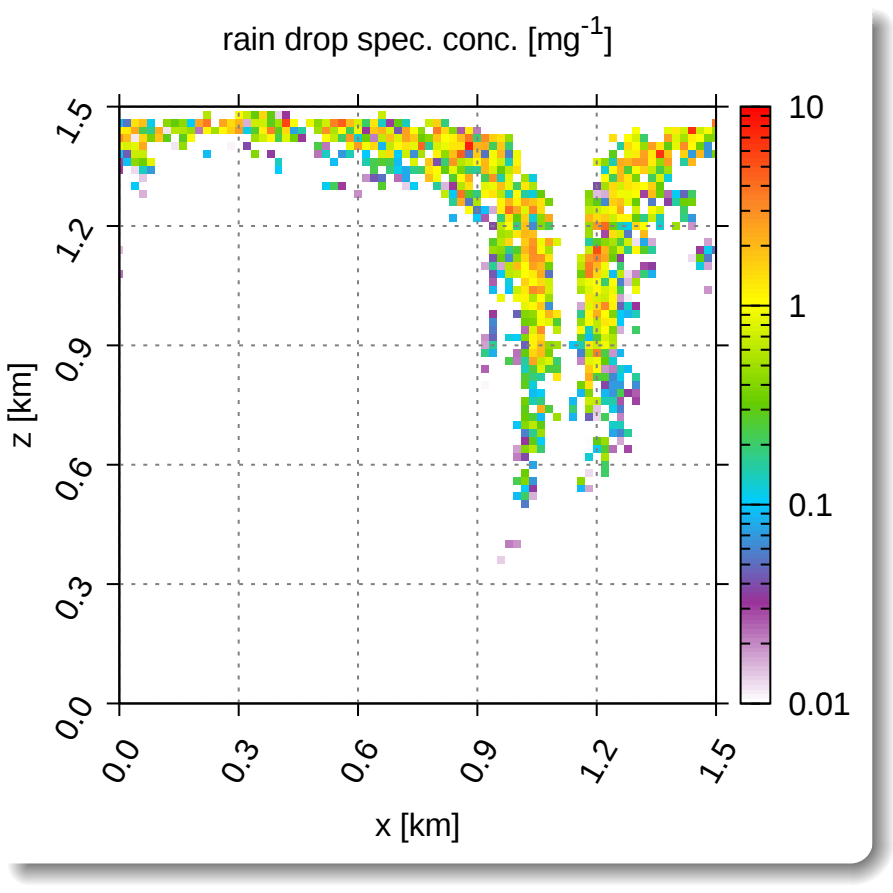
libcloudph++: example 2D prescribed-flow simulation





libcloudph++: example 2D prescribed-flow simulation





Geosci. Model Dev. Discuss., 7, 8275–8360, 2014
www.geosci-model-dev-discuss.net/7/8275/2014/
doi:10.5194/gmdd-7-8275-2014

Geoscientific Model Development
Discussions
Open Access

libcloudph++ 0.2: single-moment bulk, double-moment bulk, and particle-based warm-rain microphysics library in C++

S. Arabas[1], A. Jaruga[1], H. Pawlowska[1], and W. W. Grabowski[2,3]

[1]Institute of Geophysics, Faculty of Physics, University of Warsaw, Warsaw, Poland
[2]National Center for Atmospheric Research (NCAR), Boulder, Colorado, USA
[3]Affiliate Professor at the University of Warsaw, Warsaw, Poland

Received: 18 August 2014 – Accepted: 24 September 2014 – Published: 26 November 2014

Correspondence to: S. Arabas (sarabas@igf.fuw.edu.pl)
and H. Pawlowska (hanna.pawlowska@igf.fuw.edu.pl)

Published by Copernicus Publications on behalf of the European Geosciences Union.

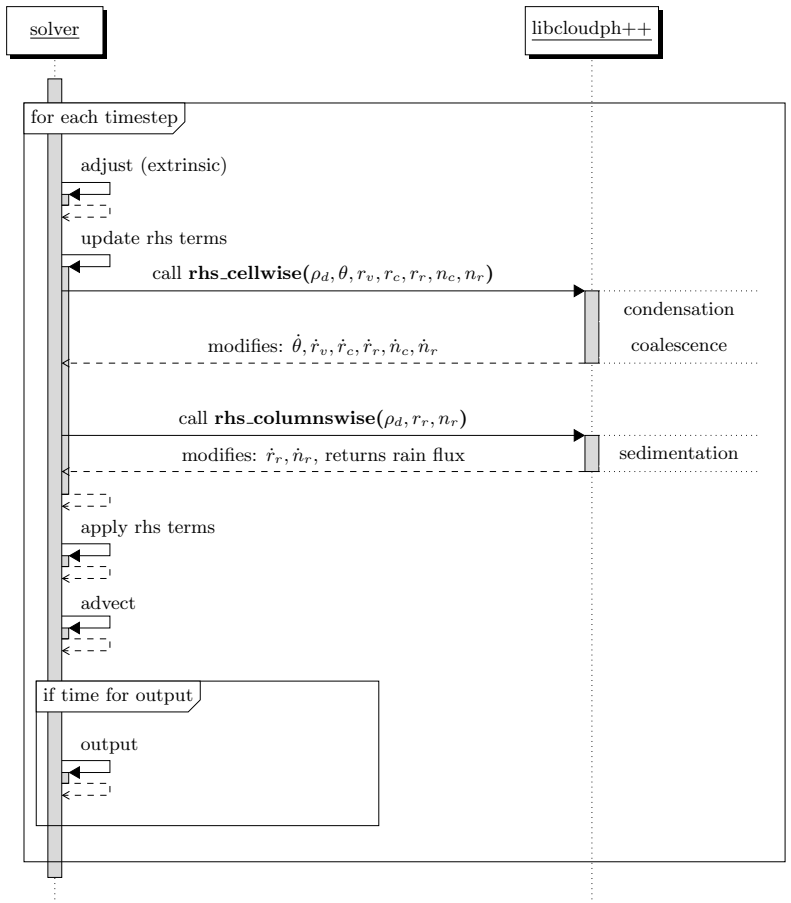
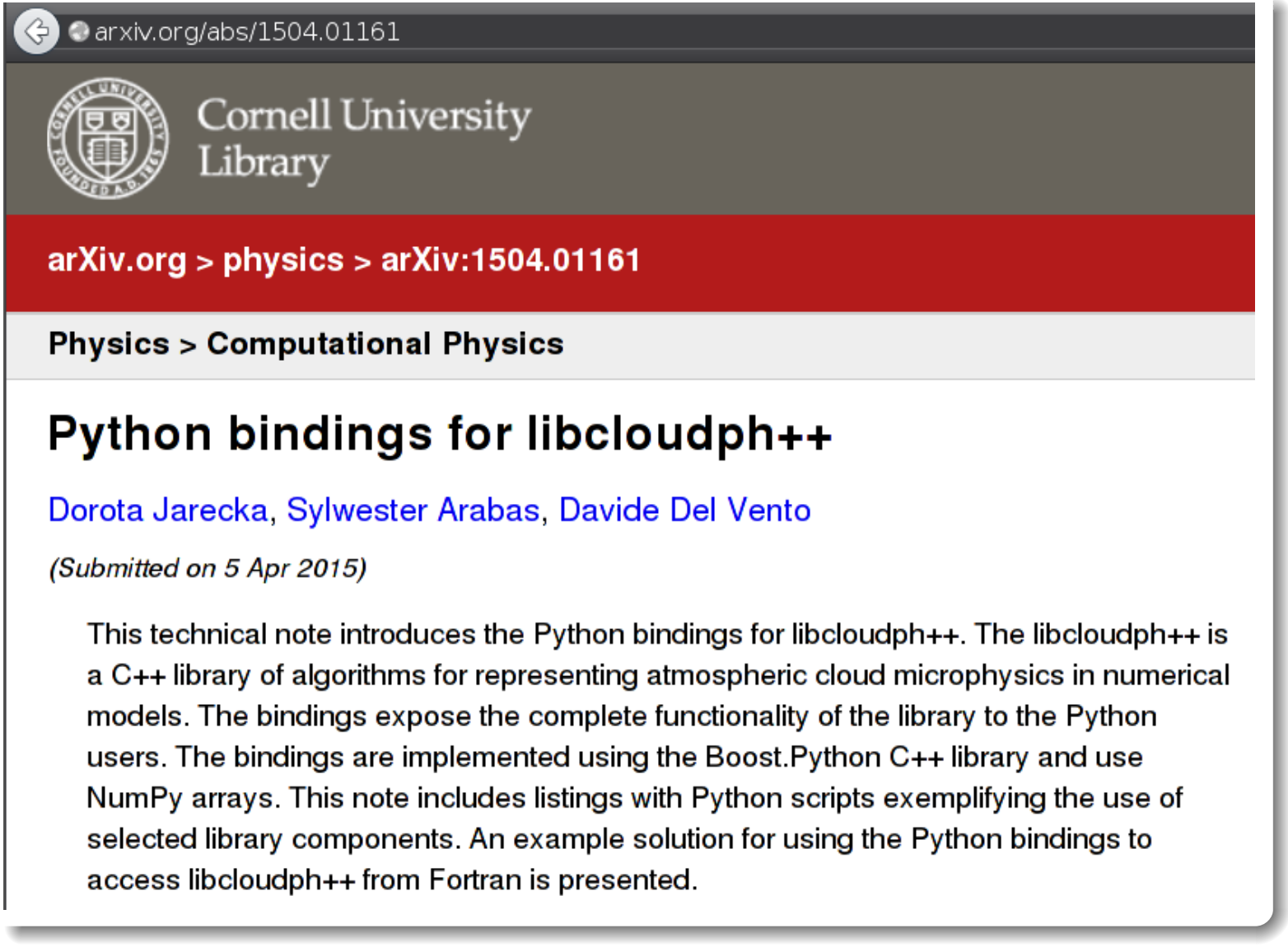


Figure 5. Sequence diagram of *libcloudph++* API calls for the double-moment bulk scheme and a prototype transport equation solver. Diagram discussed in Sect. 4.2.2. See also caption of Fig. 2 for description or diagram elements.

GMDD
7, 8275–8360, 2014



http://arxiv.org/abs/1504.01161

libcloudph++: Python bindings

C++

Python

libcloudph++: Python bindings

C++

- numerically-intensive algorithms
- concurrency, CPU/GPU

Python

libcloudph++: Python bindings

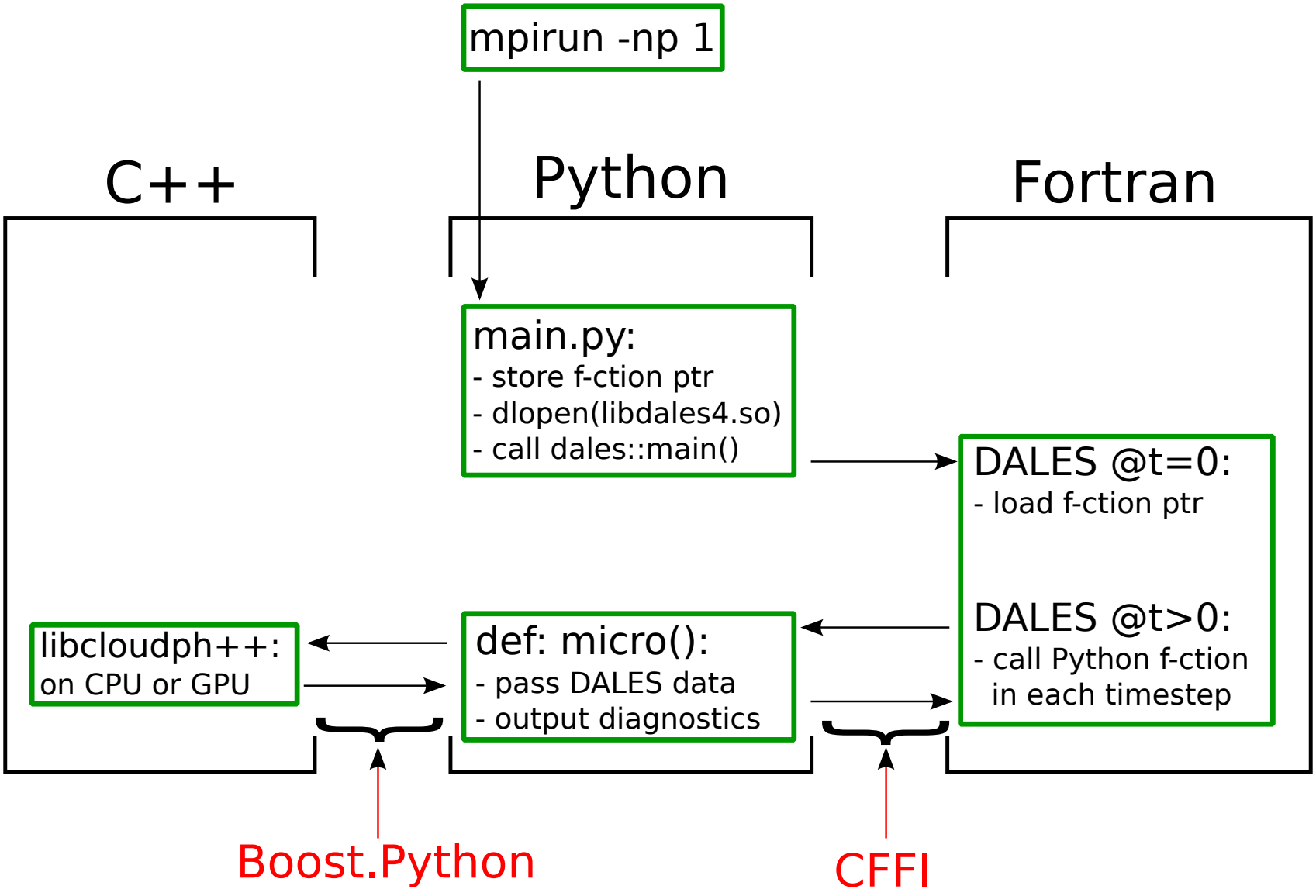
C++

- numerically-intensive algorithms
- concurrency, CPU/GPU

Python

- rapid-development of new features
- interfacing with other languages

libcloudph++: accessing from Fortran via Python



plan of the talk

introduction

libmpdata++

libcloudph++

n-dim array containers

plan of the talk

introduction

libmpdata++

libcloudph++

n-dim array containers

C++ 3D array containers (cloud modeller's perspective)

- `blitz::Array<real_t, 3>`
- `Eigen::Tensor<real_t, 3>`
- `arma::Cube<real_t>`
- `boost::multi_array<real_t, 3>`
- `std::valarray<real_t>` & `std::gslice`
- ...

C++ 3D array containers (cloud modeller's perspective)

- `blitz::Array<real_t, 3>`
- `Eigen::Tensor<real_t, 3>`
- `arma::Cube<real_t>`
- `boost::multi_array<real_t, 3>`
- `std::valarray<real_t>` & `std::gslice`
- ...

no NumPy-like de-facto standard

$\rightsquigarrow$ interoperability with libraries through C pointers

Blitz++

cons

pros

Blitz++

cons

- no development activity

pros

Blitz++

cons

- no development activity

pros

- great mailing-list support

Blitz++

cons

- no development activity
- unmaintained docs

pros

- great mailing-list support

Blitz++

cons

- no development activity
- unmaintained docs

pros

- great mailing-list support
- well-written docs

Blitz++

cons

- no development activity
- unmaintained docs
- not part of Boost (e.g., packaging)

pros

- great mailing-list support
- well-written docs

Blitz++

cons

- no development activity
- unmaintained docs
- not part of Boost (e.g., packaging)

pros

- great mailing-list support
- well-written docs
- built-in support for Boost.MPI & Boost.serialize

Blitz++

cons

- no development activity
- unmaintained docs
- not part of Boost (e.g., packaging)
- icc-only SIMD support

pros

- great mailing-list support
- well-written docs
- built-in support for Boost.MPI & Boost.serialize

Blitz++

cons

- no development activity
- unmaintained docs
- not part of Boost (e.g., packaging)
- icc-only SIMD support

pros

- great mailing-list support
- well-written docs
- built-in support for Boost.MPI & Boost.serialize
- SIMD support

Blitz++

cons

- no development activity
- unmaintained docs
- not part of Boost (e.g., packaging)
- icc-only SIMD support
- multi-screen error messages

pros

- great mailing-list support
- well-written docs
- built-in support for Boost.MPI & Boost.serialize
- SIMD support

Blitz++

cons

- no development activity
- unmaintained docs
- not part of Boost (e.g., packaging)
- icc-only SIMD support
- multi-screen error messages

pros

- great mailing-list support
- well-written docs
- built-in support for Boost.MPI & Boost.serialize
- SIMD support
- convenient debug mode

Blitz++

cons

- no development activity
- unmaintained docs
- not part of Boost (e.g., packaging)
- icc-only SIMD support
- multi-screen error messages
- no libs for binary i/o

pros

- great mailing-list support
- well-written docs
- built-in support for Boost.MPI & Boost.serialize
- SIMD support
- convenient debug mode

Blitz++

cons

- no development activity
- unmaintained docs
- not part of Boost (e.g., packaging)
- icc-only SIMD support
- multi-screen error messages
- no libs for binary i/o

pros

- great mailing-list support
- well-written docs
- built-in support for Boost.MPI & Boost.serialize
- SIMD support
- convenient debug mode
- built-in iostream i/o

Blitz++

cons

- no development activity
- unmaintained docs
- not part of Boost (e.g., packaging)
- icc-only SIMD support
- multi-screen error messages
- no libs for binary i/o
- partly preprocessor-based API

pros

- great mailing-list support
- well-written docs
- built-in support for Boost.MPI & Boost.serialize
- SIMD support
- convenient debug mode
- built-in iostream i/o

Blitz++

cons

- no development activity
- unmaintained docs
- not part of Boost (e.g., packaging)
- icc-only SIMD support
- multi-screen error messages
- no libs for binary i/o
- partly preprocessor-based API

pros

- great mailing-list support
- well-written docs
- built-in support for Boost.MPI & Boost.serialize
- SIMD support
- convenient debug mode
- built-in iostream i/o
- **well-suited API! . . .**

C++ 3D array containers (cloud modeller's perspective)

C++ 3D array containers (cloud modeller's perspective)

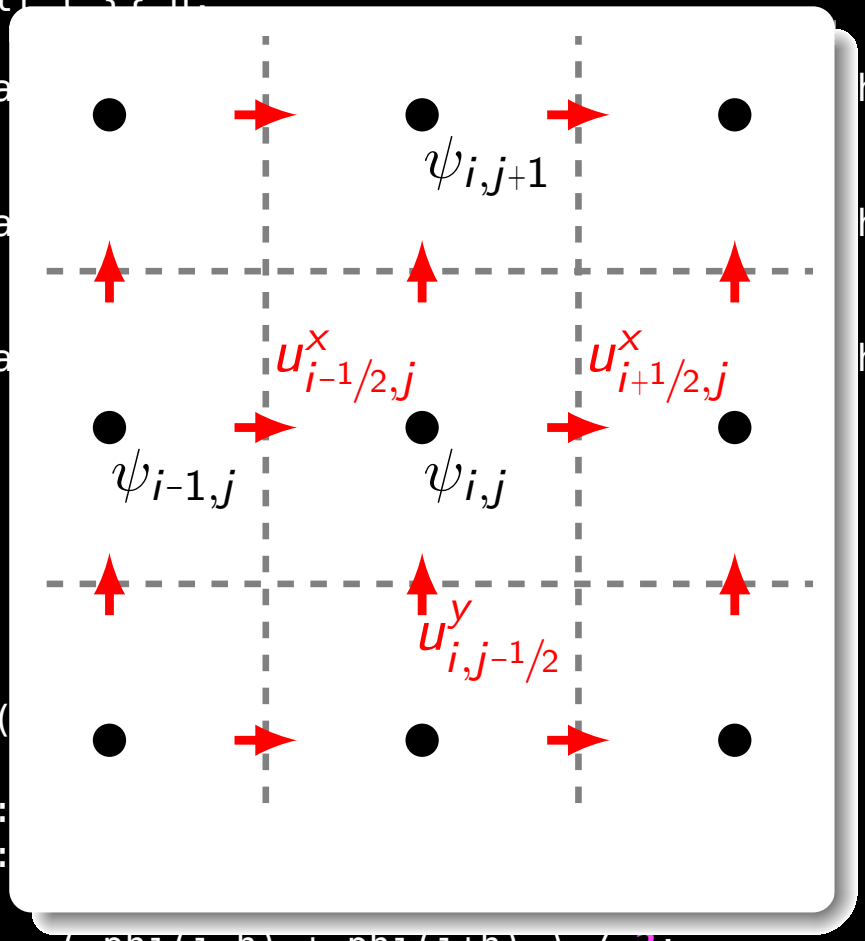
Blitz++ API: **overloadable OO index arithmetics**

```
 1 #include <blitz/array.h>
 2
 3 struct hlf_t {} h;
 4
 5 blitz::Range operator+( const blitz::Range &i, hlf_t &)
 6 { return i;  }
 7
 8 blitz::Range operator-( const blitz::Range &i, hlf_t &)
 9 { return i-1; }
10
11 blitz::Range operator^( const blitz::Range &r, hlf_t &n)
12 {
13   return blitz::Range(
14     (r - n).first(),
15     (r + n).last()
16   );
17 }
18
19 int main()
20 {
21   blitz::Range i(0,10);
22   blitz::Array<float, 1> psi(i), phi(i^h);
23
24   psi(i) = ( phi(i-h) + phi(i+h) ) / 2;
25 }
```

C++ 3D array containers (cloud modeller's perspective)

Blitz++ API: **overloadable OO index arithmetics**

```
 1 #include <blitz/array.h>
 2
 3 struct hlf_t {} h;
 4
 5 blitz::Ra                                                    hlf_t &)
 6 { return
 7
 8 blitz::Ra                                                    hlf_t &)
 9 { return
10
11 blitz::Ra                                                    hlf_t &n)
12 {
13   return
14     (r -
15     (r +
16   );
17 }
18
19 int main(
20 {
21   blitz::
22   blitz::
23
24   psi(i) = ( phi(i-h) + phi(i+h) ) / 2;
25 }
```



C++ 3D array containers (cloud modeller's perspective)

C++ 3D array containers (cloud modeller's perspective)

Blitz++ API: **multi-dim. index with single obj.**

```
 1 #include <blitz/array.h>
 2
 3 int main()
 4 {
 5   blitz::Range i(0,9), j(0,9), k(0,9);
 6
 7   blitz::Array<int, 3> psi(i,j,k);
 8
 9   std::cerr << psi(i,j,k);
10
11   blitz::RectDomain<3> ijk({i,j,k});
12
13   std::cerr << psi(ijk);
14 }
```

C++ 3D array containers (cloud modeller's perspective)

Blitz++ API: multi-dim. index with single obj.

```
 1 #include <blitz/array.h>
 2
 3 int main()
 4 {
 5   blitz::Range i(0,9), j(0,9), k(0,9);
 6
 7   blitz::Array<int, 3> psi(i,j,k);
 8
 9   std::cerr << psi(i,j,k);
10
11   blitz::RectDomain<3> ijk({i,j,k});
12
13   std::cerr << psi(ijk);
14 }
```

index permutations!

C++ 3D array containers (cloud modeller's perspective)

C++ 3D array containers (cloud modeller's perspective)

Blitz++ API: **tensor notation**

```
 1 #include <blitz/array.h>
 2
 3 int main()
 4 {
 5   blitz::Array<float,2> psi(4,4);
 6   {
 7     using namespace blitz::tensor;
 8     psi = sqrt(i*i + j*j);
 9   }
10   std::cout << psi;
11 }
```

C++ 3D array containers (cloud modeller's perspective)

C++ 3D array containers (cloud modeller's perspective)

Blitz++ API: **array-valued functions**

```
 1 //#define BZ_THREADSAFE // to enable access locks for ref counting
 2 #include <blitz/array.h>
 3
 4 enum { opt_sin, opt_cos };
 5
 6 template <int opt, typename T>
 7 auto func(T &a, typename std::enable_if<opt == opt_sin>::type* = 0)
 8 {
 9   return safeToReturn(pow(sin(a), 2)); // keeps ref count
10 }
11
12 template <int opt, typename T>
13 auto func(T &a, typename std::enable_if<(opt == opt_cos)>::type* = 0)
14 {
15   return safeToReturn(pow(cos(a), 2)); // keeps ref count
16 }
17
18 int main()
19 {
20   blitz::Array<float, 1> psi(1000);
21   {
22     using namespace blitz::tensor;
23     psi = i / 20.;
24   }
25   psi = func<opt_sin>(psi) + func<opt_cos>(psi);
26   std::cout << psi;
27 }
```

C++ 3D array containers (cloud modeller's perspective)

C++ 3D array containers (cloud modeller's perspective)

Blitz++ API: **elemental functors**

```
 1 #include <blitz/array.h>
 2
 3 struct func
 4 {
 5   float a = .25, b = .1;
 6
 7   float operator()(float x) const
 8   {
 9     return a * x + b;
10   }
11
12   // to make it accept Blitz arrays as arguments
13   BZ_DECLARE_FUNCTOR(func);
14 };
15
16 int main()
17 {
18   blitz::Array<float, 1> psi(10), x(10);
19
20   x = blitz::tensor::i;
21
22   psi = func()(x);
23 }
```

C++ 3D array containers (cloud modeller's perspective)

C++ 3D array containers (cloud modeller's perspective)

Blitz++ API: **ternary op. & reductions**

```
 1 #include <blitz/array.h>
 2
 3 int main()
 4 {
 5   blitz::Array<float, 2> psi(5,5);
 6   {
 7     using namespace blitz::tensor;
 8     psi = sqrt((i*i) + (j*j));
 9   }
10
11   // ternary operator
12   psi = where(psi>3, 0, psi);
13
14   // partial reduction
15   {
16     using namespace blitz::tensor;
17     auto xpr = sum(psi, j); // delayed eval!
18   }
19 }
```

last slide

last slide

- C++ does constitute a viable alternative to Fortran for maintainable code and reproducible results in cloud-modelling

last slide

- C++ does constitute a viable alternative to Fortran for maintainable code and reproducible results in cloud-modelling
- multi-dimensional C++ array containers:

last slide

- C++ does constitute a viable alternative to Fortran for maintainable code and reproducible results in cloud-modelling
- multi-dimensional C++ array containers:
  - Is there an alternative to Blitz++?

last slide

- C++ does constitute a viable alternative to Fortran for maintainable code and reproducible results in cloud-modelling
- multi-dimensional C++ array containers:
  - Is there an alternative to Blitz++?
  - Does Boost provide it?

last slide

- C++ does constitute a viable alternative to Fortran for maintainable code and reproducible results in cloud-modelling
- multi-dimensional C++ array containers:
  - Is there an alternative to Blitz++?
  - Does Boost provide it?
  - Should Boost provide it?

last slide

- C++ does constitute a viable alternative to Fortran for maintainable code and reproducible results in cloud-modelling
- multi-dimensional C++ array containers:
  - Is there an alternative to Blitz++?
  - Does Boost provide it?
  - Should Boost provide it?
  - Blitz++ needs new features and maintanance!

last slide

- C++ does constitute a viable alternative to Fortran for maintainable code and reproducible results in cloud-modelling
- multi-dimensional C++ array containers:
  - Is there an alternative to Blitz++?
  - Does Boost provide it?
  - Should Boost provide it?
  - Blitz++ needs new features and maintanance!
- acknowledgment to the funding agencies:
  - Development of libmpdata++ and libcloudph++ have been supported by Poland's National Science Centre (2012/06/M/ST10/00434)
  - Development of the Python bindings for libcloudph++ have been supported by the Ministry of Science and Higher Education of Poland (1119/MOB/13/2014/0)

last slide

- C++ does constitute a viable alternative to Fortran for maintainable code and reproducible results in cloud-modelling
- multi-dimensional C++ array containers:
  - Is there an alternative to Blitz++?
  - Does Boost provide it?
  - Should Boost provide it?
  - Blitz++ needs new features and maintanance!
- acknowledgment to the funding agencies:
  - Development of libmpdata++ and libcloudph++ have been supported by Poland's National Science Centre (2012/06/M/ST10/00434)
  - Development of the Python bindings for libcloudph++ have been supported by the Ministry of Science and Higher Education of Poland (1119/MOB/13/2014/0)
- thank you for your attention!