Object-oriented numerics in C++, Python and modern Fortran

a case study comparison

Sylwester Arabas, Dorota Jarecka, Anna Jaruga

Faculty of Physics, University of Warsaw

Maciej Fijałkowski

PyPy Team

UCAR SEA Software Engineering Conference
Boulder, 2nd April 2013

background

Object oriented programming (OOP) *"has become recognised as the almost unique successful paradigm for creating complex software"*

Press et al. 2007: Numerical Recipes – The Art of Scientific Computing

recognised $\neq$ adopted

background

Object oriented programming (OOP) *"has become recognised as the almost unique successful paradigm for creating complex software"*

Press et al. 2007: Numerical Recipes – The Art of Scientific Computing

recognised $\neq$ adopted

background

Application of OOP for numerical modelling software may help to:

- maintain modularity and separation of program logic layers (e.g. separation of numerical algorithms, parallelisation mechanisms, data input/output, error handling and the description of physical processes); and

- **shorten and simplify the source code by reproducing the mathematical notation used in the literature**.

both contribute to software maintainability and auditability

background

Application of OOP for numerical modelling software may help to:

- maintain modularity and separation of program logic layers (e.g. separation of numerical algorithms, parallelisation mechanisms, data input/output, error handling and the description of physical processes); and
- **shorten and simplify the source code by reproducing the mathematical notation used in the literature**.

both contribute to software maintainability and auditability

background

Application of OOP for numerical modelling software may help to:

- maintain modularity and separation of program logic layers (e.g. separation of numerical algorithms, parallelisation mechanisms, data input/output, error handling and the description of physical processes); and

- **shorten and simplify the source code by reproducing the mathematical notation used in the literature**.

both contribute to software maintainability and auditability

background

Application of OOP for numerical modelling software may help to:

- maintain modularity and separation of program logic layers (e.g. separation of numerical algorithms, parallelisation mechanisms, data input/output, error handling and the description of physical processes); and

- **shorten and simplify the source code by reproducing the mathematical notation used in the literature**.

both contribute to software maintainability and auditability

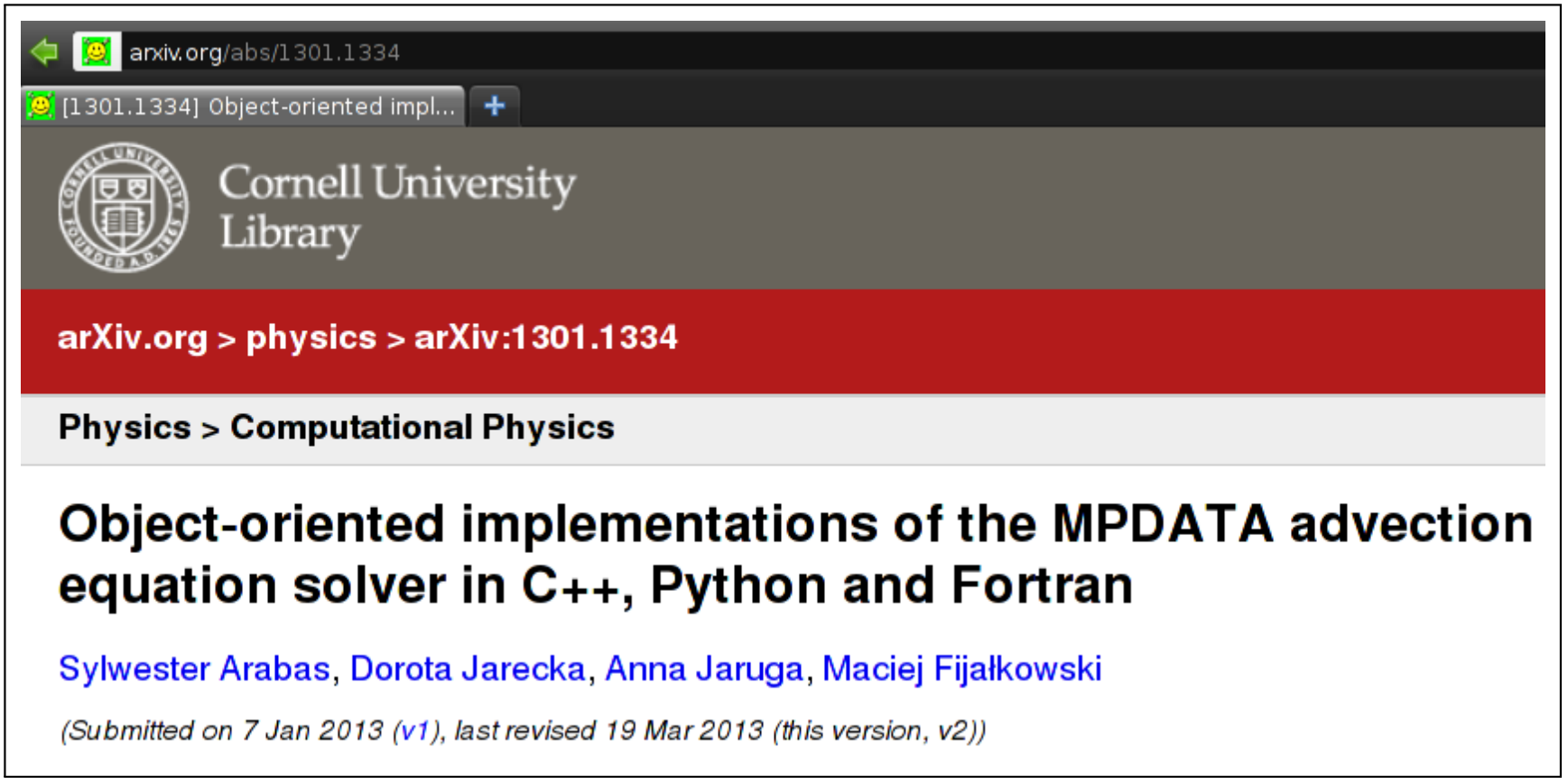
background

we need software maintainability and auditability!

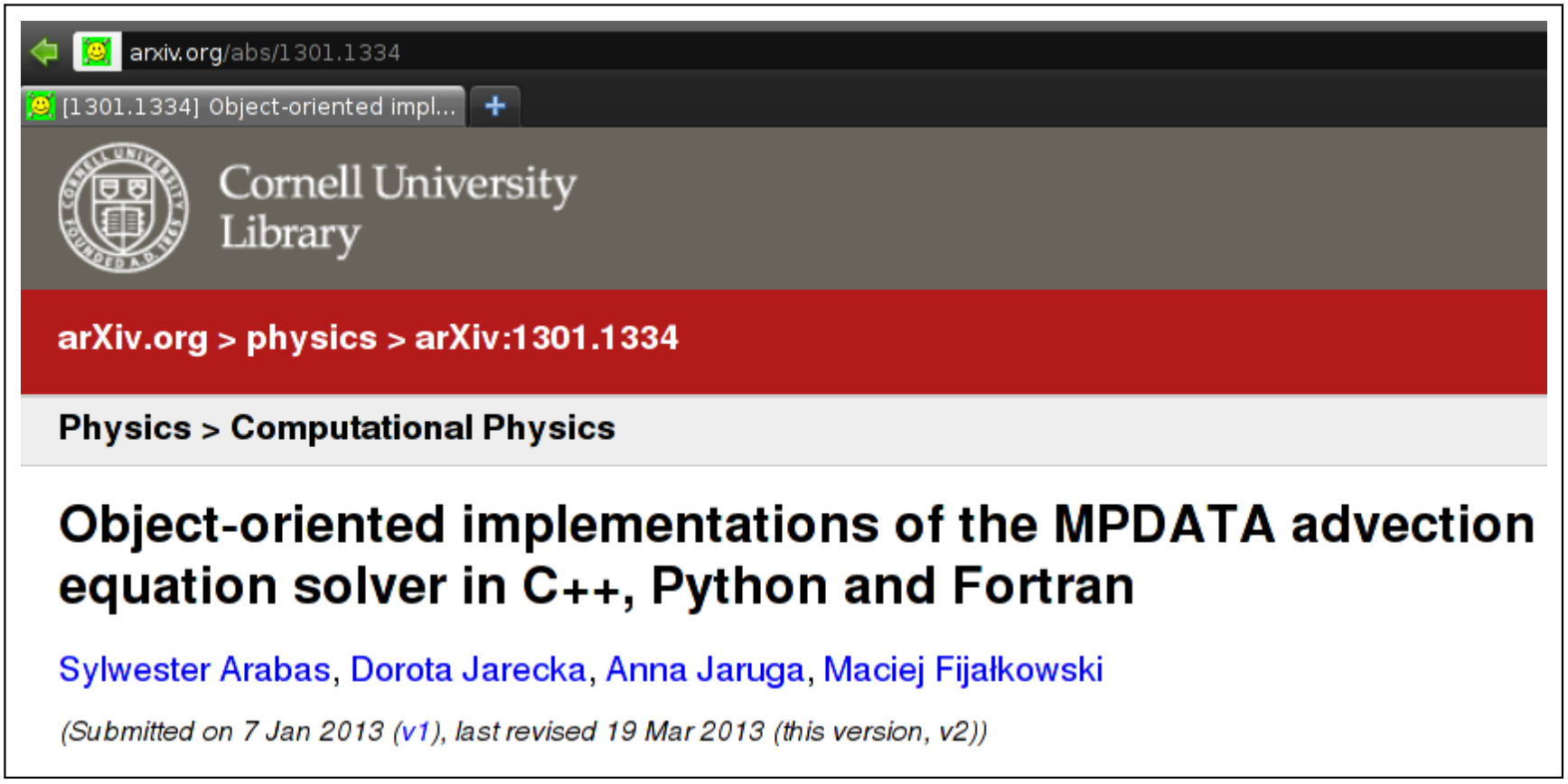
Merali 2010 (Nature, vol. 467, p. 775-777)

(climate modelling context)

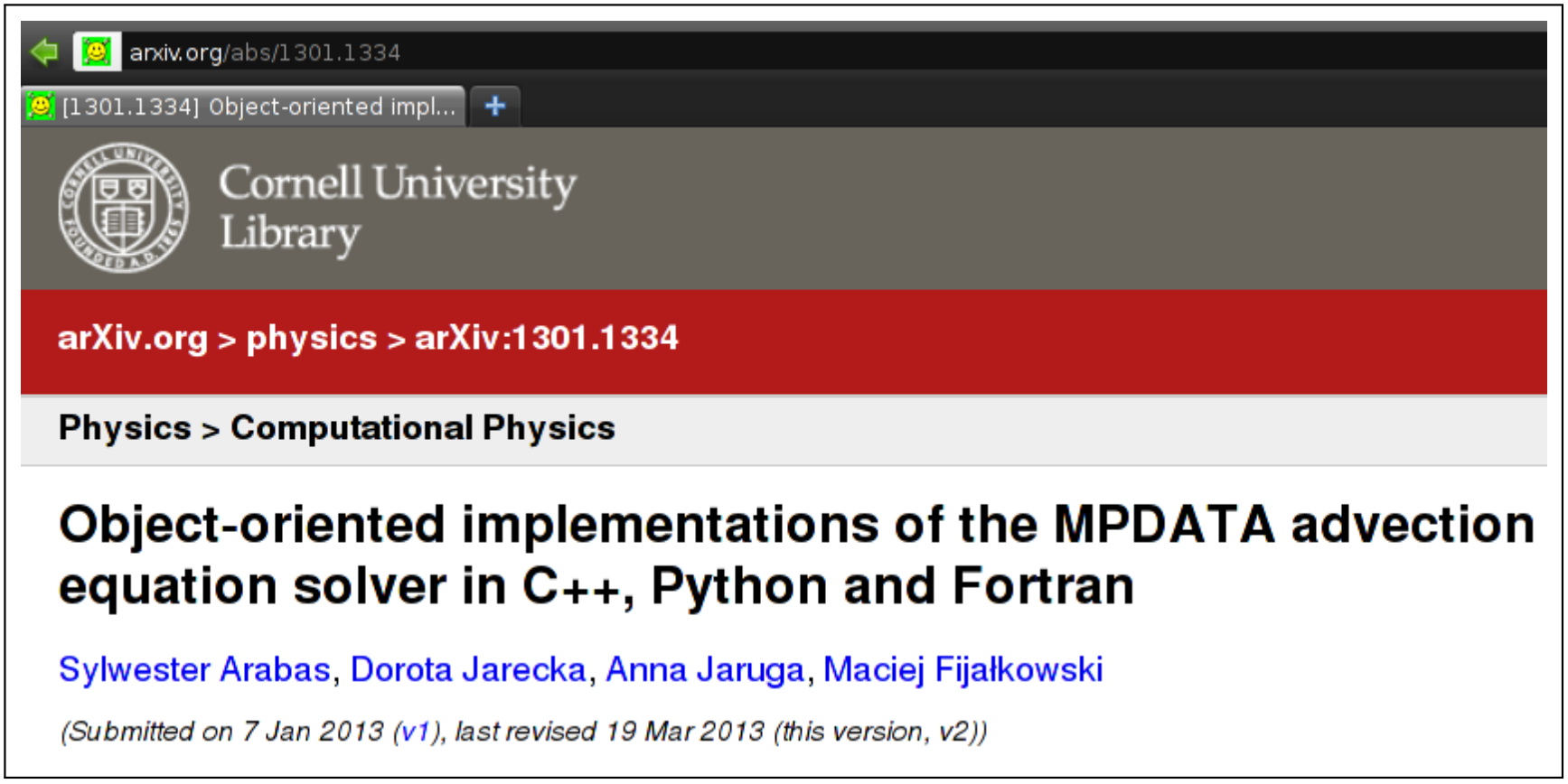
UNIVERSITAS VARSOVIENSIS



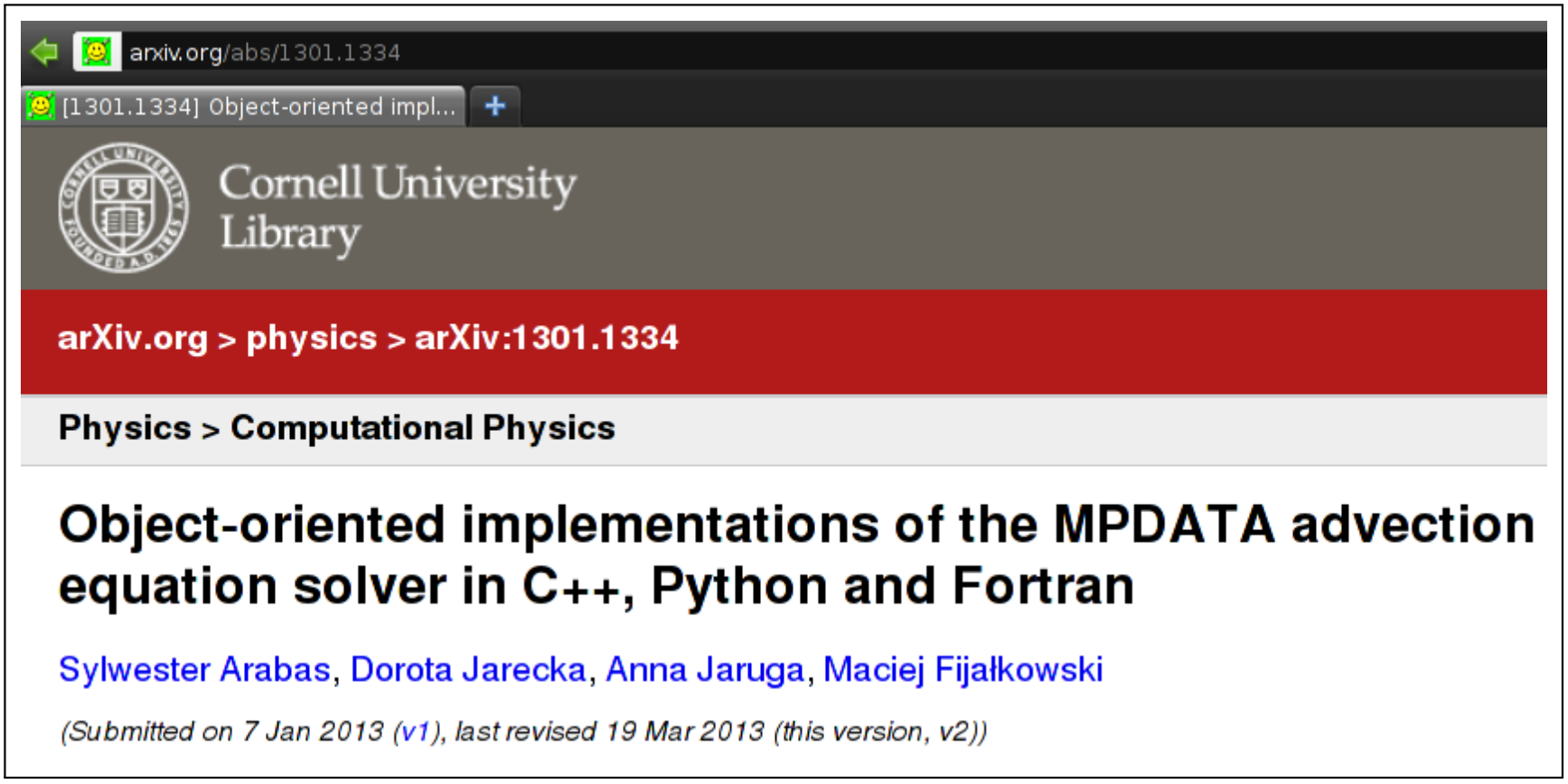
- idea:
  - take an algorithm used in the cores of weather/climate models
  - implement it using OOP in C++, Python and Fortran
  - discuss **all** the code (inlined in the paper text)
  - compare language/library features and performance
- submitted to Comp. Phys. Comm.
- code: github.com/slayoo/mpdata/



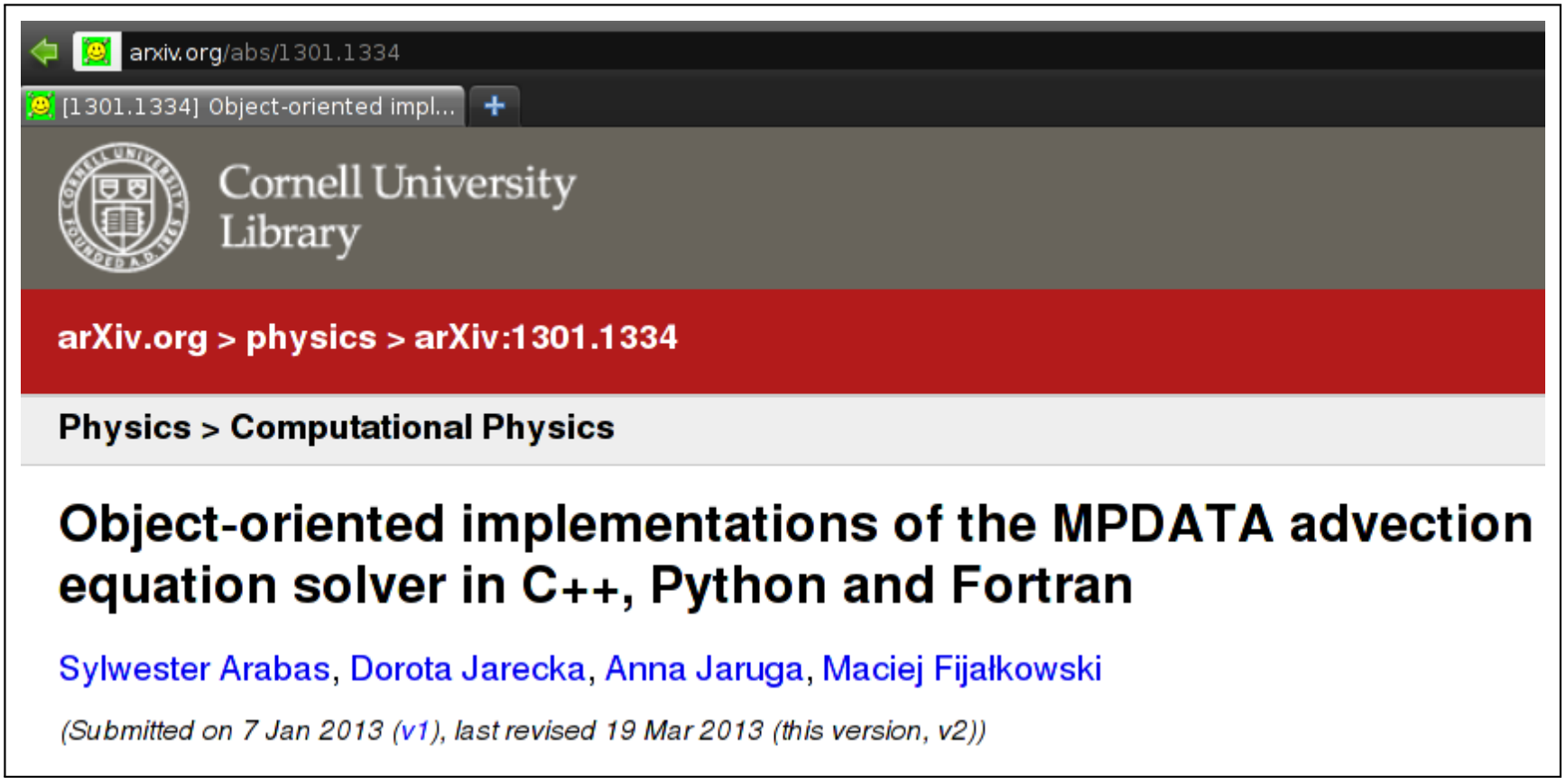
- idea:
  - take an algorithm used in the cores of weather/climate models
  - implement it using OOP in C++, Python and Fortran
  - discuss **all** the code (inlined in the paper text)
  - compare language/library features and performance
- submitted to Comp. Phys. Comm.
- code: github.com/slayoo/mpdata/



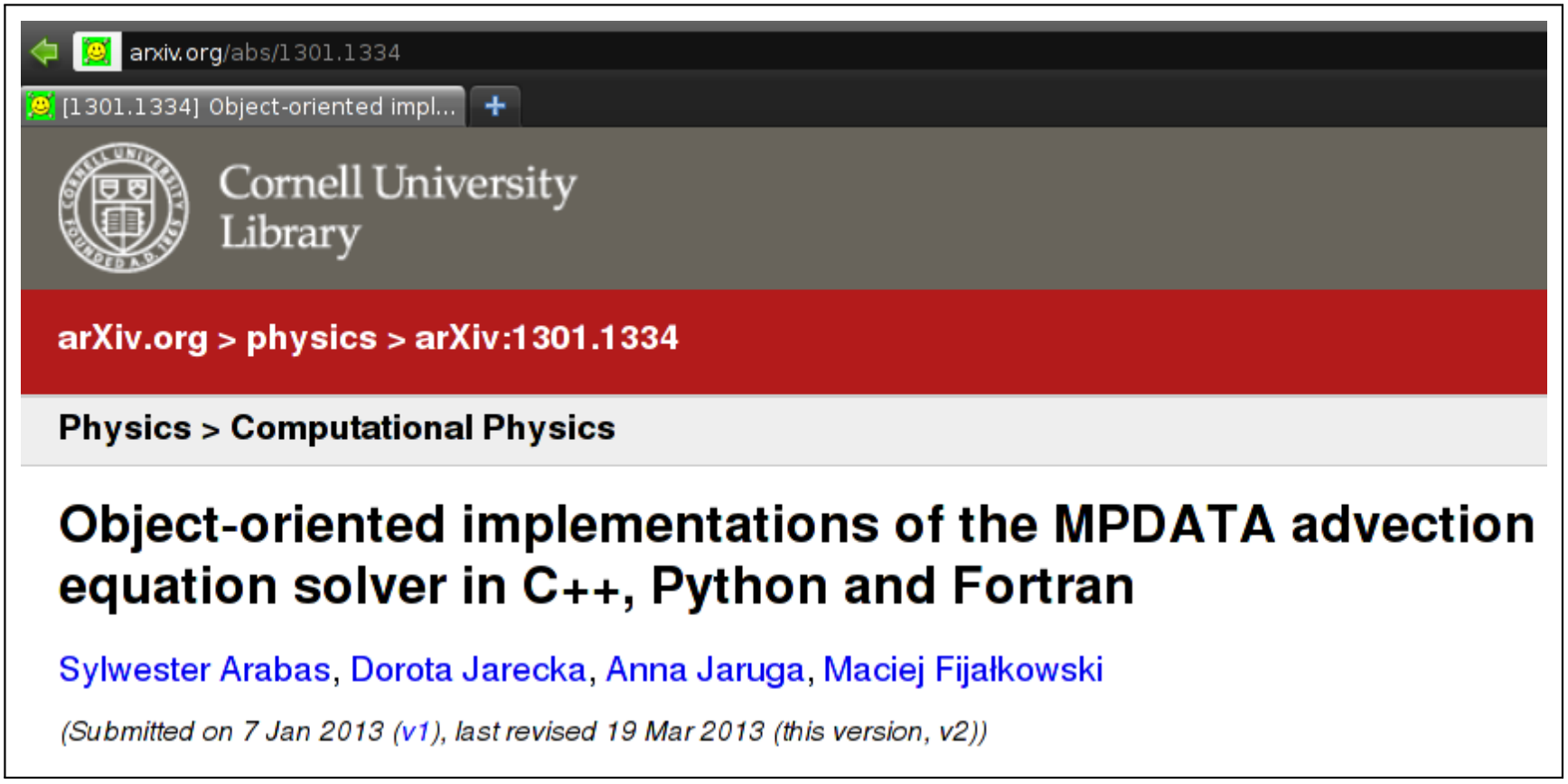
- idea:
  - take an algorithm used in the cores of weather/climate models
  - implement it using OOP in C++, Python and Fortran
  - discuss **all** the code (inlined in the paper text)
  - compare language/library features and performance
- submitted to Comp. Phys. Comm.
- code: github.com/slayoo/mpdata/



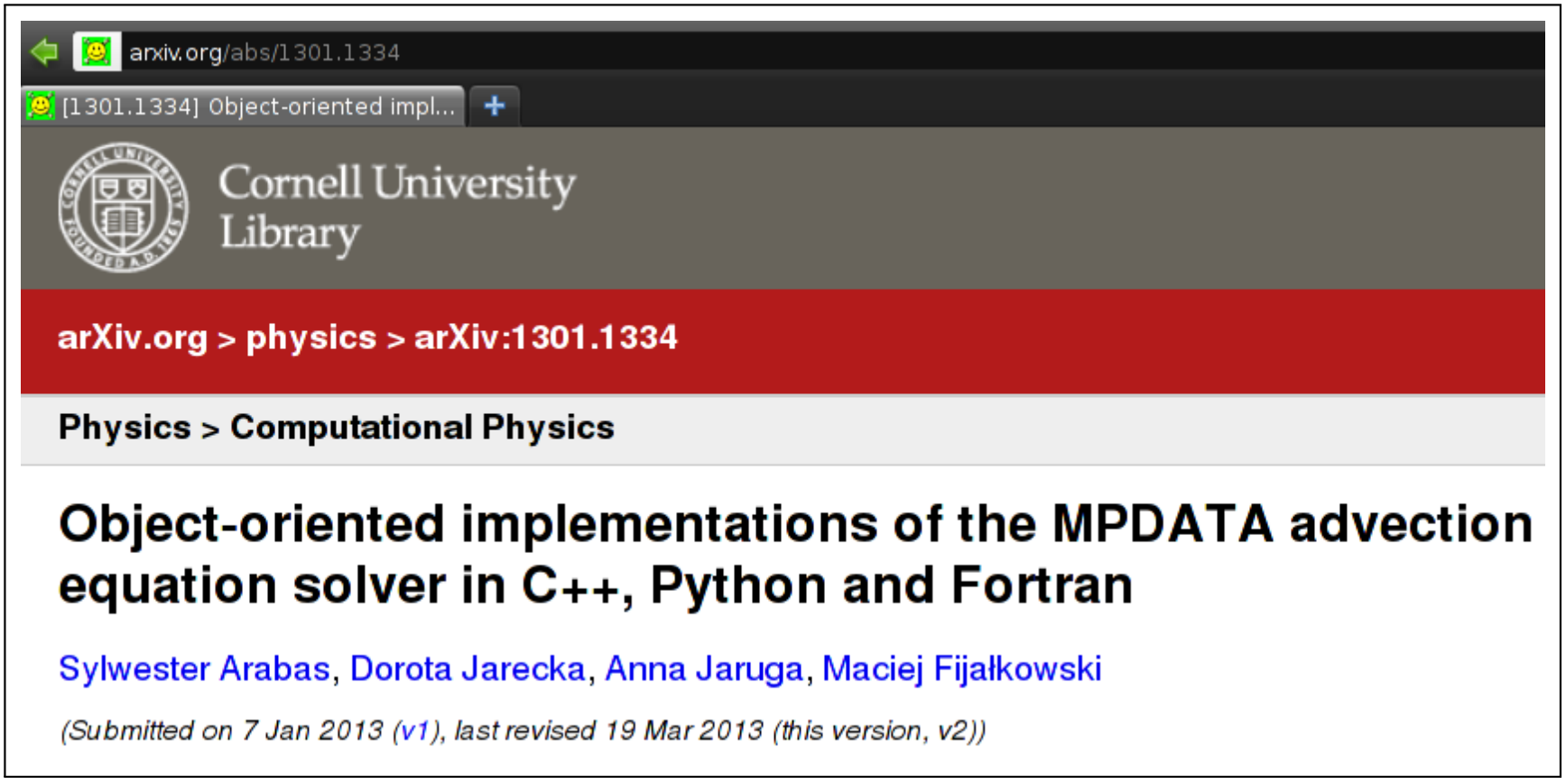
- idea:
  - take an algorithm used in the cores of weather/climate models
  - implement it using OOP in C++, Python and Fortran
  - discuss **all** the code (inlined in the paper text)
  - compare language/library features and performance
- submitted to Comp. Phys. Comm.
- code: github.com/slayoo/mpdata/



- idea:
  - take an algorithm used in the cores of weather/climate models
  - implement it using OOP in C++, Python and Fortran
  - discuss **all** the code (inlined in the paper text)
  - compare language/library features and performance
- submitted to Comp. Phys. Comm.
- code: github.com/slayoo/mpdata/



- idea:
  - take an algorithm used in the cores of weather/climate models
  - implement it using OOP in C++, Python and Fortran
  - discuss **all** the code (inlined in the paper text)
  - compare language/library features and performance
- submitted to Comp. Phys. Comm.
- code: github.com/slayoo/mpdata/



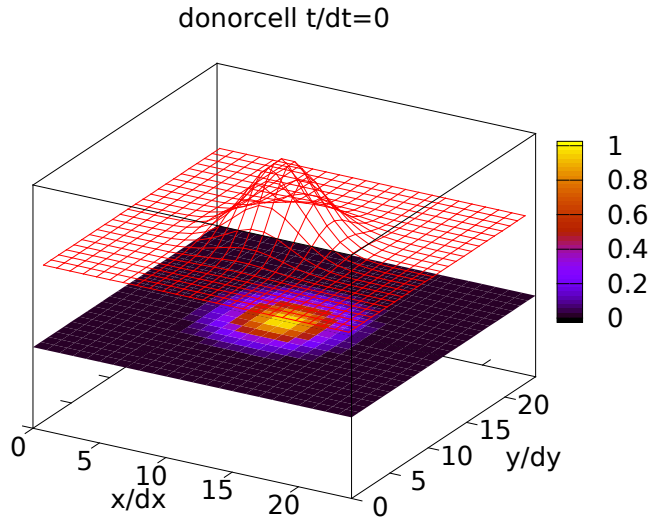
- idea:
  - take an algorithm used in the cores of weather/climate models
  - implement it using OOP in C++, Python and Fortran
  - discuss **all** the code (inlined in the paper text)
  - compare language/library features and performance
- submitted to Comp. Phys. Comm.
- code: github.com/slayoo/mpdata/

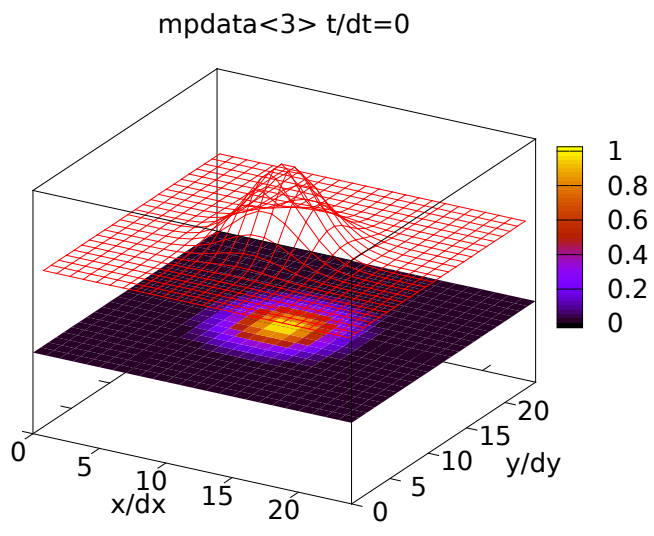
UNIVERSITAS VARSOVIENSIS

plan of this talk

- background
- MPDATA in pictures & formulæ
- MPDATA in C++, Python & Fortran
  - highlights from three OOP implementations
  - performance evaluation
- language choice tradeoffs
- future plans and a take-home message

MPDATA (Smolarkiewicz 1984) in pictures



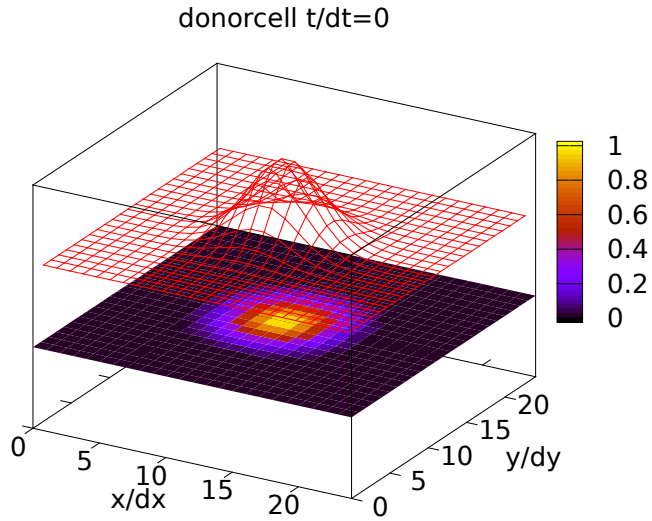


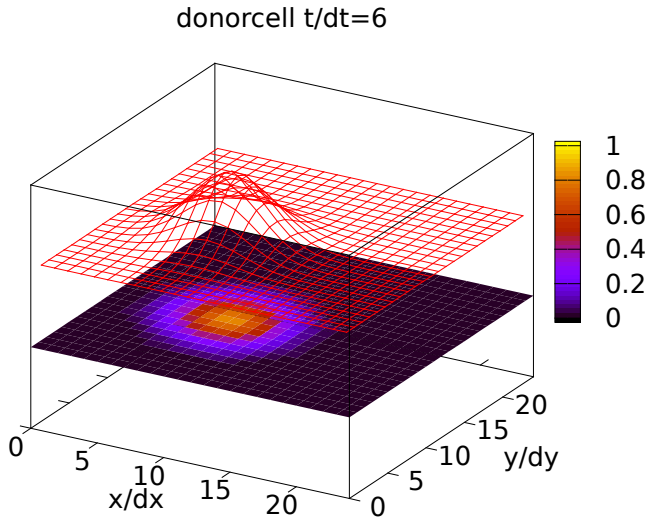
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

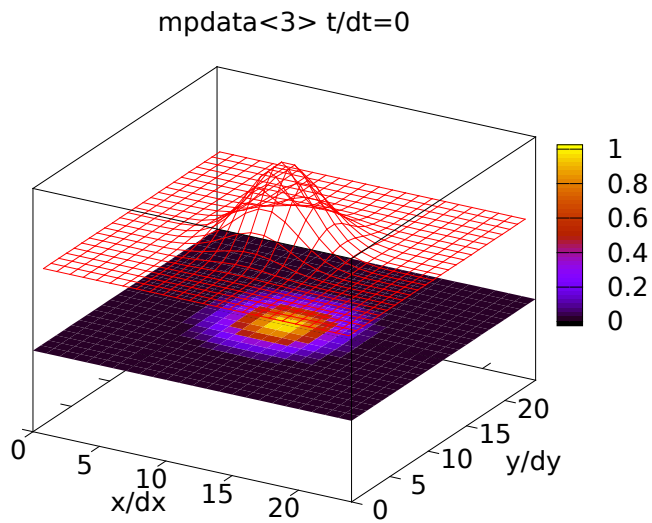
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

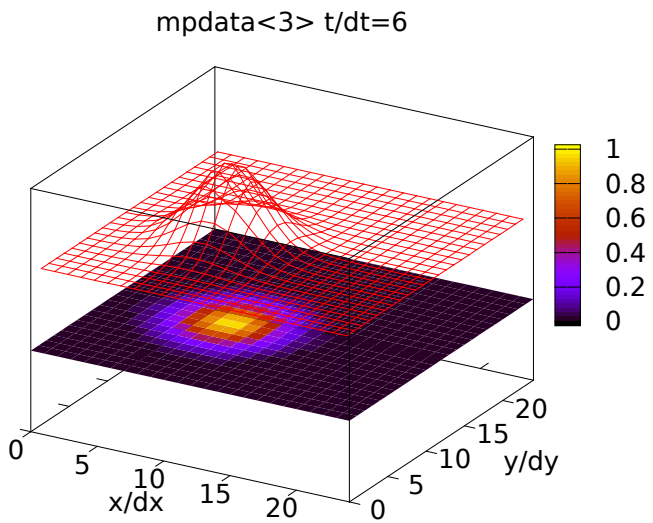
UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures







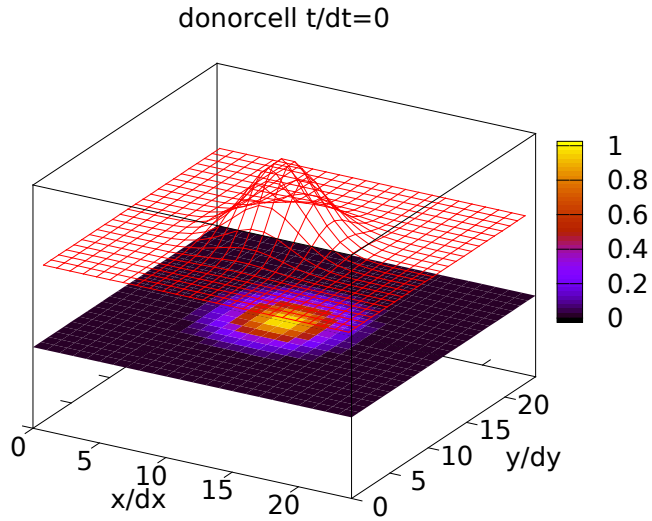


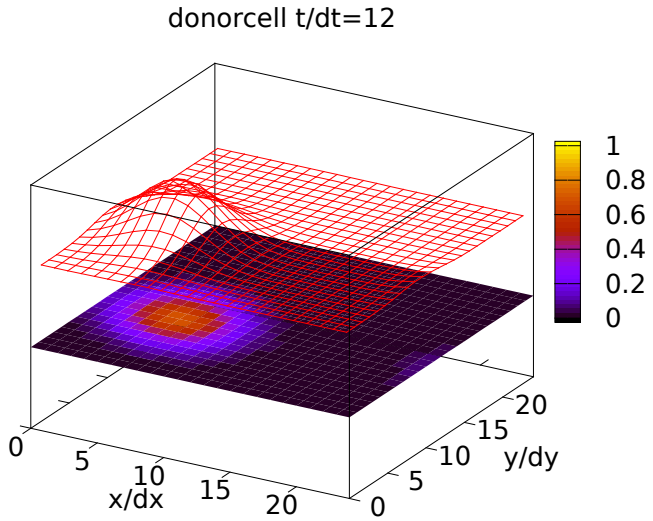
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

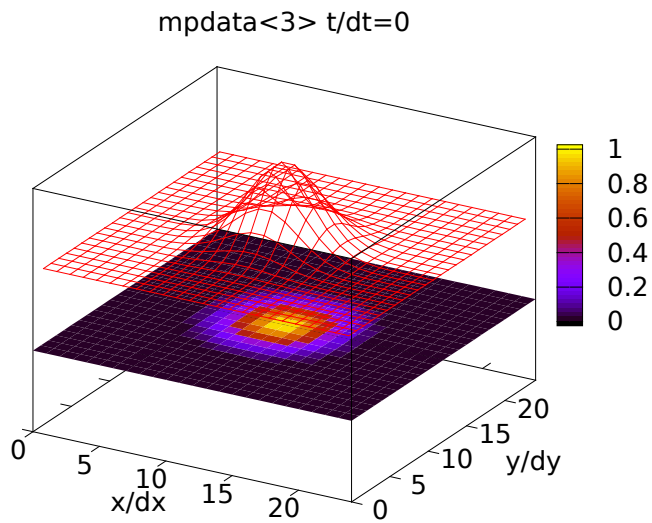
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

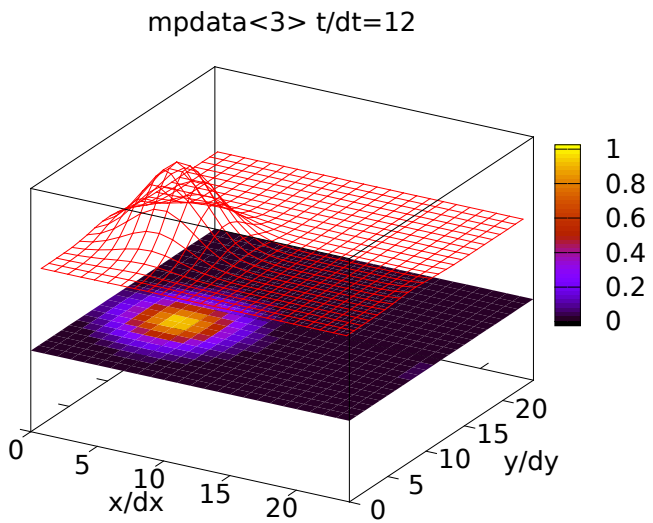
UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures







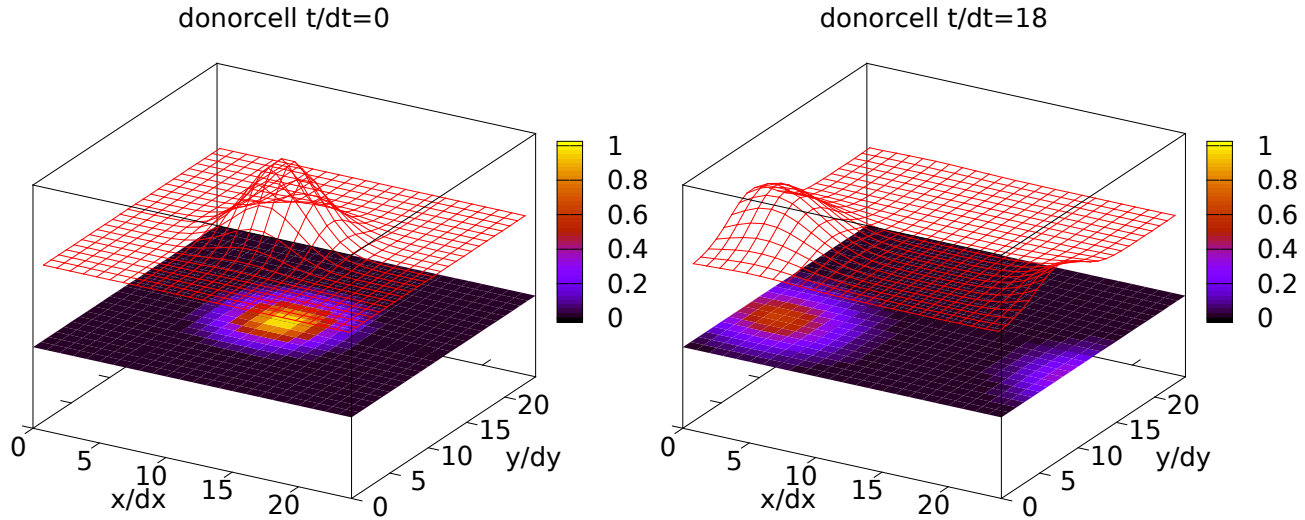


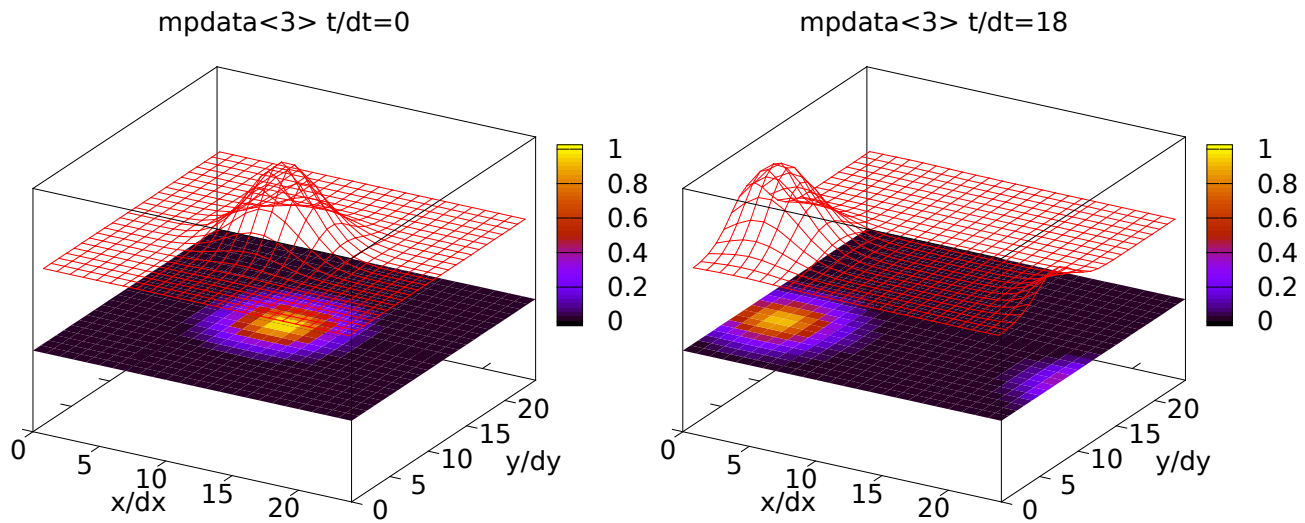
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures



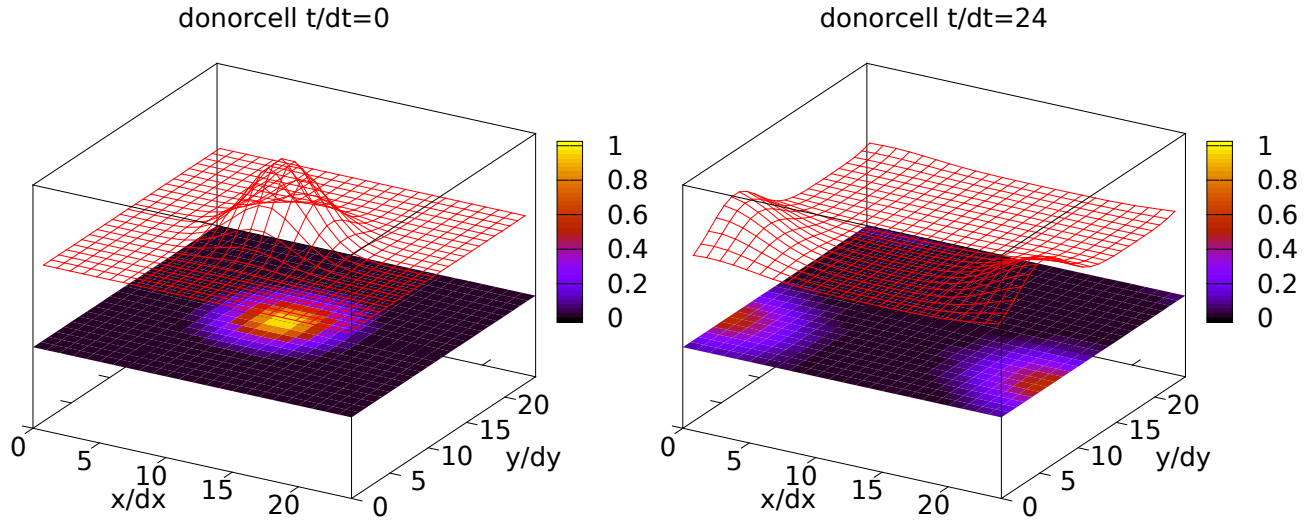


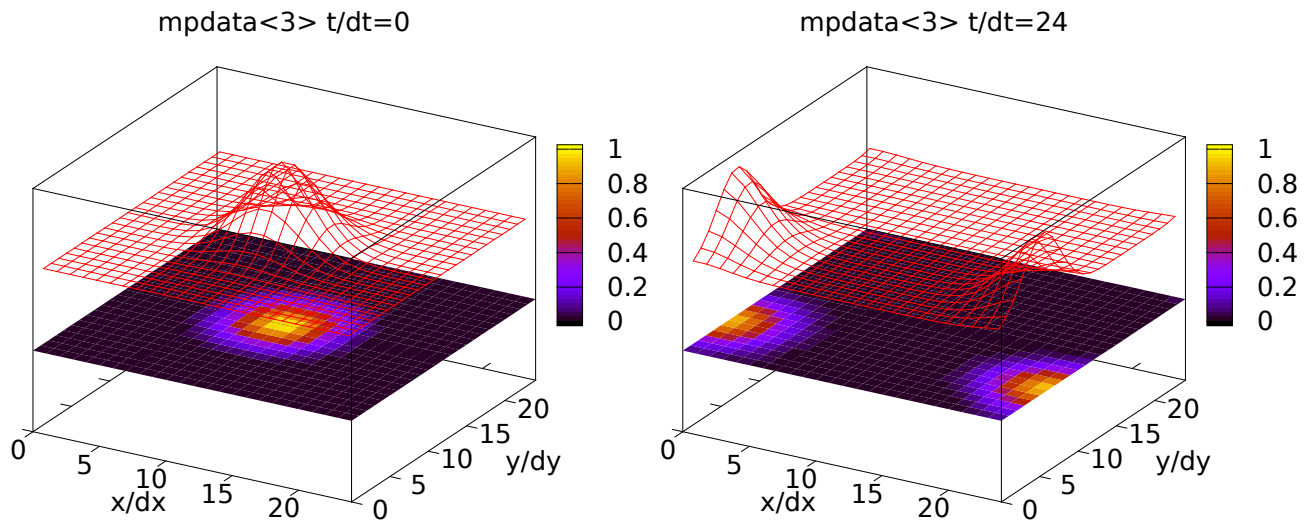
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures



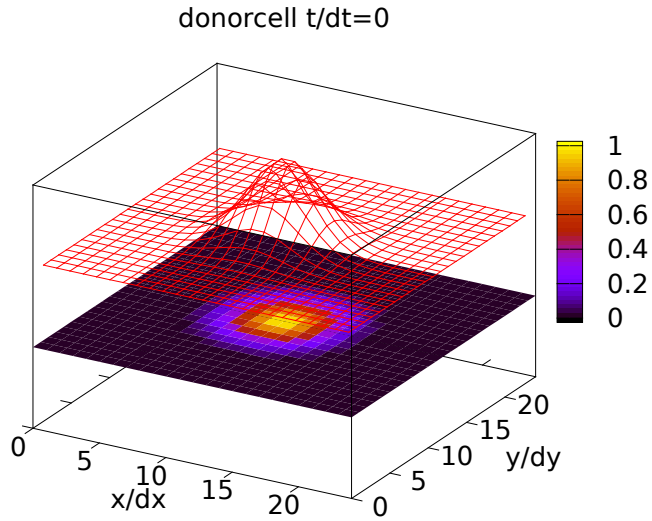


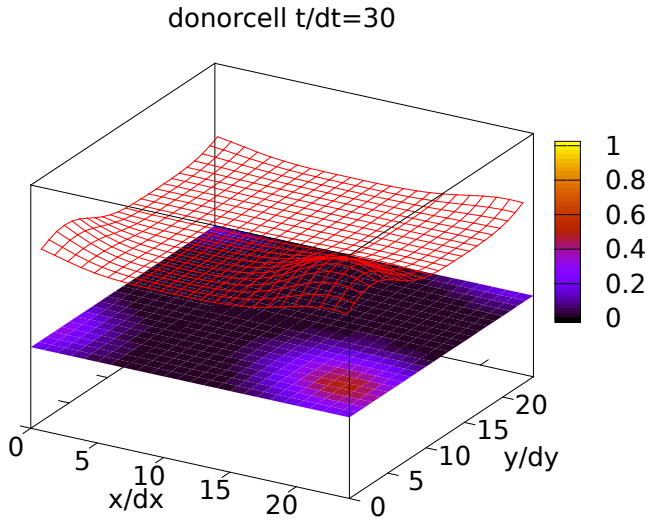
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

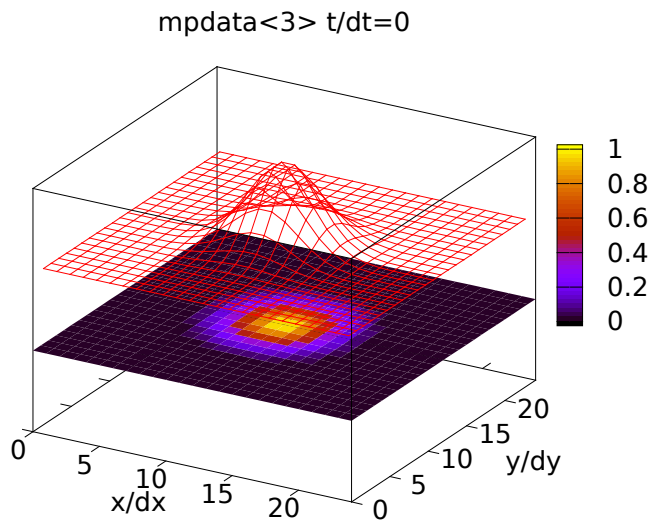
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

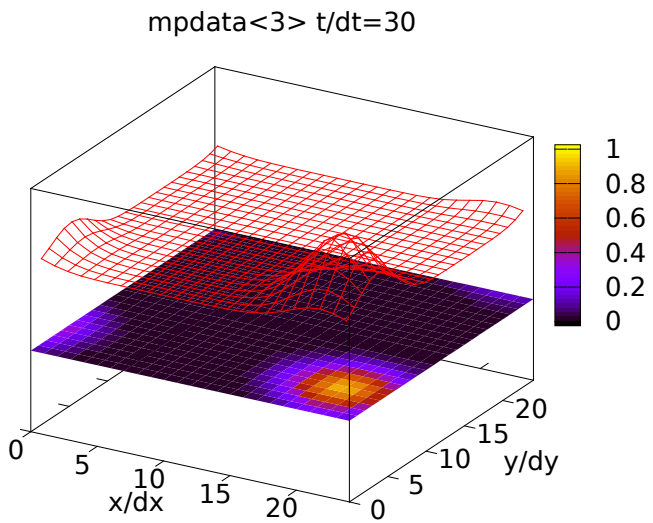
UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures







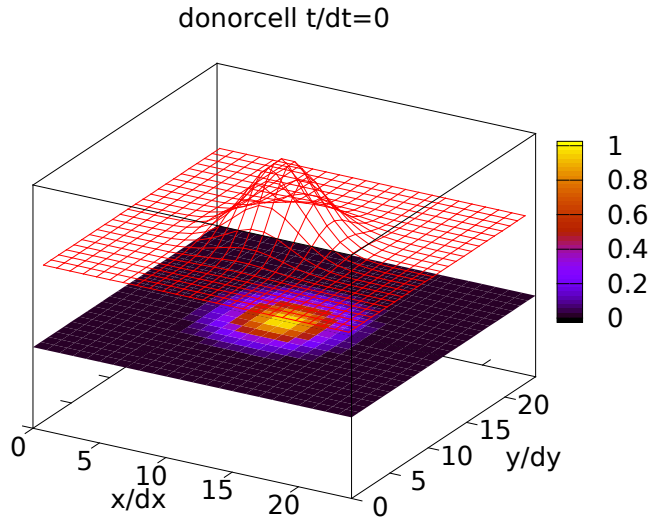


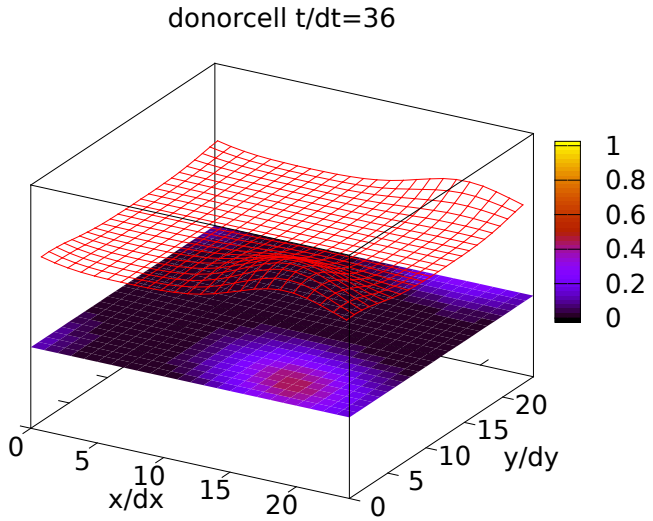
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

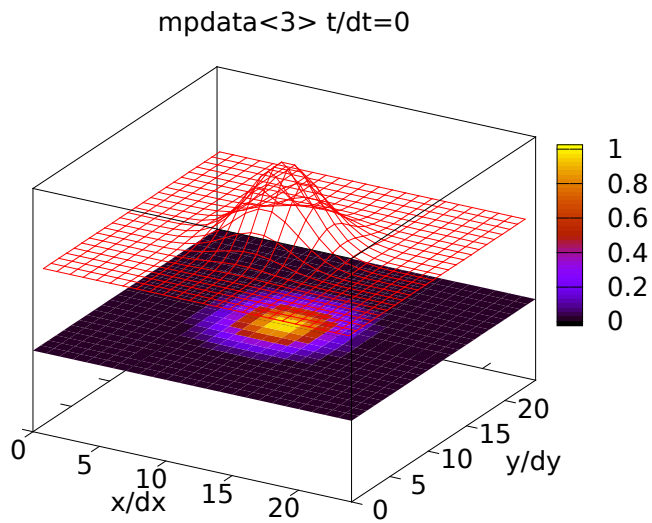
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

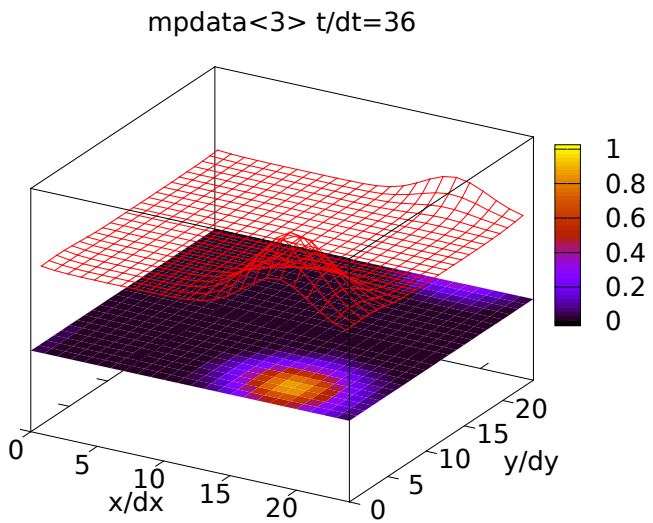
UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures







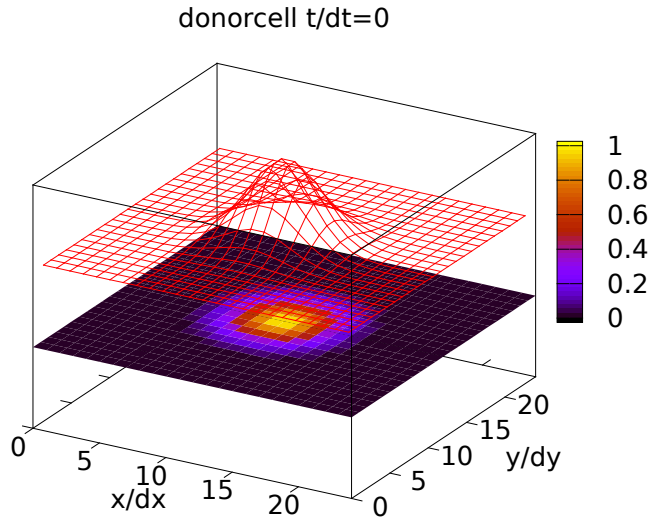


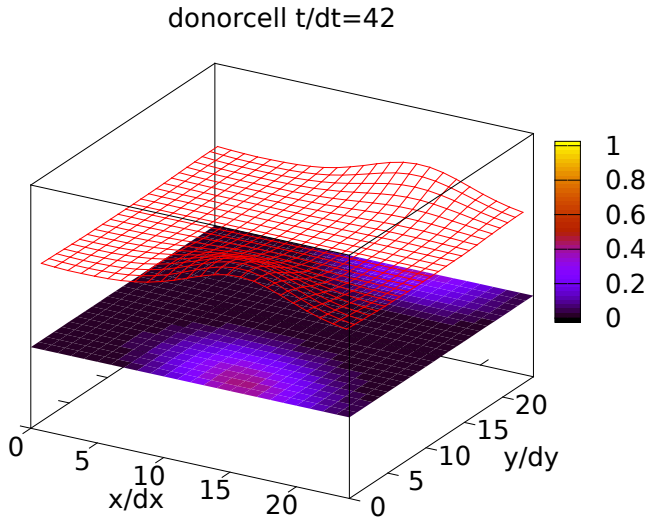
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

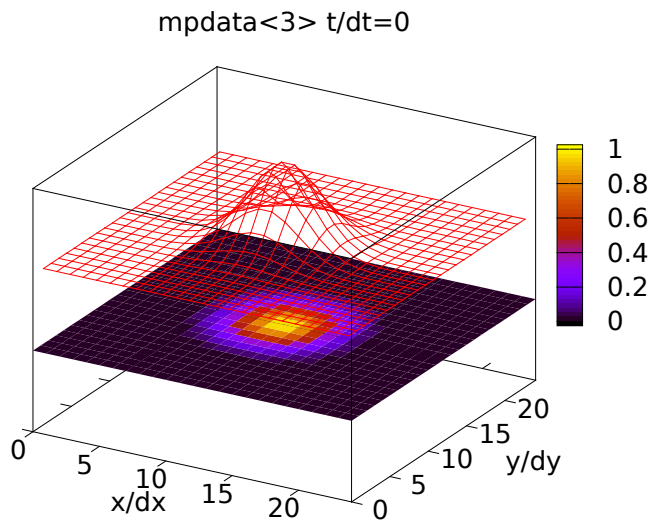
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

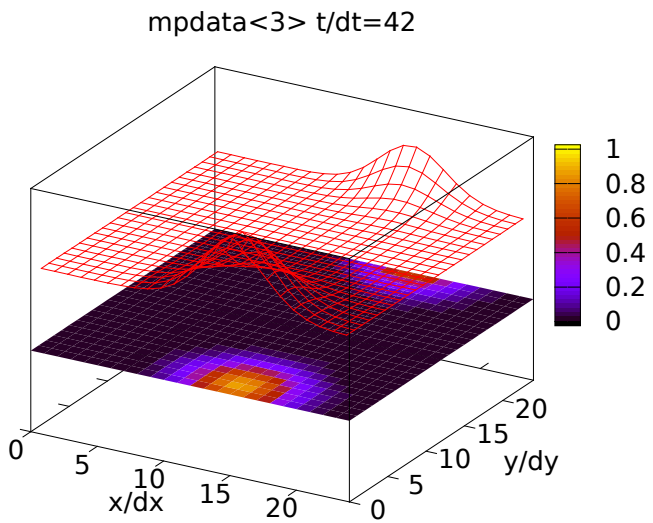
UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures







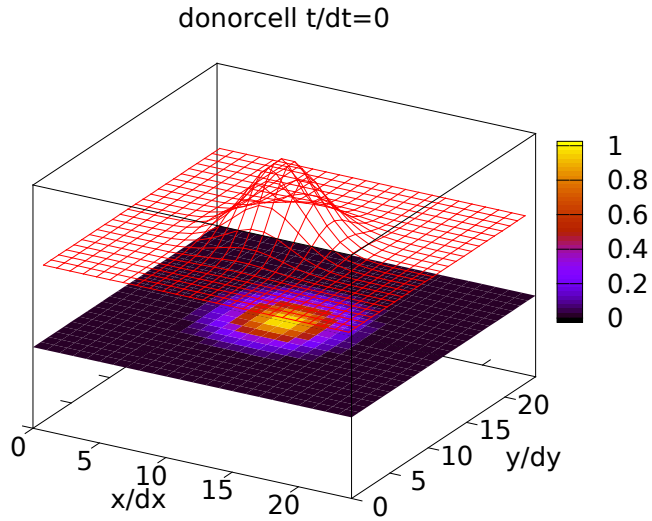


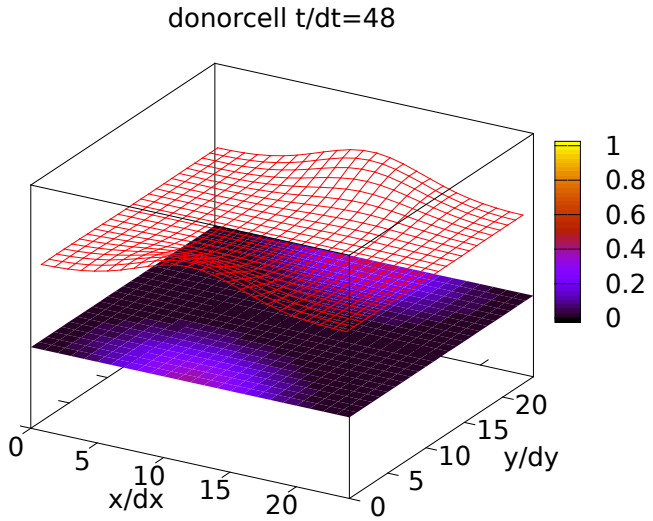
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

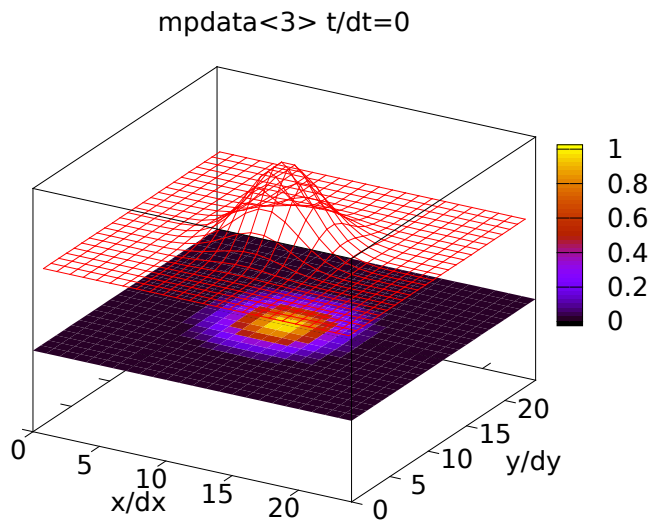
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

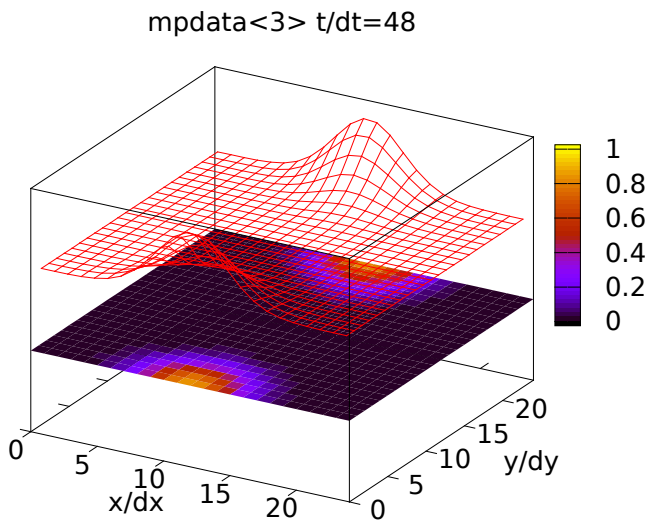
UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures







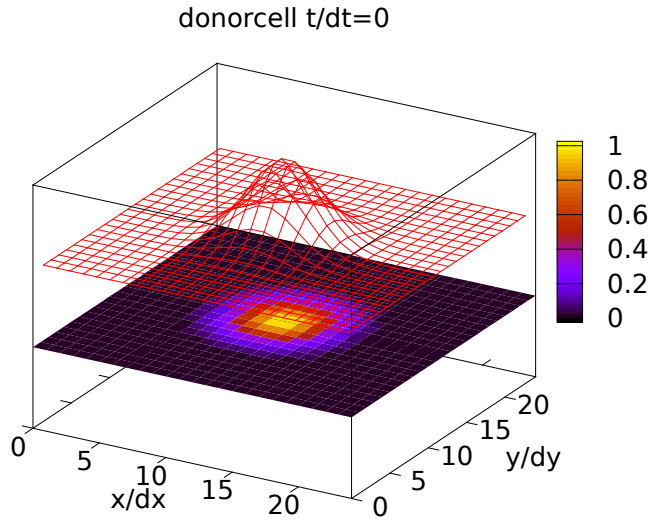


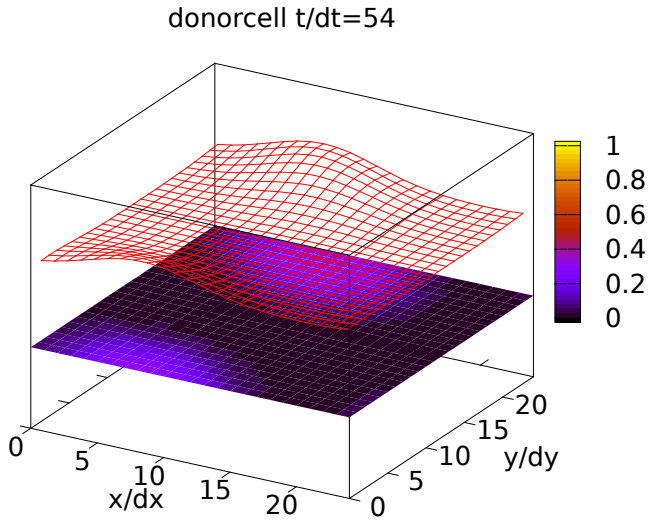
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

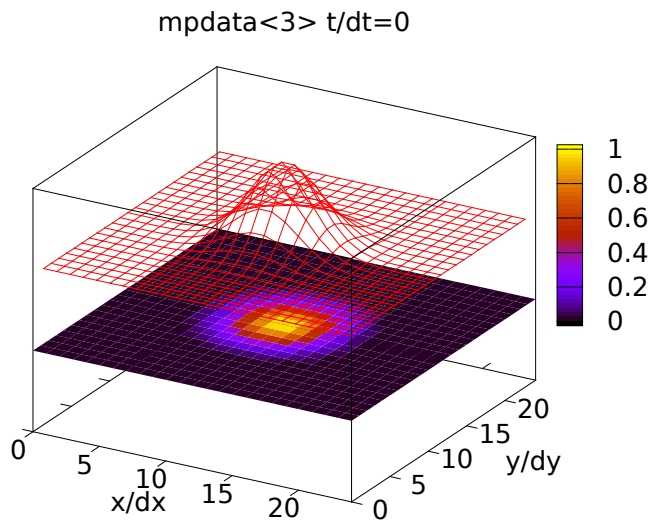
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

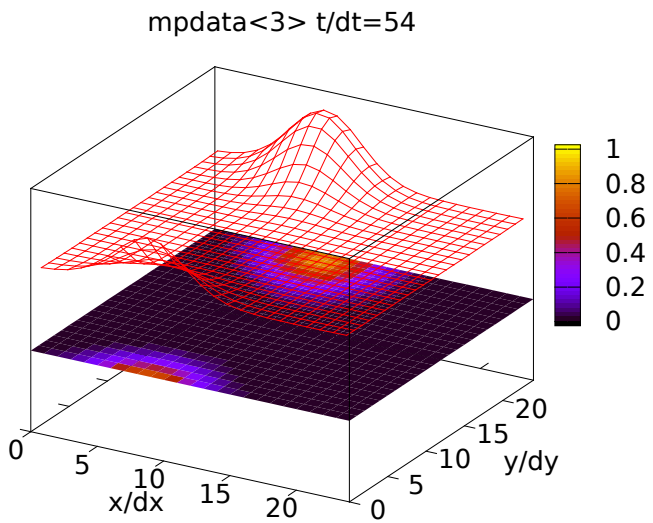
UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures







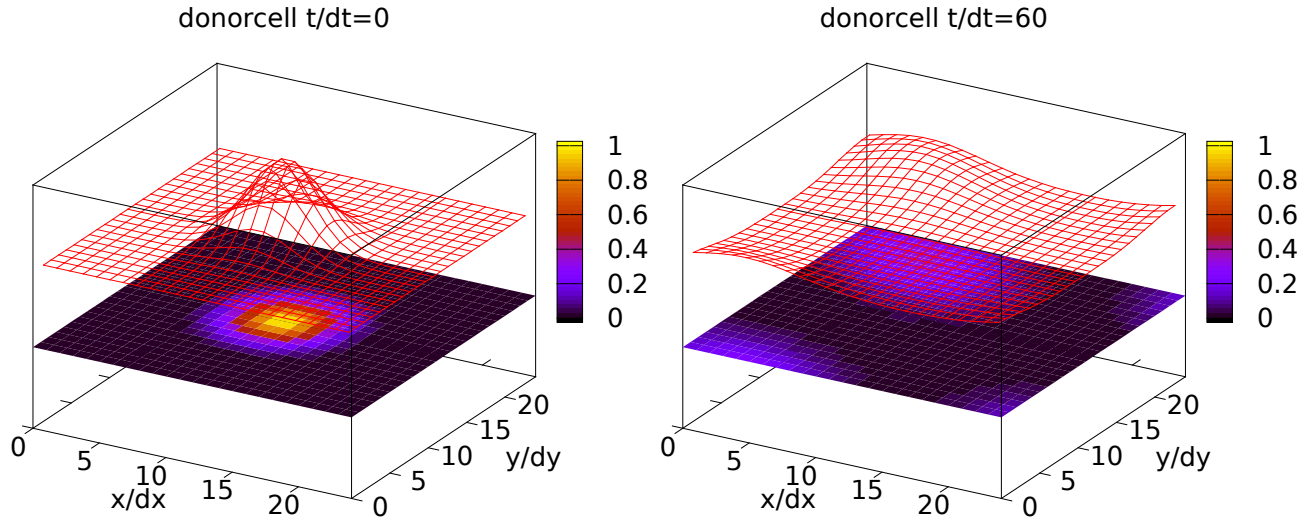


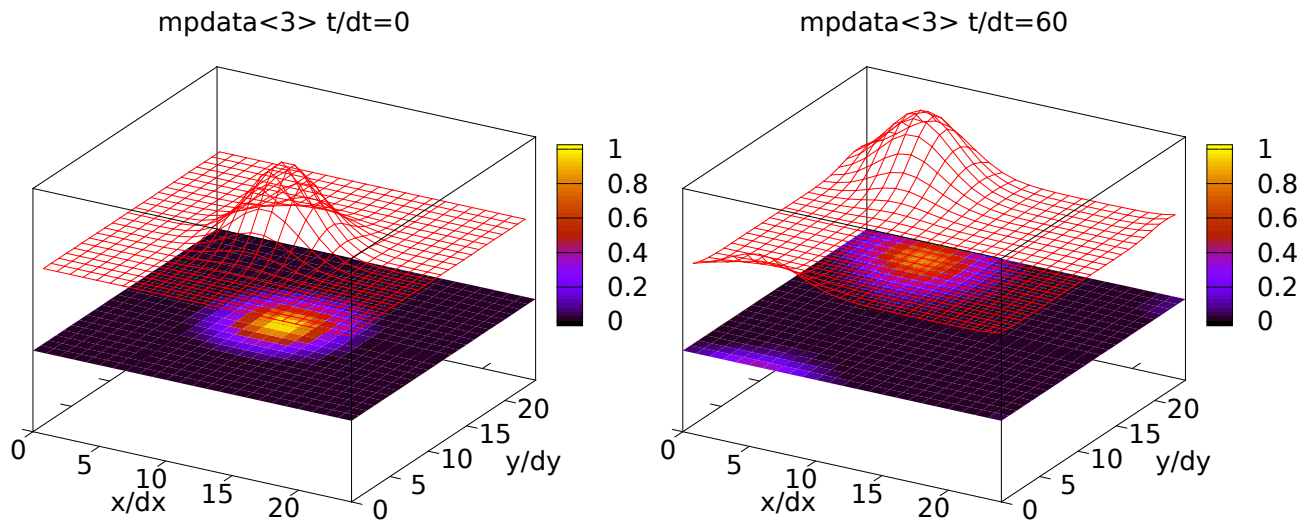
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures



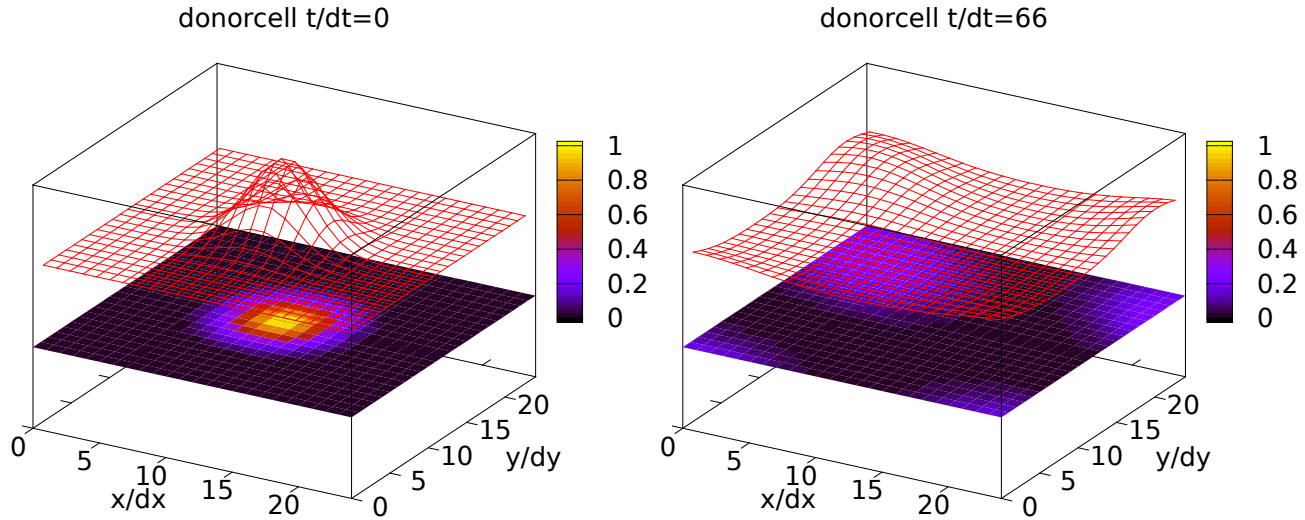


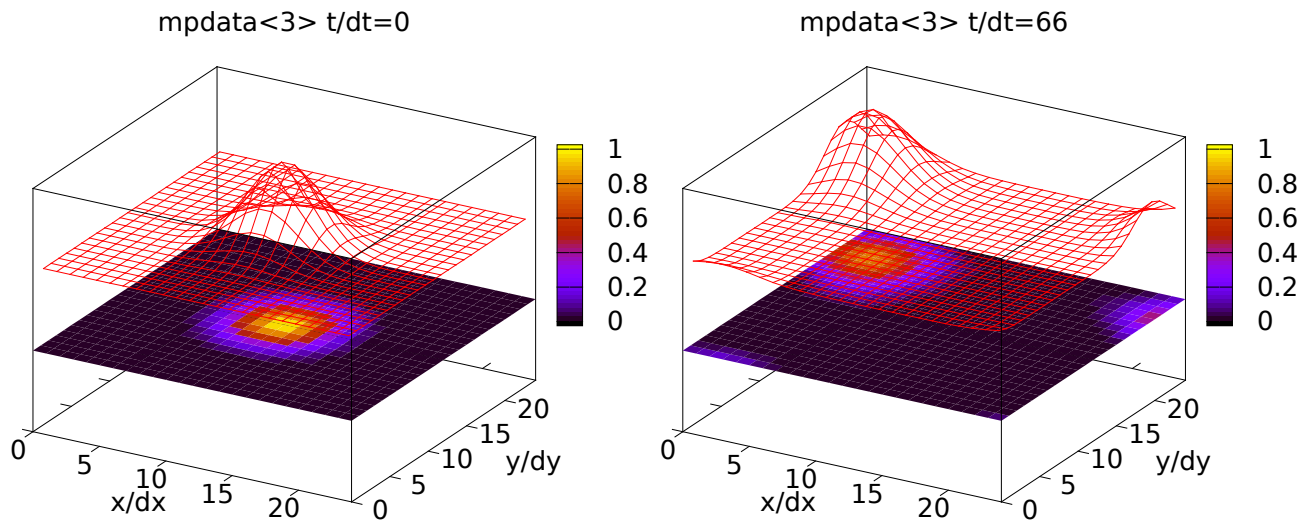
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures



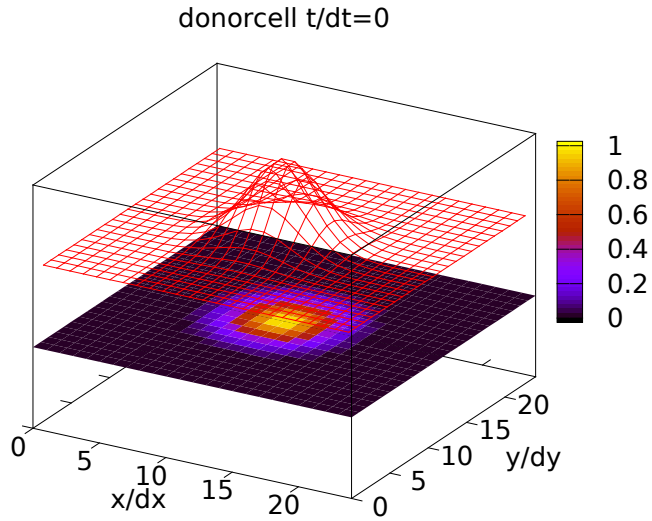


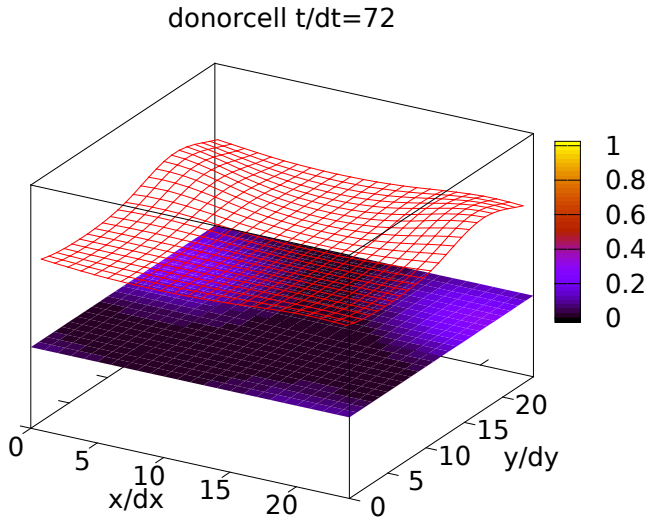
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

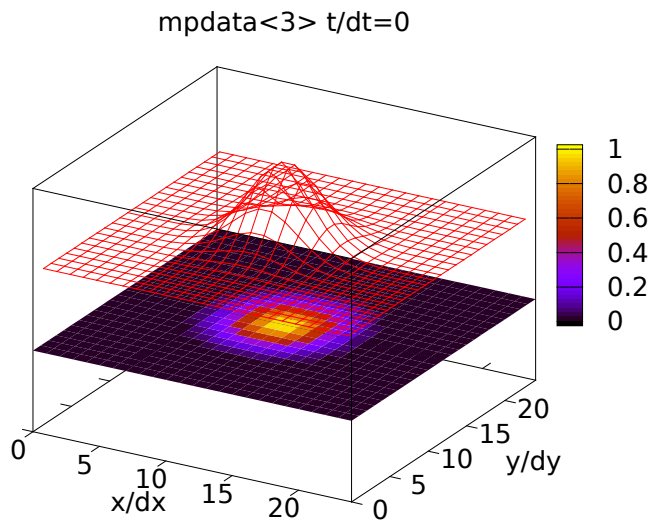
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

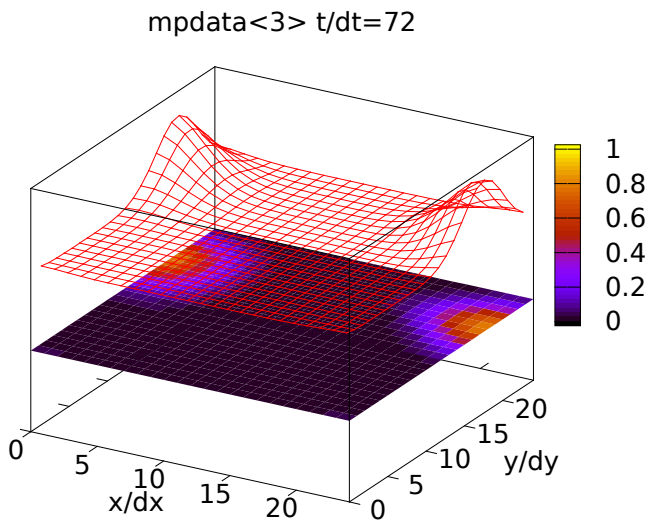
UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures







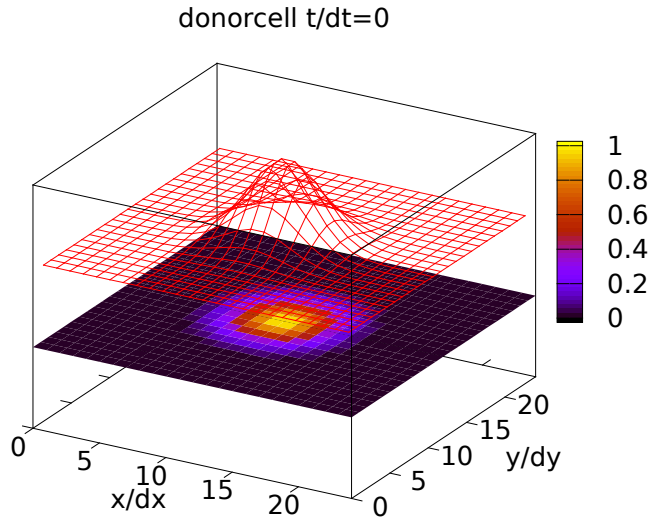


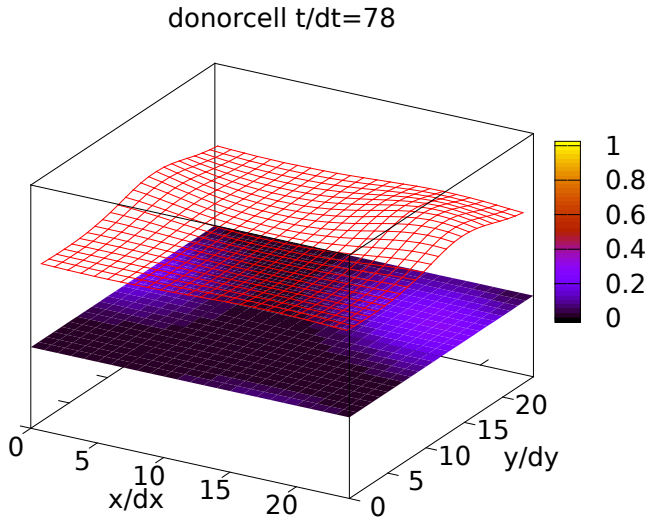
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

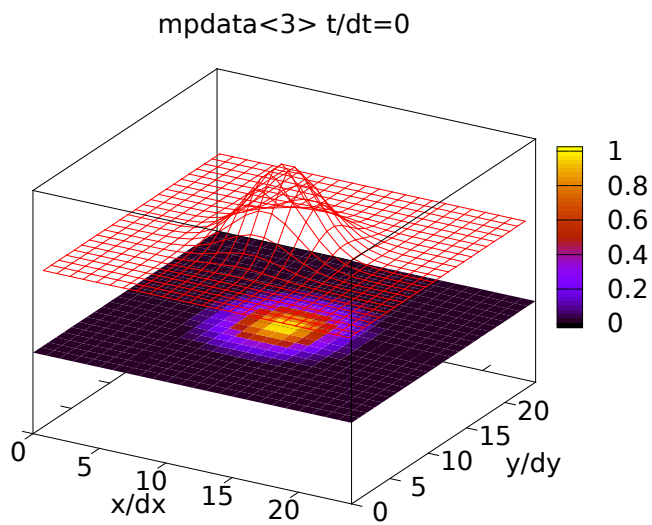
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

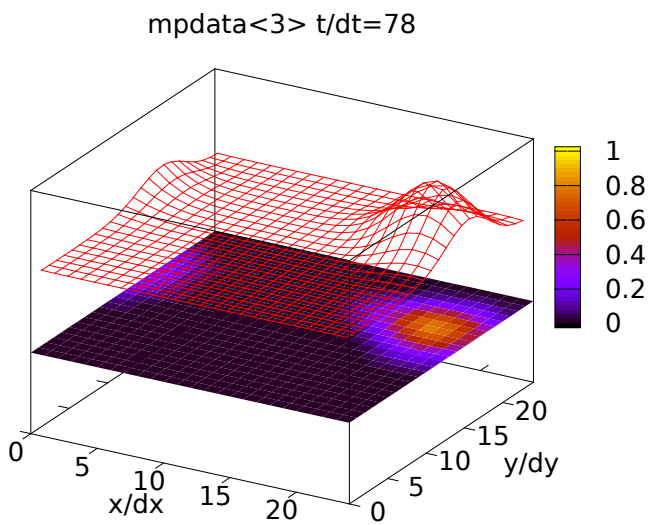
UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures







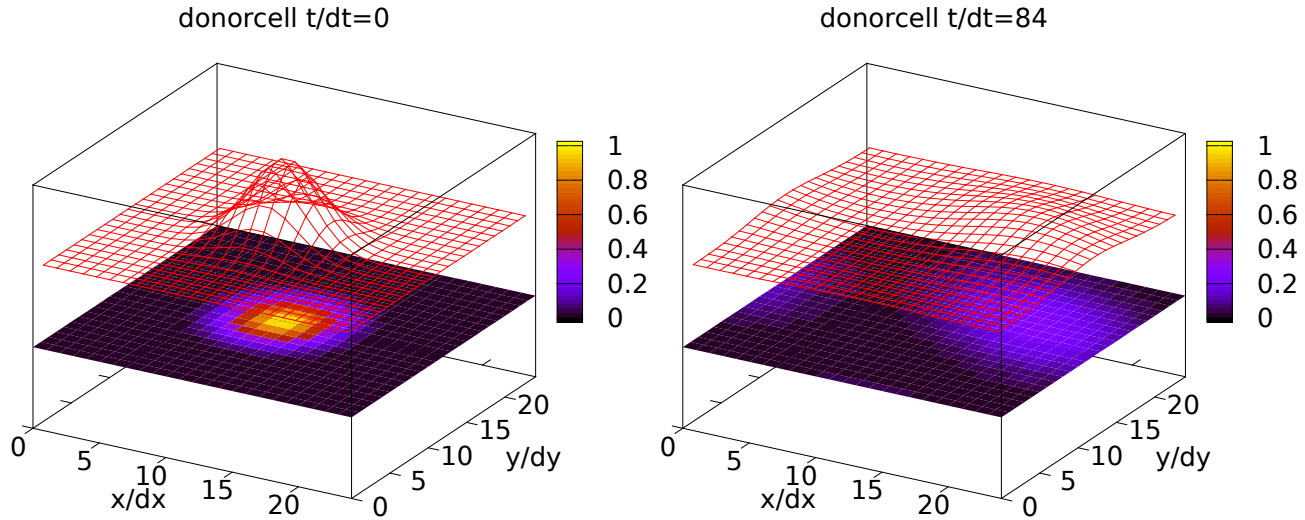


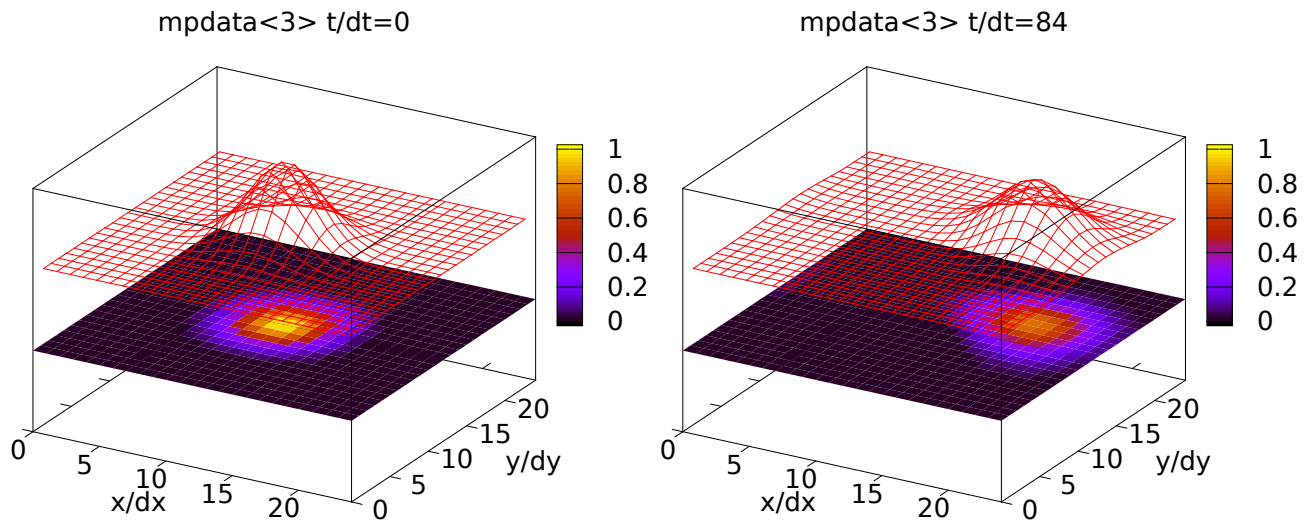
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures



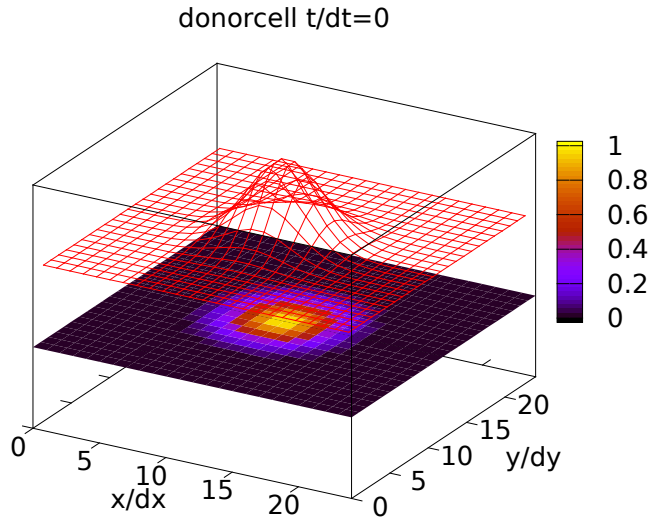


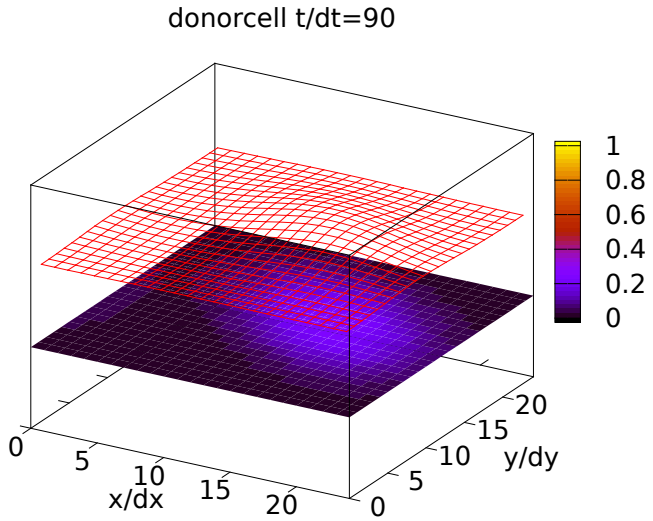
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

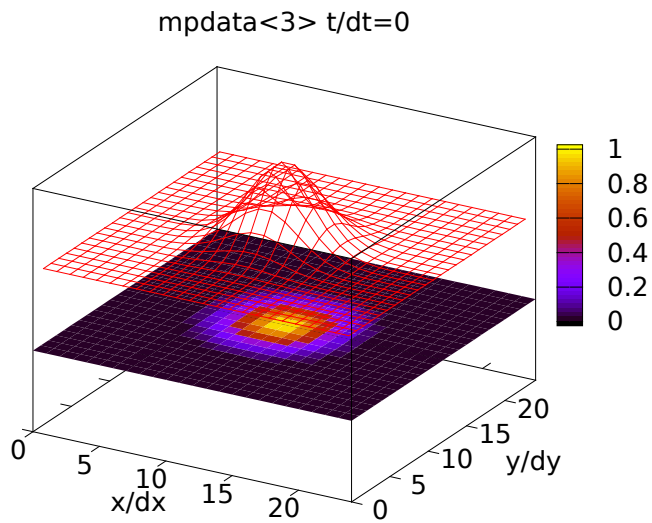
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

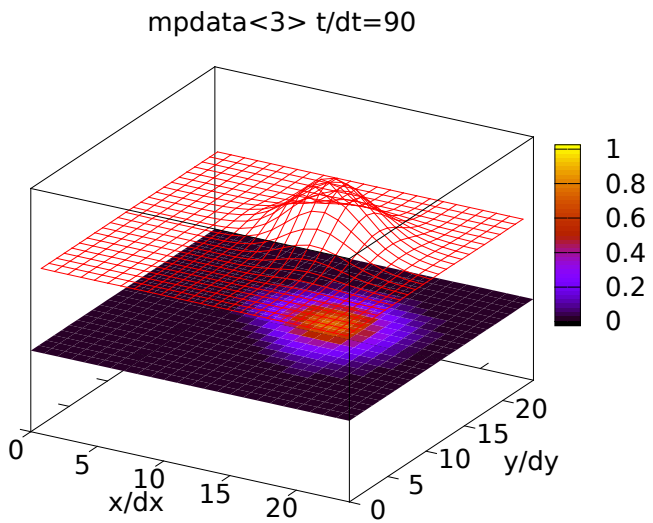
UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures





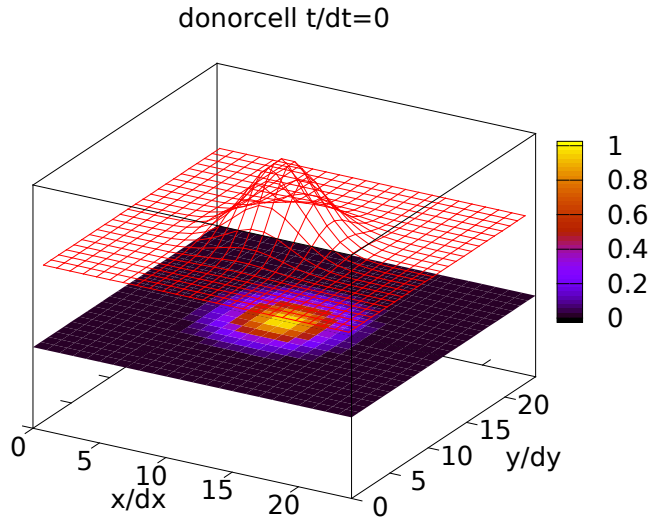


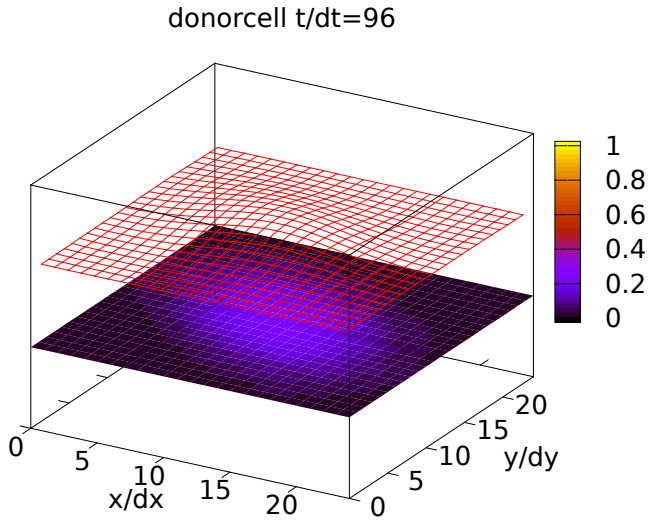


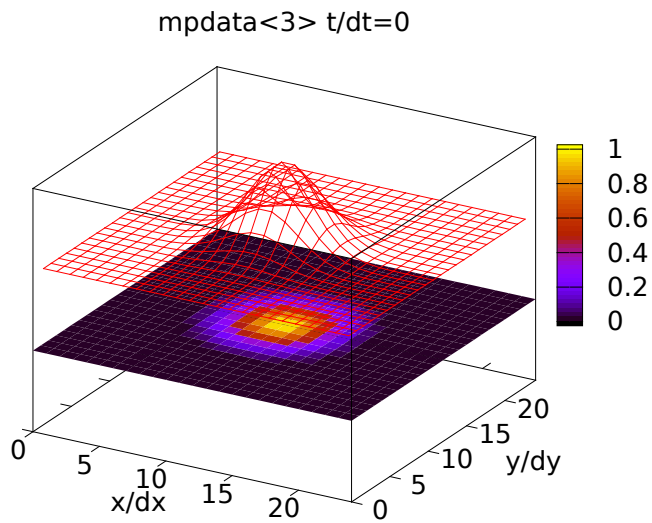
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

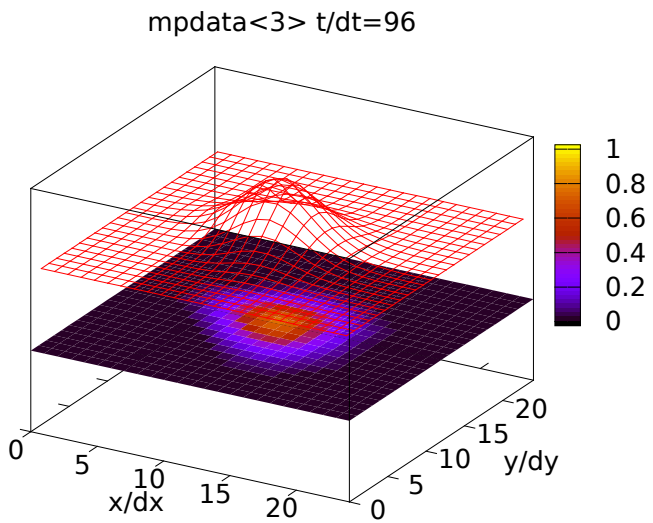
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

MPDATA (Smolarkiewicz 1984) in pictures





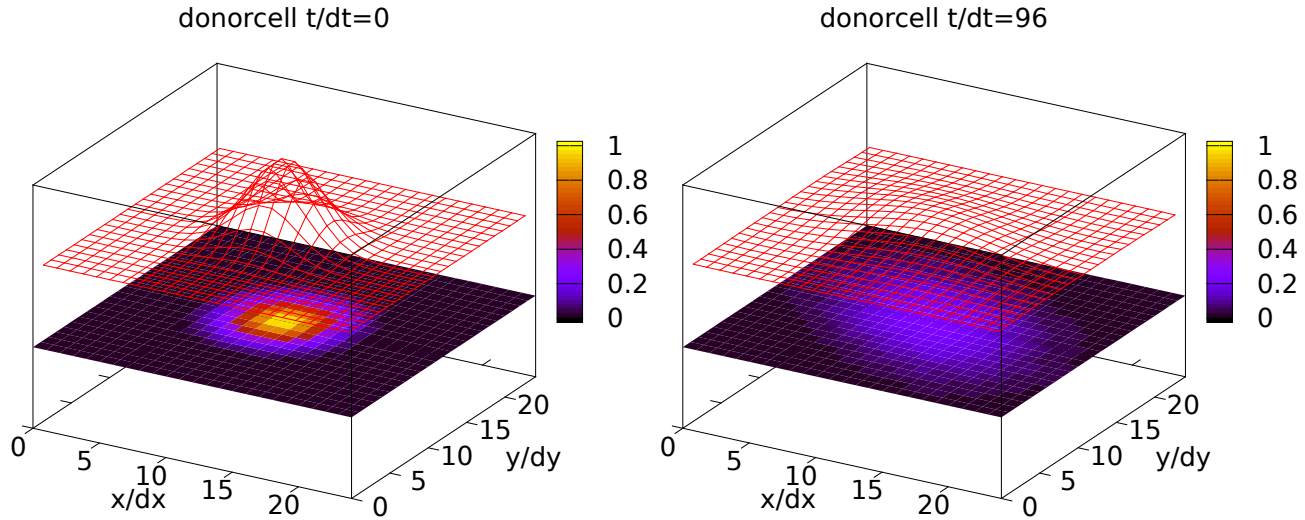


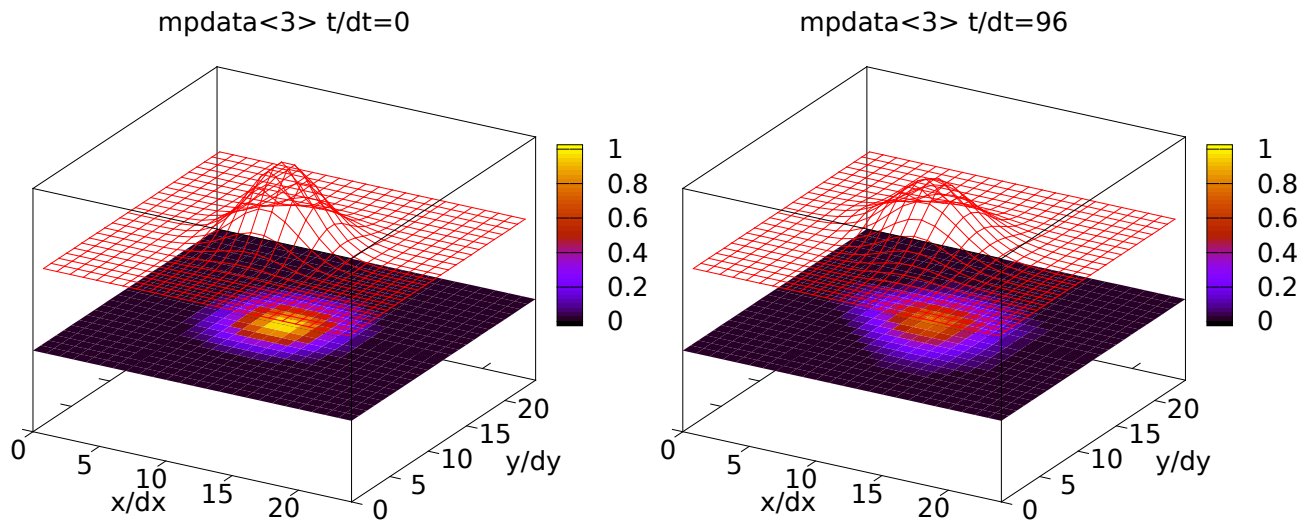


- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

MPDATA (Smolarkiewicz 1984) in pictures



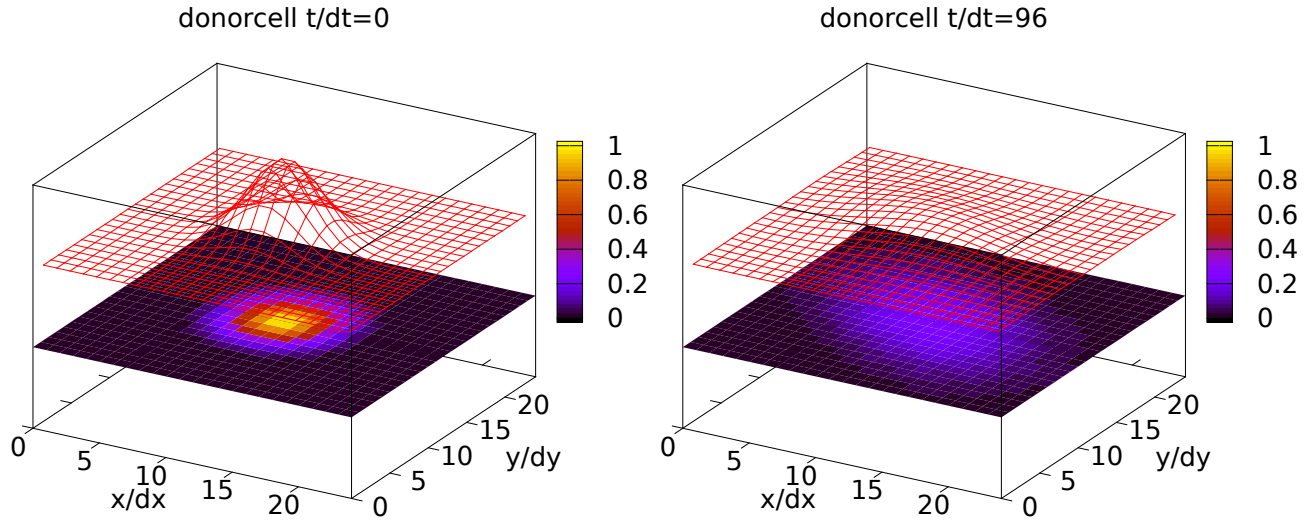


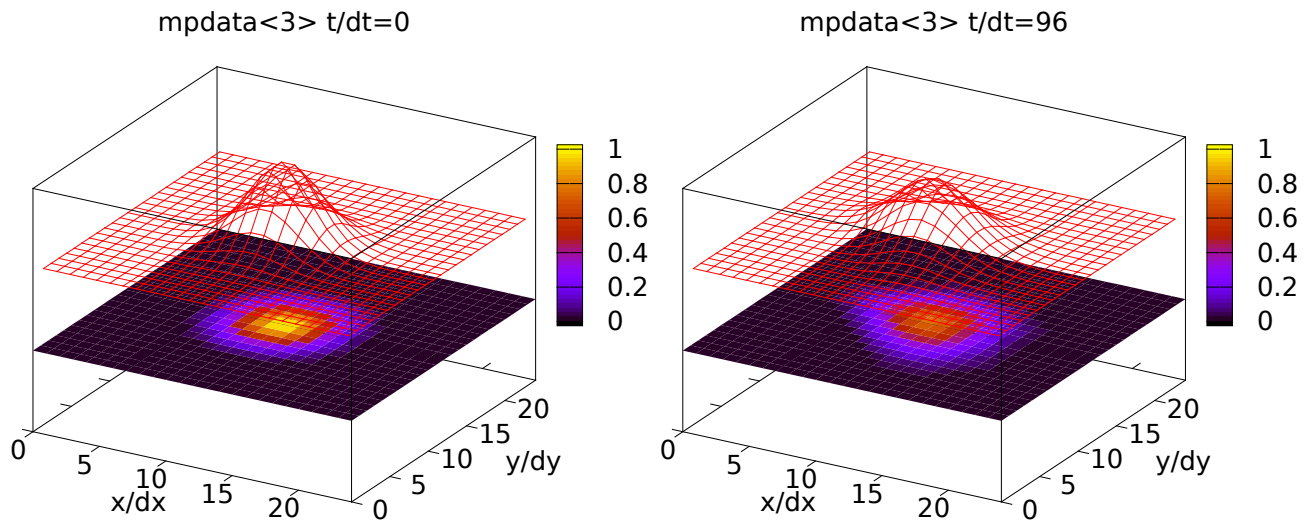
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures



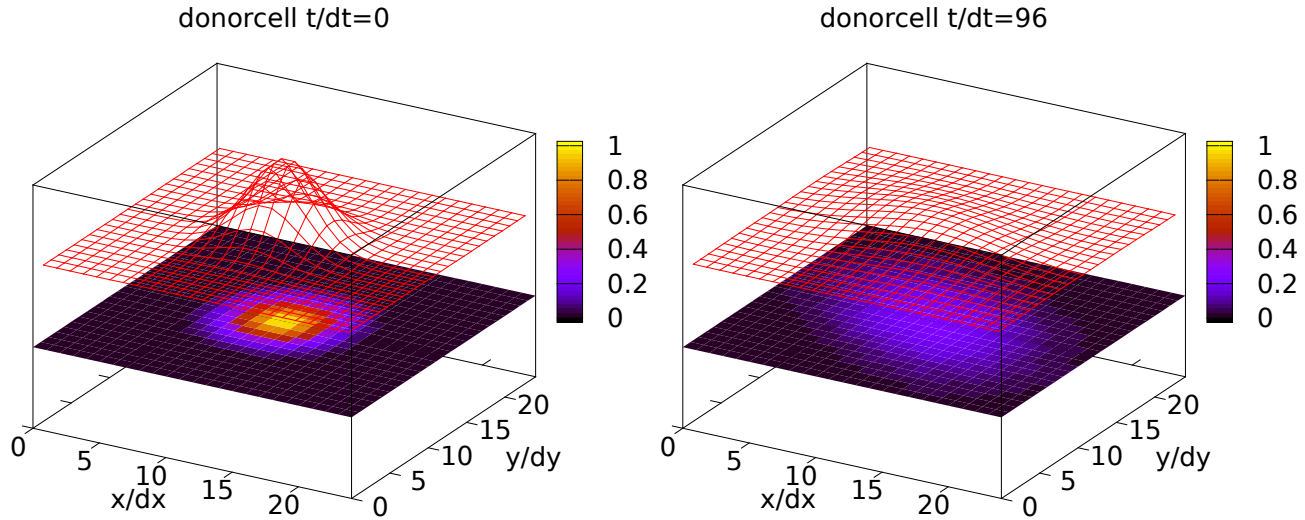


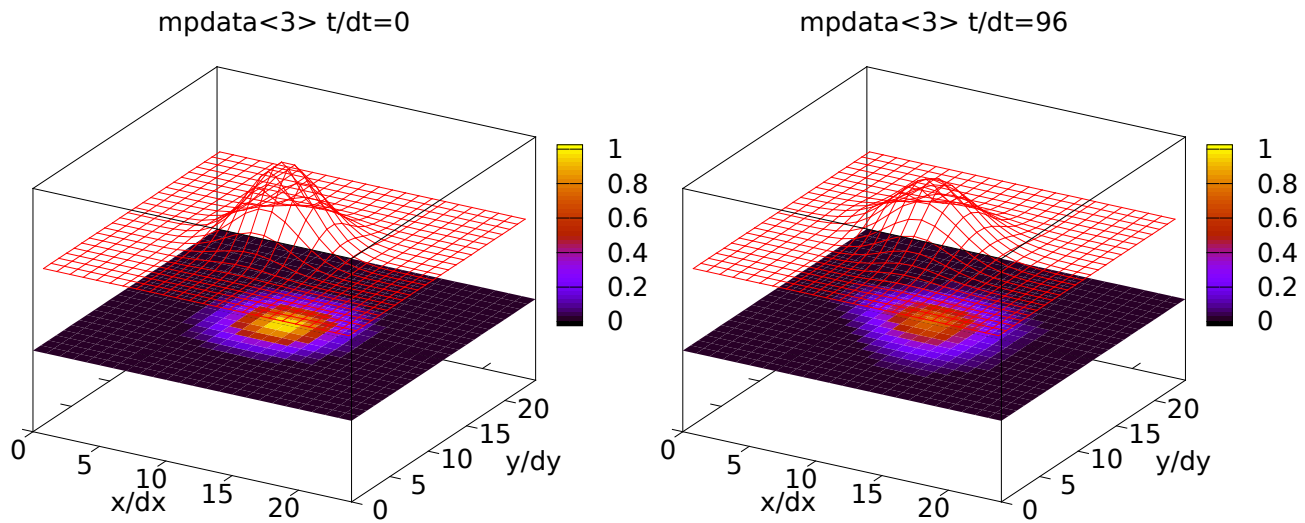
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures



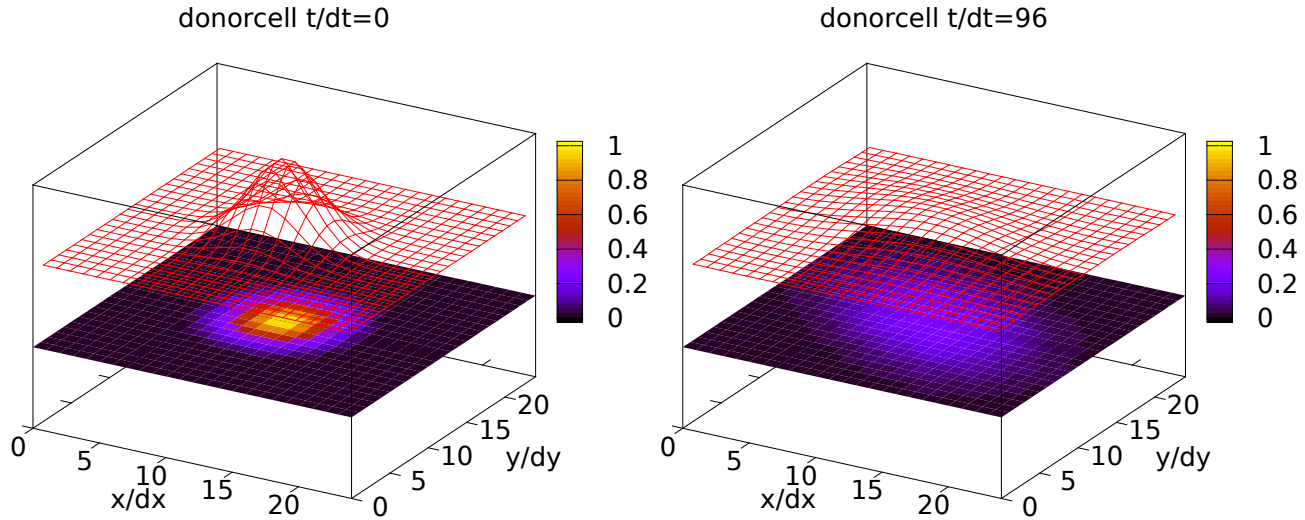


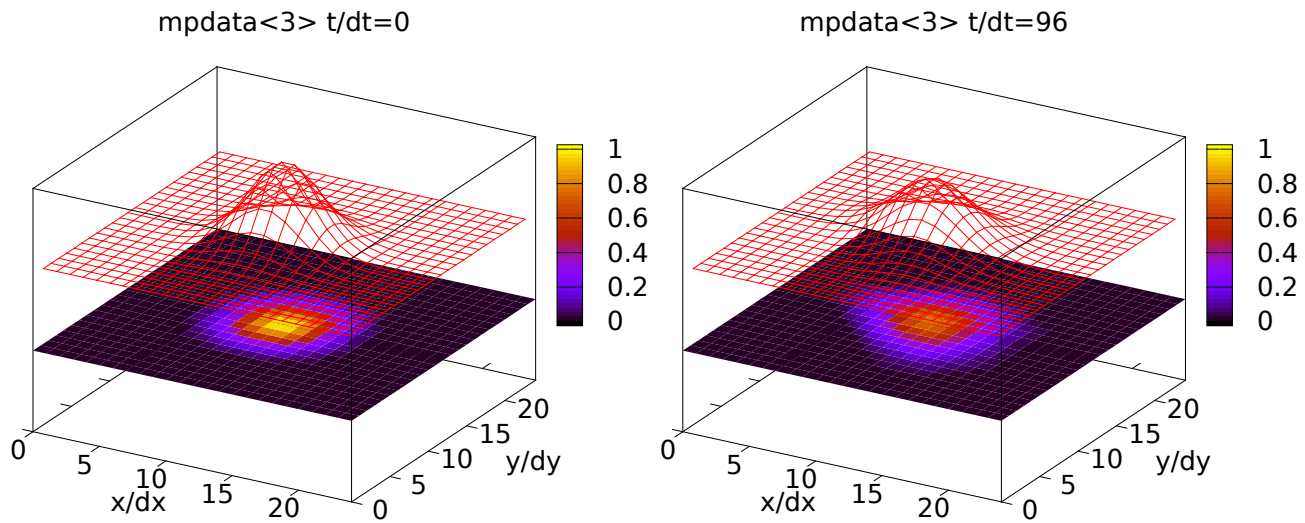
- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures



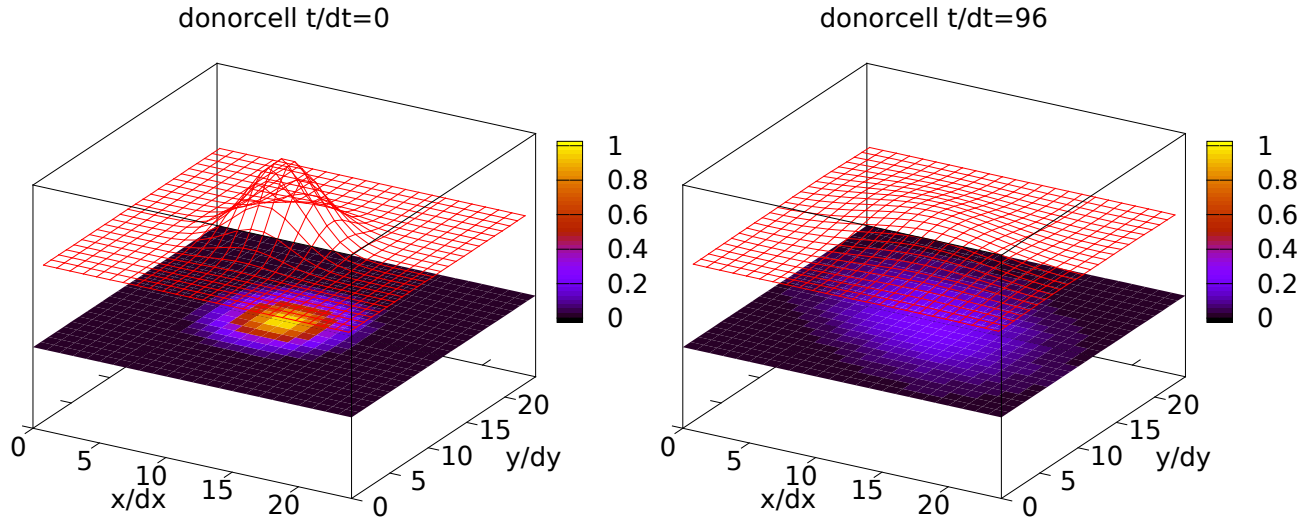


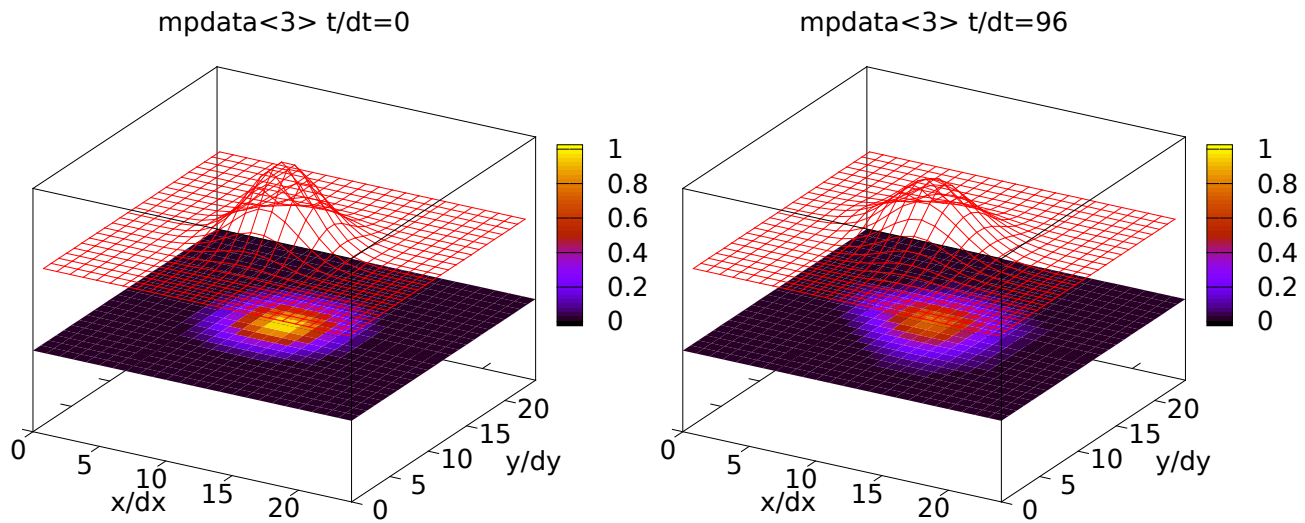
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures



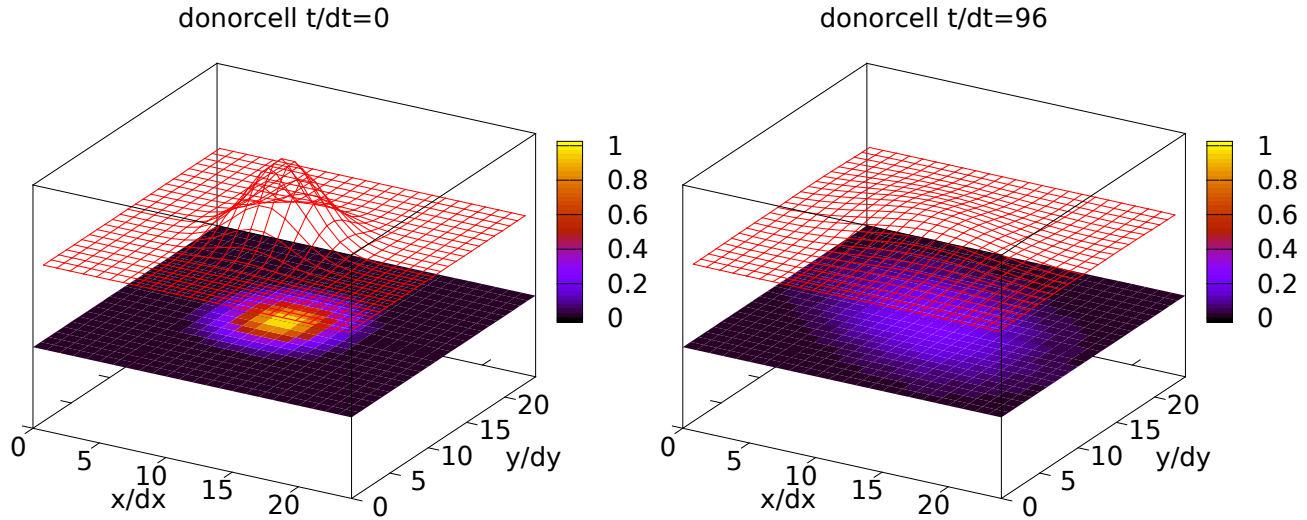


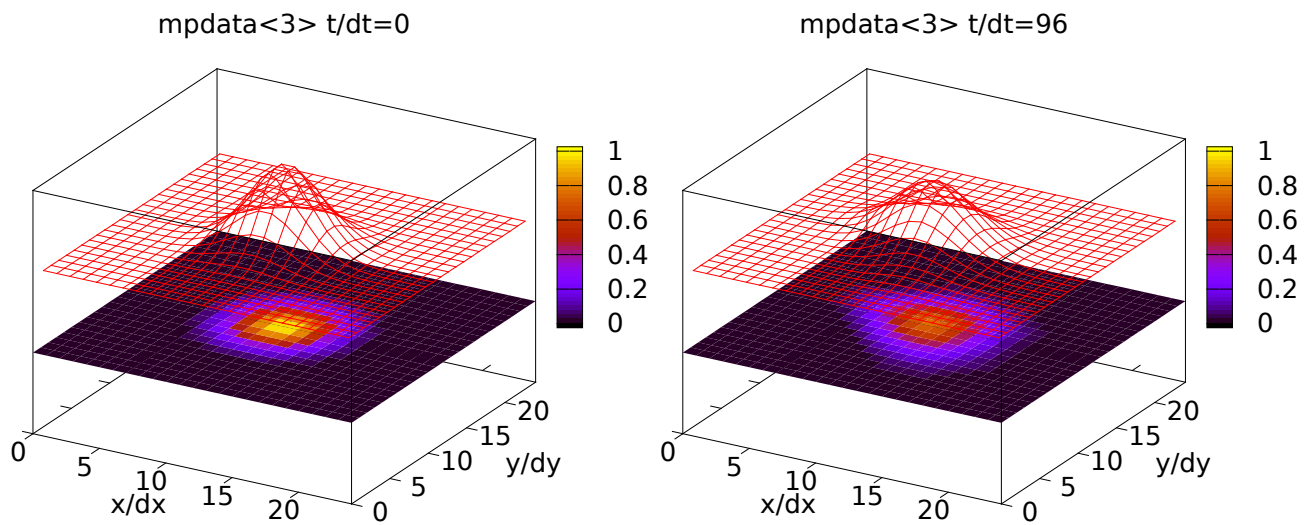
https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

UNIVERSITAS VARSOVIENSIS

MPDATA (Smolarkiewicz 1984) in pictures

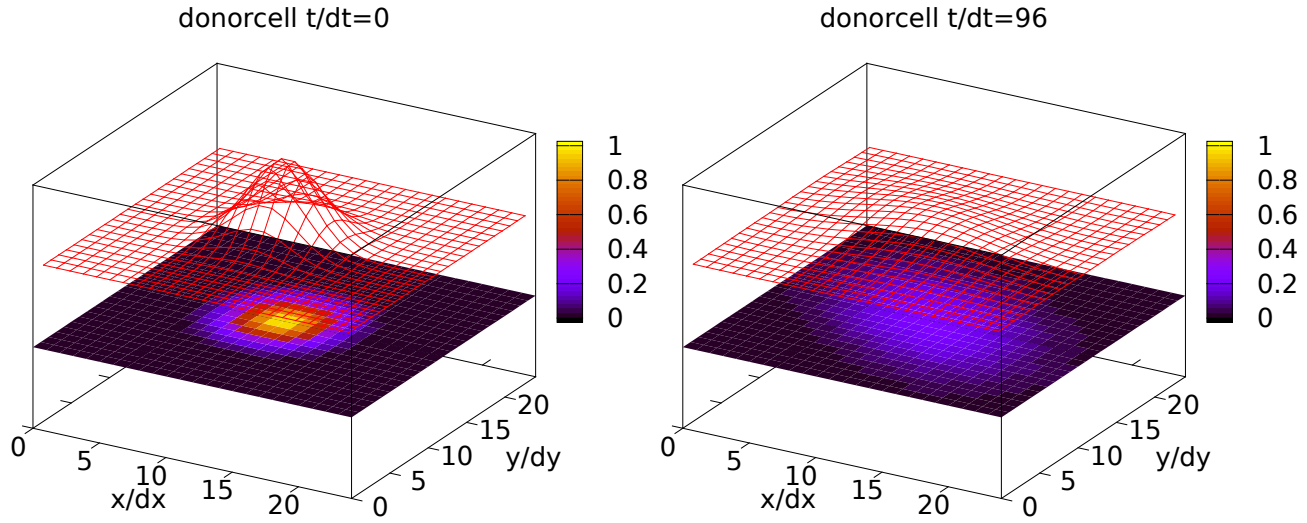


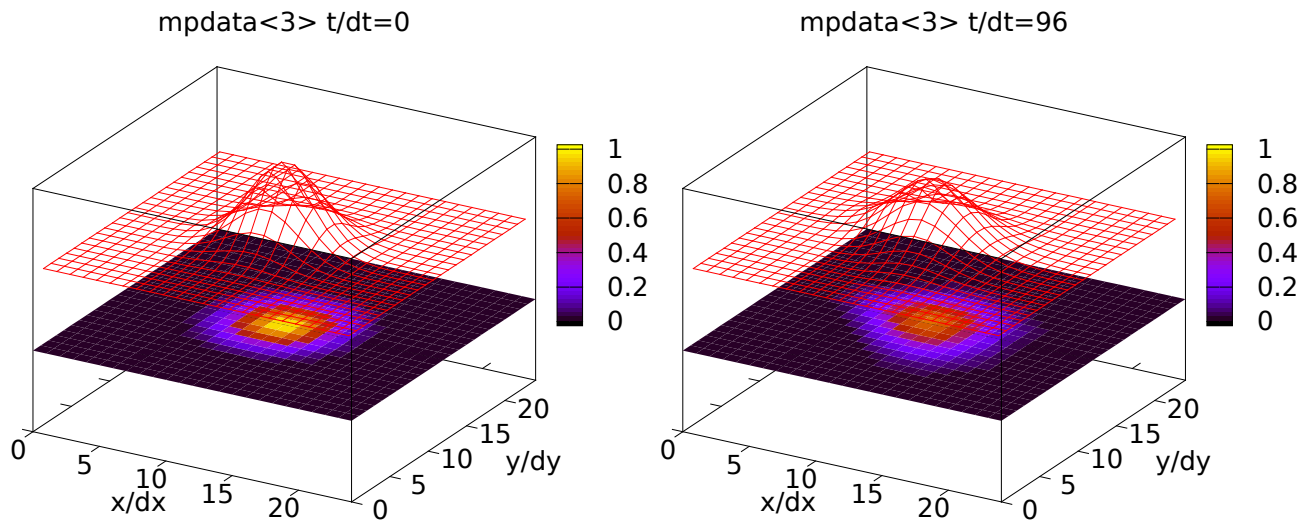


https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

MPDATA (Smolarkiewicz 1984) in pictures





- 2^{nd}-order accurate in space and time
- non-oscillatory (optionally)
- multi-dimensional & sign-preserving by design
- readily parallelisable
- proven applicability in weather-, climate- and ocean simulations
- NCAR-developed
- no free & open-source nor OOP impl. to date

https://github.com/slayoo/mpdata/blob/master/paper/cpp/plot.cpp

MPDATA (Smolarkiewicz 1984) in formulæ

- advection equation: $\partial_t \psi = -\nabla \cdot (\vec{v}\psi)$
- two time-level forward-in-time scheme:

$$\psi^{[n+1]} = \psi^{[n]} - \sum_{\mathrm{d}} Adv(\psi^{[n]}, \vec{C})$$

$$\vec{C} = \vec{v} \cdot \frac{\Delta t}{\Delta x}$$

- donor-cell formula on an Arakawa-C grid:

$$\psi^{[n+1]}_{[i,j]} = \psi^{[n]}_{[i,j]} - \sum_{d=0}^{N-1} \left(F\left[\psi^{[n]}_{[i,j]}, \psi^{[n]}_{[i,j]+\pi^d_{1,0}}, C^{[d]}_{[i,j]+\pi^d_{1/2,0}}\right] - F\left[\psi^{[n]}_{[i,j]+\pi^d_{-1,0}}, \psi^{[n]}_{[i,j]}, C^{[d]}_{[i,j]+\pi^d_{-1/2,0}}\right]\right)$$

$$F(\psi_L, \psi_R, C) = \max(C, 0) \cdot \psi_L + \min(C, 0) \cdot \psi_R = \frac{C + |C|}{2} \cdot \psi_L + \frac{C - |C|}{2} \cdot \psi_R$$

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d

MPDATA (Smolarkiewicz 1984) in formulæ

- advection equation: $\partial_t \psi = -\nabla \cdot (\vec{v}\psi)$
- two time-level forward-in-time scheme:

$$\psi^{[n+1]} = \psi^{[n]} - \sum_{\mathrm{d}} Adv(\psi^{[n]}, \vec{C})$$

$$\vec{C} = \vec{v} \cdot \frac{\Delta t}{\Delta x}$$

- donor-cell formula on an Arakawa-C grid:

$$\psi^{[n+1]}_{[i,j]} = \psi^{[n]}_{[i,j]} - \sum_{d=0}^{N-1} \left(F\left[\psi^{[n]}_{[i,j]}, \psi^{[n]}_{[i,j]+\pi^d_{1,0}}, C^{[d]}_{[i,j]+\pi^d_{1/2,0}}\right] - F\left[\psi^{[n]}_{[i,j]+\pi^d_{-1,0}}, \psi^{[n]}_{[i,j]}, C^{[d]}_{[i,j]+\pi^d_{-1/2,0}}\right] \right)$$

$$F(\psi_L, \psi_R, C) = \max(C, 0) \cdot \psi_L + \min(C, 0) \cdot \psi_R = \frac{C + |C|}{2} \cdot \psi_L + \frac{C - |C|}{2} \cdot \psi_R$$

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d

MPDATA (Smolarkiewicz 1984) in formulæ

- MPDATA: corrective donor-cell steps using C':

$$C'^{[d]}_{[i,j]+\pi^d_{1/2,0}} = \left| C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \right| \cdot \left[1 - \left| C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \right| \right] \cdot A^{[d]}_{[i,j]}(\psi) - \sum_{q=0, q\neq d}^{N} C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \cdot \overline{C}^{[q]}_{[i,j]+\pi^d_{1/2,0}} \cdot B^{[d]}_{[i,j]}(\psi)$$

$$\overline{C}^{[q]}_{[i,j]+\pi^d_{1/2,0}} = \frac{1}{4} \cdot \left(C^{[q]}_{[i,j]+\pi^d_{1,1/2}} + C^{[q]}_{[i,j]+\pi^d_{0,1/2}} + C^{[q]}_{[i,j]+\pi^d_{1,-1/2}} + C^{[q]}_{[i,j]+\pi^d_{0,-1/2}} \right)$$

$$A^{[d]}_{[i,j]} = \frac{\psi_{[i,j]+\pi^d_{1,0}} - \psi_{[i,j]}}{\psi_{[i,j]+\pi^d_{1,0}} + \psi_{[i,j]}}$$

$$B^{[d]}_{[i,j]} = \frac{1}{2} \frac{\psi_{[i,j]+\pi^d_{1,1}} + \psi_{[i,j]+\pi^d_{0,1}} - \psi_{[i,j]+\pi^d_{1,-1}} - \psi_{[i,j]+\pi^d_{0,-1}}}{\psi_{[i,j]+\pi^d_{1,1}} + \psi_{[i,j]+\pi^d_{0,1}} + \psi_{[i,j]+\pi^d_{1,-1}} + \psi_{[i,j]+\pi^d_{0,-1}}}$$

- altogether: 0.06 kLOC in LaTeX

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d

MPDATA (Smolarkiewicz 1984) in formulæ

- MPDATA: corrective donor-cell steps using C':

$$C'^{[d]}_{[i,j]+\pi^d_{1/2,0}} = \left| C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \right| \cdot \left[1 - \left| C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \right| \right] \cdot A^{[d]}_{[i,j]}(\psi) - \sum_{q=0, q\neq d}^{N} C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \cdot \overline{C}^{[q]}_{[i,j]+\pi^d_{1/2,0}} \cdot B^{[d]}_{[i,j]}(\psi)$$

$$\overline{C}^{[q]}_{[i,j]+\pi^d_{1/2,0}} = \frac{1}{4} \cdot \left(C^{[q]}_{[i,j]+\pi^d_{1,1/2}} + C^{[q]}_{[i,j]+\pi^d_{0,1/2}} + C^{[q]}_{[i,j]+\pi^d_{1,-1/2}} + C^{[q]}_{[i,j]+\pi^d_{0,-1/2}} \right)$$

$$A^{[d]}_{[i,j]} = \frac{\psi_{[i,j]+\pi^d_{1,0}} - \psi_{[i,j]}}{\psi_{[i,j]+\pi^d_{1,0}} + \psi_{[i,j]}}$$

$$B^{[d]}_{[i,j]} = \frac{1}{2} \frac{\psi_{[i,j]+\pi^d_{1,1}} + \psi_{[i,j]+\pi^d_{0,1}} - \psi_{[i,j]+\pi^d_{1,-1}} - \psi_{[i,j]+\pi^d_{0,-1}}}{\psi_{[i,j]+\pi^d_{1,1}} + \psi_{[i,j]+\pi^d_{0,1}} + \psi_{[i,j]+\pi^d_{1,-1}} + \psi_{[i,j]+\pi^d_{0,-1}}}$$

- **altogether: 0.06 kLOC in LaTeX**

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d

MPDATA (Smolarkiewicz 1984) in formulæ

- "traditional" numerics:
  - floating-point arrays: ψ, $C^{[0]}$, $C^{[1]}$
  - integers: n, d, i, j
  - statements
  - ...
  - #LOC $\gg$ #LOC in LaTeX, human-readable?

- OO numerics:
  - instance of an array-of-floats class: ψ
  - instance of a vector-of-arrays class: C
  - instance of an array-index class: i, j
  - ...
  - instance of an array-valued-expression class: $\psi_{[i+1,j]} + \psi_{[i-1,j]}$
  - ...
  - how could it make one's life easier?
  - are there any solutions applicable in HPC?

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d

MPDATA (Smolarkiewicz 1984) in formulæ

- "traditional" numerics:
  - floating-point arrays: ψ, $C^{[0]}$, $C^{[1]}$
  - integers: n, d, i, j
  - statements
  - ...
  - #LOC $\gg$ #LOC in LaTeX, human-readable?

- OO numerics:
  - instance of an array-of-floats class: ψ
  - instance of a vector-of-arrays class: C
  - instance of an array-index class: i, j
  - ...
  - instance of an array-valued-expression class: $\psi_{[i+1,j]} + \psi_{[i-1,j]}$
  - ...
  - how could it make one's life easier?
  - are there any solutions applicable in HPC?

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d

MPDATA (Smolarkiewicz 1984) in formulæ

- "traditional" numerics:
  - floating-point arrays: ψ, $C^{[0]}$, $C^{[1]}$
  - integers: n, d, i, j
  - statements
  - ...
  - #LOC $\gg$ #LOC in LaTeX, human-readable?

- OO numerics:
  - instance of an array-of-floats class: ψ
  - instance of a vector-of-arrays class: C
  - instance of an array-index class: i, j
  - ...
  - instance of an array-valued-expression class: $\psi_{[i+1,j]} + \psi_{[i-1,j]}$
  - ...
  - how could it make one's life easier?
  - are there any solutions applicable in HPC?

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^{d}_{a,b}$ - permutation of (a,b) of order d

MPDATA (Smolarkiewicz 1984) in formulæ

- "traditional" numerics:
  - floating-point arrays: ψ, $C^{[0]}$, $C^{[1]}$
  - integers: n, d, i, j
  - statements
  - ...
  - #LOC $\gg$ #LOC in LaTeX, human-readable?
- OO numerics:
  - instance of an array-of-floats class: ψ
  - instance of a vector-of-arrays class: C
  - instance of an array-index class: i, j
  - ...
  - instance of an array-valued-expression class: $\psi_{[i+1,j]} + \psi_{[i-1,j]}$
  - ...
  - how could it make one's life easier?
  - are there any solutions applicable in HPC?

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d

MPDATA (Smolarkiewicz 1984) in formulæ

- "traditional" numerics:
  - floating-point arrays: ψ, $C^{[0]}$, $C^{[1]}$
  - integers: n, d, i, j
  - statements
  - ...
  - #LOC $\gg$ #LOC in LaTeX, human-readable?
- OO numerics:
  - instance of an array-of-floats class: ψ
  - instance of a vector-of-arrays class: C
  - instance of an array-index class: i, j
  - ...
  - instance of an array-valued-expression class: $\psi_{[i+1,j]} + \psi_{[i-1,j]}$
  - ...
  - how could it make one's life easier?
  - are there any solutions applicable in HPC?

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d

MPDATA (Smolarkiewicz 1984) in formulæ

- "traditional" numerics:
  - floating-point arrays: ψ, $C^{[0]}$, $C^{[1]}$
  - integers: n, d, i, j
  - statements
  - ...
  - #LOC $\gg$ #LOC in LaTeX, human-readable?
- OO numerics:
  - instance of an array-of-floats class: ψ
  - instance of a vector-of-arrays class: C
  - instance of an array-index class: i, j
  - ...
  - instance of an array-valued-expression class: $\psi_{[i+1,j]} + \psi_{[i-1,j]}$
  - ...
  - how could it make one's life easier?
  - are there any solutions applicable in HPC?

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d

MPDATA (Smolarkiewicz 1984) in formulæ

- "traditional" numerics:
  - floating-point arrays: ψ, $C^{[0]}$, $C^{[1]}$
  - integers: n, d, i, j
  - statements
  - ...
  - #LOC $\gg$ #LOC in LaTeX, human-readable?
- OO numerics:
  - instance of an array-of-floats class: ψ
  - instance of a vector-of-arrays class: C
  - instance of an array-index class: i, j
  - ...
  - instance of an array-valued-expression class: $\psi_{[i+1,j]} + \psi_{[i-1,j]}$
  - ...
  - how could it make one's life easier?
  - are there any solutions applicable in HPC?

symbols:

ψ - conservative dependent variable (scalar field)

$\vec{C}$ - Courant number (vector field)

n - time level

d - dimension

i, j - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d

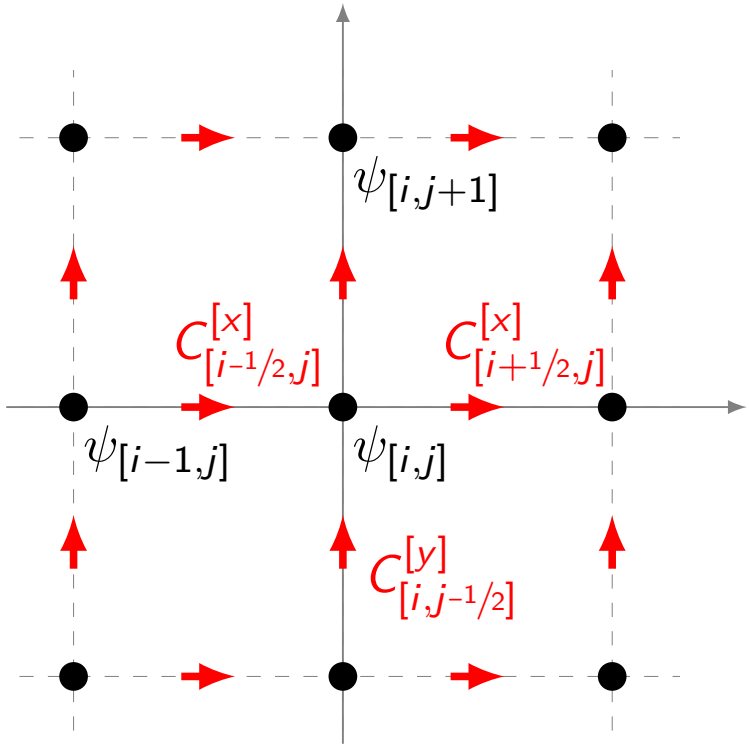
MPDATA in C++, Python and Fortran: prerequisites

OOP language/library set-ups for fast number crunching:
(in context of structured meshes, stencil-type algorithms)

- C++ & Blitz++
- Python & NumPy
- Fortran 200x
- ...[1]

[1] suggestions welcome on what else to try out

example (1/3): Arakawa-C grid fractional indices



example (1/3): Arakawa-C grid fractional indices

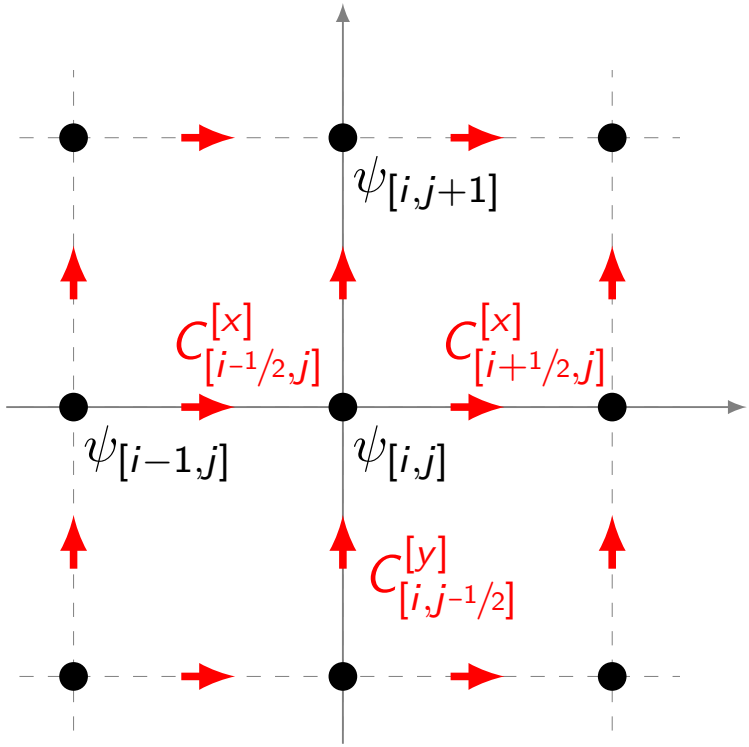
C++ / Blitz++:

(C++)

```
#include <blitz/array.h>
using arr_t = blitz::Array<real_t, 2>;
using rng_t = blitz::Range;
using idx_t = blitz::RectDomain<2>;
```

(C++)

```
struct hlf_t {} h;

inline rng_t operator+(const rng_t &i, const hlf_t &)
{
  return i;
}

inline rng_t operator-(const rng_t &i, const hlf_t &)
{
  return i-1;
}
```



example (1/3): Arakawa-C grid fractional indices

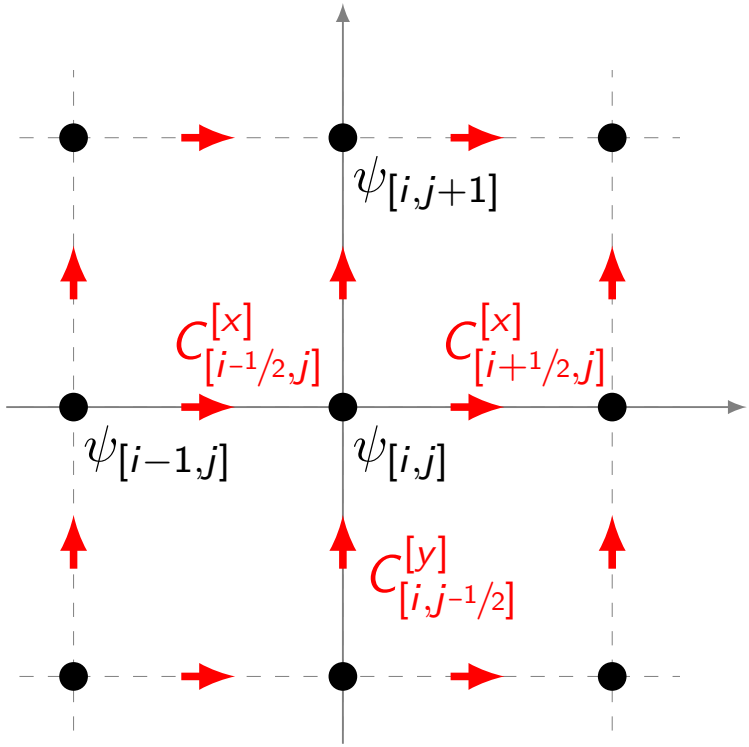
Python / NumPy:

(Python)

```
class shift():
  def __init__(self, plus, mnus):
    self.plus = plus
    self.mnus = mnus
  def __radd__(self, arg):
    return type(arg)(
      arg.start + self.plus,
      arg.stop  + self.plus
    )
  def __rsub__(self, arg):
    return type(arg)(
      arg.start - self.mnus,
      arg.stop  - self.mnus
    )
```

(Python)

```
one = shift(1,1)
hlf = shift(0,1)
```

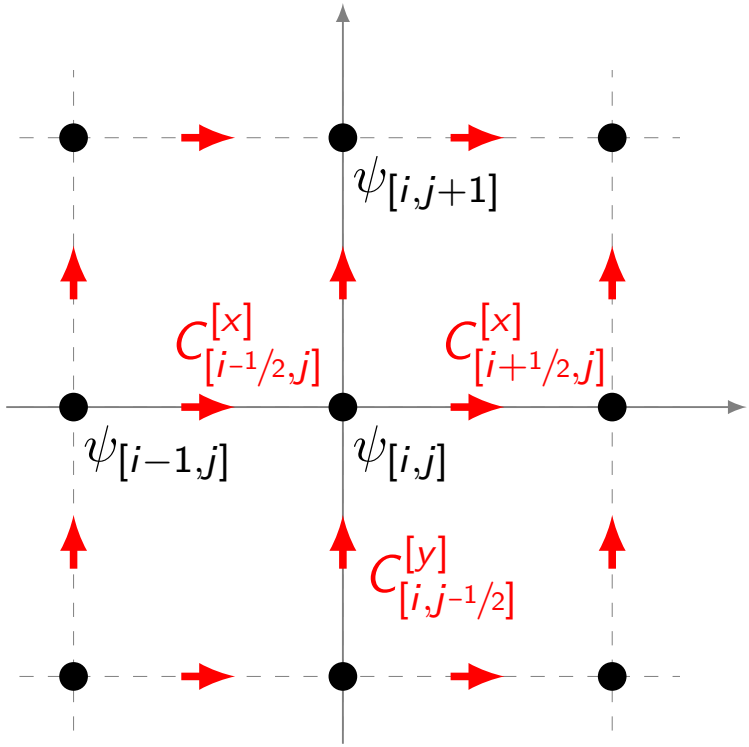


example (1/3): Arakawa-C grid fractional indices

Fortran:

(Fortran)

```
module arakawa_c_m
  implicit none

  type :: half_t
  end type

  type(half_t) :: h

  interface operator (+)
    module procedure ph
  end interface

  interface operator (-)
    module procedure mh
  end interface

  contains

  elemental function ph(i, h) result (return)
    integer, intent(in) :: i
    type(half_t), intent(in) :: h
    integer :: return
    return = i
  end function

  elemental function mh(i, h) result (return)
    integer, intent(in) :: i
    type(half_t), intent(in) :: h
    integer :: return
    return = i - 1
  end function
end module
```



example (2/3): dimensional indirection

$$\psi^{[n+1]}_{[i,j]} = \psi^{[n]}_{[i,j]} - \sum_{d=0}^{N-1} \left(F\left[\psi^{[n]}_{[i,j]}, \psi^{[n]}_{[i,j]+\pi^d_{1,0}}, C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \right] - F\left[\psi^{[n]}_{[i,j]+\pi^d_{-1,0}}, \psi^{[n]}_{[i,j]}, C^{[d]}_{[i,j]+\pi^d_{-1/2,0}} \right] \right)$$

C++ / Blitz++:

(C++)

```
template<int d>
inline auto donorcell(
  const arr_t &psi, const arr_t &C,
  const rng_t &i, const rng_t &j
) return_macro(
  F(
    psi(pi<d>(i,   j)),
    psi(pi<d>(i+1, j)),
      C(pi<d>(i+h, j))
  ) -
  F(
    psi(pi<d>(i-1, j)),
    psi(pi<d>(i,   j)),
      C(pi<d>(i-h, j))
  )
)
```

(C++)

```
void donorcell_op(
  const arrvec_t &psi, const int n,
  const arrvec_t &C,
  const rng_t &i, const rng_t &j
) {
  psi[n+1](i,j) = psi[n](i,j)
    - donorcell<0>(psi[n], C[0], i, j)
    - donorcell<1>(psi[n], C[1], j, i);
}
```

pi() returns an instance of blitz::RectDomain

example (2/3): dimensional indirection

$$\psi^{[n+1]}_{[i,j]} = \psi^{[n]}_{[i,j]} - \sum_{d=0}^{N-1} \left(F\left[\psi^{[n]}_{[i,j]}, \psi^{[n]}_{[i,j]+\pi^d_{1,0}}, C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \right] - F\left[\psi^{[n]}_{[i,j]+\pi^d_{-1,0}}, \psi^{[n]}_{[i,j]}, C^{[d]}_{[i,j]+\pi^d_{-1/2,0}} \right] \right)$$

Python / NumPy:

(Python)

```
def donorcell(d, psi, C, i, j):
  return (
    f(
      psi[pi(d, i,     j)],
      psi[pi(d, i+one, j)],
        C[pi(d, i+hlf, j)]
    ) -
    f(
      psi[pi(d, i-one, j)],
      psi[pi(d, i,     j)],
        C[pi(d, i-hlf, j)]
    )
  )
```

(Python)

```
def donorcell_op(psi, n, C, i, j):
  psi[n+1][i,j] = (psi[n][i,j]
    - donorcell(0, psi[n], C[0], i, j)
    - donorcell(1, psi[n], C[1], j, i)
  )
```

pi() returns a tuple of slices

example (2/3): dimensional indirection

Fortran:

(Fortran)

```
function donorcell(d, psi, C, i, j) result (return)
  integer :: d
  integer, intent(in) :: i(2), j(2)
  real(real_t) :: return(span(d, i, j), span(d, j, i))
  real(real_t), allocatable, intent(in) :: psi(:,:), C(:,:)
  return = (                                                &
    F(                                                      &
      pi(d, psi, i,   j),                                   &
      pi(d, psi, i+1, j),                                   &
      pi(d,   C, i+h, j)                                    &
    ) -                                                     &
    F(                                                      &
      pi(d, psi, i-1, j),                                   &
      pi(d, psi, i,   j),                                   &
      pi(d, C,   i-h, j)                                    &
    )                                                       &
  )
end function
```

(Fortran)

```
subroutine donorcell_op(psi, n, C, i, j)
  class(arrvec_t), allocatable :: psi
  class(arrvec_t), pointer :: C
  integer, intent(in) :: n
  integer, intent(in) :: i(2), j(2)

  real(real_t), pointer :: ptr(:,:)
  ptr => pi(0, psi%at(n+1)%p%a, i, j)
  ptr = pi(0, psi%at(n)%p%a, i, j)                          &
    - donorcell(0, psi%at(n)%p%a, C%at(0)%p%a, i, j)        &
    - donorcell(1, psi%at(n)%p%a, C%at(1)%p%a, j, i)
end subroutine
```

pi() returns pointer to a slab of an allocatable

example (3/3): array-valued functions

C++ / Blitz++:

(C++)

```
#define return_macro(expr) \
  -> decltype(safeToReturn(expr)) \
{ return safeToReturn(expr); }
```

(C++)

```
template<class nom_t, class den_t>
inline auto mpdata_frac(
  const nom_t &nom, const den_t &den
) return_macro(
  where(den > 0, nom / den, 0)
)
```

(C++)

```
template<int d>
inline auto mpdata_A(const arr_t &psi,
  const rng_t &i, const rng_t &j
) return_macro(
  mpdata_frac(
    psi(pi<d>(i+1, j)) - psi(pi<d>(i,j)),
    psi(pi<d>(i+1, j)) + psi(pi<d>(i,j))
  )
)
```

$$A^{[d]}_{[i,j]} = \frac{\psi_{[i,j]+\pi^d_{1,0}} - \psi_{[i,j]}}{\psi_{[i,j]+\pi^d_{1,0}} + \psi_{[i,j]}}$$

return type: C++11's auto $\rightsquigarrow$ array expression

no temporary objects: by design

example (3/3): array-valued functions

Python / NumPy:

(Python)
```
def mpdata_frac(nom, den):
  return numpy.where(den > 0, nom/den, 0)
```

(Python)
```
def mpdata_A(d, psi, i, j):
  return mpdata_frac(
    psi[pi(d, i+one, j)] - psi[pi(d, i, j)],
    psi[pi(d, i+one, j)] + psi[pi(d, i, j)]
  )
```

$$A^{[d]}_{[i,j]} = \frac{\psi_{[i,j]+\pi^d_{1,0}} - \psi_{[i,j]}}{\psi_{[i,j]+\pi^d_{1,0}} + \psi_{[i,j]}}$$

return type: n/a

temporary objects: interpreter-dependant

example (3/3): array-valued functions

Fortran:

(Fortran)

```
function mpdata_frac(nom, den) result (return)
  real(real_t), intent(in) :: nom(:,:), den(:,:)
  real(real_t) :: return(size(nom, 1), size(nom, 2))
  where (den > 0)
    return = nom / den
  elsewhere
    return = 0
  end where
end function
```

(Fortran)

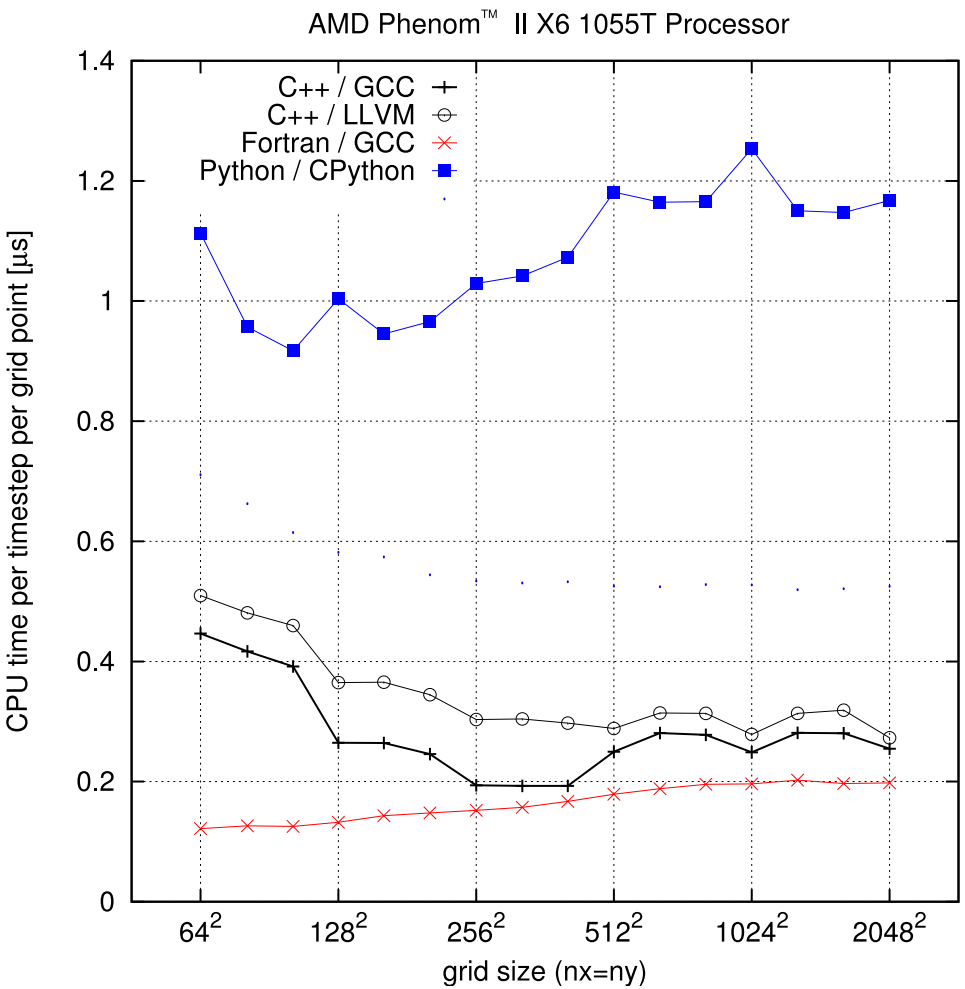
```
function mpdata_A(d, psi, i, j) result (return)
  integer :: d
  real(real_t), allocatable, intent(in) :: psi(:,:)
  integer, intent(in) :: i(2), j(2)
  real(real_t) :: return(span(d, i, j), span(d, j, i))
  return = mpdata_frac(                                  &
    pi(d, psi, i+1, j) - pi(d, psi, i, j),               &
    pi(d, psi, i+1, j) + pi(d, psi, i, j)                &
  )
end function
```

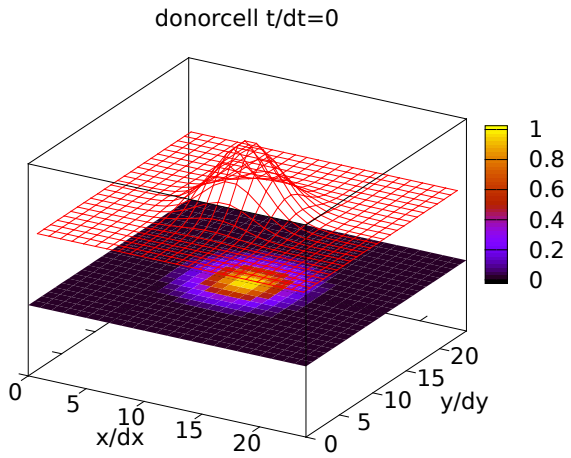
$$A^{[d]}_{[i,j]} = \frac{\psi_{[i,j]+\pi^d_{1,0}} - \psi_{[i,j]}}{\psi_{[i,j]+\pi^d_{1,0}} + \psi_{[i,j]}}$$

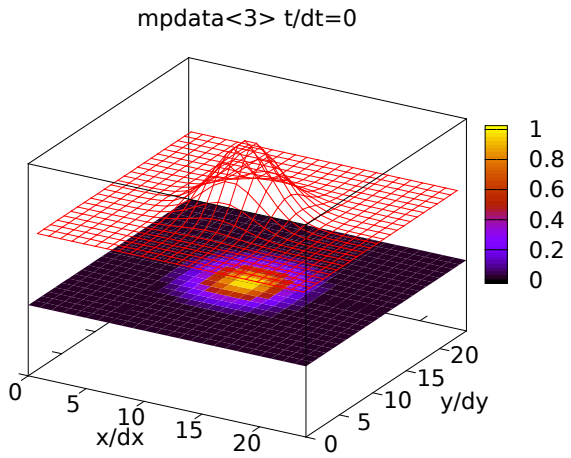
return type: "dimension" (array)

temporary objects: compiler-dependant

MPDATA in C++, Python and Fortran: performance



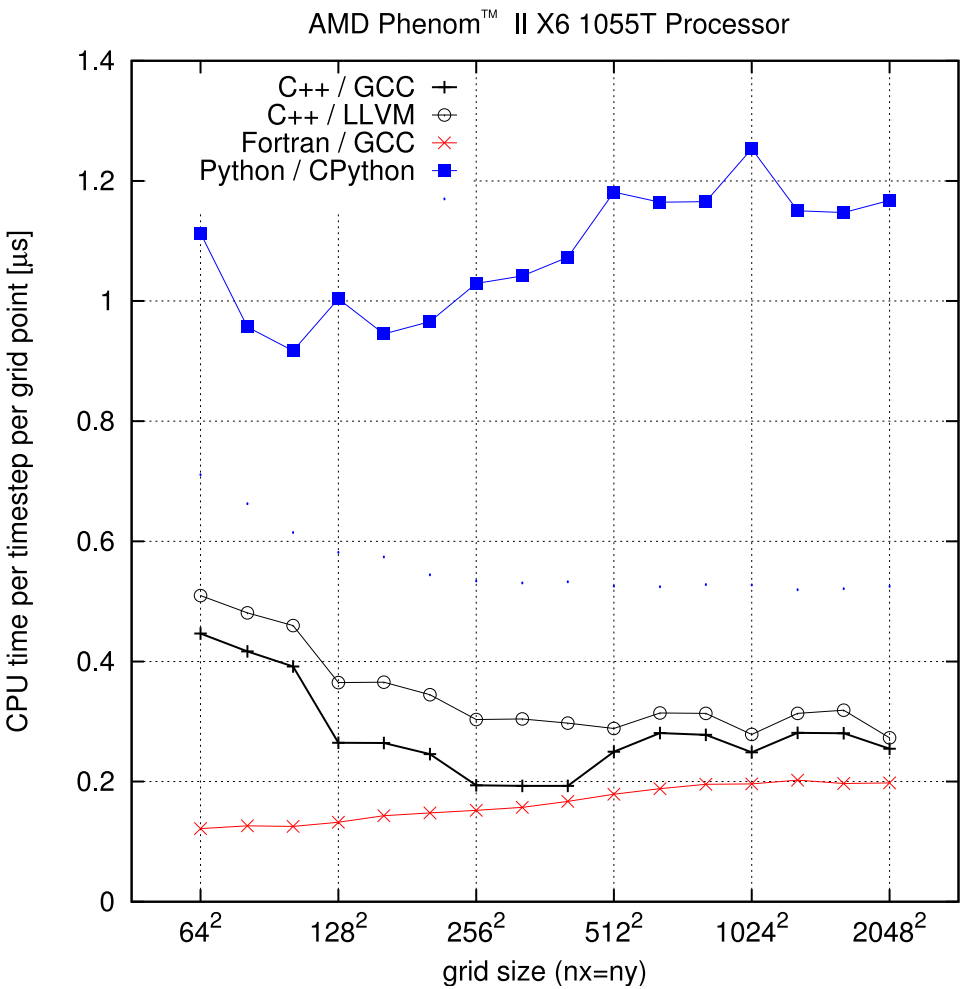


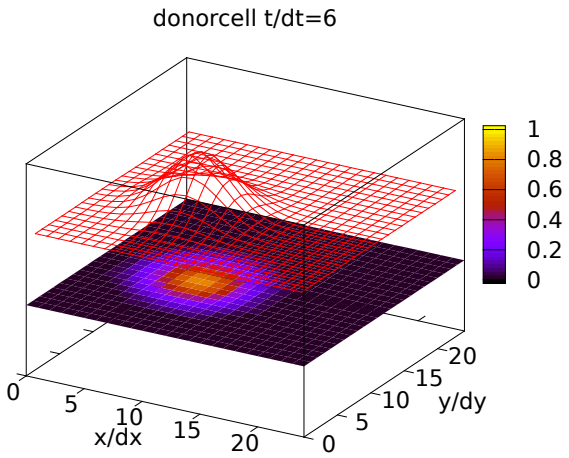


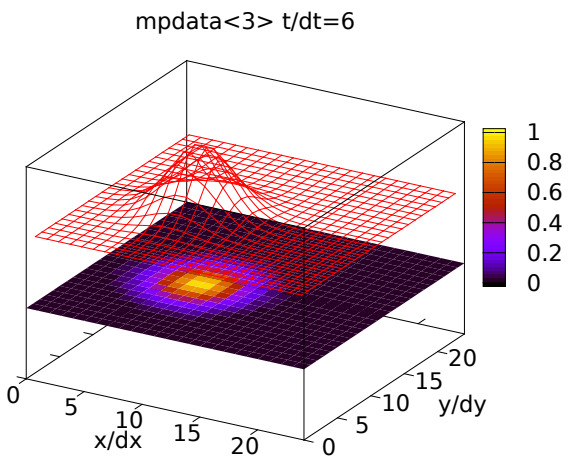
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



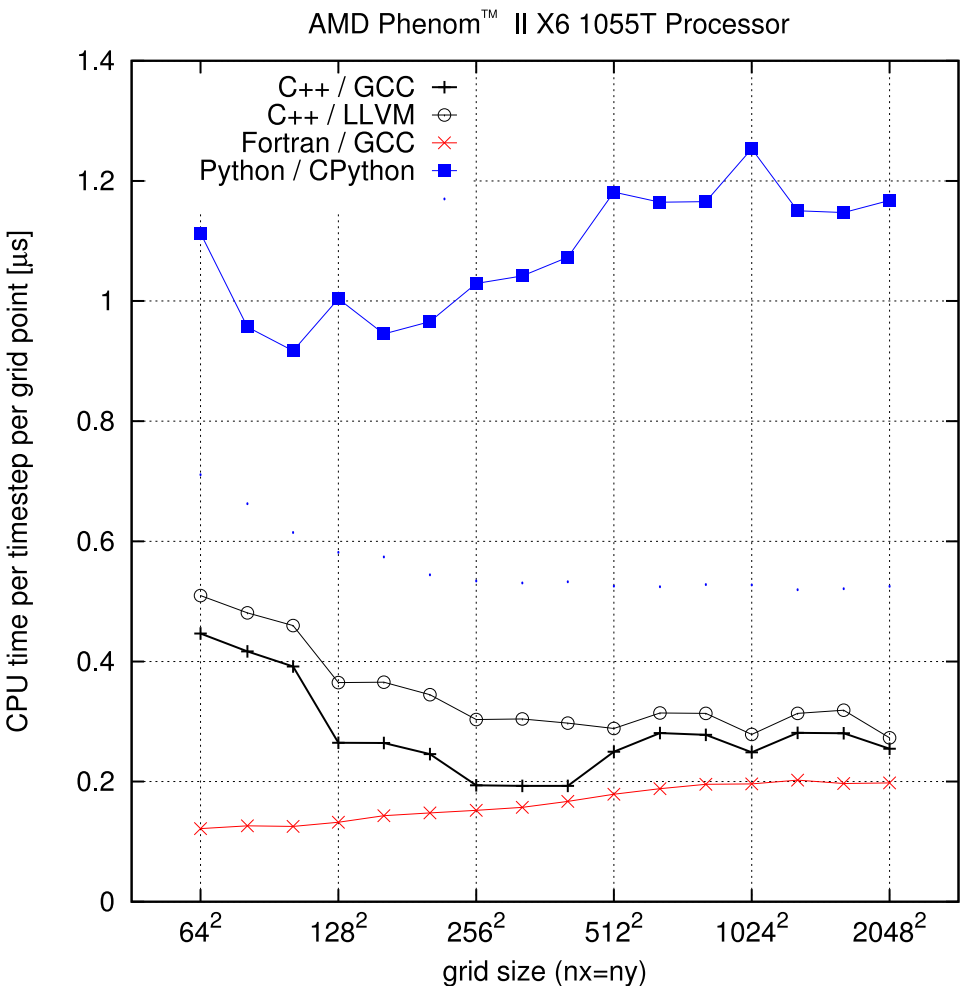


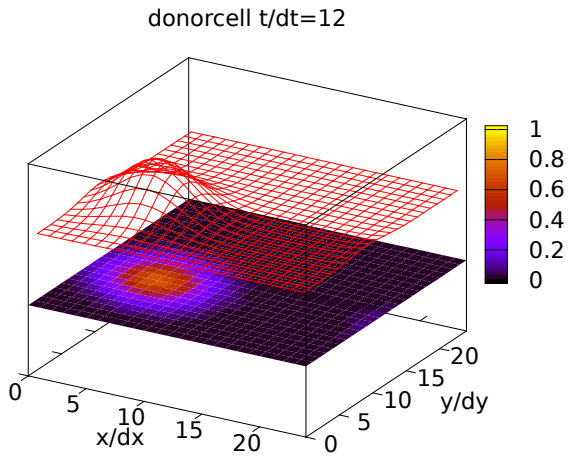


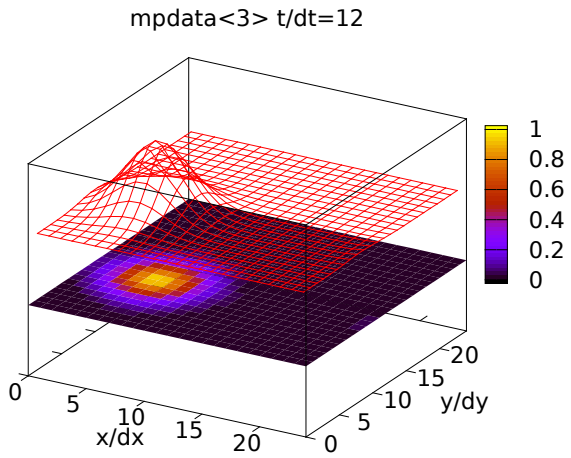
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



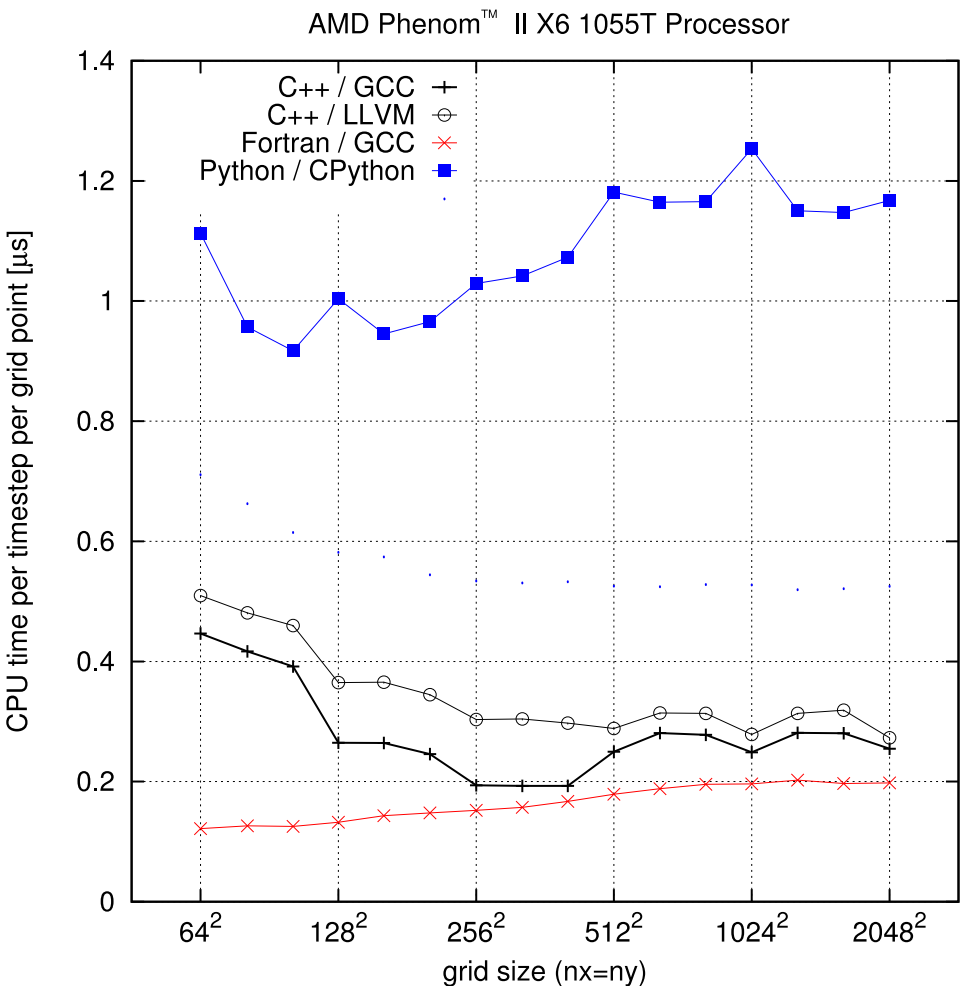


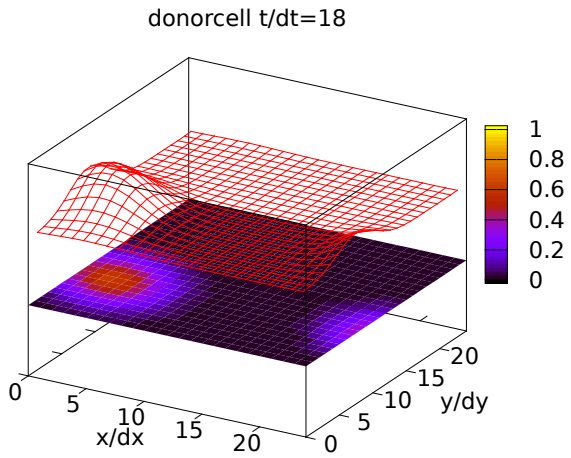


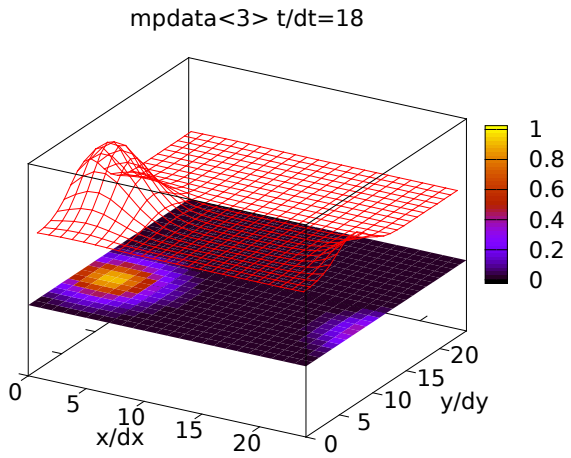
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



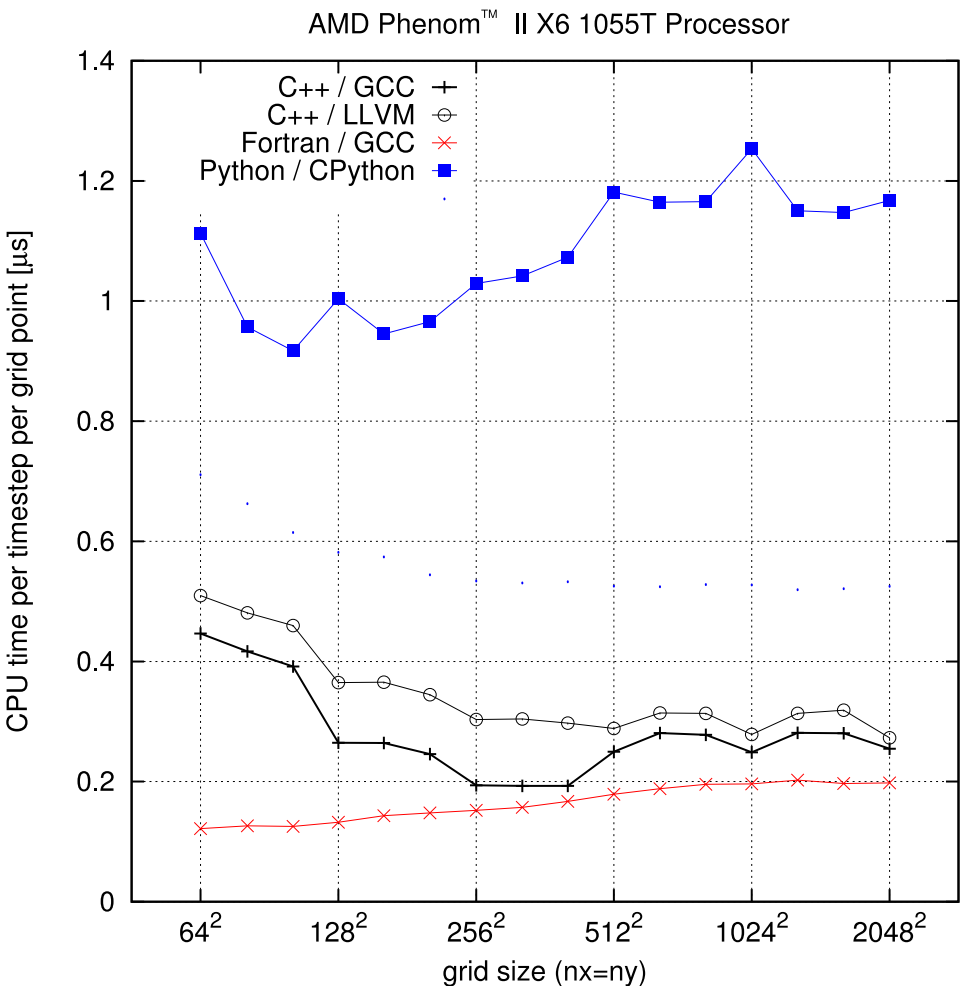


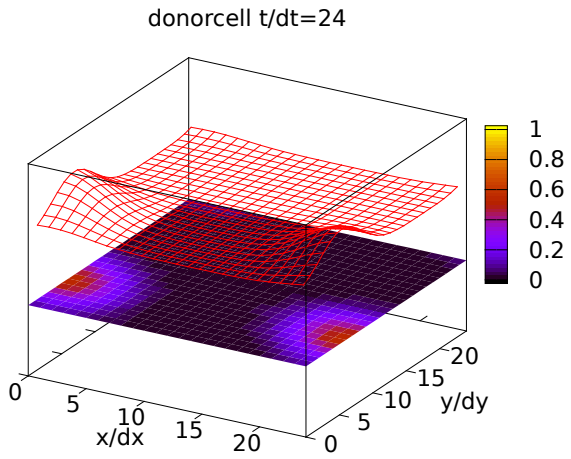


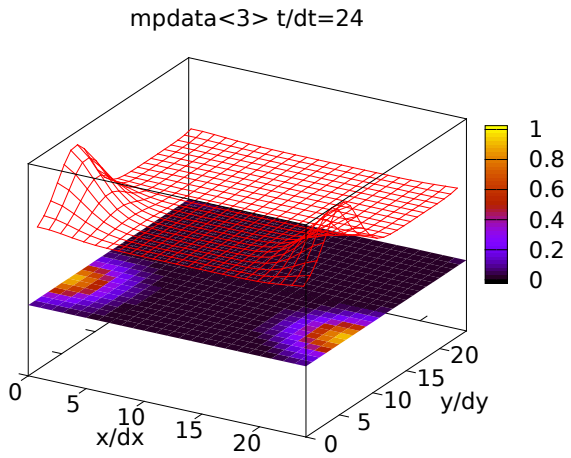
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



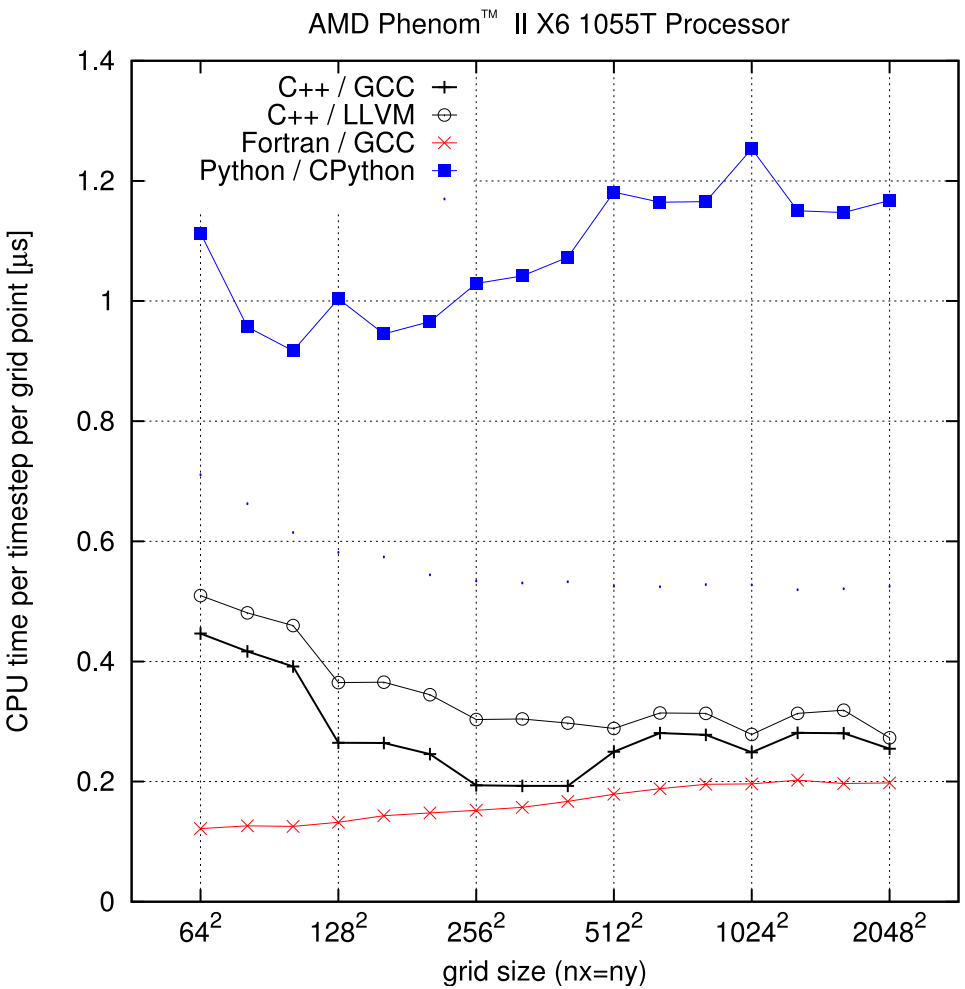


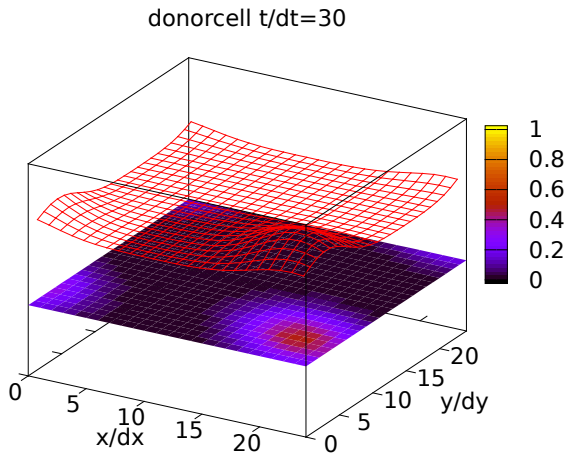


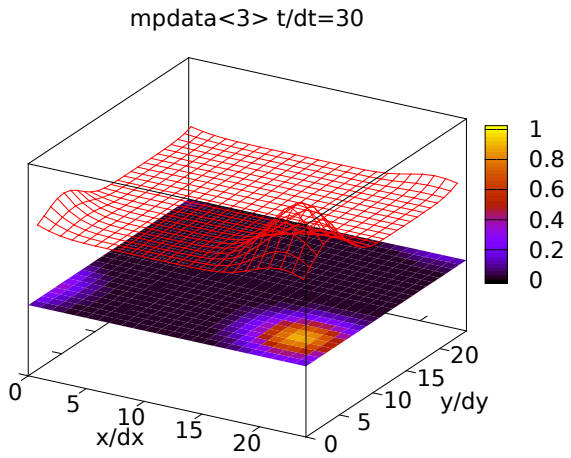
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



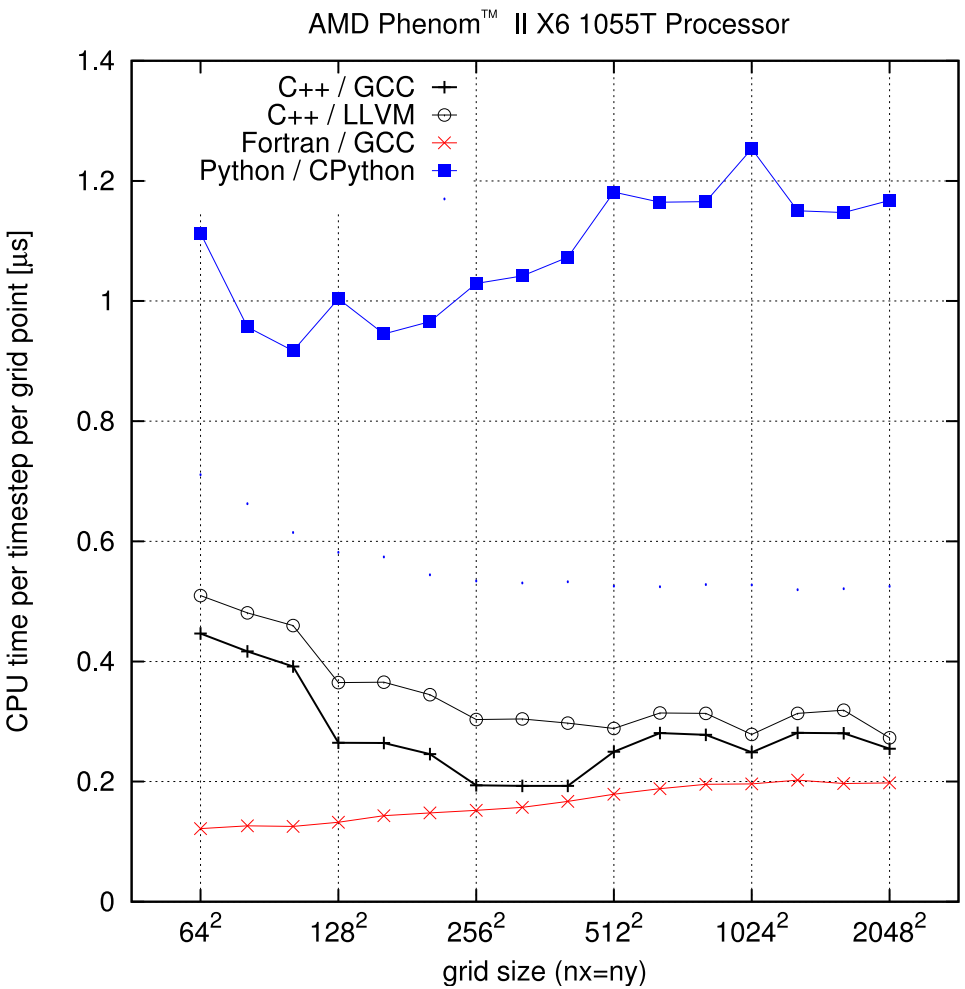


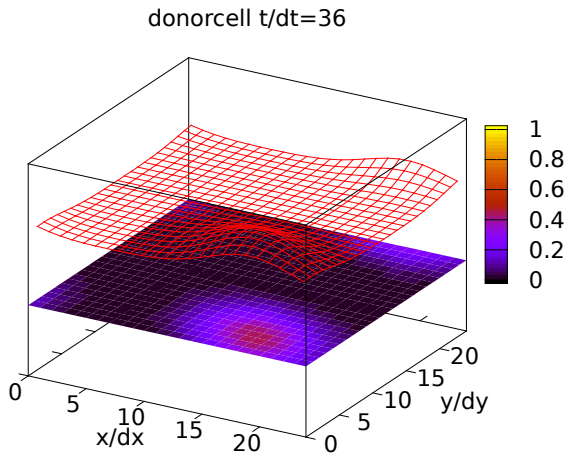


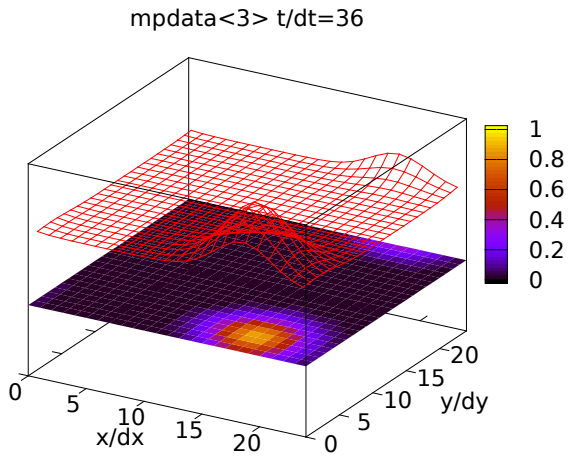
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



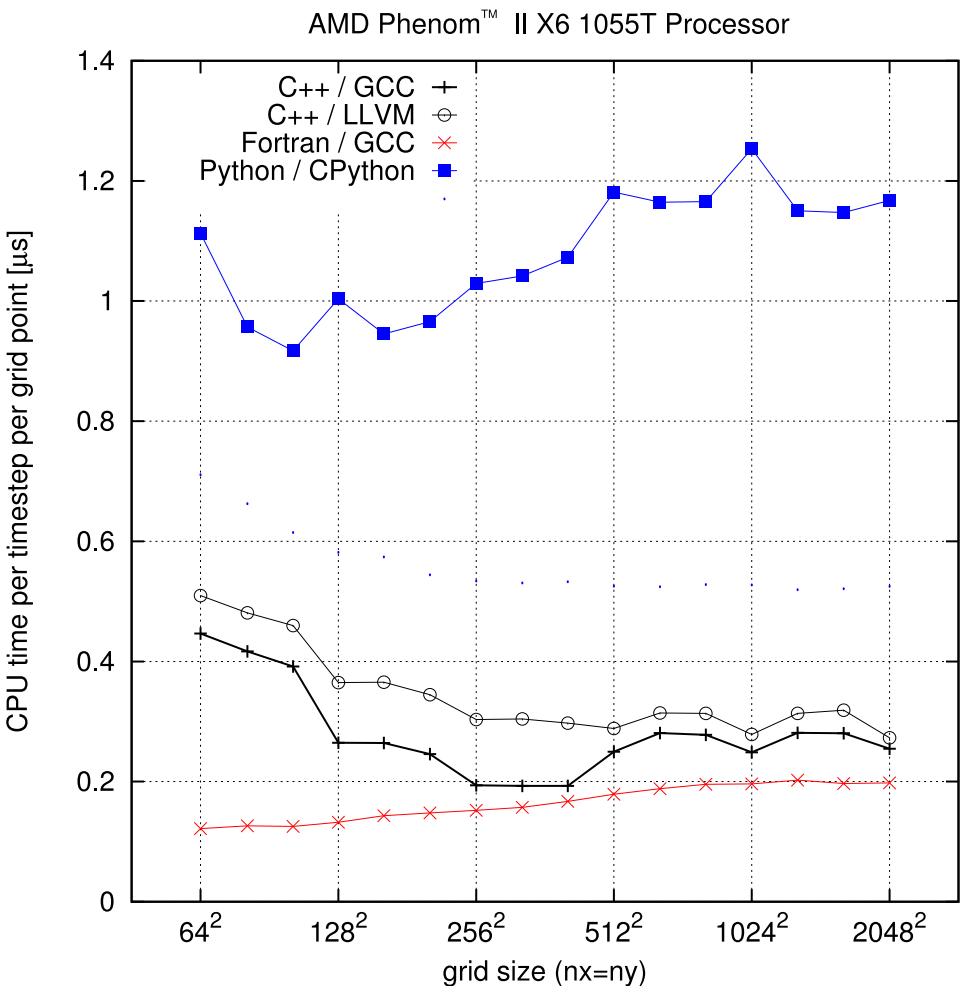


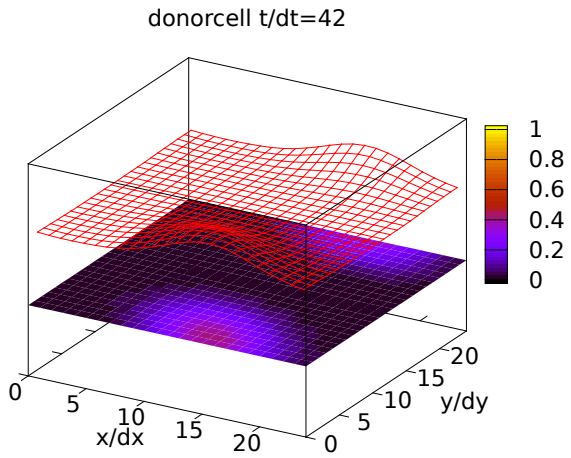


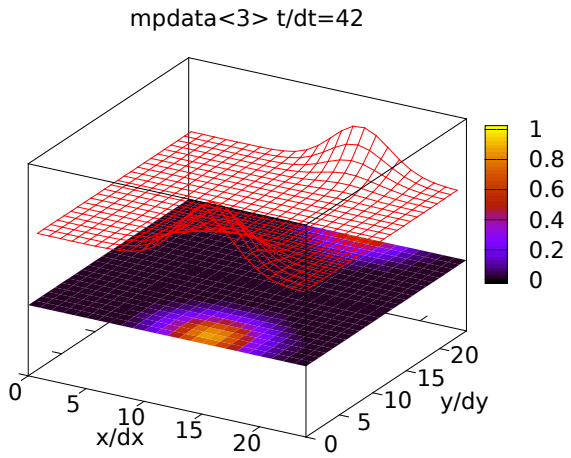
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



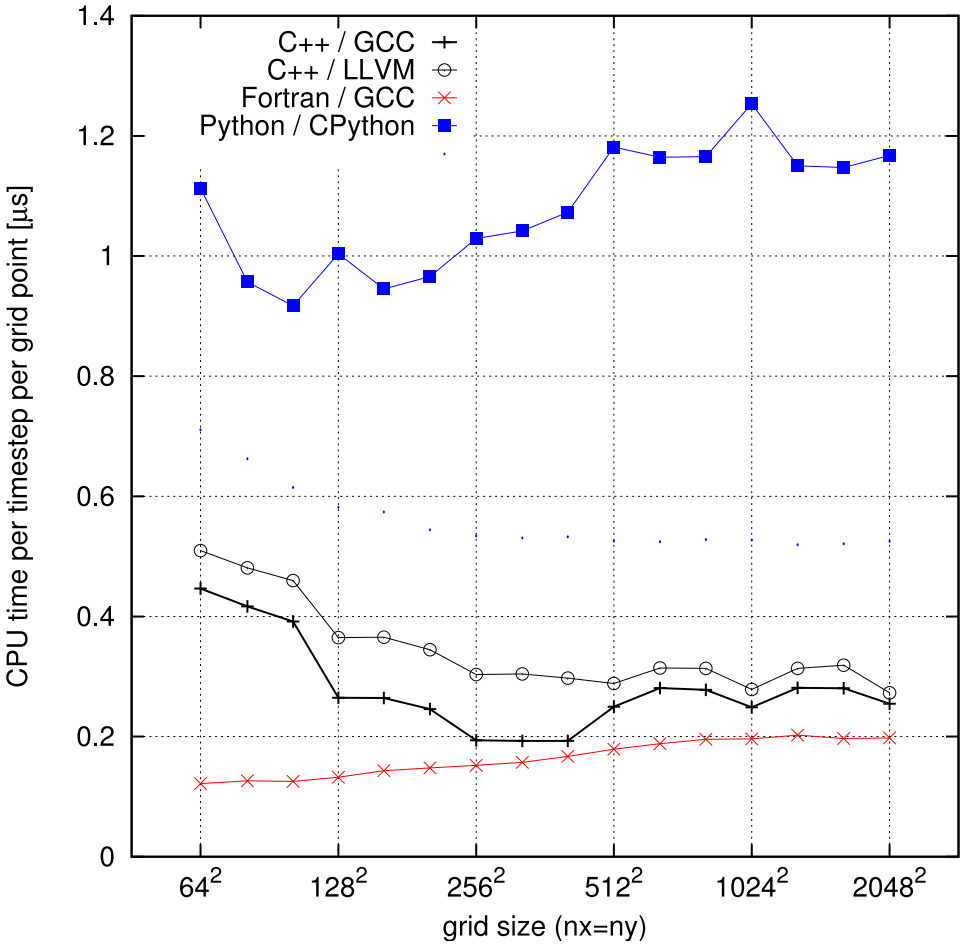


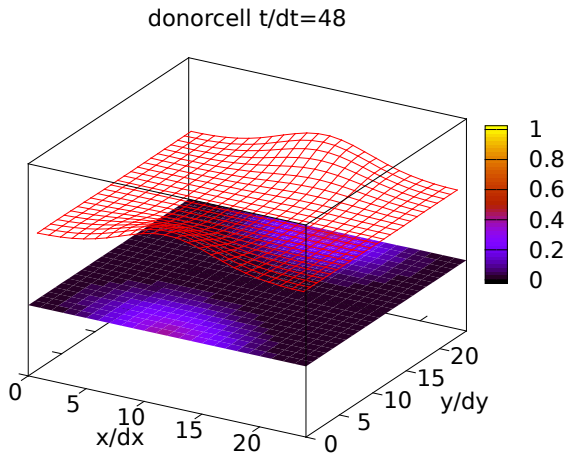


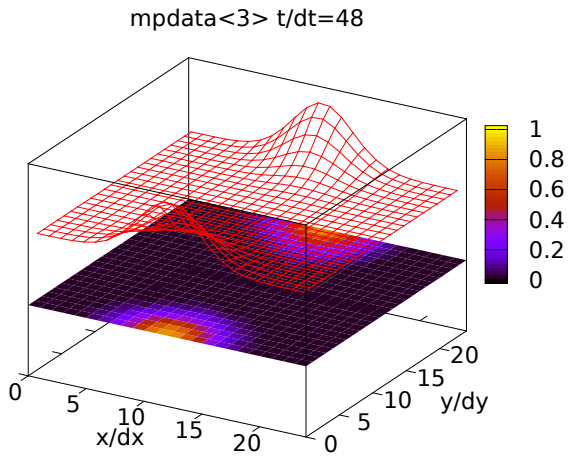
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



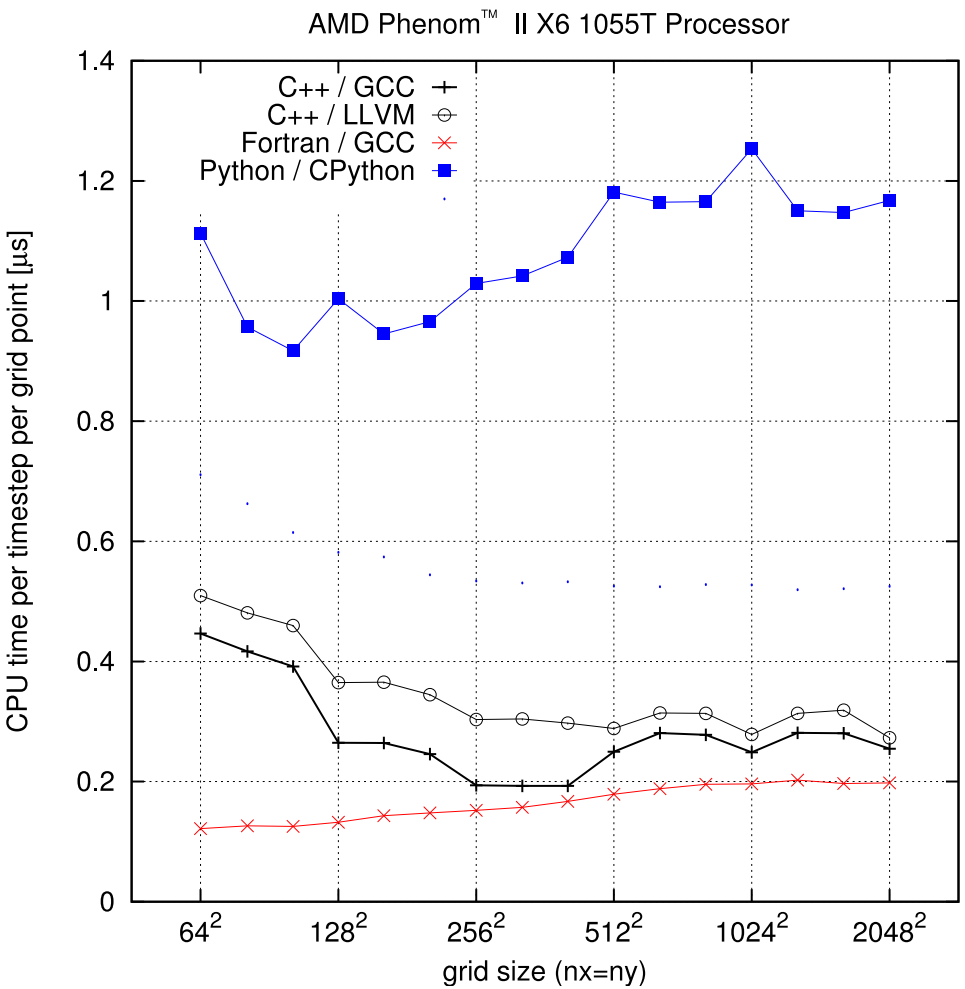


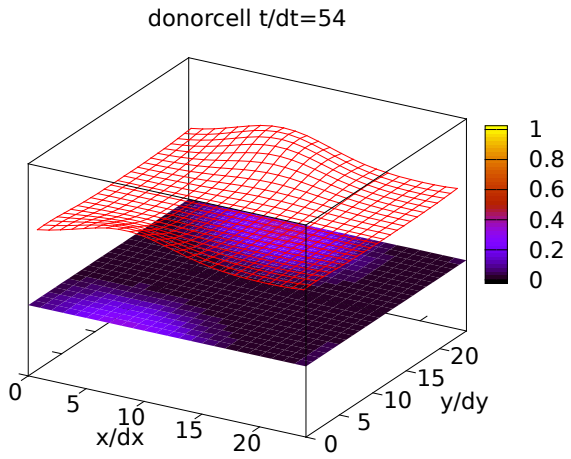


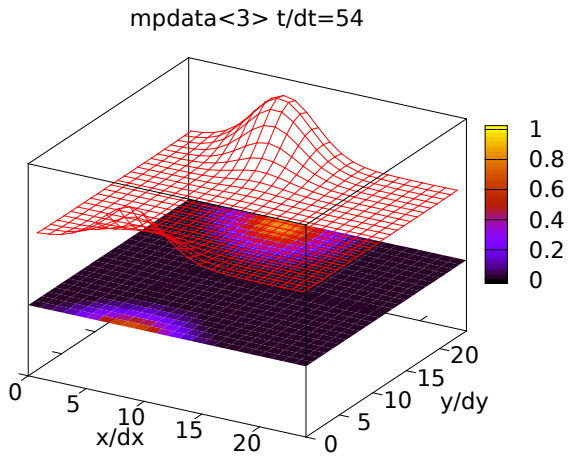
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



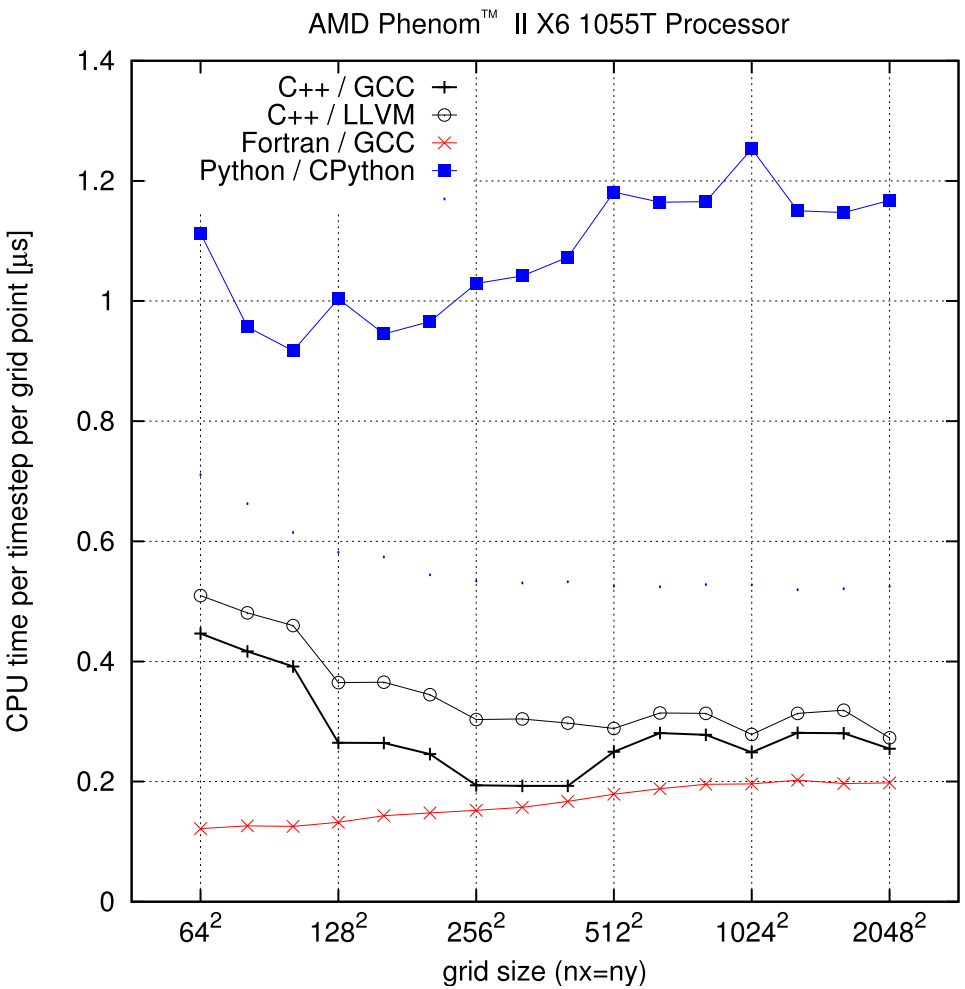


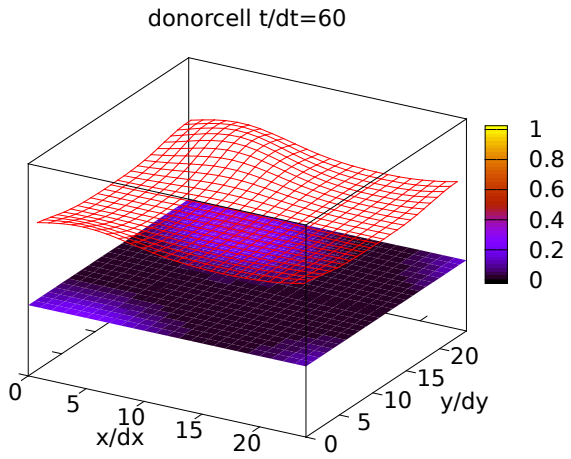


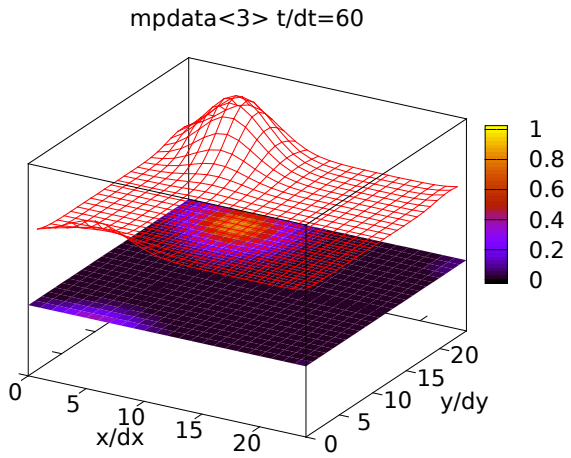
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



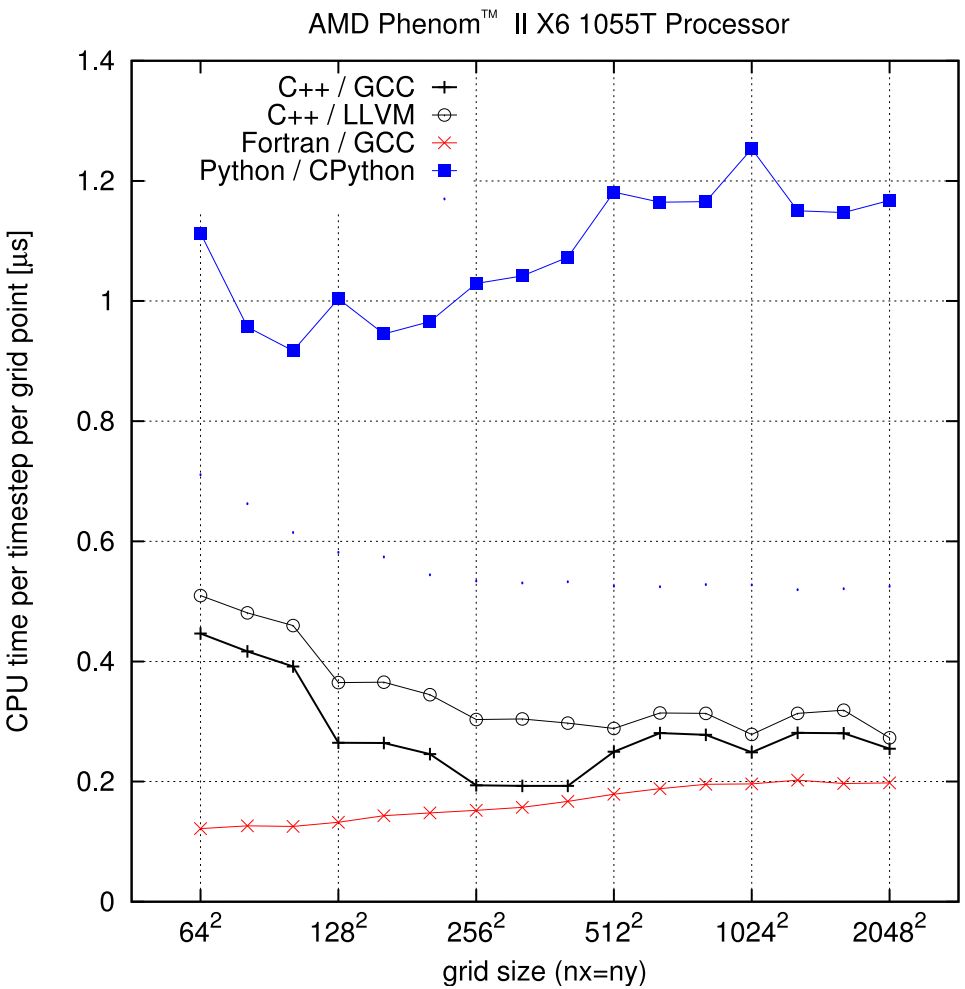


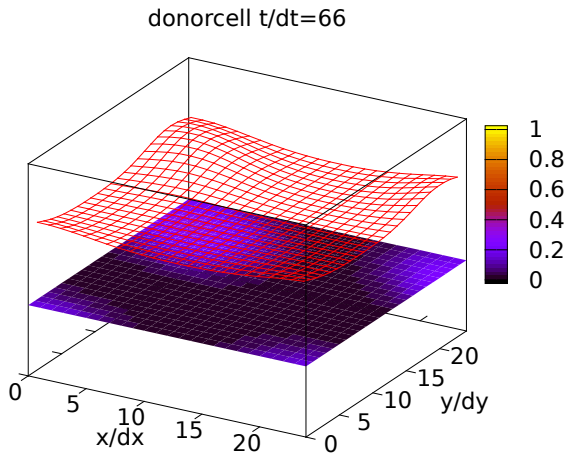


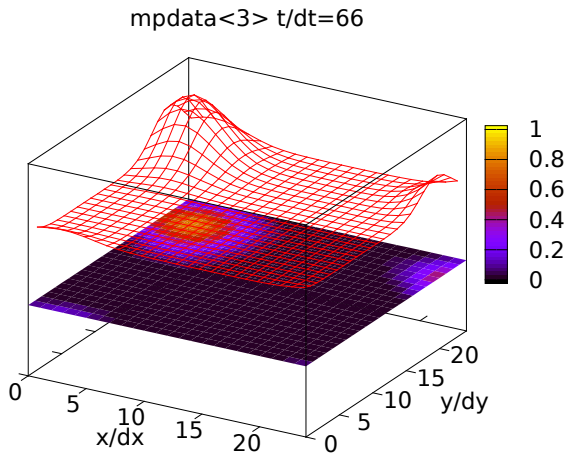
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



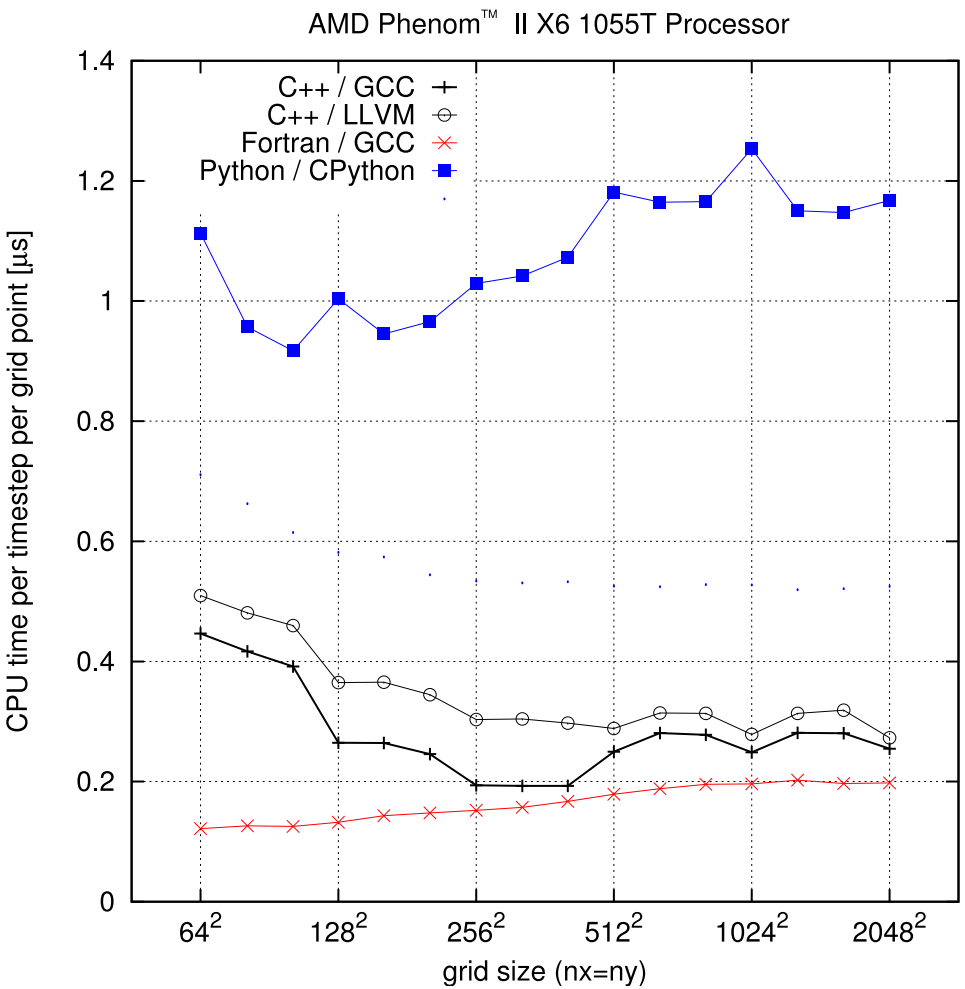


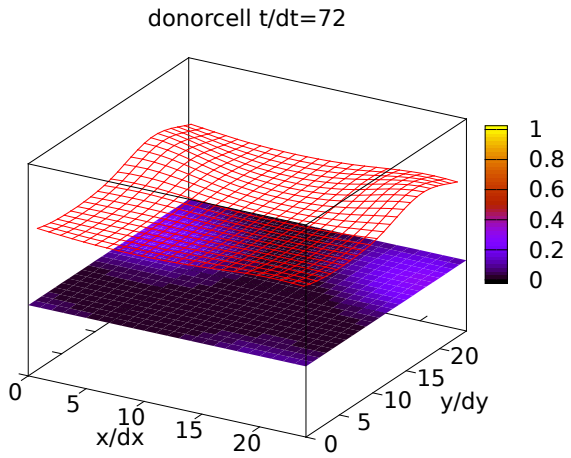


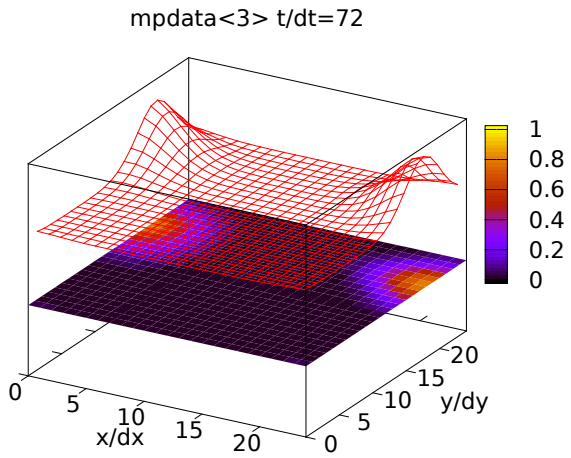
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



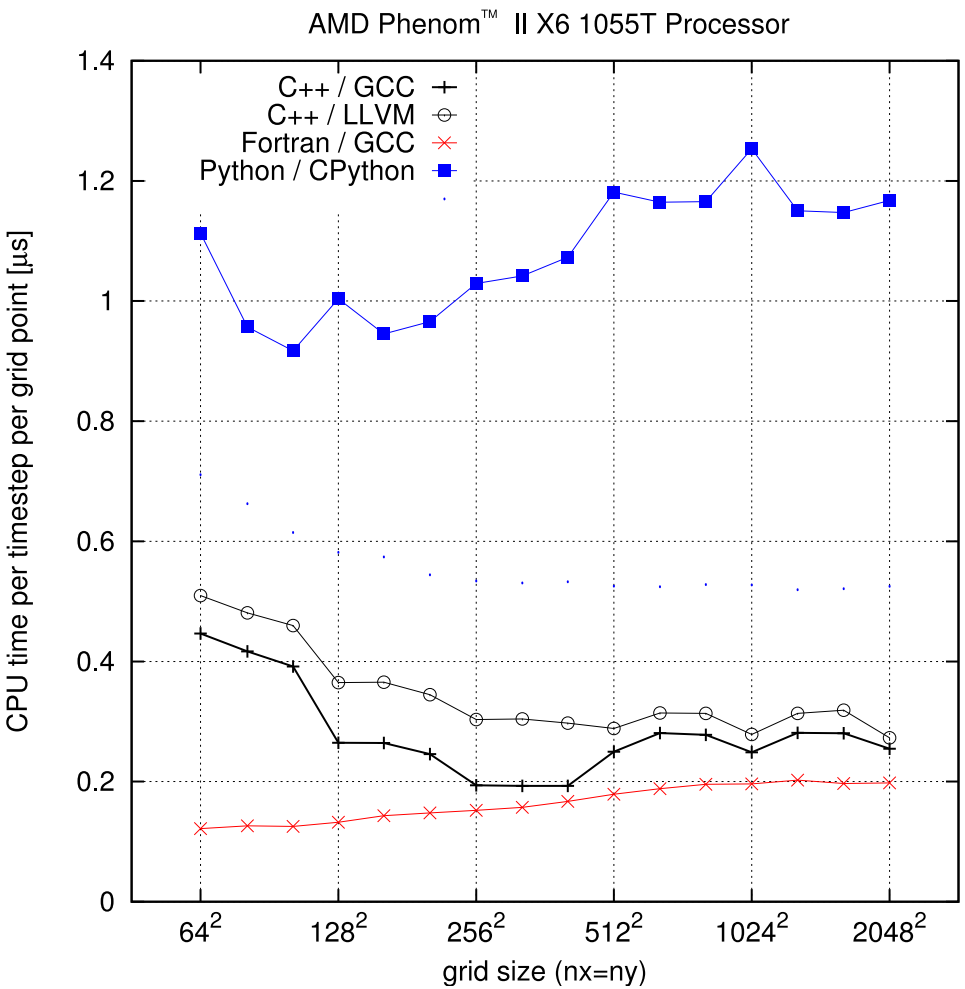


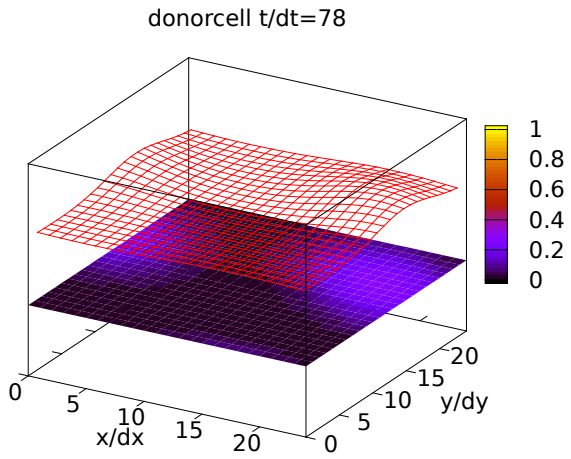


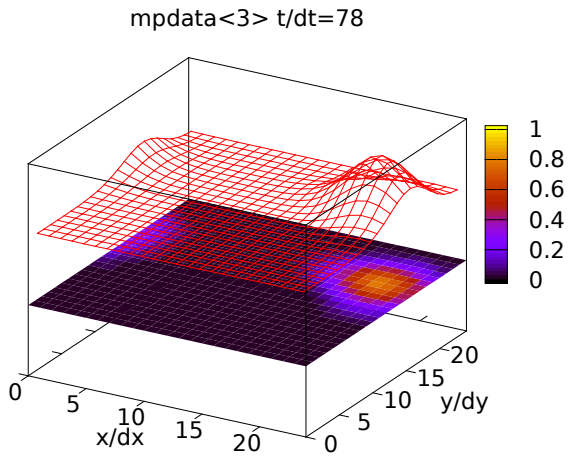
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



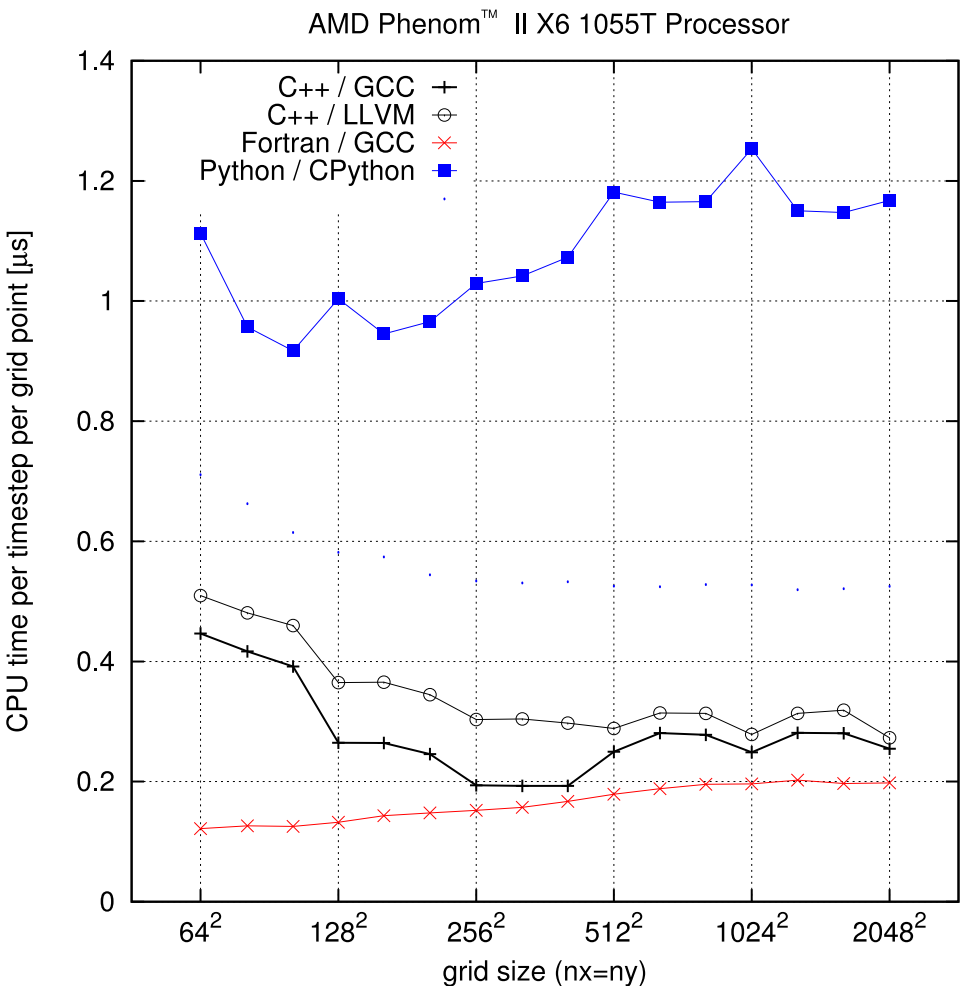


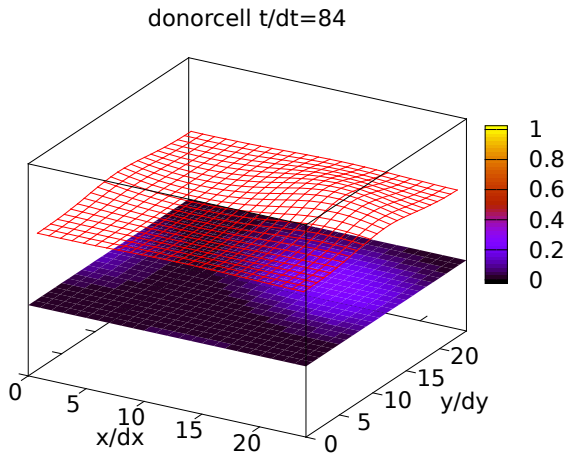


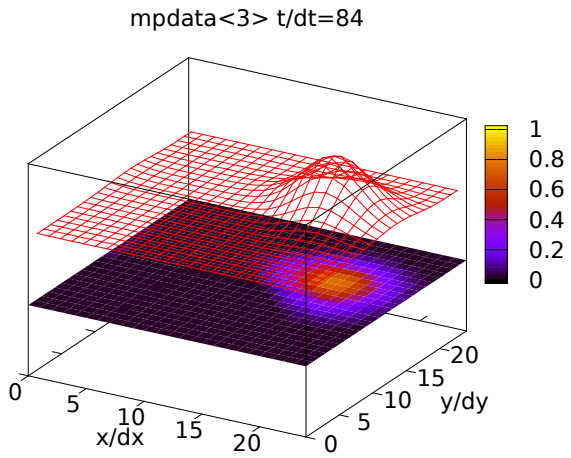
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



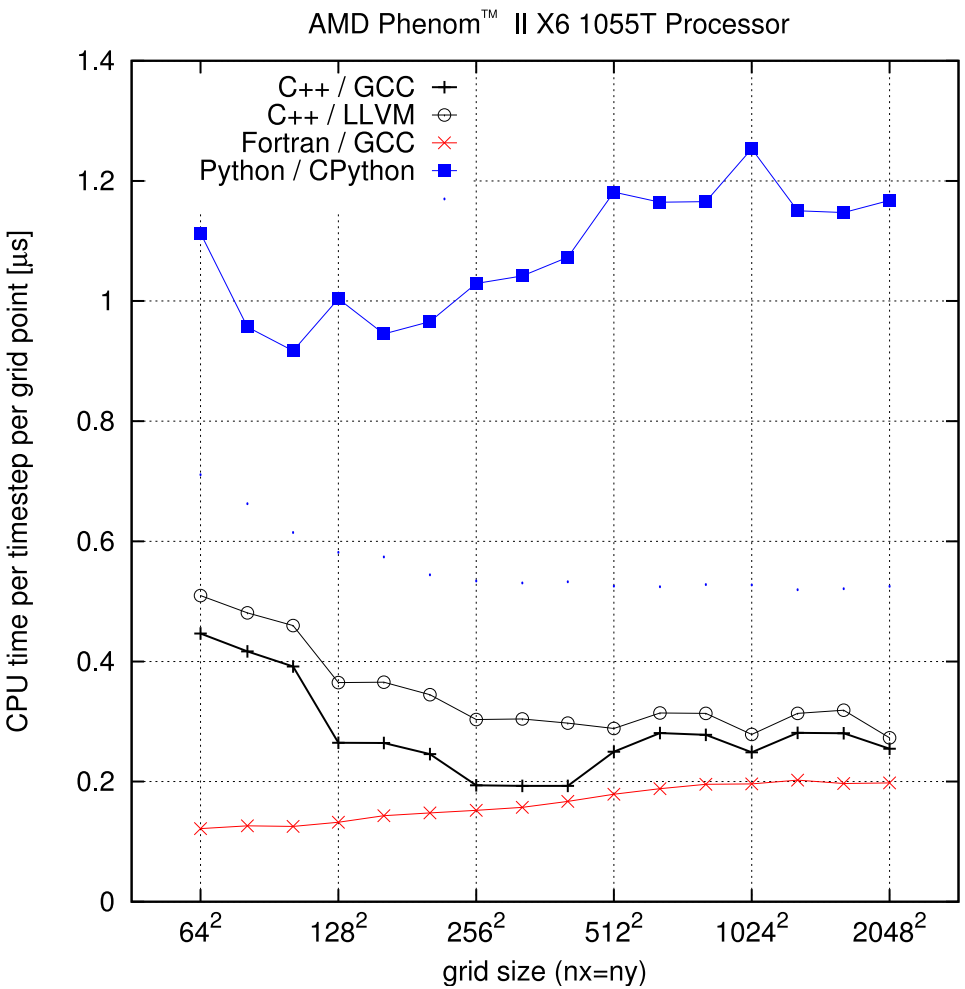


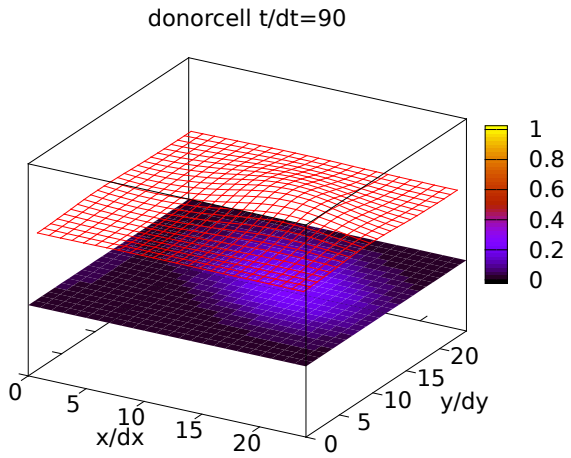


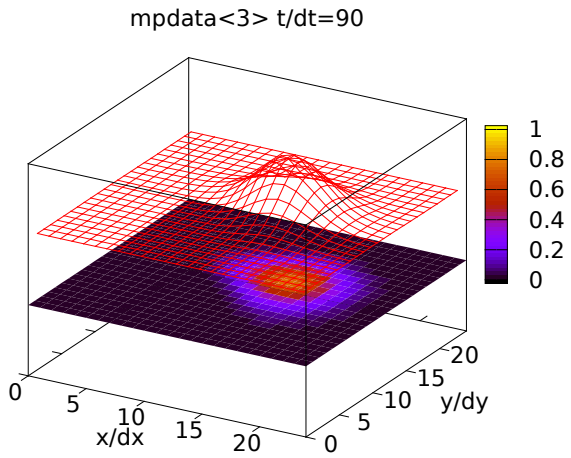
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



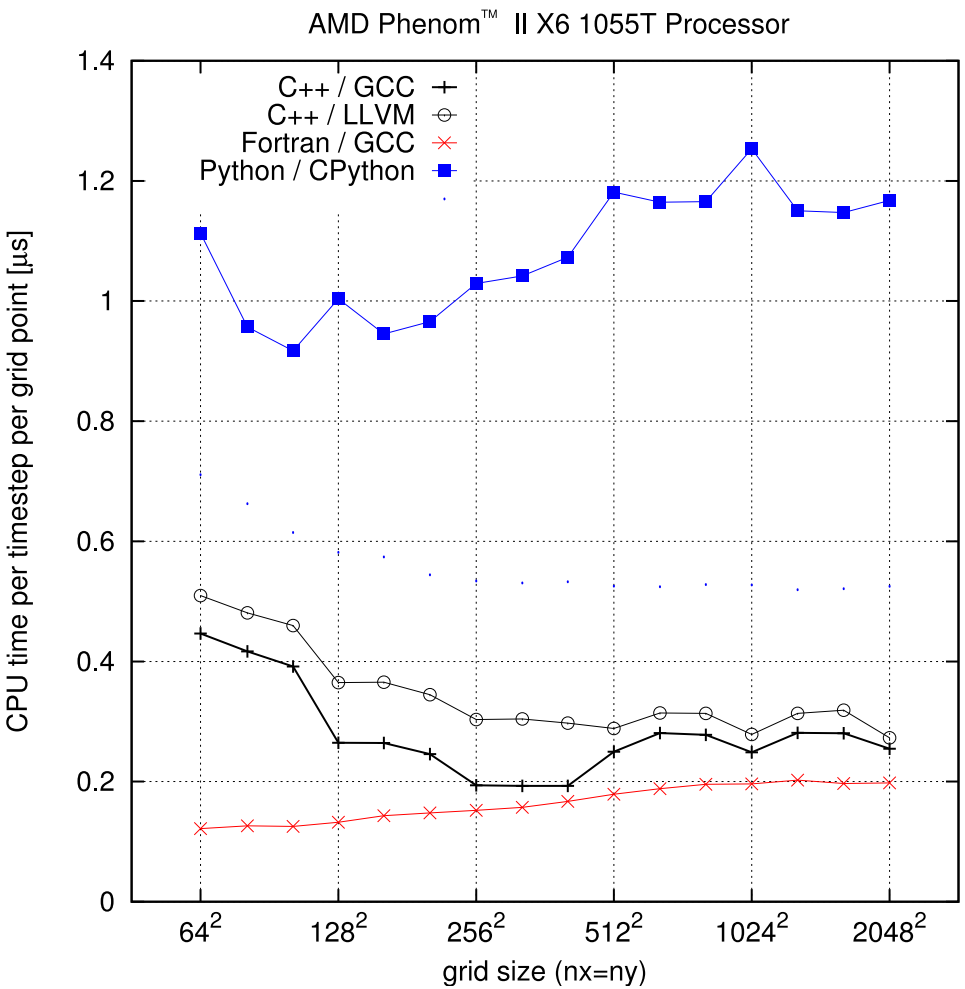


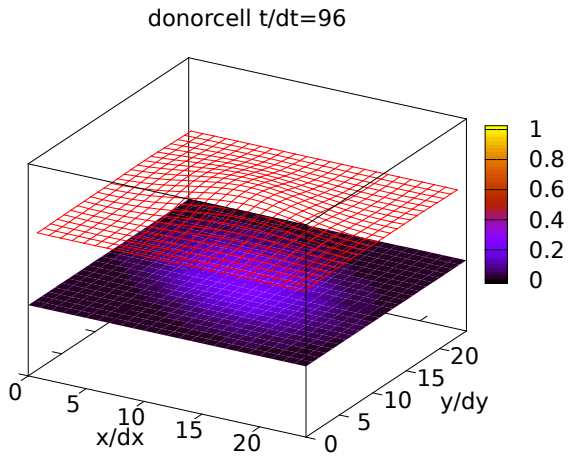


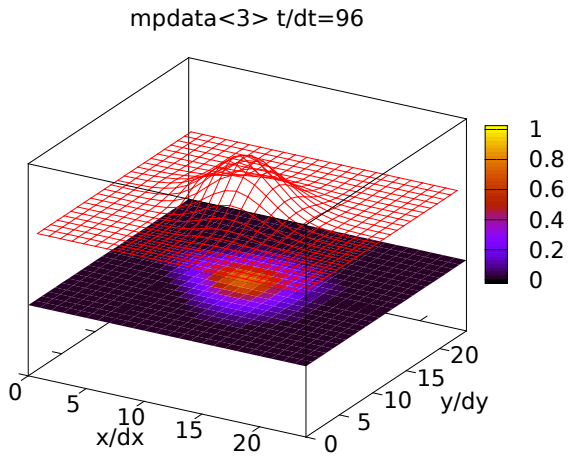
Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance



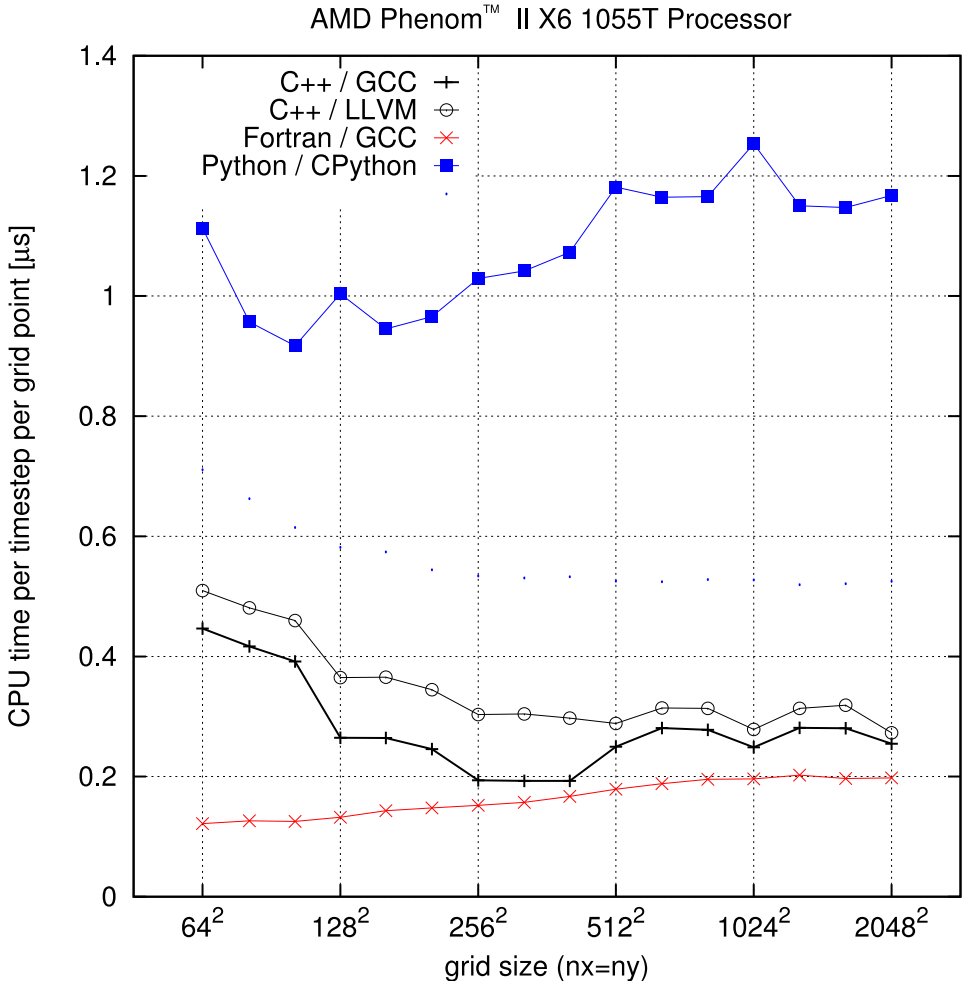


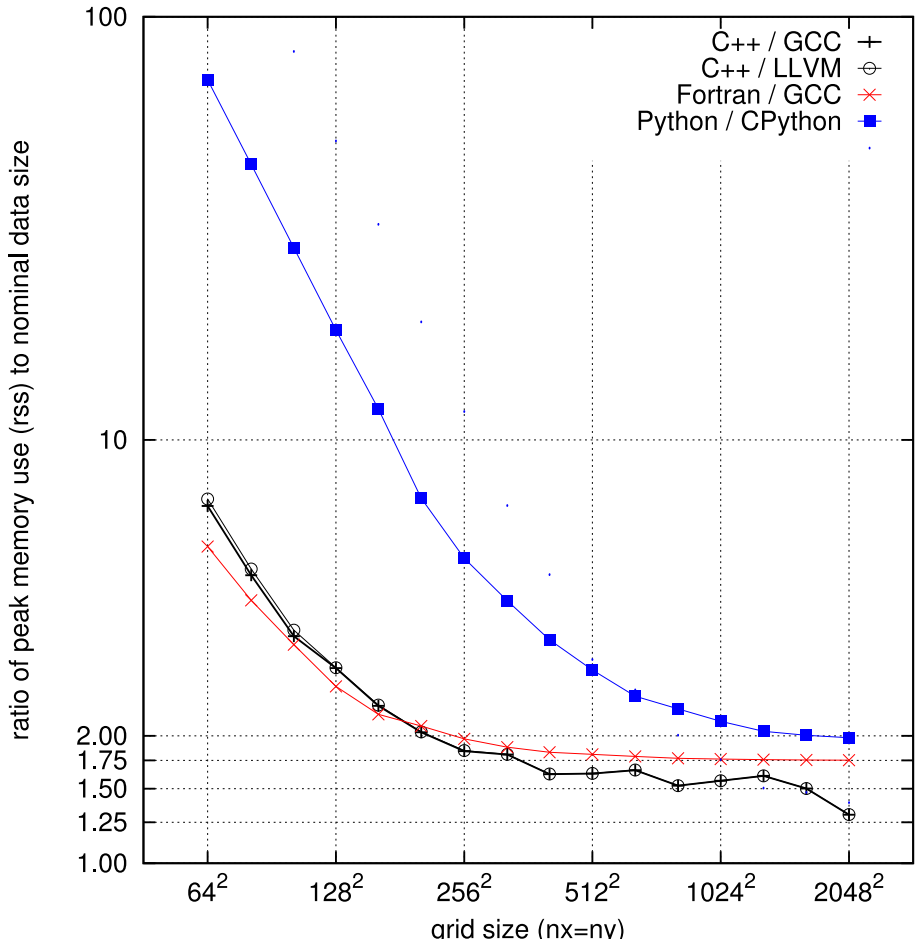


Is there any way to improve Python's performance?

UNIVERSITAS VARSOVIENSIS

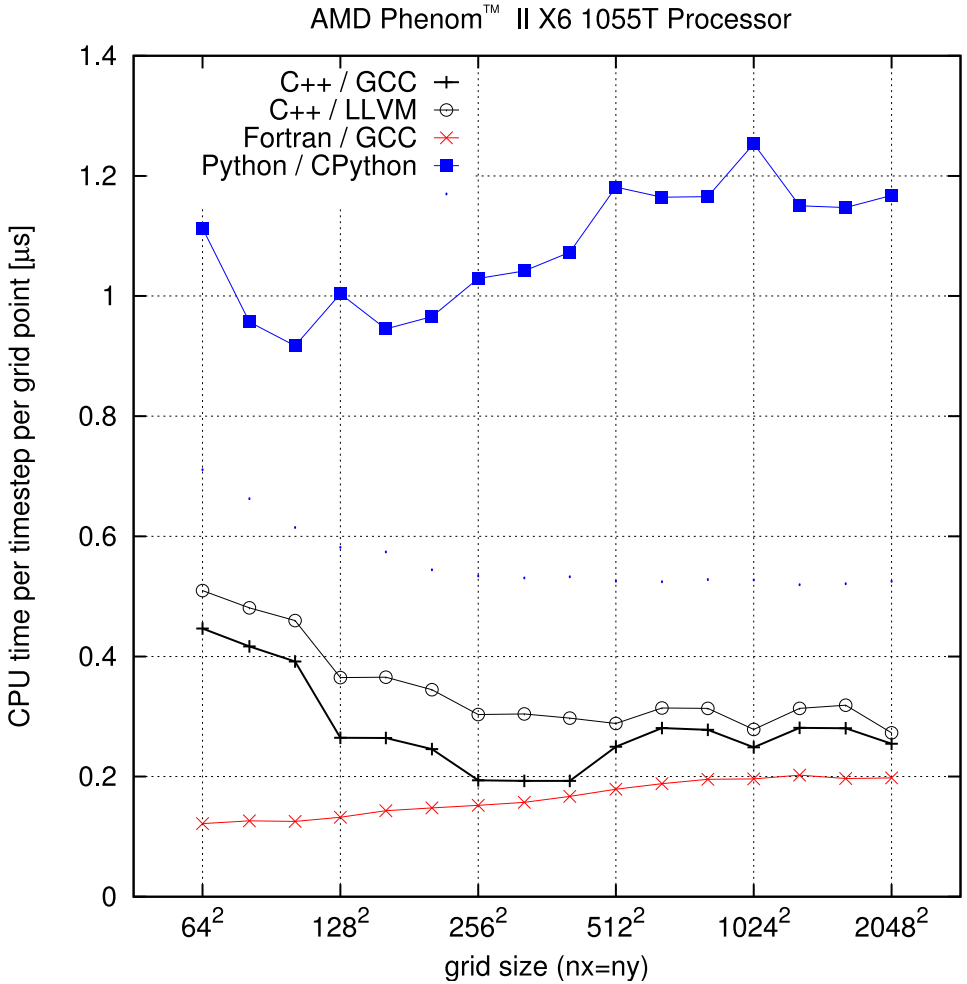
MPDATA in C++, Python and Fortran: performance

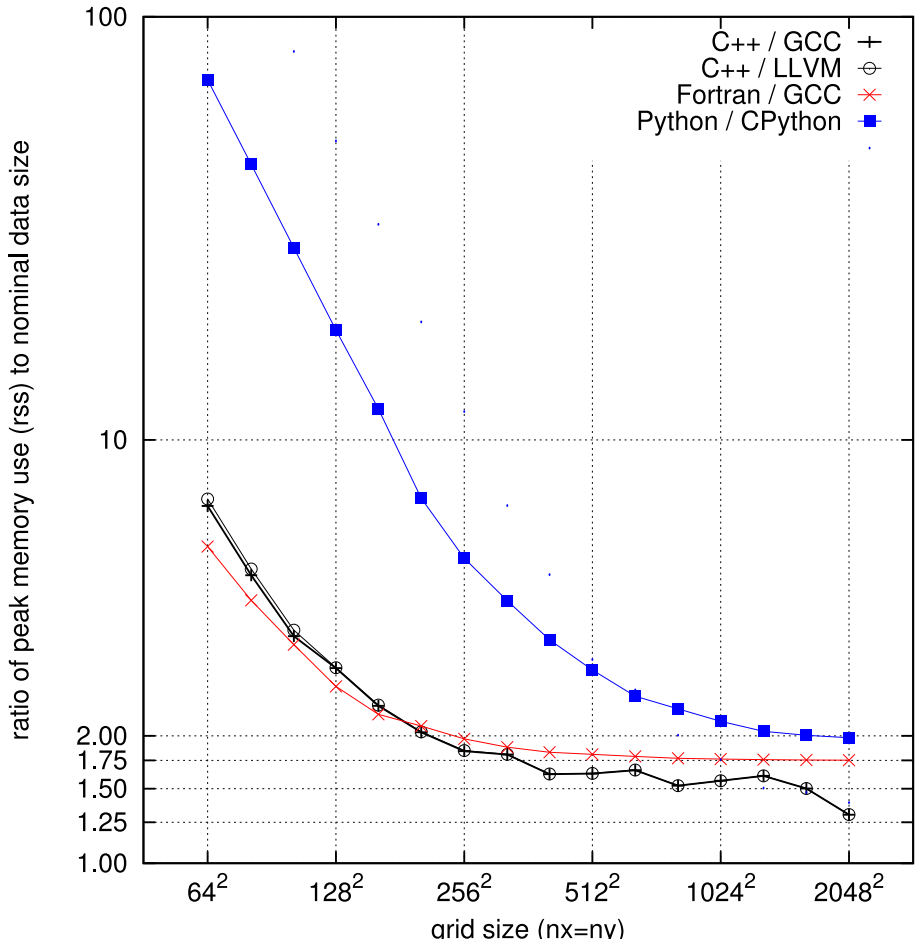




Is there any way to improve Python's performance?

MPDATA in C++, Python and Fortran: performance





Is there any way to improve Python's performance?

speeding up NumPy code with PyPy

- what's PyPy?
  - alternative implementation of Python
    equipped with just-in-time compiler (JIT)
  - developed with the aim of improving Python's performance
    while maintaining compatibility with CPython
  - more info: `http://pypy.org`

- why to use PyPy (in this context)?
  - PyPy's built-in NumPy implementation
    features JIT-powered lazy-evaluation mechanism
    $\rightsquigarrow$ potential improvement in speed and memory consumption
  - switching to PyPy does not require code modifications!
    (in contrast to Numexpr, Cython, Numba, ...)

UNIVERSITAS VARSOVIENSIS

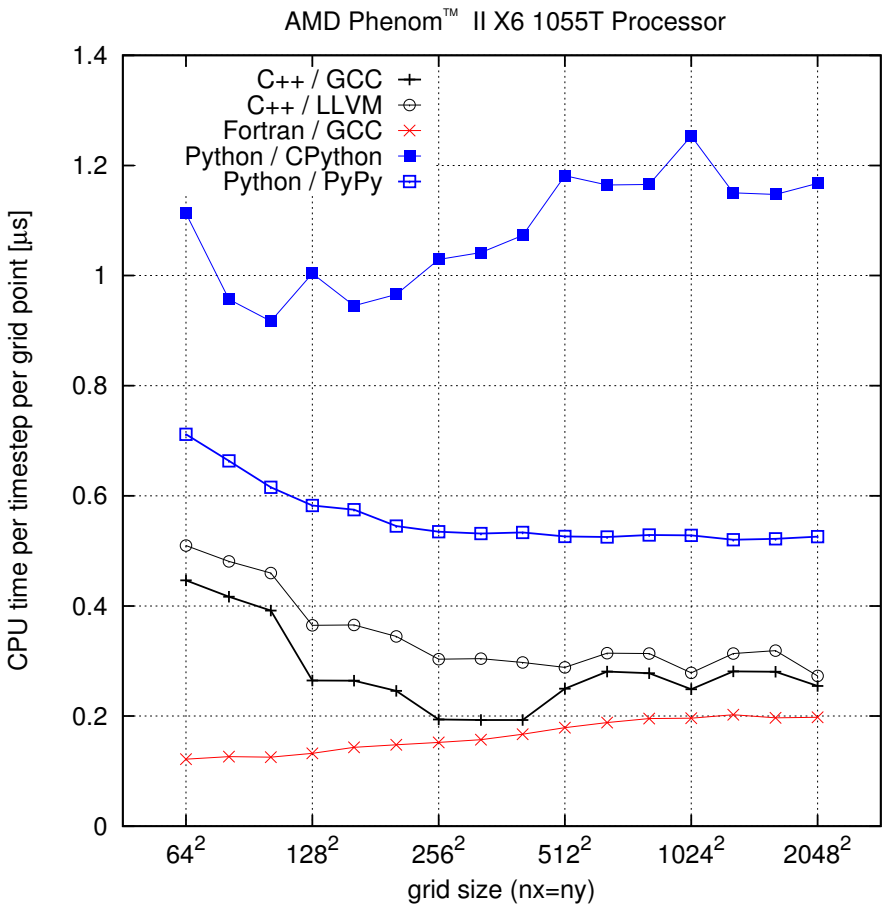
speeding up NumPy code with PyPy

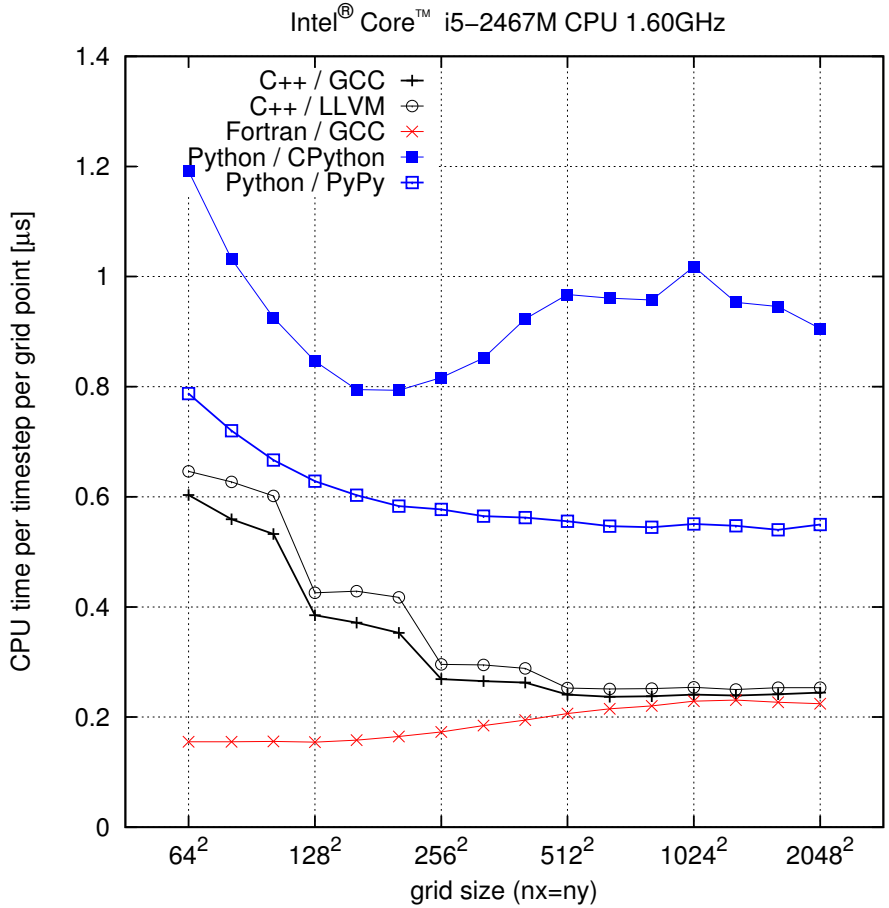
- what's PyPy?
  - alternative implementation of Python
    equipped with just-in-time compiler (JIT)
  - developed with the aim of improving Python's performance
    while maintaining compatibility with CPython
  - more info: `http://pypy.org`

- why to use PyPy (in this context)?
  - PyPy's built-in NumPy implementation
    features JIT-powered lazy-evaluation mechanism
    $\rightsquigarrow$ potential improvement in speed and memory consumption
  - switching to PyPy does not require code modifications!
    (in contrast to Numexpr, Cython, Numba, ...)

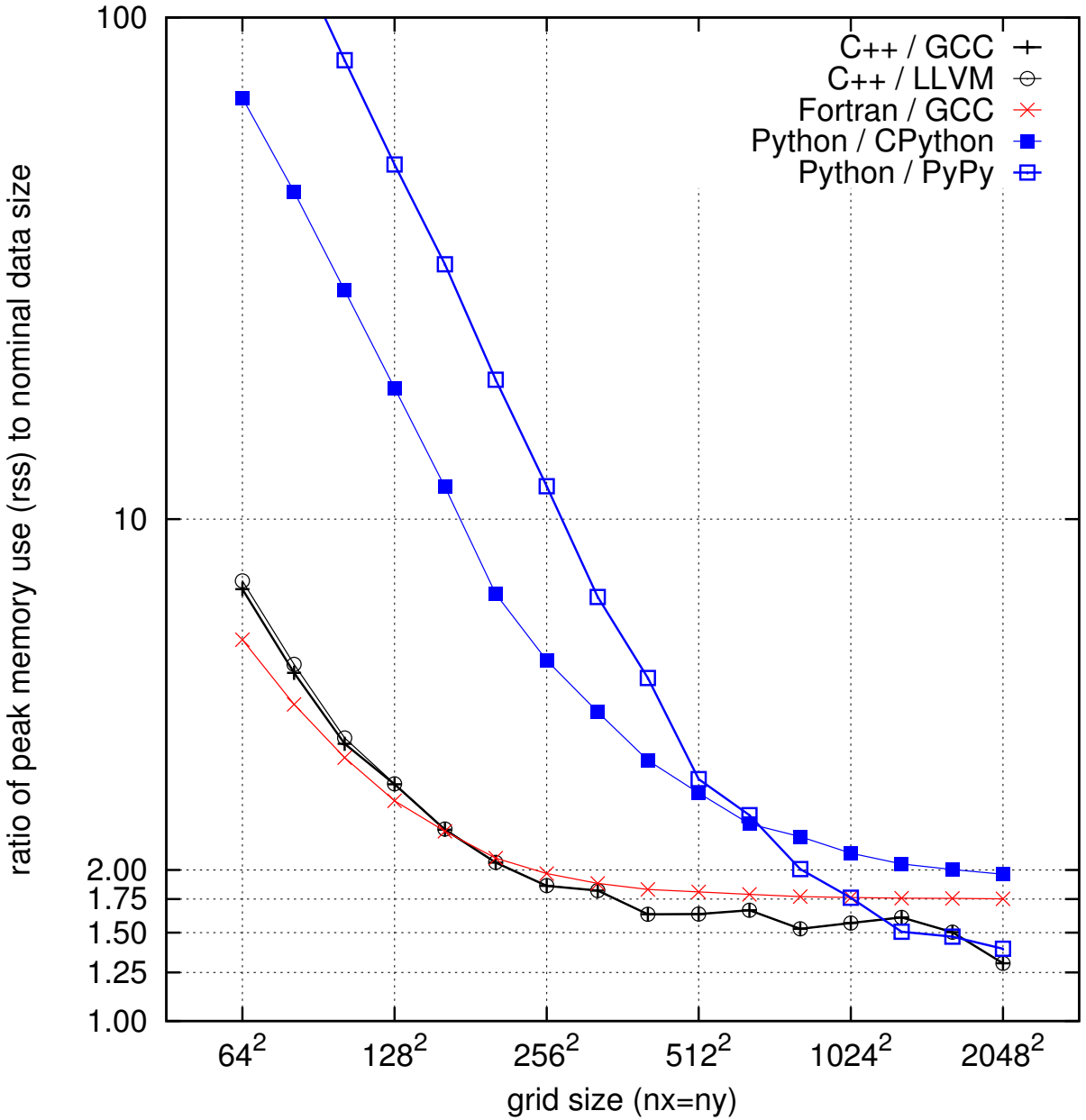
UNIVERSITAS VARSOVIENSIS

MPDATA in C++, Python and Fortran: performance





MPDATA in C++, Python and Fortran: performance



OOP numerics: language choice tradeoffs

capabilities for representing blackboard abstractions:

- C++, Python and Fortran provide comparable functionalities for compact representation of mathematical abstractions thanks to:
  - loop-free array arithmetics
  - array-valued expressions and functions
  - indirections allowing permuting array indices
  - fractional indexing through operator overloading
  - ...
- Fortran's limitations:
  - no built-in mechanism for code reuse on different types (templates in C++, duck-typing in Python)
  - no function calls on lhs
  - no array's of array's (e.g. components of a vector field)

OOP numerics: language choice tradeoffs

capabilities for representing blackboard abstractions:

- C++, Python and Fortran provide comparable functionalities for compact representation of mathematical abstractions thanks to:
  - loop-free array arithmetics
  - array-valued expressions and functions
  - indirections allowing permuting array indices
  - fractional indexing through operator overloading
  - ...
- Fortran's limitations:
  - no built-in mechanism for code reuse on different types (templates in C++, duck-typing in Python)
  - no function calls on lhs
  - no array's of array's (e.g. components of a vector field)

OOP numerics: language choice tradeoffs

performance:

- CPU times & memory consumption: no single winner
- Blitz++ performance on par with Fortran (GCC, same options)
- significant performance gain when switching from CPython to PyPy
- what if coding and maintenance time/cost taken into account?

Python	ca. 200 LOC
C++	ca. 300 LOC
Fortran	ca. 500 LOC

OOP numerics: language choice tradeoffs

performance:

- CPU times & memory consumption: no single winner
- Blitz++ performance on par with Fortran (GCC, same options)
- significant performance gain when switching from CPython to PyPy
- what if coding and maintenance time/cost taken into account?

Python	ca. 200 LOC
C++	ca. 300 LOC
Fortran	ca. 500 LOC

OOP numerics: language choice tradeoffs

performance:

- CPU times & memory consumption: no single winner
- Blitz++ performance on par with Fortran (GCC, same options)
- significant performance gain when switching from CPython to PyPy
- what if coding and maintenance time/cost taken into account?

Python	ca. 200 LOC
C++	ca. 300 LOC
Fortran	ca. 500 LOC

OOP numerics: language choice tradeoffs

performance:

- CPU times & memory consumption: no single winner
- Blitz++ performance on par with Fortran (GCC, same options)
- significant performance gain when switching from CPython to PyPy
- what if coding and maintenance time/cost taken into account?

Python	ca. 200 LOC
C++	ca. 300 LOC
Fortran	ca. 500 LOC

OOP numerics: language choice tradeoffs

ease of use and abuse:

- syntax brevity (e.g. LOC, # distinct keywords): Python wins
- learning curve steepness, abuse-resistance: Python wins
- debugging issues:
  - C++: indecipherable compiler messages (templates)
  - Fortran: immature OOP support in compilers and debuggers
- performance predictability (no temporary objects):
  - C++/Blitz++: by design
  - Python: interpreter dependant
  - Fortran: compiler dependant

OOP numerics: language choice tradeoffs

ease of use and abuse:

- syntax brevity (e.g. LOC, # distinct keywords): Python wins
- learning curve steepness, abuse-resistance: Python wins
- debugging issues:
  - C++: indecipherable compiler messages (templates)
  - Fortran: immature OOP support in compilers and debuggers
- performance predictability (no temporary objects):
  - C++/Blitz++: by design
  - Python: interpreter dependant
  - Fortran: compiler dependant

OOP numerics: language choice tradeoffs

ease of use and abuse:

- syntax brevity (e.g. LOC, # distinct keywords): Python wins
- learning curve steepness, abuse-resistance: Python wins
- debugging issues:
  - C++: indecipherable compiler messages (templates)
  - Fortran: immature OOP support in compilers and debuggers
- performance predictability (no temporary objects):
  - C++/Blitz++: by design
  - Python: interpreter dependant
  - Fortran: compiler dependant

OOP numerics: language choice tradeoffs

ease of use and abuse:

- syntax brevity (e.g. LOC, # distinct keywords): Python wins
- learning curve steepness, abuse-resistance: Python wins
- debugging issues:
  - C++: indecipherable compiler messages (templates)
  - Fortran: immature OOP support in compilers and debuggers
- performance predictability (no temporary objects):
  - C++/Blitz++: by design
  - Python: interpreter dependant
  - Fortran: compiler dependant

OOP numerics: language choice tradeoffs

other issues:

- programmers' community size: C++ and Python win
  - $\rightsquigarrow$ trained personnel
  - $\rightsquigarrow$ reusable software components
  - $\rightsquigarrow$ information resources
- issues with Fortran:
  - OOP features did not gain popularity among users
  - ... neither among library authors
  - ... neither among standard library authors
  - no longer routinely taught at the universities
  - lack of standard exception handling mechanism
- issues with C++:
  - C++ community $\neq$ Blitz++ community
  - numerous alternative to Blitz++ (in contrast to NumPy)
- issues with Python:
  - tricky to leverage multi-core CPUs through shared-memory parallelisation

OOP numerics: language choice tradeoffs

other issues:

- programmers' community size: C++ and Python win
  - $\rightsquigarrow$ trained personnel
  - $\rightsquigarrow$ reusable software components
  - $\rightsquigarrow$ information resources
- issues with Fortran:
  - OOP features did not gain popularity among users
  - . . . neither among library authors
  - . . . neither among standard library authors
  - no longer routinely taught at the universities
  - lack of standard exception handling mechanism
- issues with C++:
  - C++ community $\neq$ Blitz++ community
  - numerous alternative to Blitz++ (in contrast to NumPy)
- issues with Python:
  - tricky to leverage multi-core CPUs through shared-memory parallelisation

OOP numerics: language choice tradeoffs

other issues:

- programmers' community size: C++ and Python win
  - $\rightsquigarrow$ trained personnel
  - $\rightsquigarrow$ reusable software components
  - $\rightsquigarrow$ information resources
- issues with Fortran:
  - OOP features did not gain popularity among users
  - . . . neither among library authors
  - . . . neither among standard library authors
  - no longer routinely taught at the universities
  - lack of standard exception handling mechanism
- issues with C++:
  - C++ community $\neq$ Blitz++ community
  - numerous alternative to Blitz++ (in contrast to NumPy)
- issues with Python:
  - tricky to leverage multi-core CPUs through shared-memory parallelisation

OOP numerics: language choice tradeoffs

other issues:

- programmers' community size: C++ and Python win
  - $\rightsquigarrow$ trained personnel
  - $\rightsquigarrow$ reusable software components
  - $\rightsquigarrow$ information resources
- issues with Fortran:
  - OOP features did not gain popularity among users
  - . . . neither among library authors
  - . . . neither among standard library authors
  - no longer routinely taught at the universities
  - lack of standard exception handling mechanism
- issues with C++:
  - C++ community $\neq$ Blitz++ community
  - numerous alternative to Blitz++ (in contrast to NumPy)
- issues with Python:
  - tricky to leverage multi-core CPUs through shared-memory parallelisation

OOP numerics: language choice tradeoffs

other issues:

- programmers' community size: C++ and Python win
  - $\rightsquigarrow$ trained personnel
  - $\rightsquigarrow$ reusable software components
  - $\rightsquigarrow$ information resources
- issues with Fortran:
  - OOP features did not gain popularity among users
  - . . . neither among library authors
  - . . . neither among standard library authors
  - no longer routinely taught at the universities
  - lack of standard exception handling mechanism
- issues with C++:
  - C++ community $\neq$ Blitz++ community
  - numerous alternative to Blitz++ (in contrast to NumPy)
- issues with Python:
  - tricky to leverage multi-core CPUs through shared-memory parallelisation

OOP numerics: language choice tradeoffs

other issues:

- programmers' community size: C++ and Python win
  - $\rightsquigarrow$ trained personnel
  - $\rightsquigarrow$ reusable software components
  - $\rightsquigarrow$ information resources
- issues with Fortran:
  - OOP features did not gain popularity among users
  - ... neither among library authors
  - ... neither among standard library authors
  - no longer routinely taught at the universities
  - lack of standard exception handling mechanism
- issues with C++:
  - C++ community $\neq$ Blitz++ community
  - numerous alternative to Blitz++ (in contrast to NumPy)
- issues with Python:
  - tricky to leverage multi-core CPUs through shared-memory parallelisation

OOP numerics: language choice tradeoffs

other issues:

- programmers' community size: C++ and Python win
  - $\rightsquigarrow$ trained personnel
  - $\rightsquigarrow$ reusable software components
  - $\rightsquigarrow$ information resources
- issues with Fortran:
  - OOP features did not gain popularity among users
  - ... neither among library authors
  - ... neither among standard library authors
  - no longer routinely taught at the universities
  - lack of standard exception handling mechanism
- issues with C++:
  - C++ community $\neq$ Blitz++ community
  - numerous alternative to Blitz++ (in contrast to NumPy)
- issues with Python:
  - tricky to leverage multi-core CPUs through shared-memory parallelisation

work in progress & future plans

mpdata-oop C++, Py, F08 codes presented in the arXiv paper

use: benchmarking, didactics

repo: `http://github.com/slayoo/mpdata`

libmpdata++ C++ library of parallel MPDATA-based solvers

deps: Blitz++, OpenMP/Boost.Thread, Boost.MPI

use: dynamical core for models of geophysical flows

repo: `http://github.com/slayoo/advoocat`

libcmphscs++ C++ library of cloud μ-physics algorithms

deps: Blitz++, Thrust (GPU/OpenMP)

use: cloud/aerosol/precipitation μ-physics package

icicle 2D model of aerosol-cloud-aerosol interactions

deps: libmpdata++, libcmphscs++, ...

icicLES OOP LES system for cloud-physics research

deps: libmpdata++, libcmphscs++, ...

news: Polish National Science Centre funds granted!

plan: 3 years, ca. 3 full-time positions (50/50 sci/dev),
co-op: Wojciech Grabowski, Piotr Smolarkiewicz

all GPL-licensed

work in progress & future plans

mpdata-oop C++, Py, F08 codes presented in the arXiv paper

use: benchmarking, didactics

repo: `http://github.com/slayoo/mpdata`

libmpdata++ C++ library of parallel MPDATA-based solvers

deps: Blitz++, OpenMP/Boost.Thread, Boost.MPI

use: dynamical core for models of geophysical flows

repo: `http://github.com/slayoo/advoocat`

libcmphscs++ C++ library of cloud μ-physics algorithms

deps: Blitz++, Thrust (GPU/OpenMP)

use: cloud/aerosol/precipitation μ-physics package

icicle 2D model of aerosol-cloud-aerosol interactions

deps: libmpdata++, libcmphscs++, ...

icicLES OOP LES system for cloud-physics research

deps: libmpdata++, libcmphscs++, ...

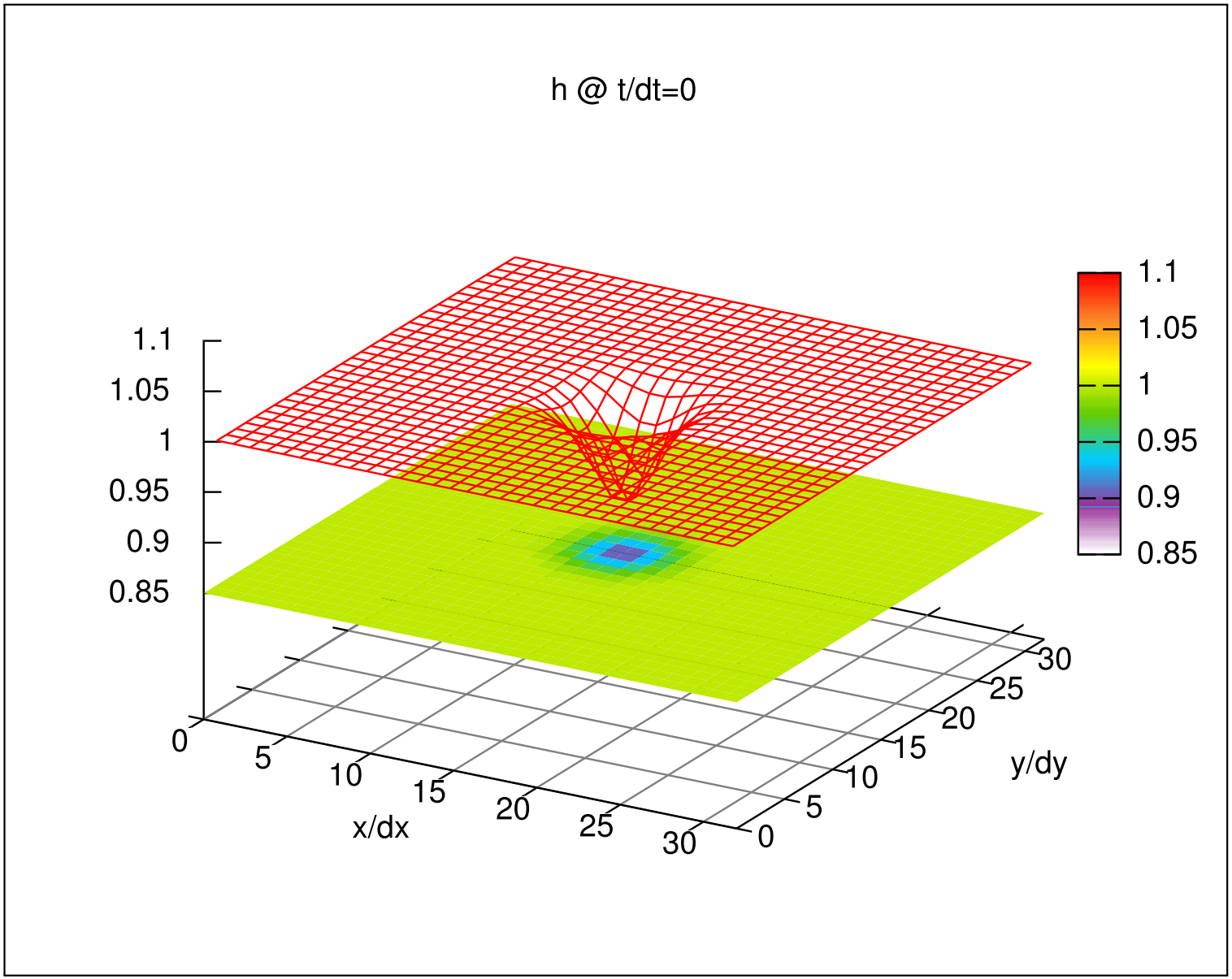
news: Polish National Science Centre funds granted!

plan: 3 years, ca. 3 full-time positions (50/50 sci/dev),
co-op: Wojciech Grabowski, Piotr Smolarkiewicz

all GPL-licensed

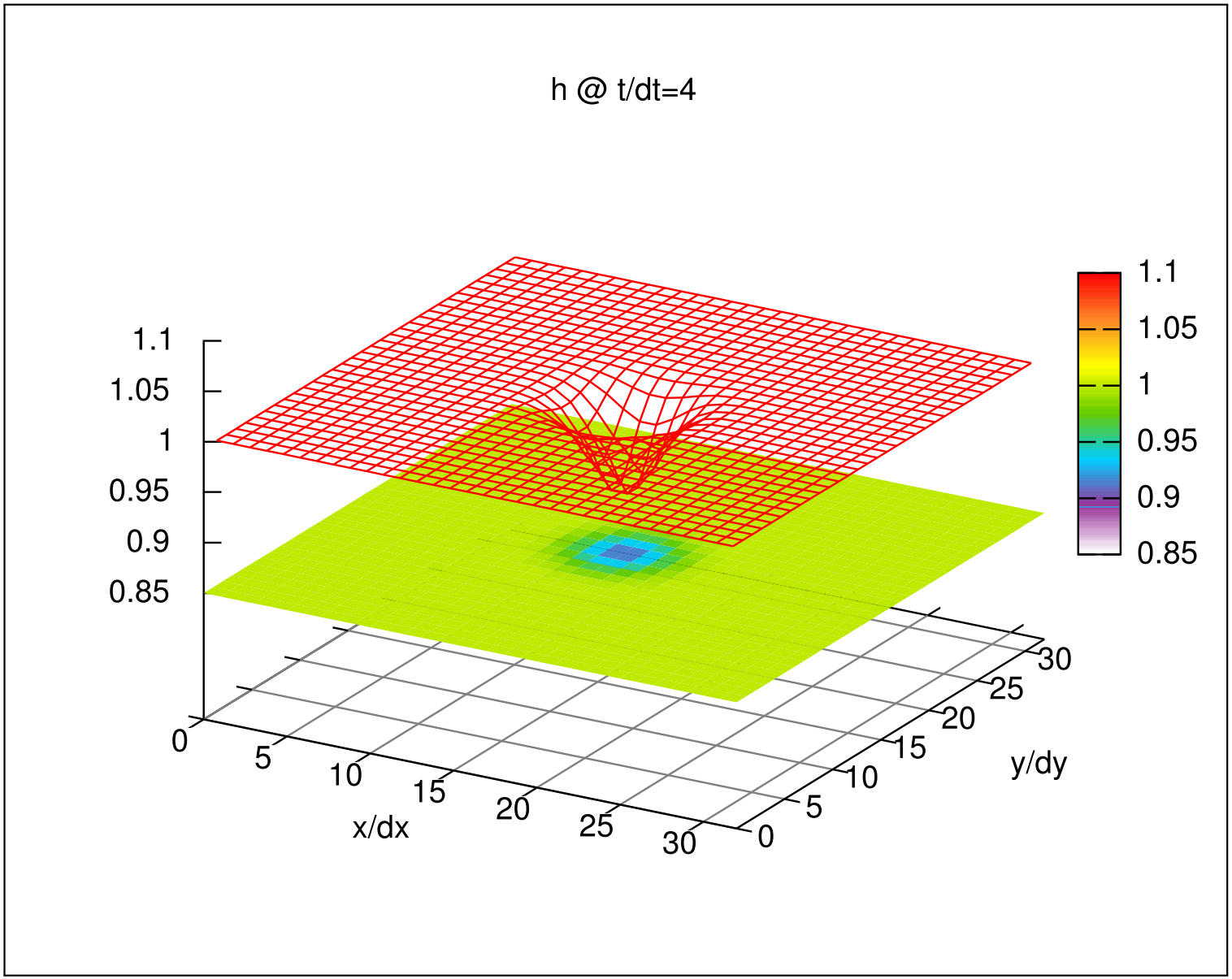
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



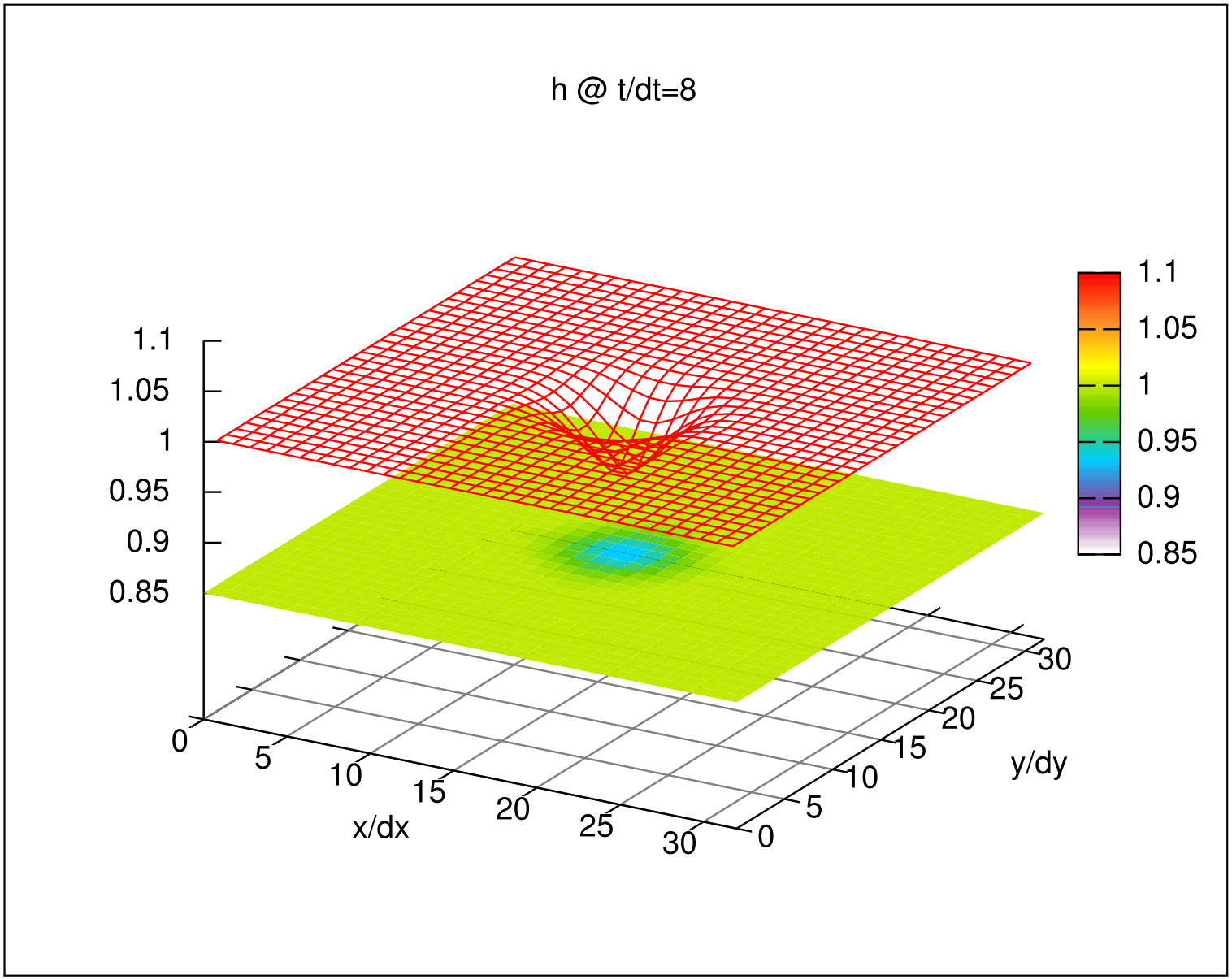
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



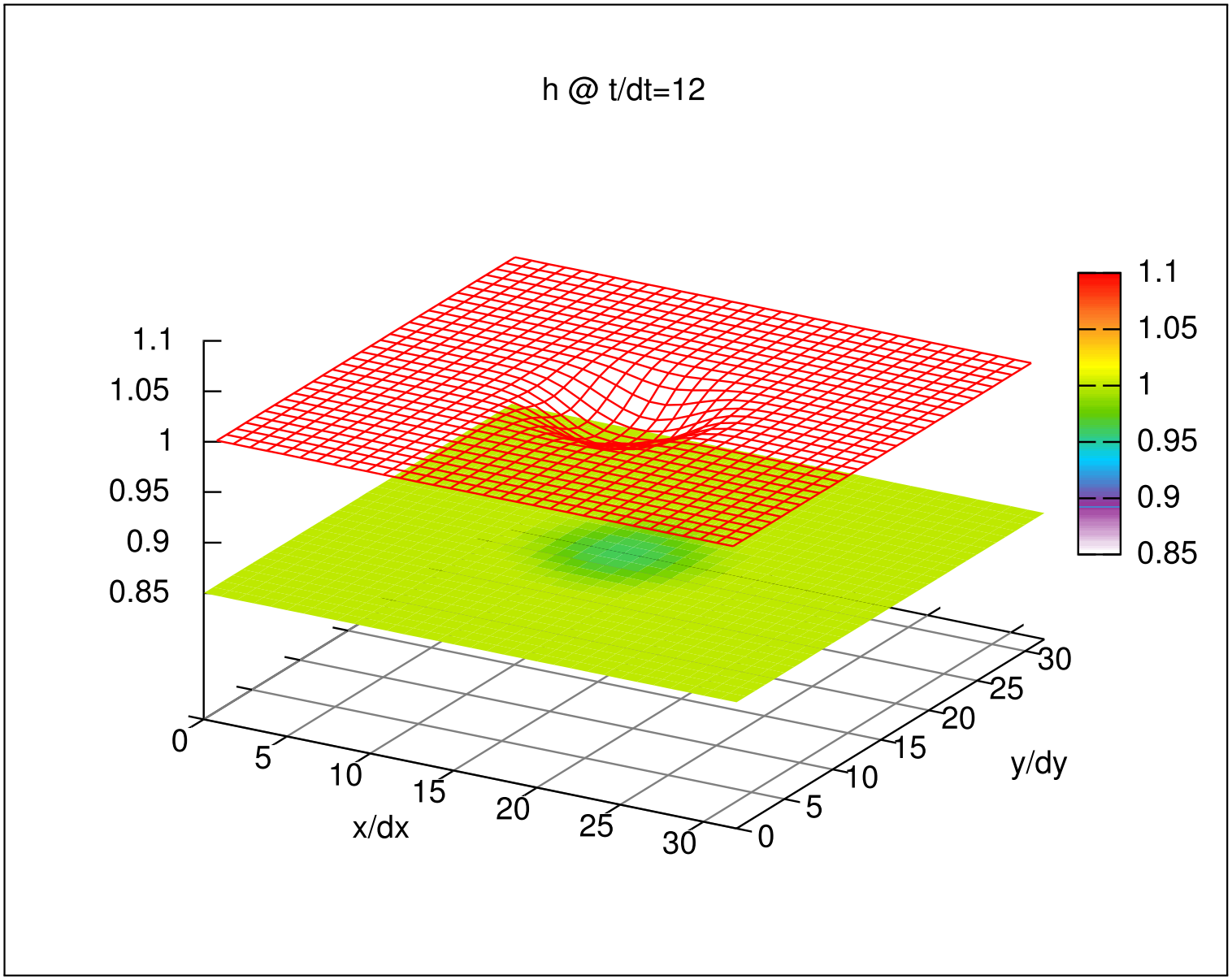
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



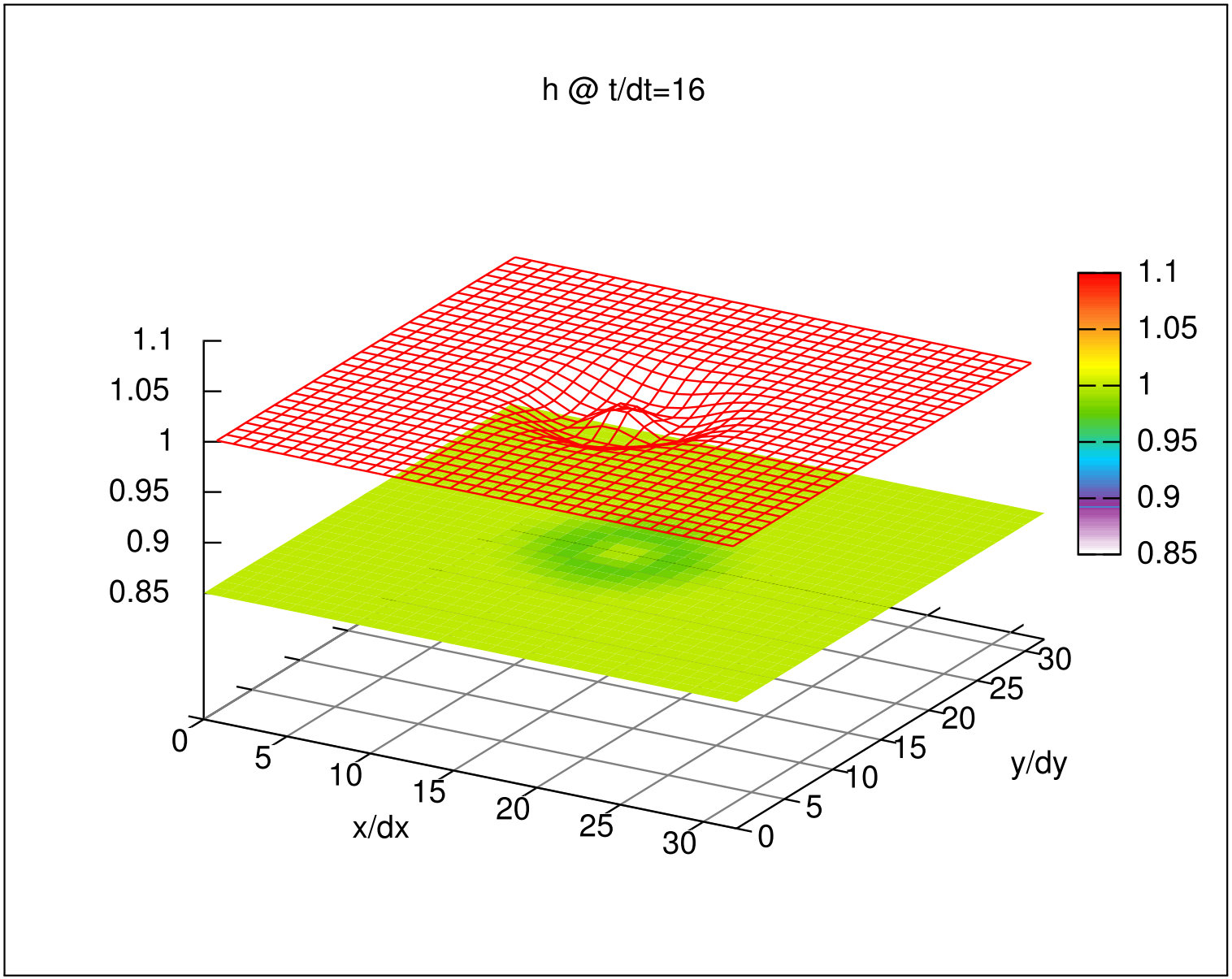
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



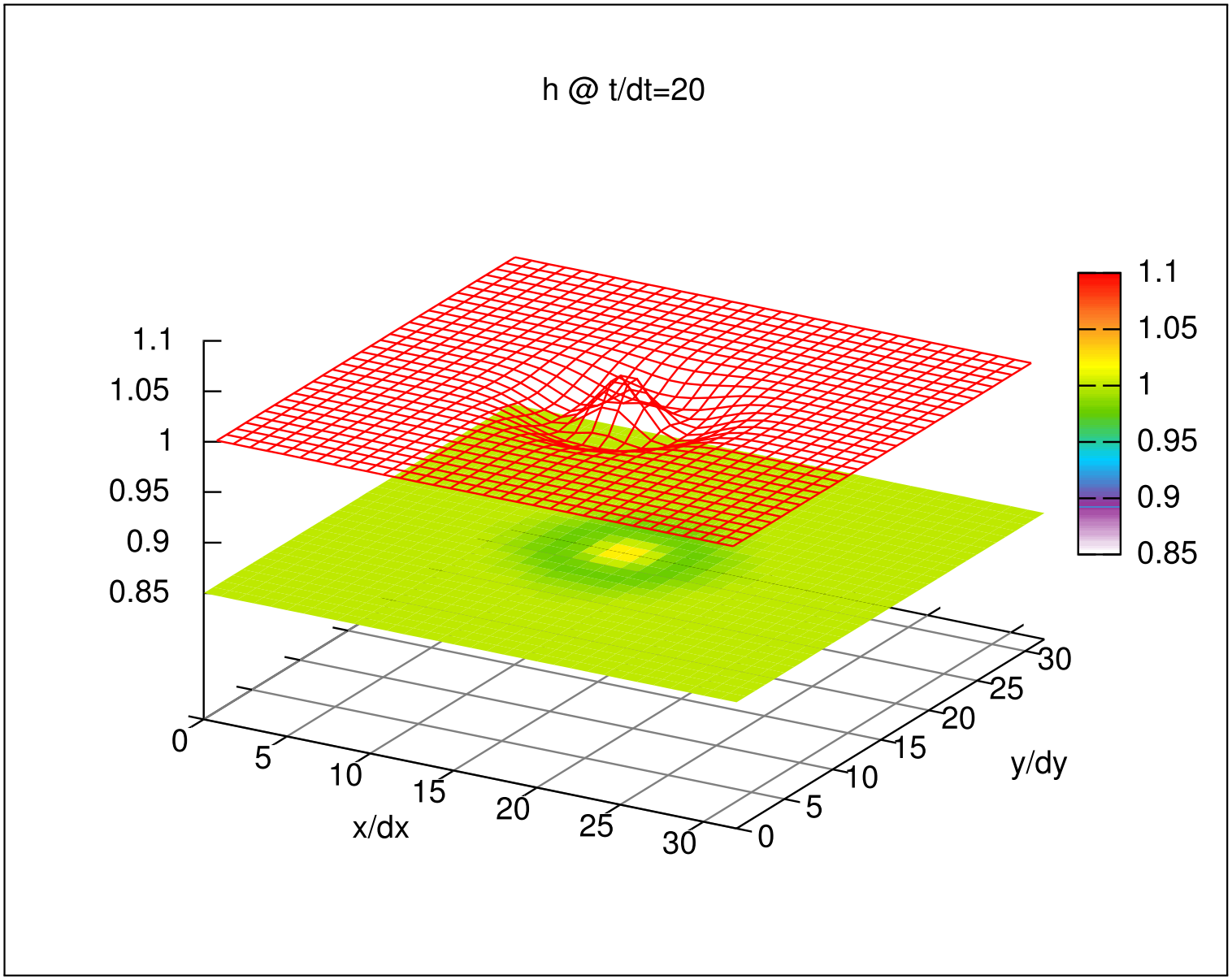
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



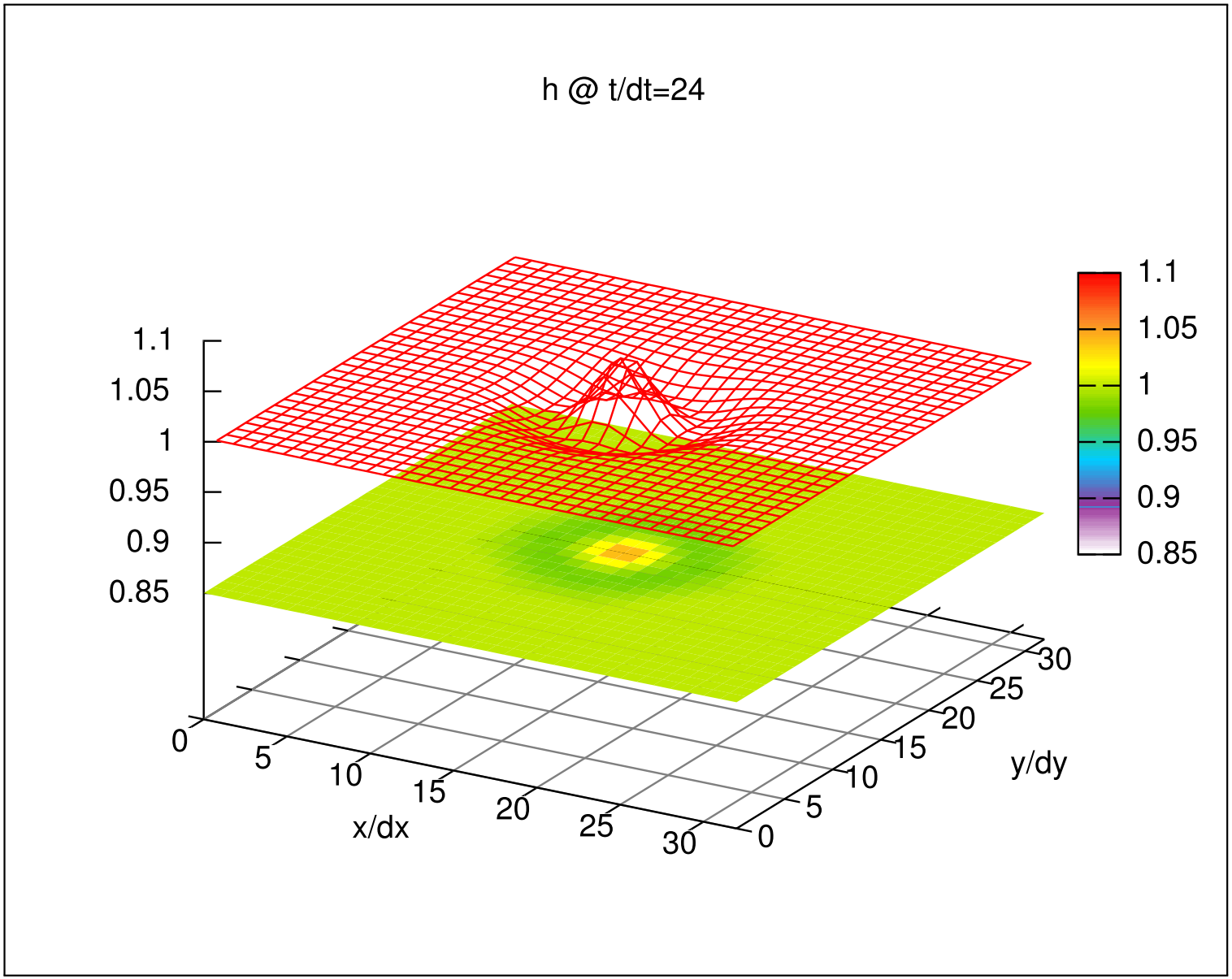
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



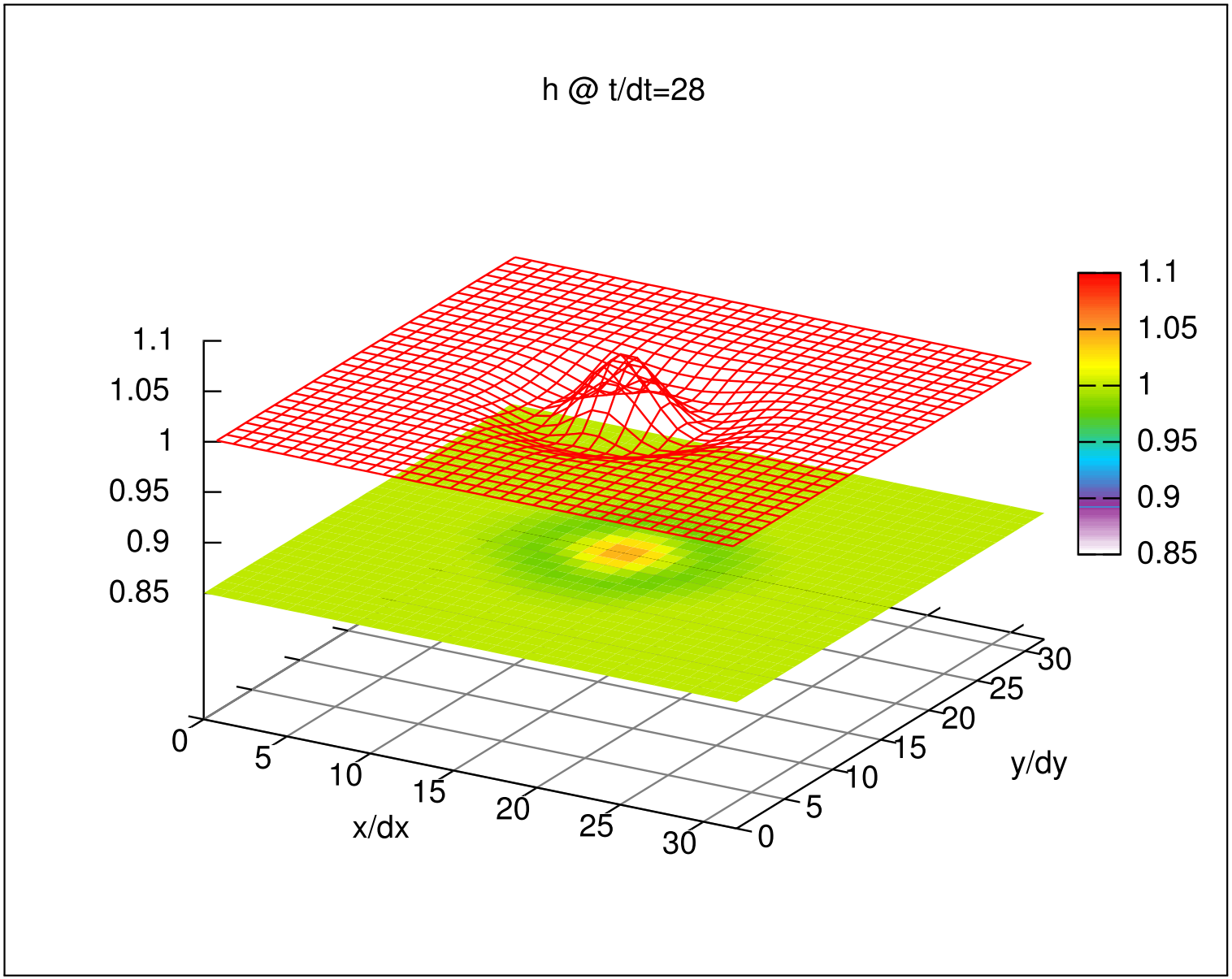
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



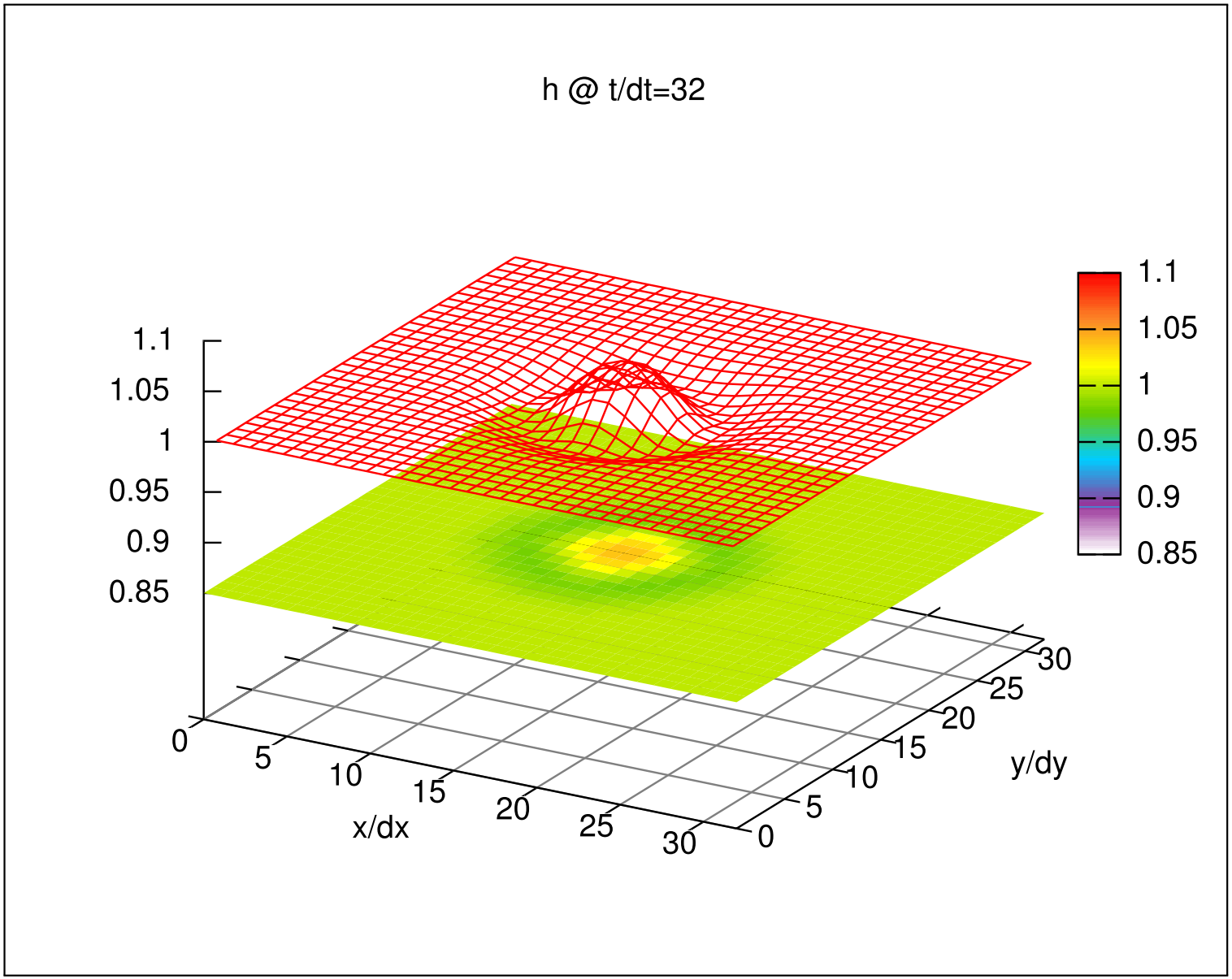
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



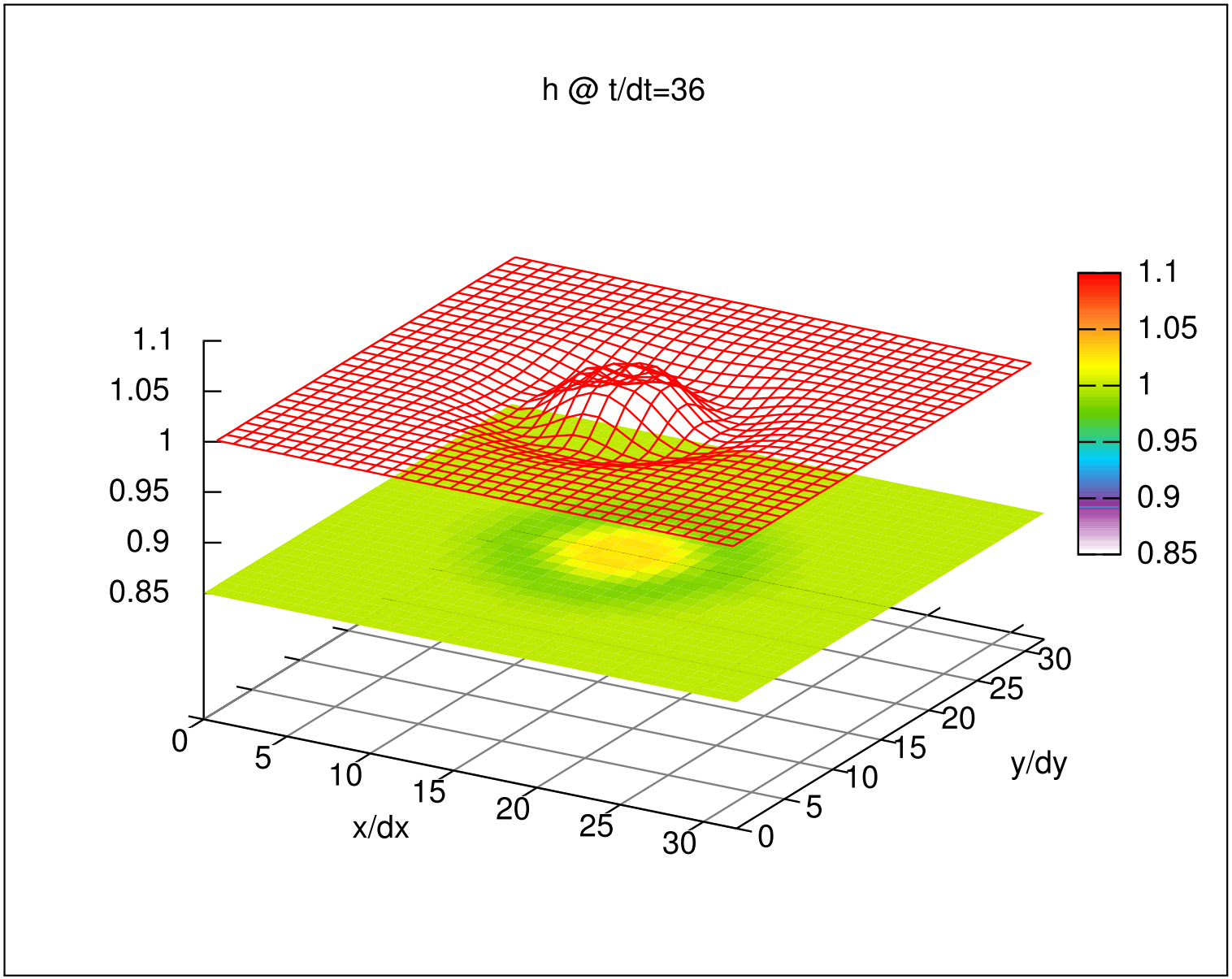
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



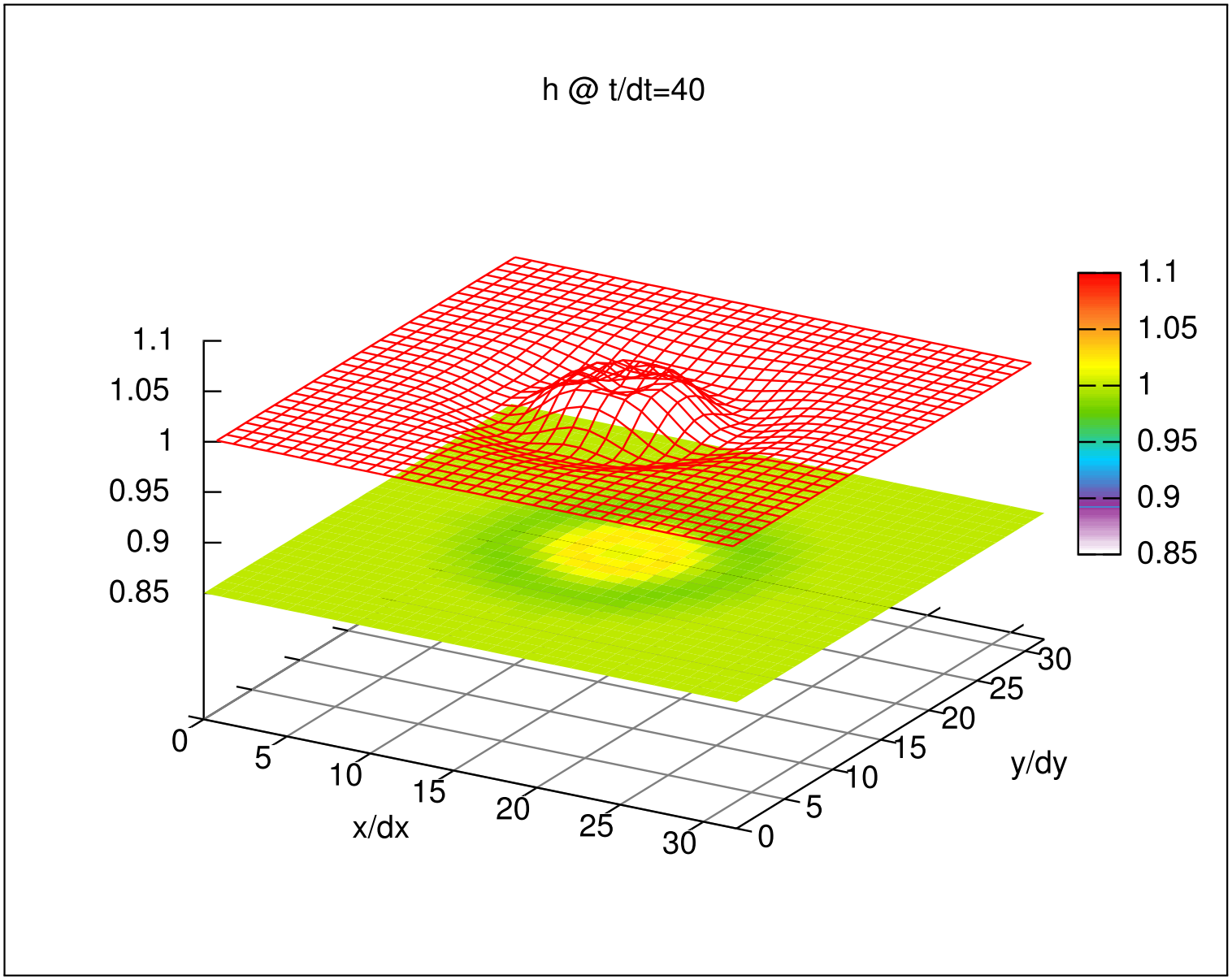
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



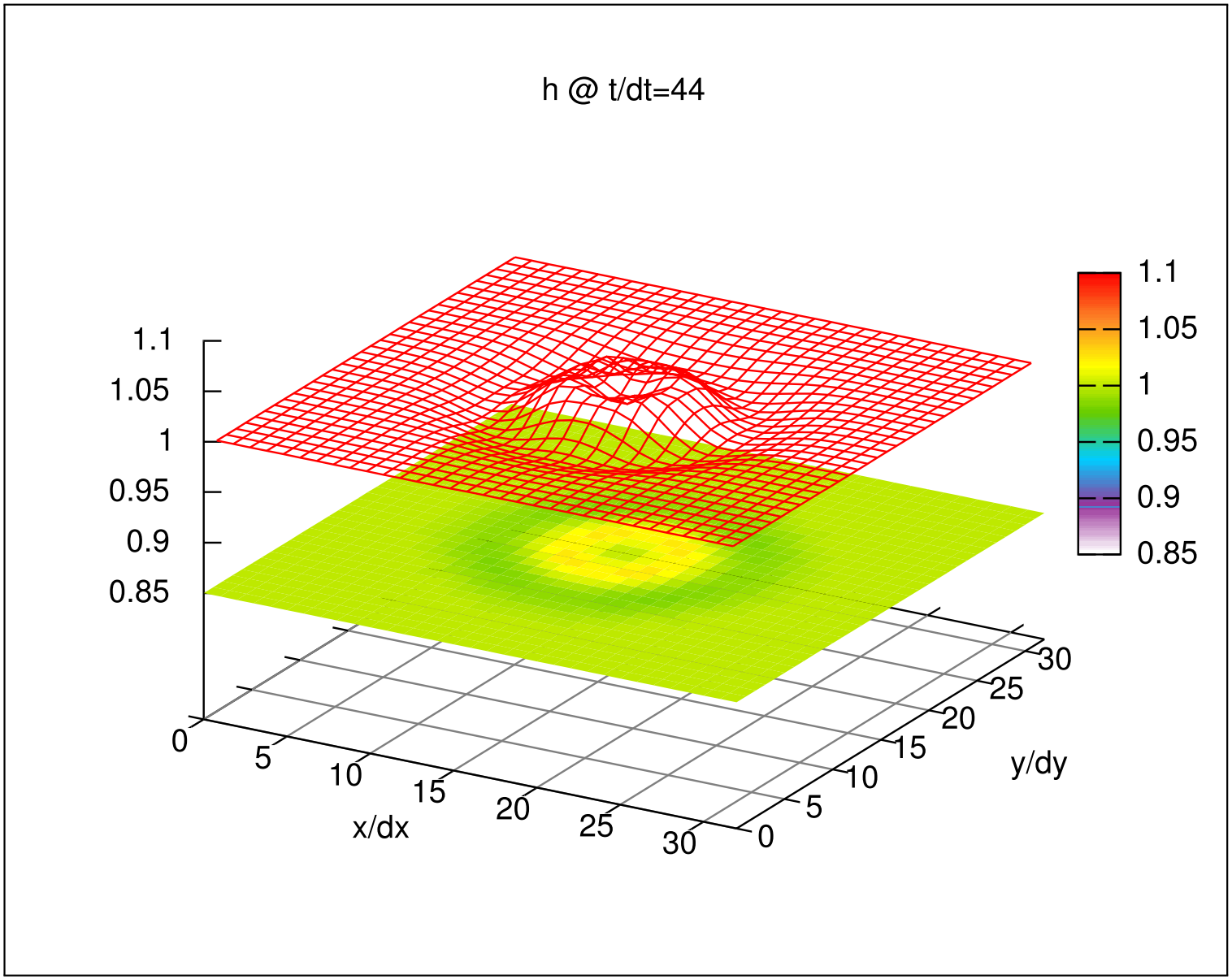
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



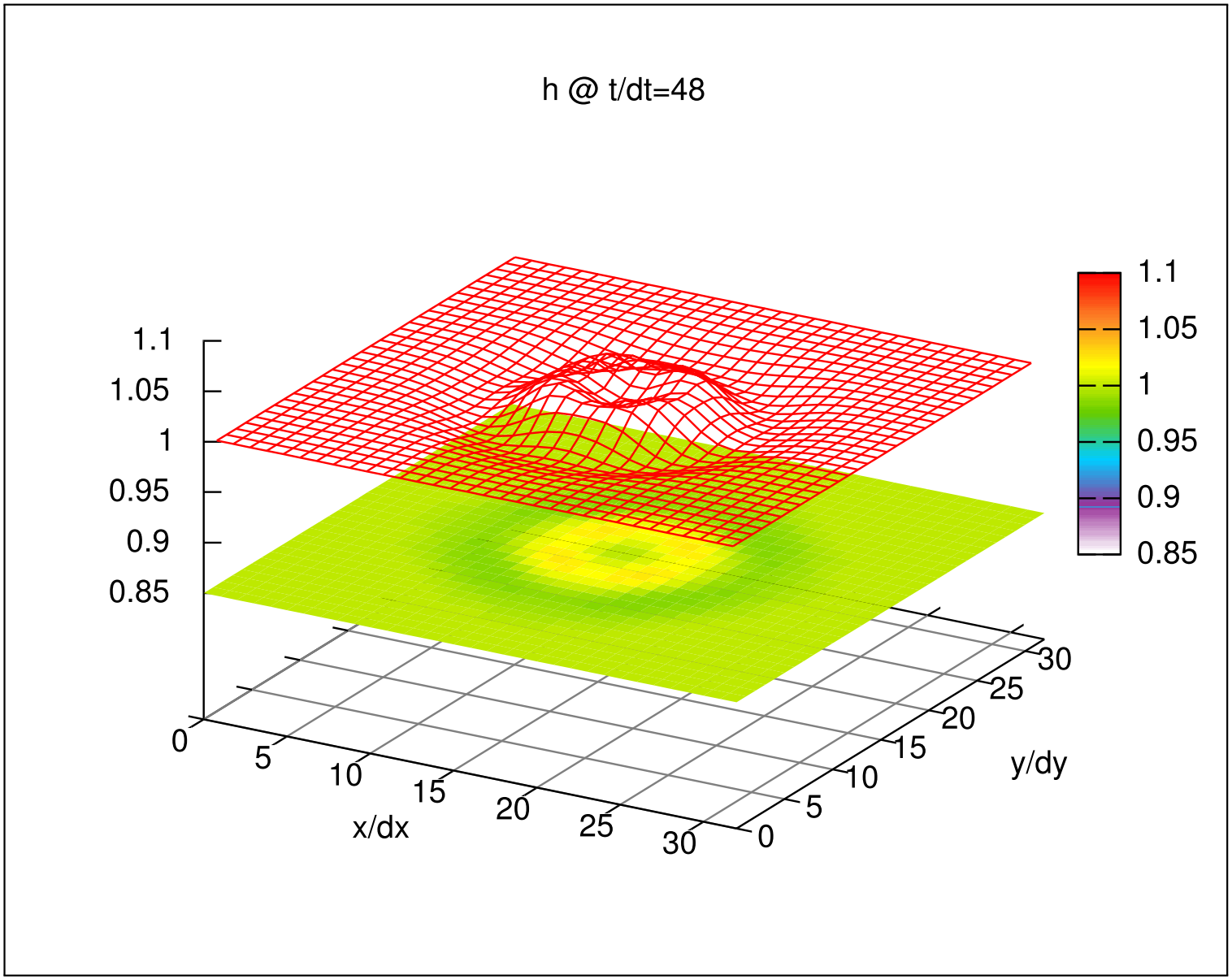
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



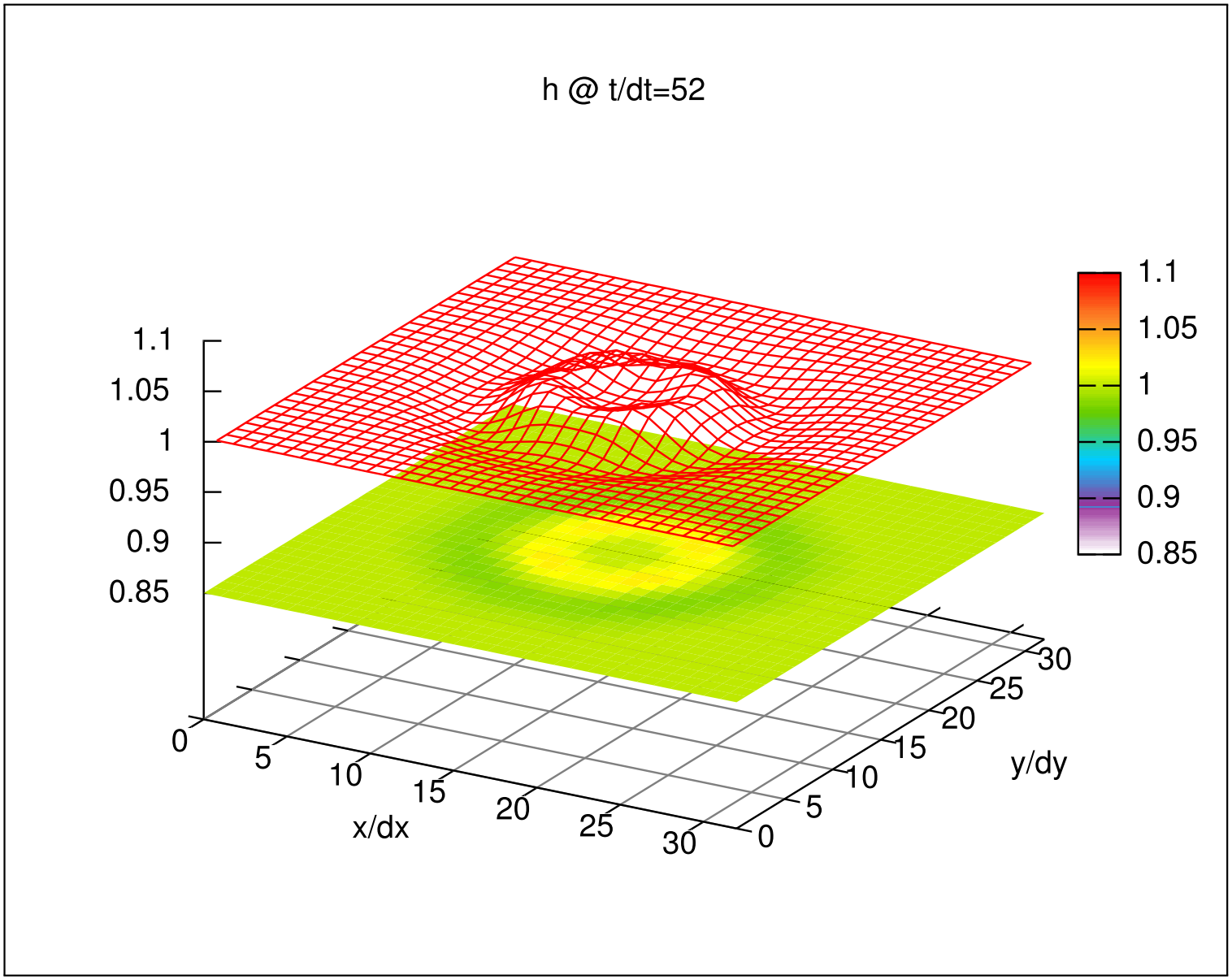
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



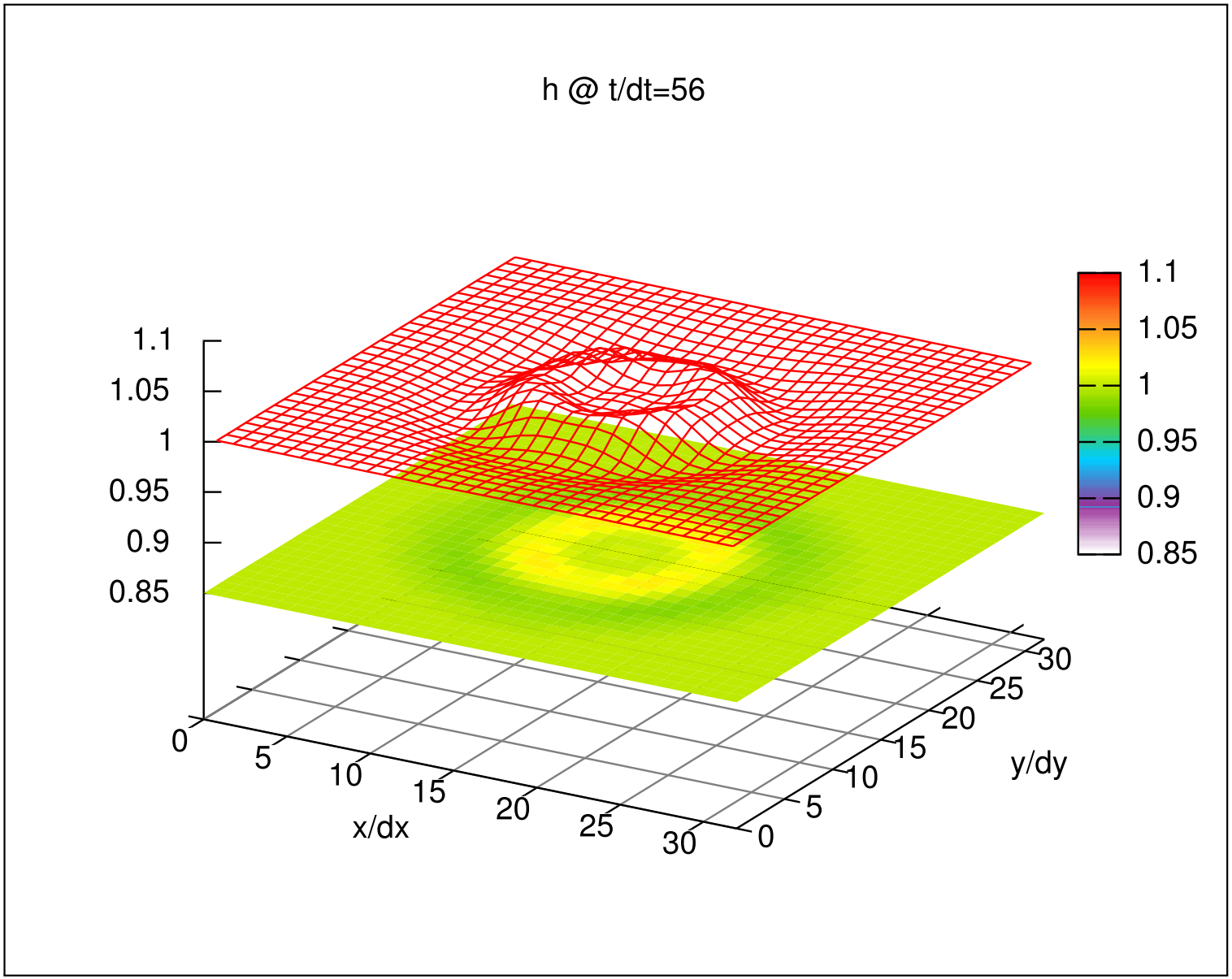
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



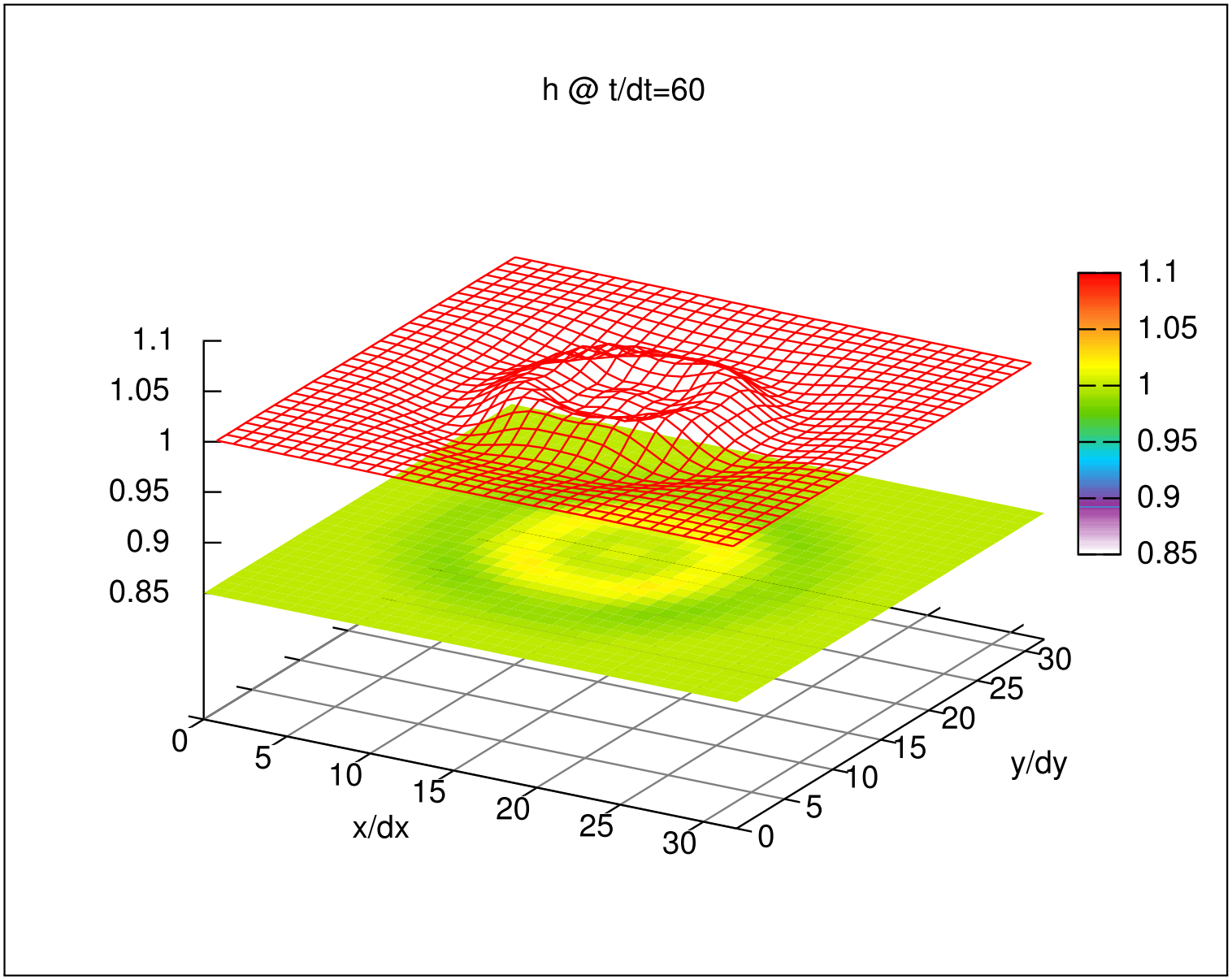
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



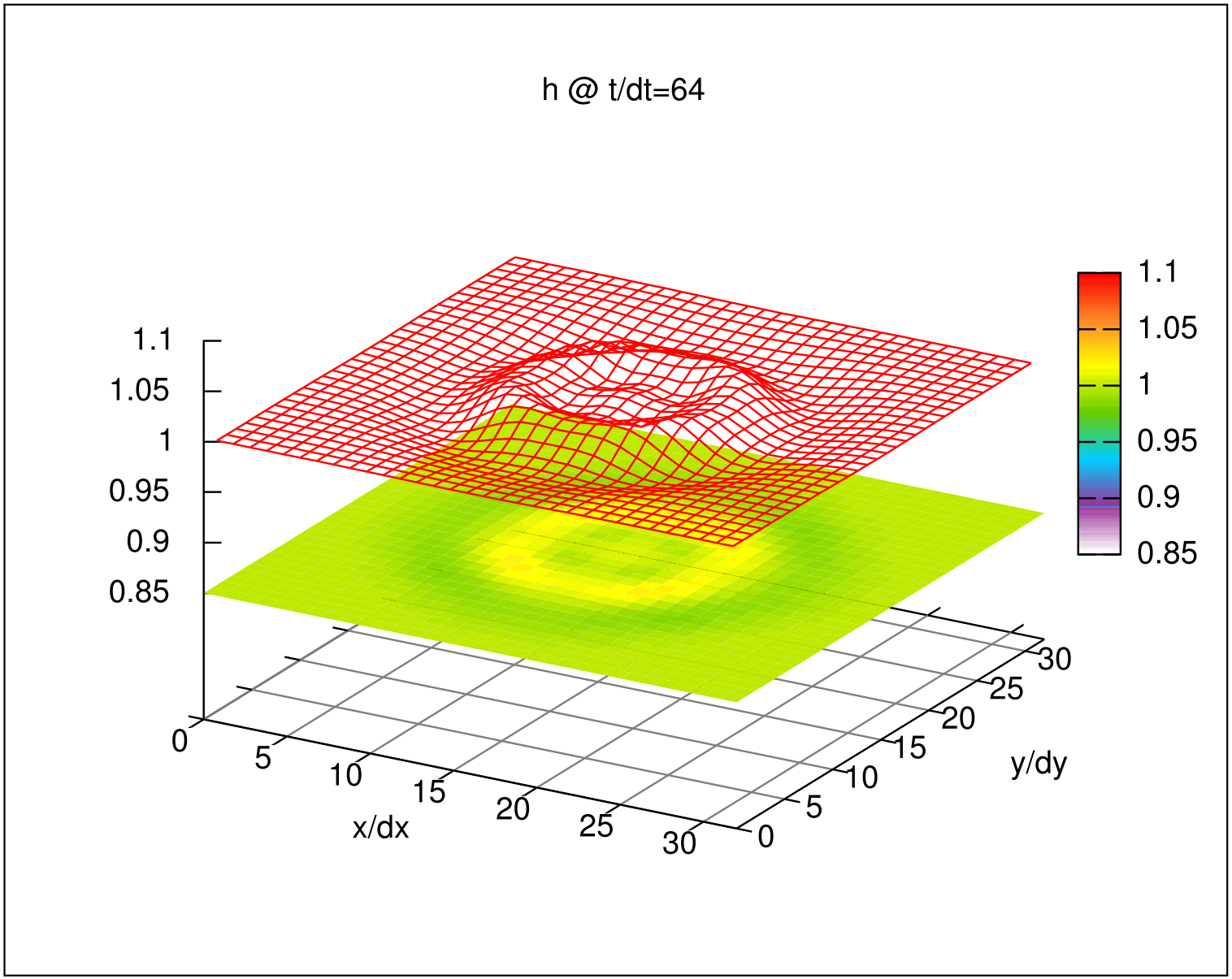
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



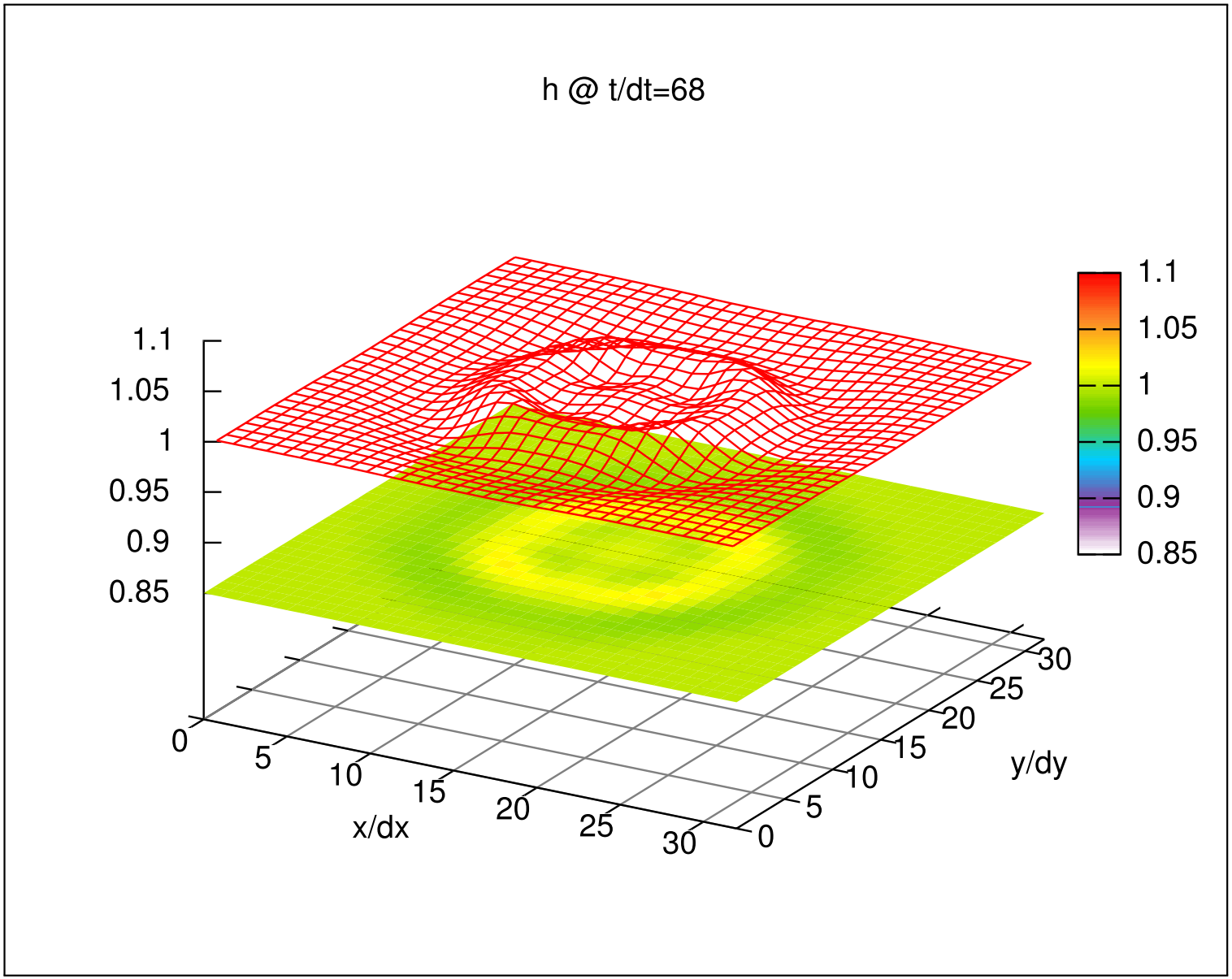
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



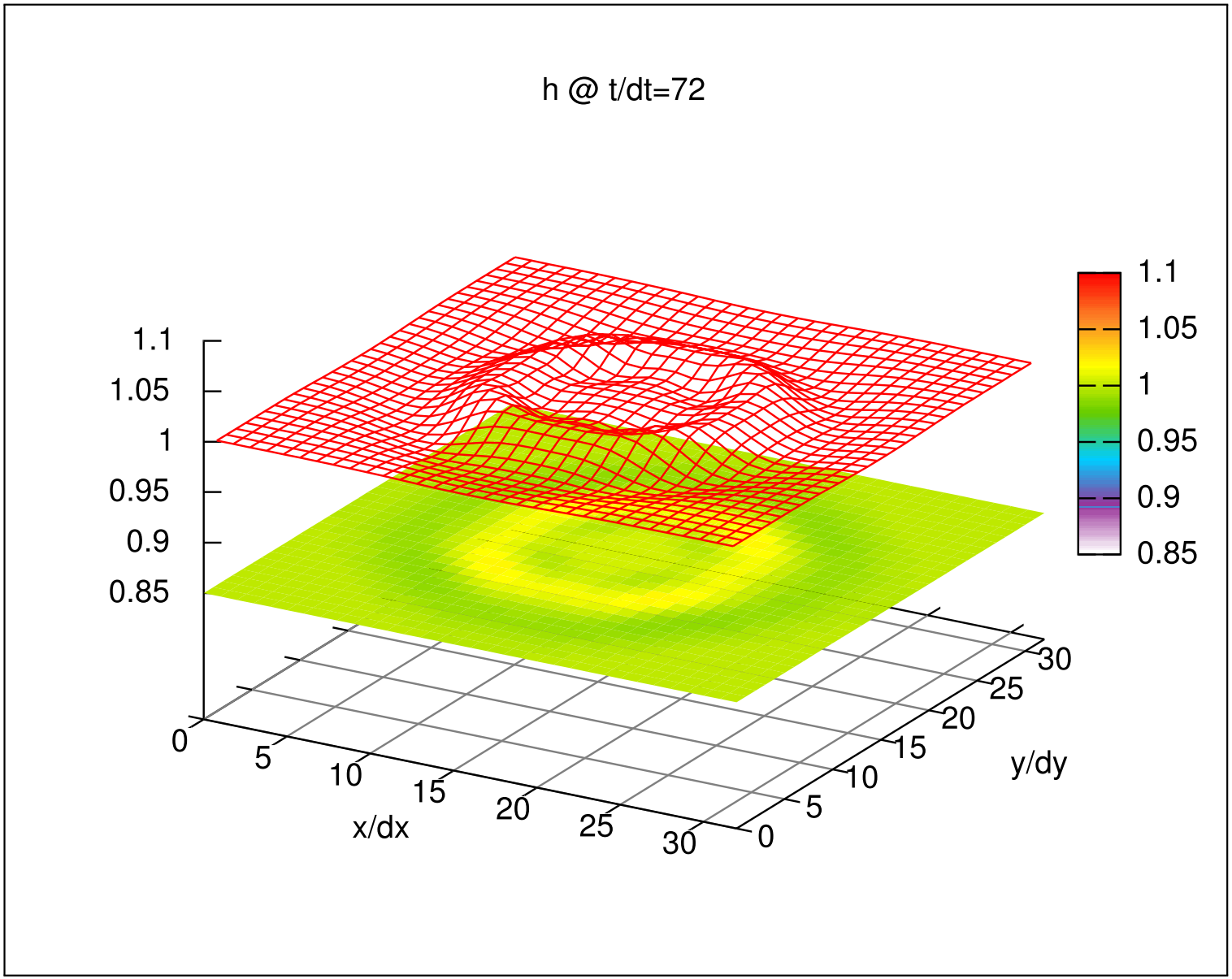
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



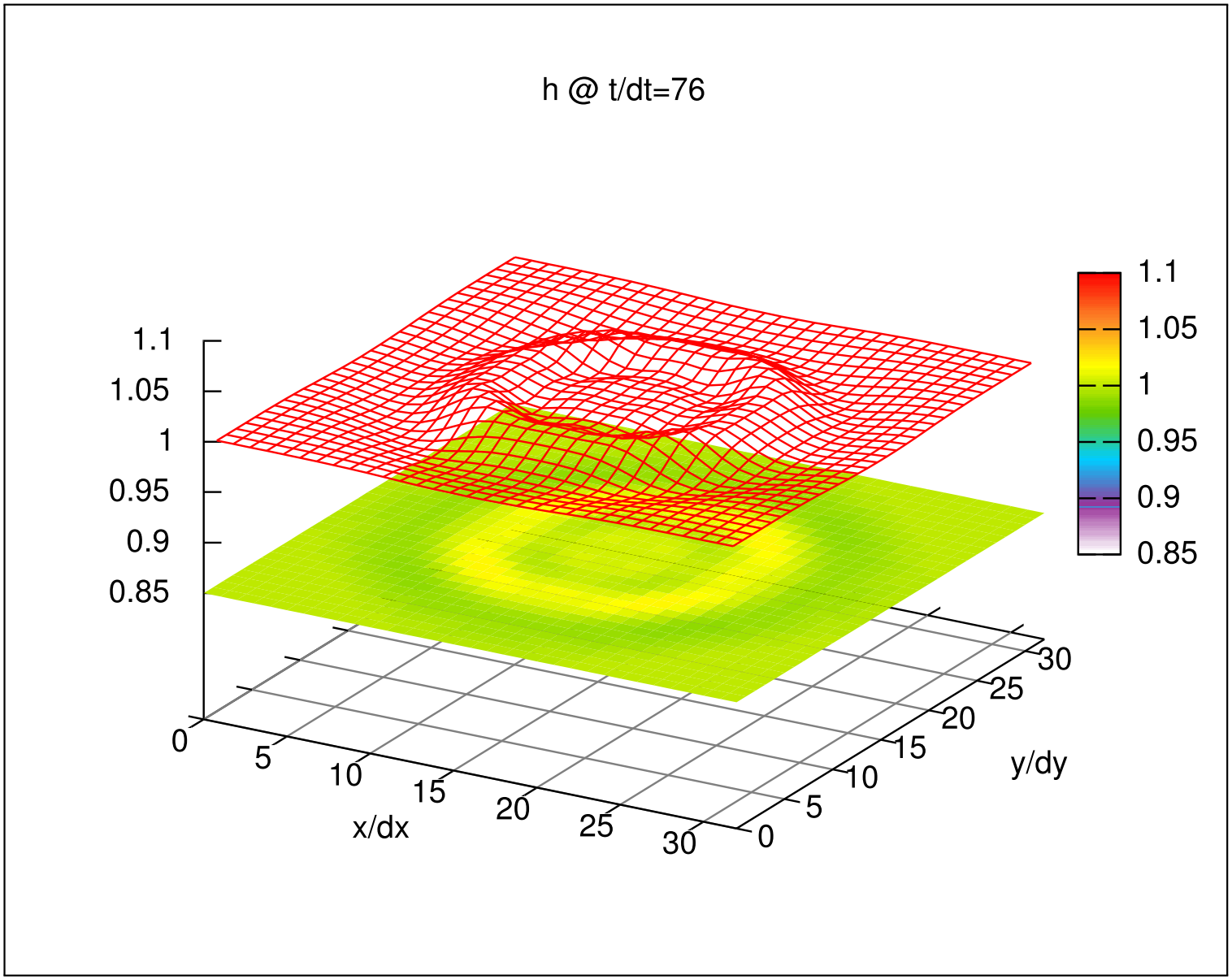
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)



work in progress & future plans

mpdata-oop C++, Py, F08 codes presented in the arXiv paper

use: benchmarking, didactics

repo: `http://github.com/slayoo/mpdata`

libmpdata++ C++ library of parallel MPDATA-based solvers

deps: Blitz++, OpenMP/Boost.Thread, Boost.MPI

use: dynamical core for models of geophysical flows

repo: `http://github.com/slayoo/advoocat`

libcmphscs++ C++ library of cloud μ-physics algorithms

deps: Blitz++, Thrust (GPU/OpenMP)

use: cloud/aerosol/precipitation μ-physics package

icicle 2D model of aerosol-cloud-aerosol interactions

deps: libmpdata++, libcmphscs++, ...

icicLES OOP LES system for cloud-physics research

deps: libmpdata++, libcmphscs++, ...

news: Polish National Science Centre funds granted!

plan: 3 years, ca. 3 full-time positions (50/50 sci/dev),
co-op: Wojciech Grabowski, Piotr Smolarkiewicz

all GPL-licensed

work in progress & future plans

mpdata-oop C++, Py, F08 codes presented in the arXiv paper

- use: benchmarking, didactics
- repo: `http://github.com/slayoo/mpdata`

libmpdata++ C++ library of parallel MPDATA-based solvers

- deps: Blitz++, OpenMP/Boost.Thread, Boost.MPI
- use: dynamical core for models of geophysical flows
- repo: `http://github.com/slayoo/advoocat`

libcmphscs++ C++ library of cloud μ-physics algorithms

- deps: Blitz++, Thrust (GPU/OpenMP)
- use: cloud/aerosol/precipitation μ-physics package

icicle 2D model of aerosol-cloud-aerosol interactions

- deps: libmpdata++, libcmphscs++, ...

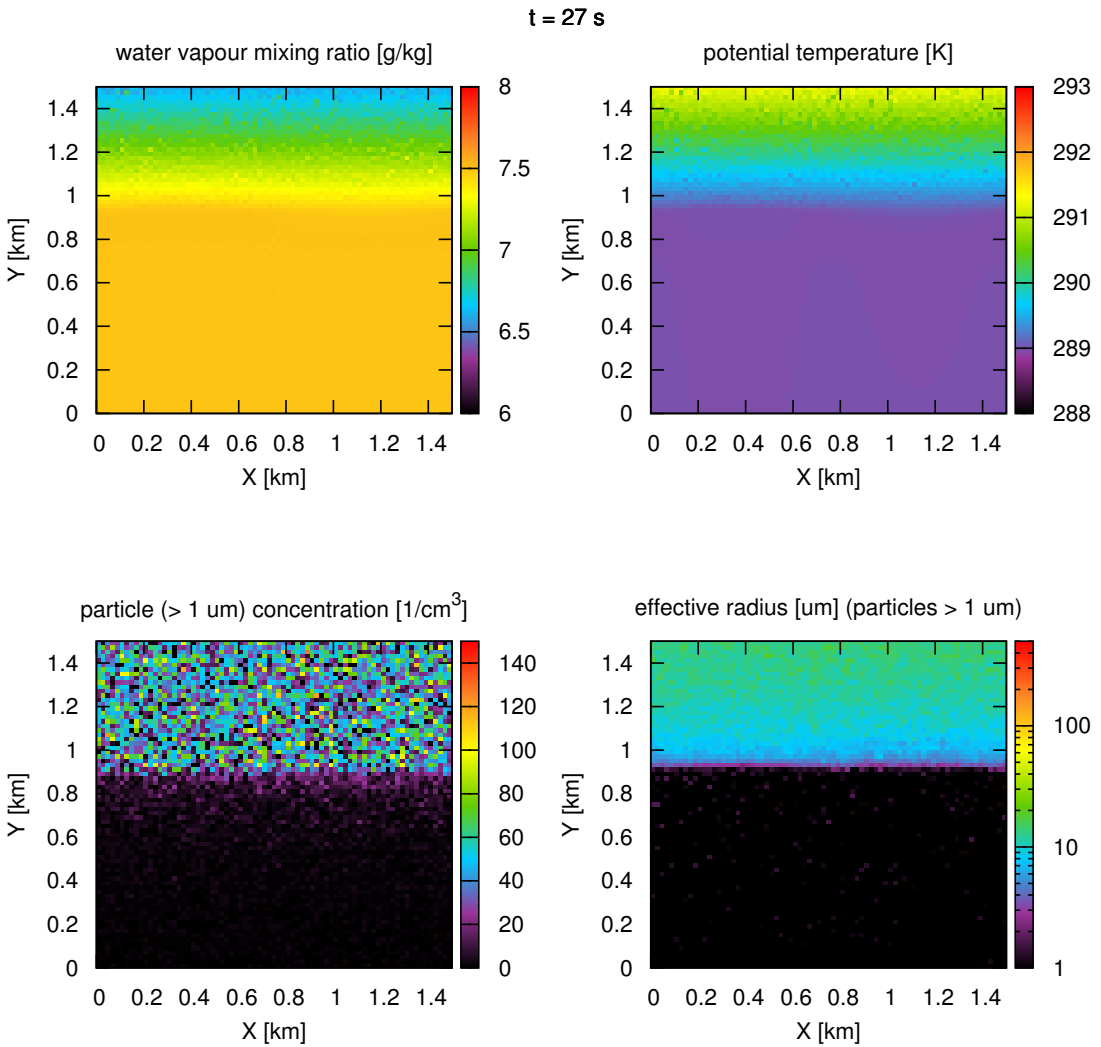
icicLES OOP LES system for cloud-physics research

- deps: libmpdata++, libcmphscs++, ...
- news: Polish National Science Centre funds granted!
- plan: 3 years, ca. 3 full-time positions (50/50 sci/dev), co-op: Wojciech Grabowski, Piotr Smolarkiewicz

all GPL-licensed

work in progress & future plans

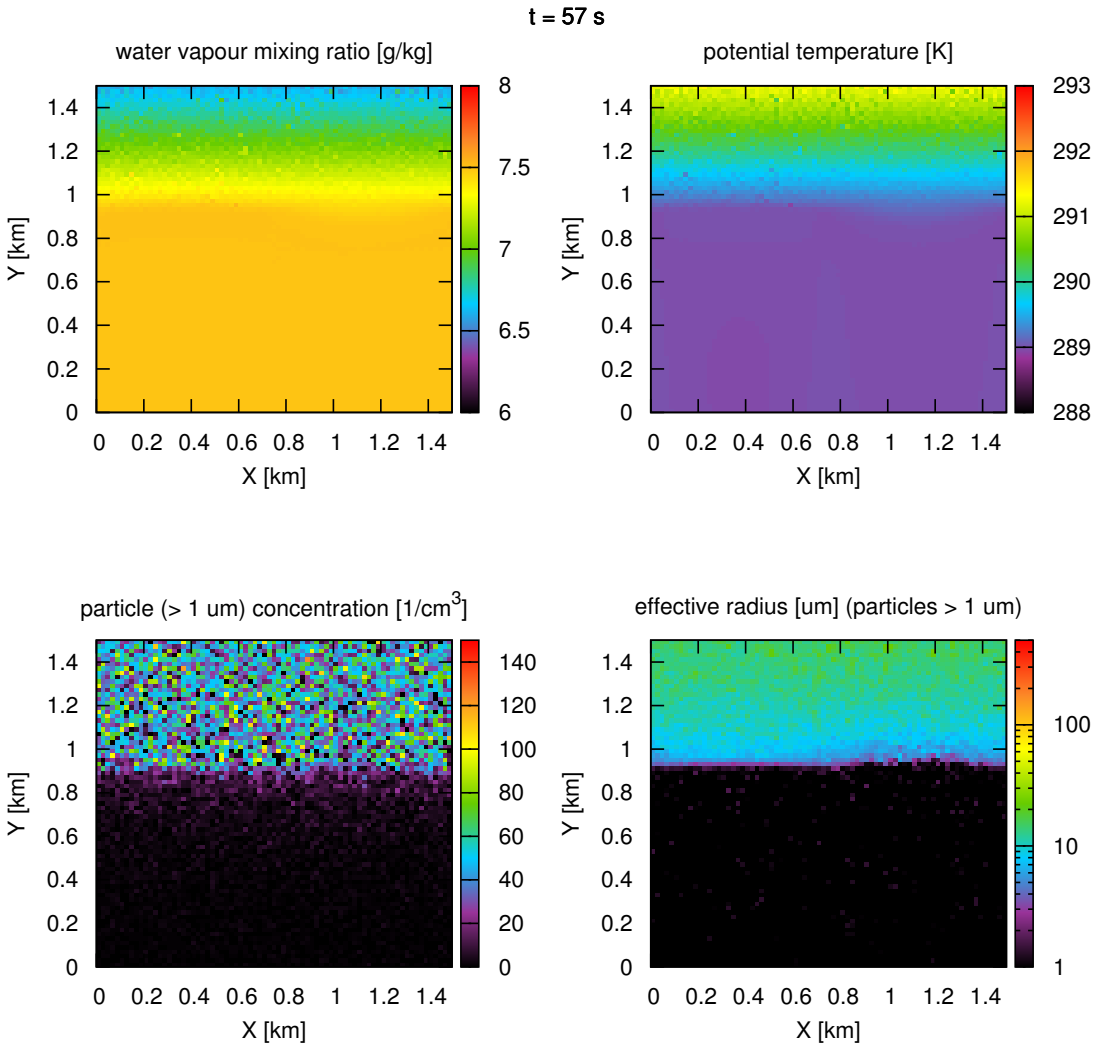
particle-based Monte-Carlo μ-physics (using Thrust)



all GPL-licensed

work in progress & future plans

particle-based Monte-Carlo μ-physics (using Thrust)

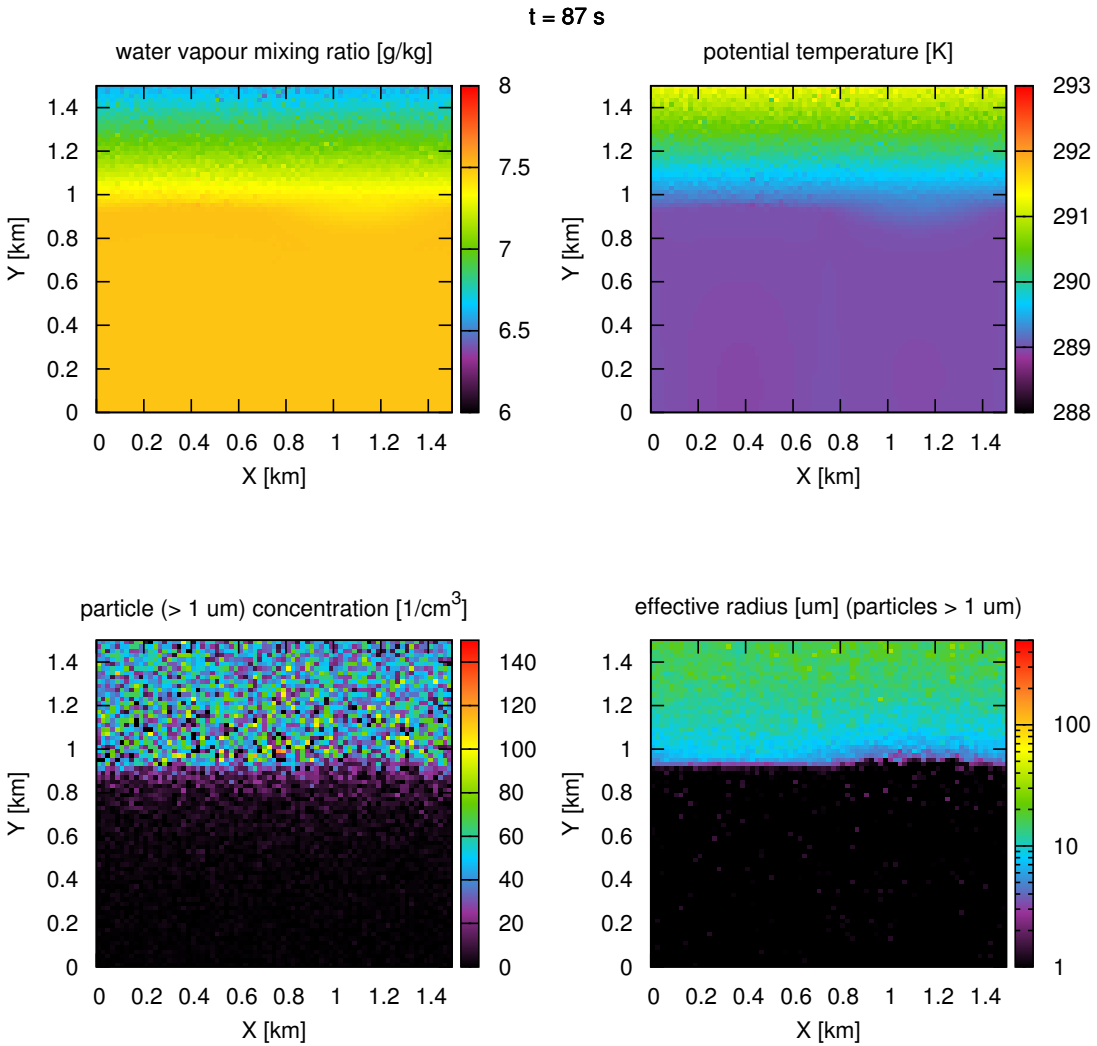


all GPL-licensed

UNIVERSITAS VARSOVIENSIS

work in progress & future plans

particle-based Monte-Carlo μ-physics (using Thrust)

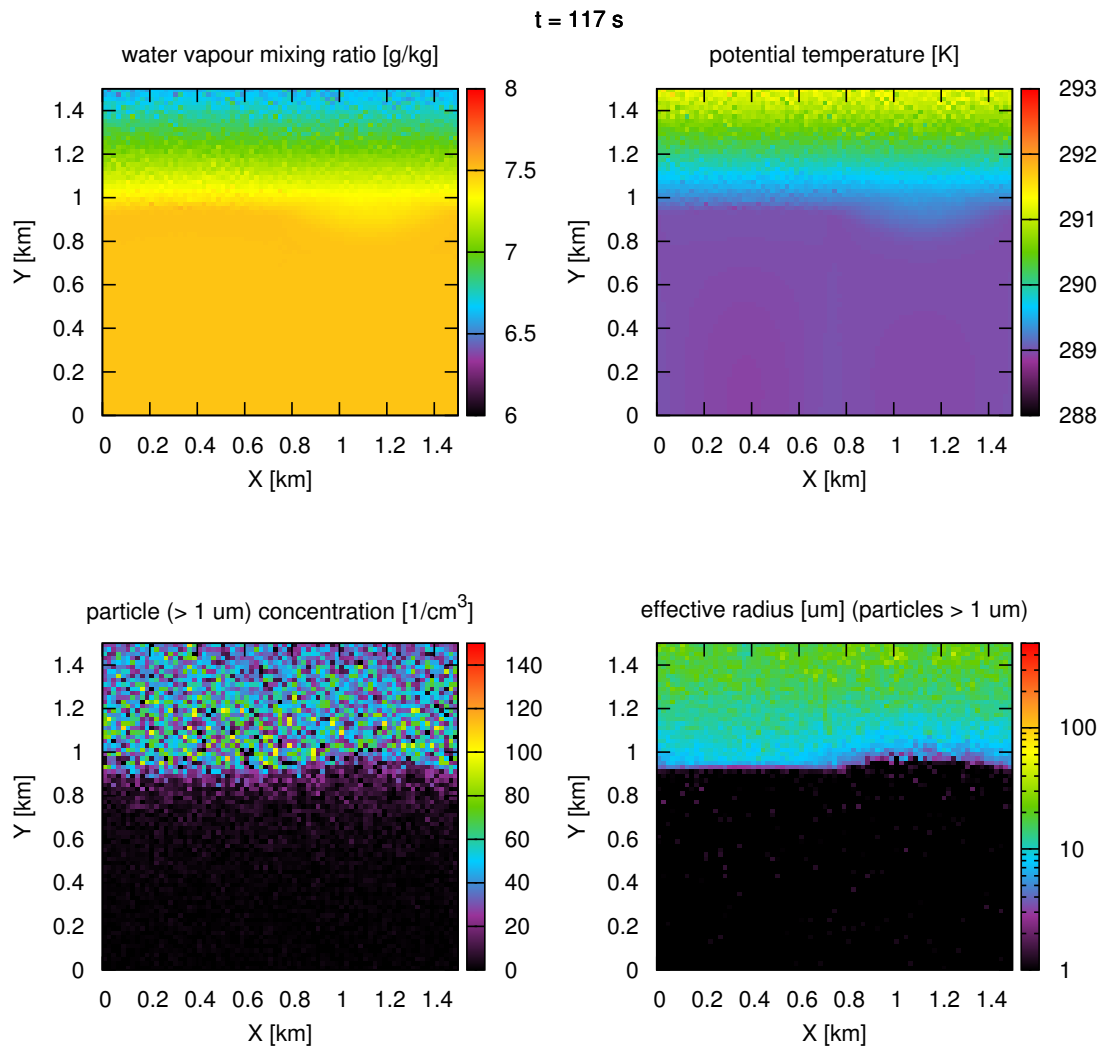


all GPL-licensed

UNIVERSITAS VARSOVIENSIS

work in progress & future plans

particle-based Monte-Carlo μ-physics (using Thrust)

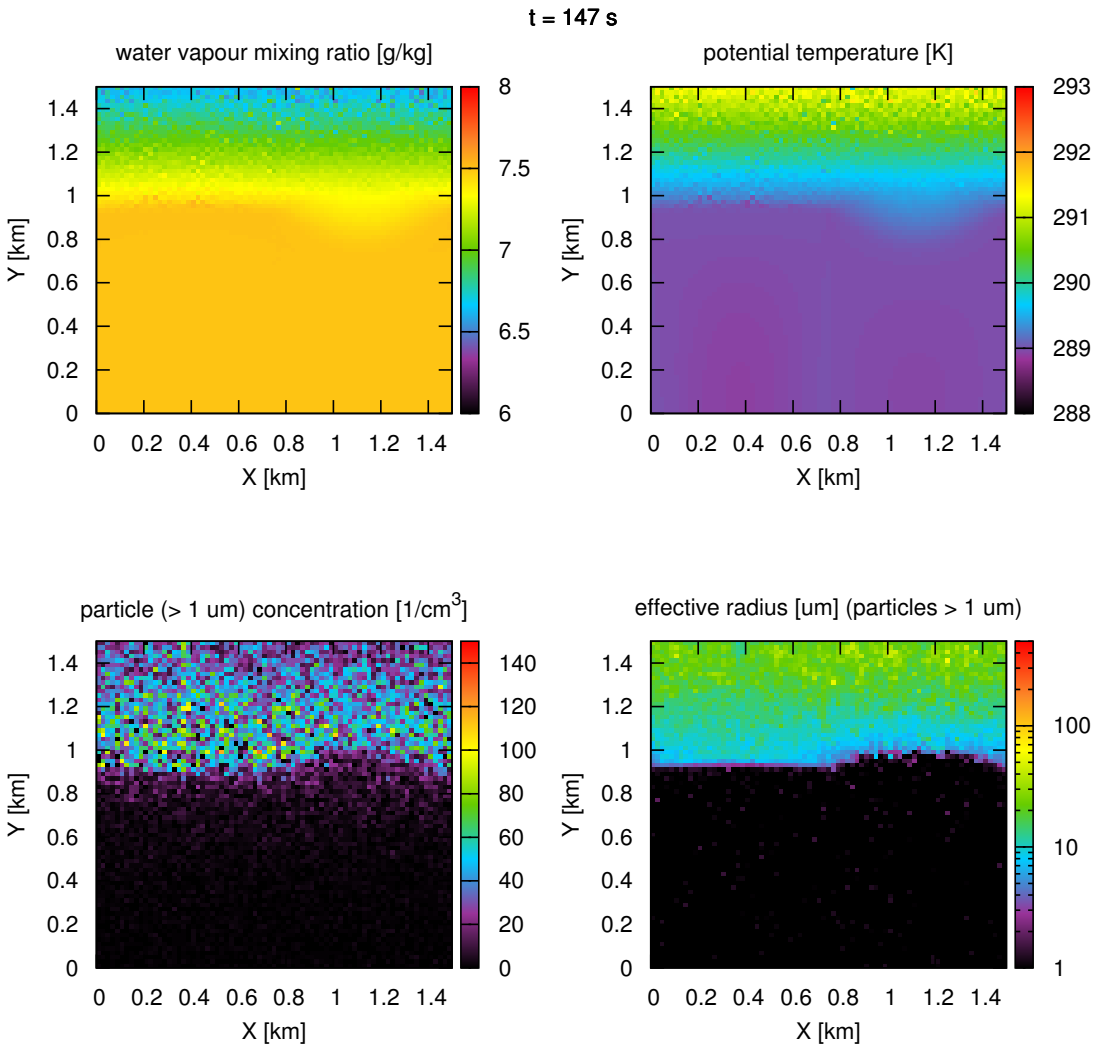


all GPL-licensed

UNIVERSITAS VARSOVIENSIS

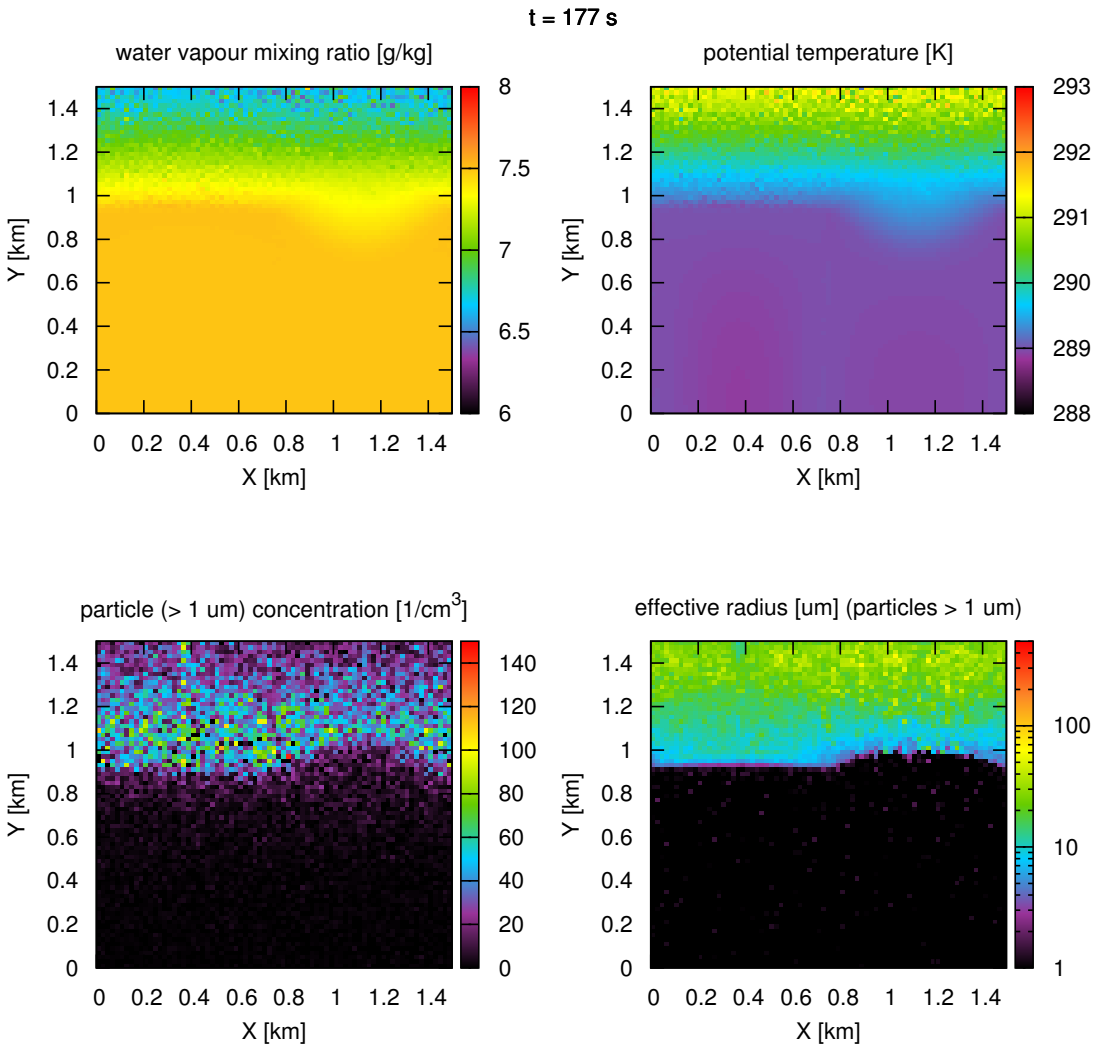
work in progress & future plans

particle-based Monte-Carlo μ-physics (using Thrust)



work in progress & future plans

particle-based Monte-Carlo μ-physics (using Thrust)

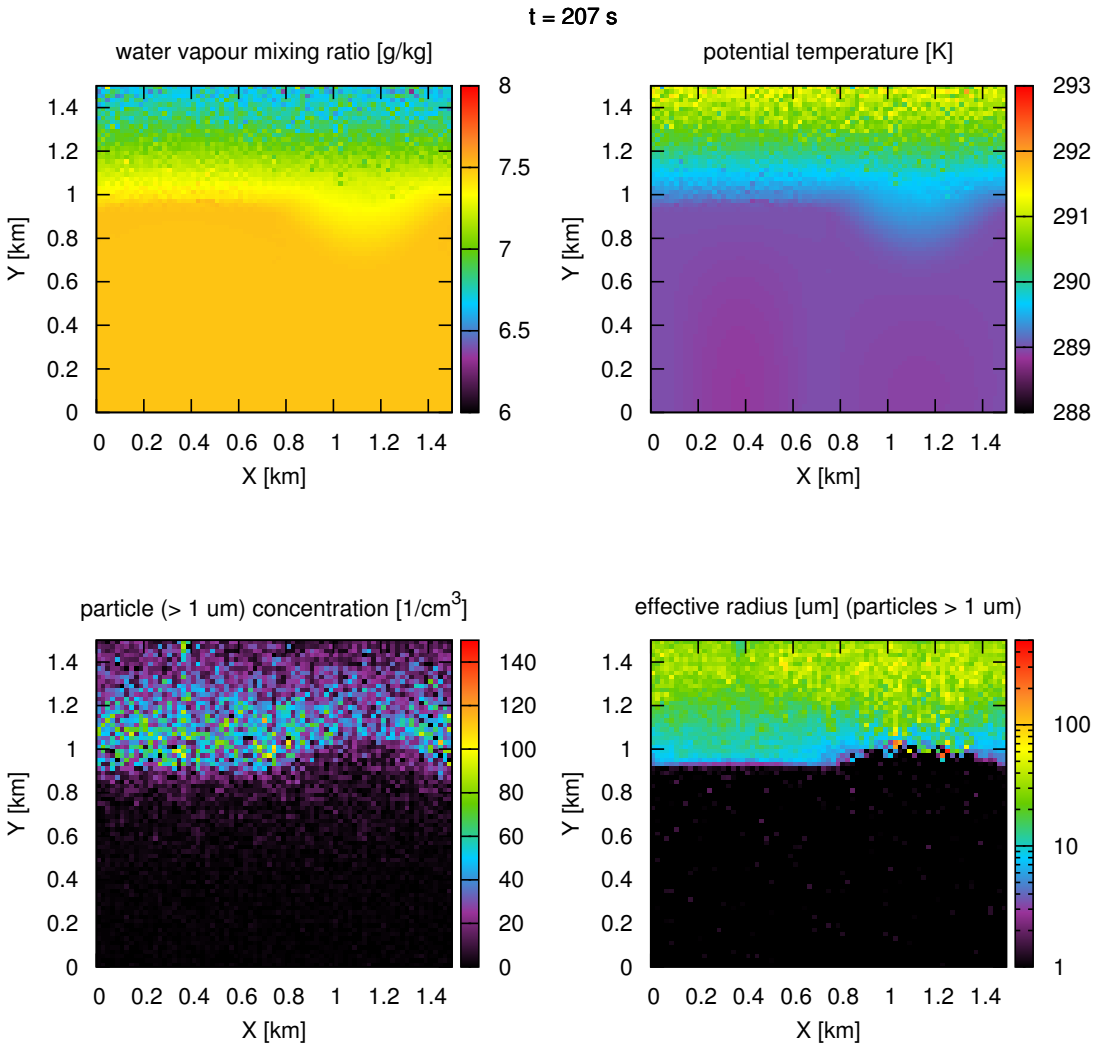


all GPL-licensed

UNIVERSITAS VARSOVIENSIS

work in progress & future plans

particle-based Monte-Carlo μ-physics (using Thrust)

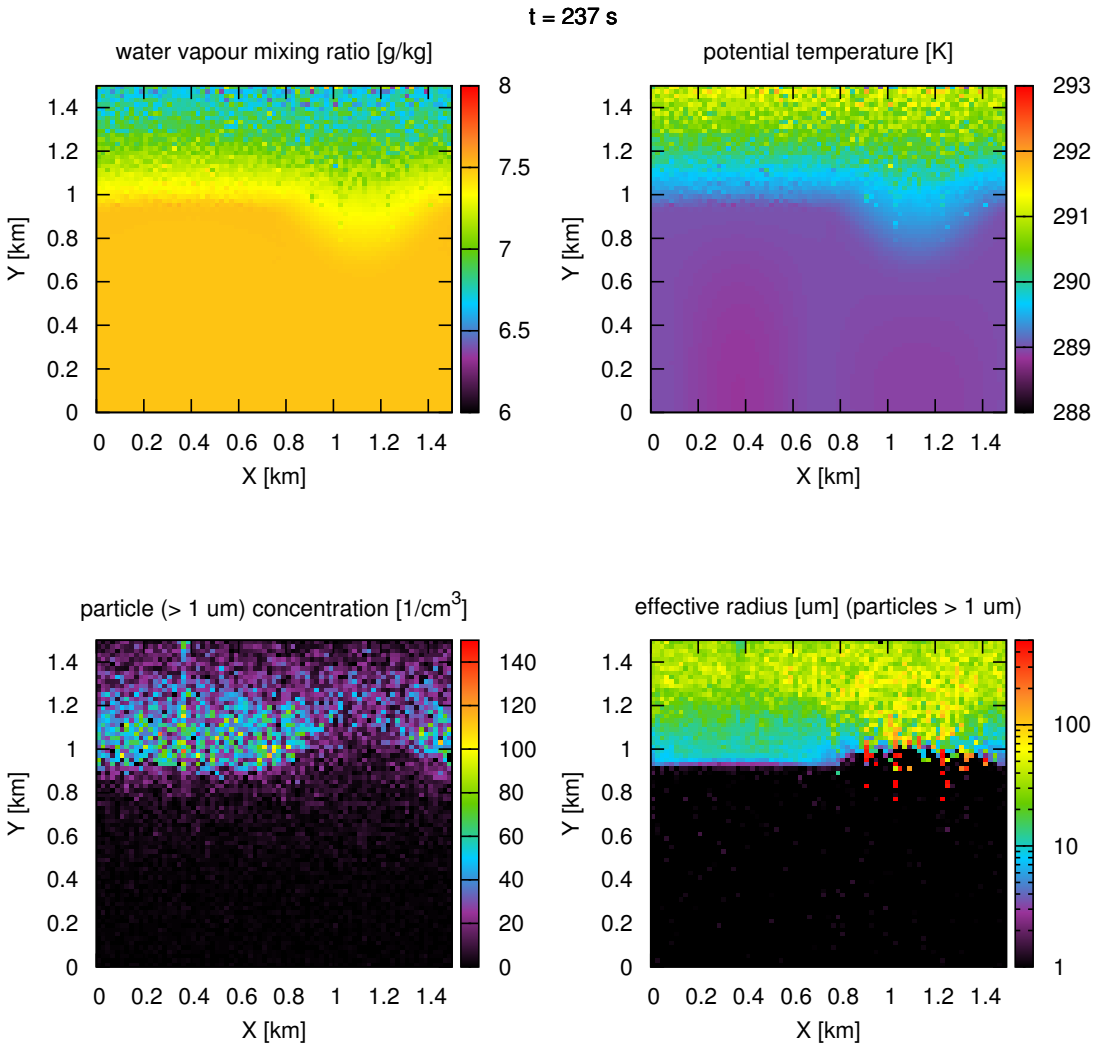


all GPL-licensed

UNIVERSITAS VARSOVIENSIS

work in progress & future plans

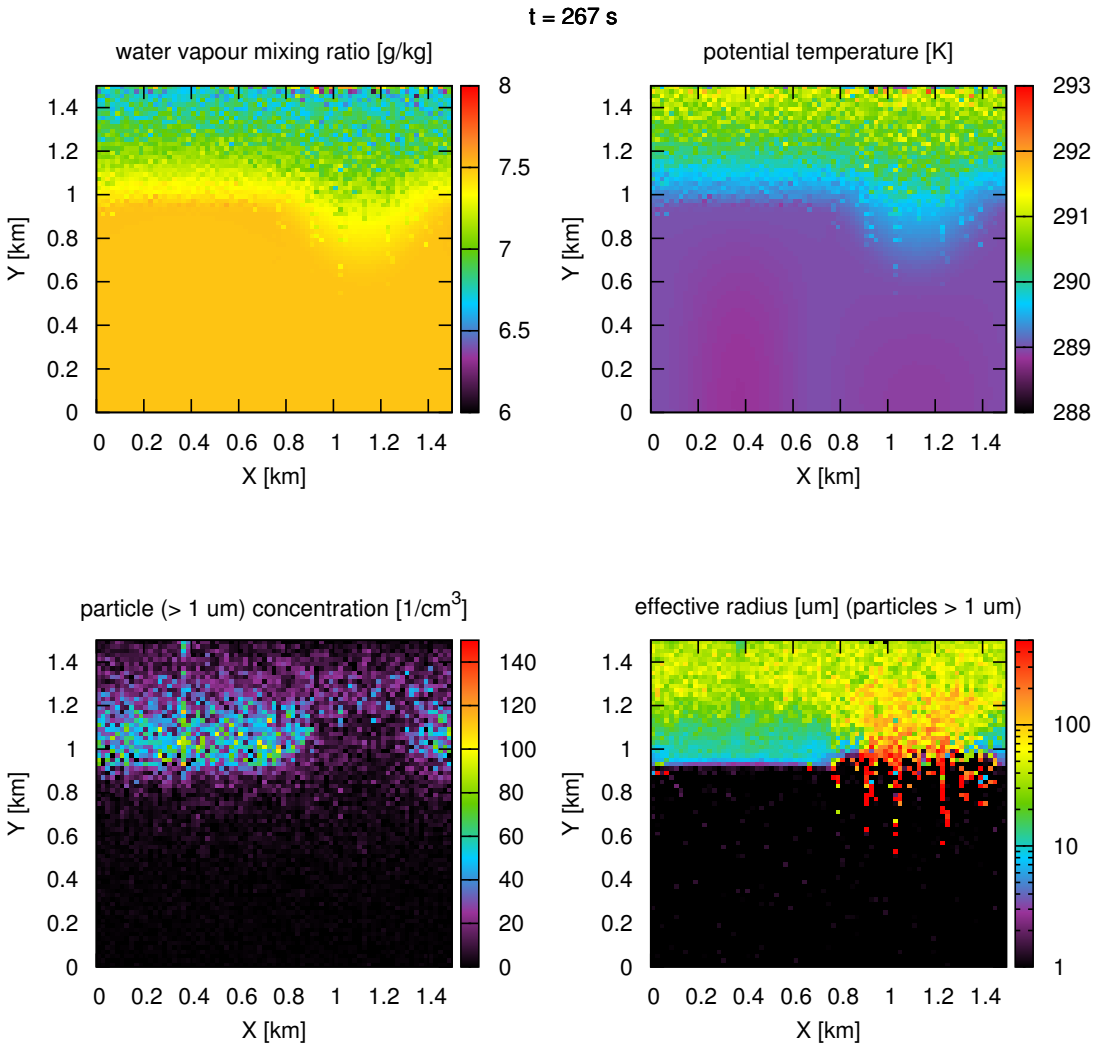
particle-based Monte-Carlo μ-physics (using Thrust)



all GPL-licensed

work in progress & future plans

particle-based Monte-Carlo μ-physics (using Thrust)

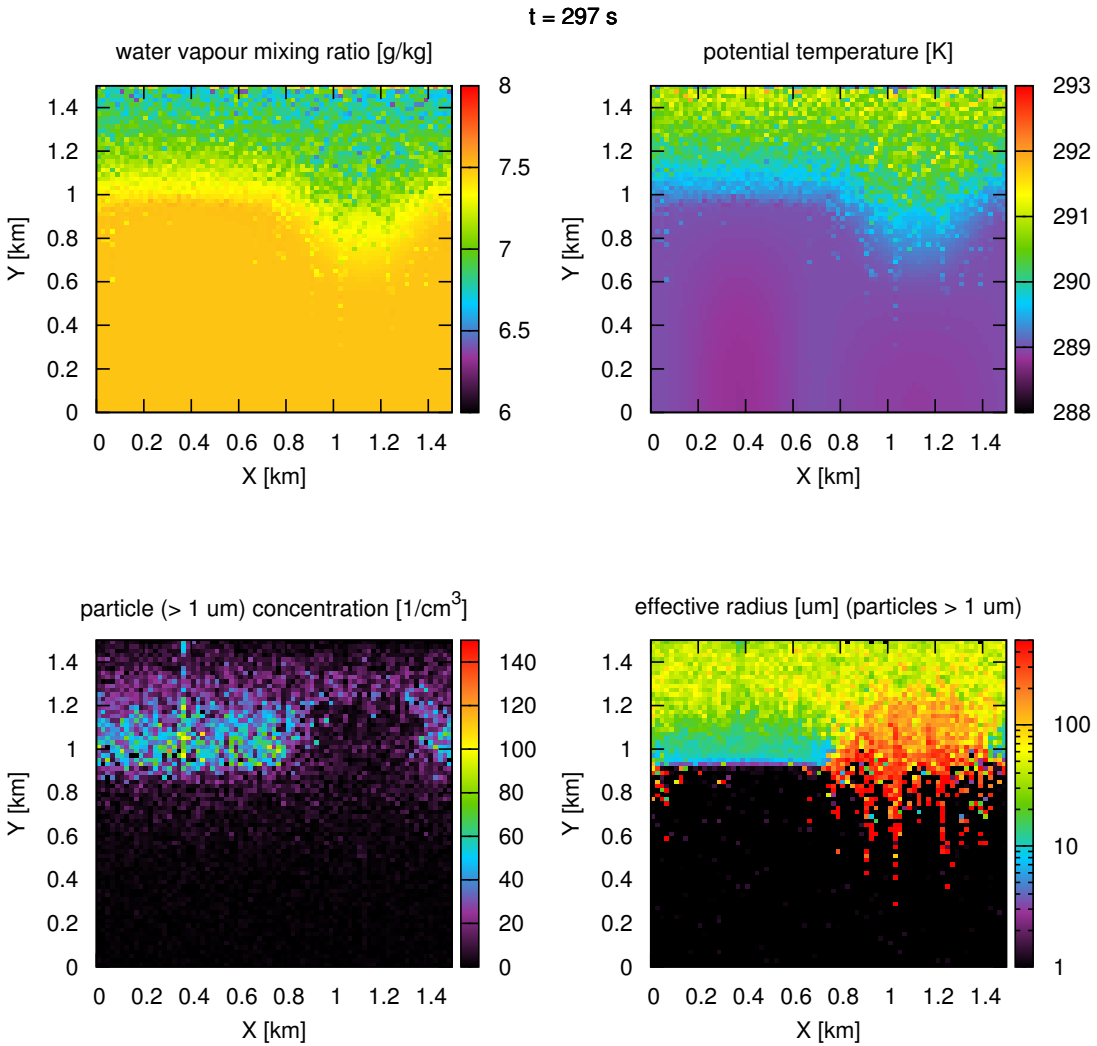


all GPL-licensed

UNIVERSITAS VARSOVIENSIS

work in progress & future plans

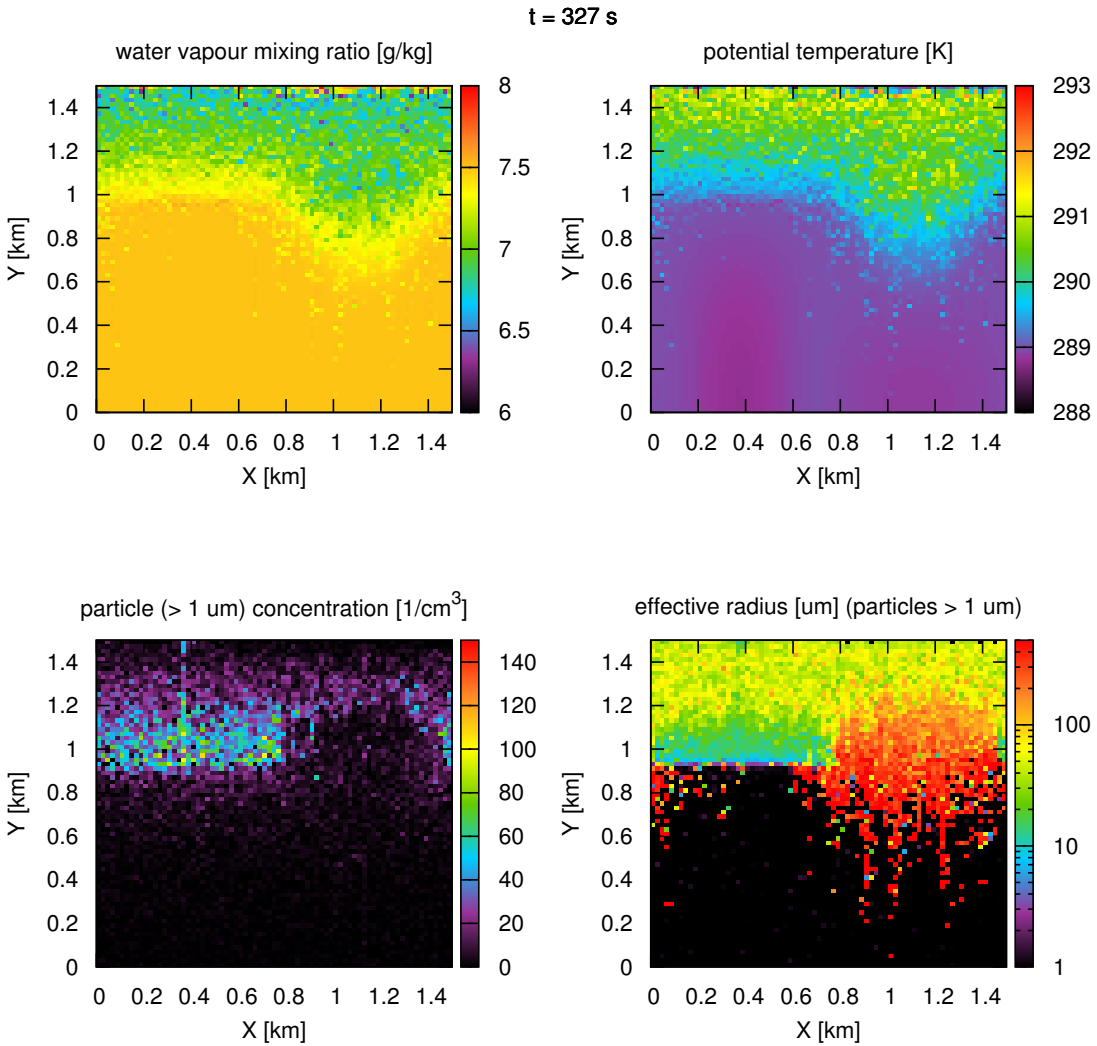
particle-based Monte-Carlo μ-physics (using Thrust)



all GPL-licensed

work in progress & future plans

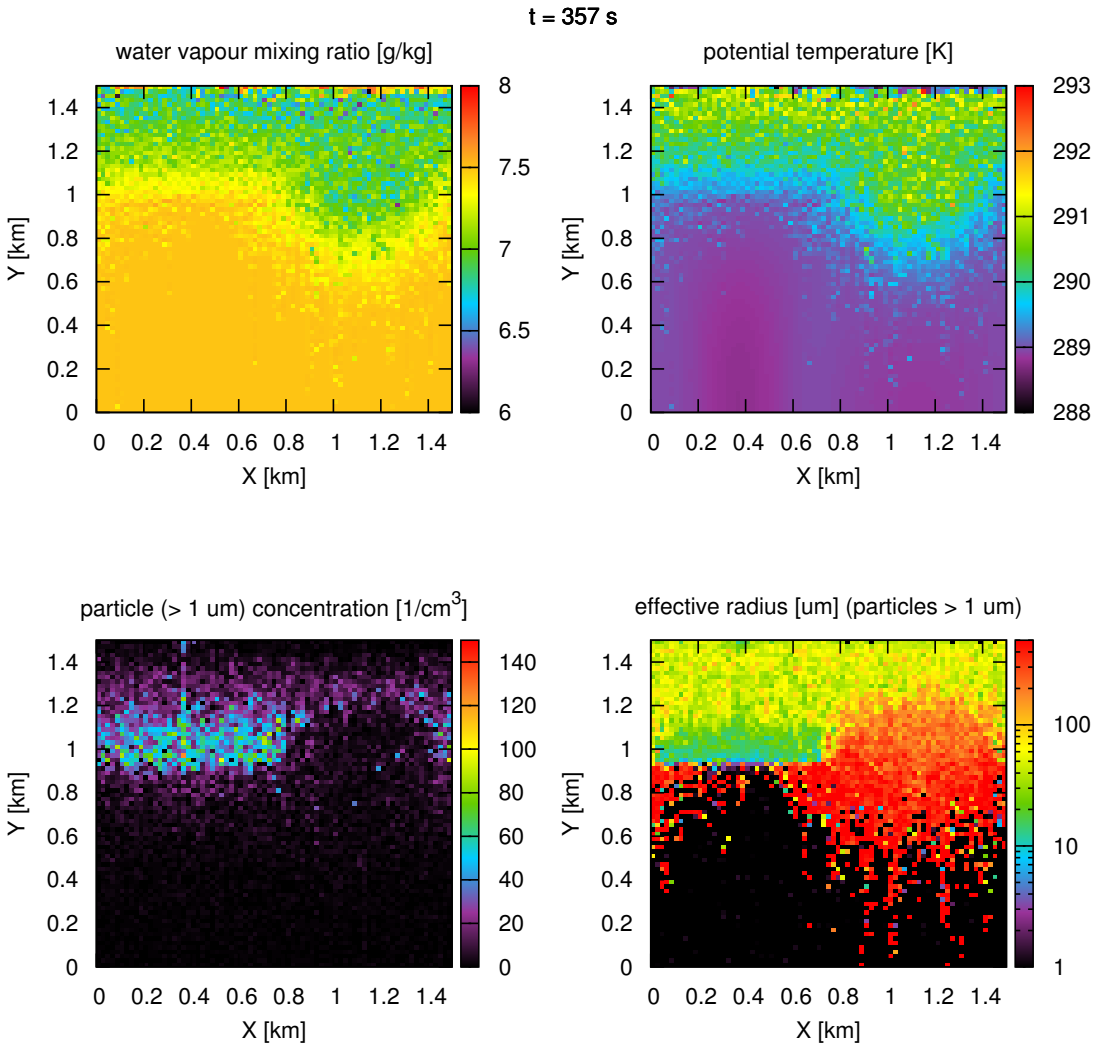
particle-based Monte-Carlo μ-physics (using Thrust)



all GPL-licensed

work in progress & future plans

particle-based Monte-Carlo μ-physics (using Thrust)



work in progress & future plans

mpdata-oop C++, Py, F08 codes presented in the arXiv paper

- use: benchmarking, didactics
- repo: `http://github.com/slayoo/mpdata`

libmpdata++ C++ library of parallel MPDATA-based solvers

- deps: Blitz++, OpenMP/Boost.Thread, Boost.MPI
- use: dynamical core for models of geophysical flows
- repo: `http://github.com/slayoo/advoocat`

libcmphscs++ C++ library of cloud μ-physics algorithms

- deps: Blitz++, Thrust (GPU/OpenMP)
- use: cloud/aerosol/precipitation μ-physics package

icicle 2D model of aerosol-cloud-aerosol interactions

- deps: libmpdata++, libcmphscs++, ...

icicLES OOP LES system for cloud-physics research

- deps: libmpdata++, libcmphscs++, ...
- news: Polish National Science Centre funds granted!
- plan: 3 years, ca. 3 full-time positions (50/50 sci/dev), co-op: Wojciech Grabowski, Piotr Smolarkiewicz

all GPL-licensed

work in progress & future plans

mpdata-oop C++, Py, F08 codes presented in the arXiv paper

- use: benchmarking, didactics
- repo: `http://github.com/slayoo/mpdata`

libmpdata++ C++ library of parallel MPDATA-based solvers

- deps: Blitz++, OpenMP/Boost.Thread, Boost.MPI
- use: dynamical core for models of geophysical flows
- repo: `http://github.com/slayoo/advoocat`

libcmphscs++ C++ library of cloud μ-physics algorithms

- deps: Blitz++, Thrust (GPU/OpenMP)
- use: cloud/aerosol/precipitation μ-physics package

icicle 2D model of aerosol-cloud-aerosol interactions

- deps: libmpdata++, libcmphscs++, ...

icicLES OOP LES system for cloud-physics research

- deps: libmpdata++, libcmphscs++, ...
- news: Polish National Science Centre funds granted!
- plan: 3 years, ca. 3 full-time positions (50/50 sci/dev), co-op: Wojciech Grabowski, Piotr Smolarkiewicz

all GPL-licensed

a take-home message

C++11/Blitz++,
Python/NumPyPy,
and Fortran 2008

since very recently (2010s!)
offer similar and unprecedented possibilities
for matching the mathematical "blackboard abstractions"
in high-performance computing applications
using object-oriented programming

all are available as free & open-source solutions

cleverly used may significantly improve
code auditability and maintainability

a take-home message

C++11/Blitz++,
Python/NumPyPy,
and Fortran 2008

since very recently (2010s!)
offer similar and unprecedented possibilities
for matching the mathematical "blackboard abstractions"
in high-performance computing applications
using object-oriented programming

all are available as free & open-source solutions

cleverly used may significantly improve
code auditability and maintainability

a take-home message

C++11/Blitz++,
Python/NumPyPy,
and Fortran 2008

since very recently (2010s!)
offer similar and unprecedented possibilities
for matching the mathematical "blackboard abstractions"
in high-performance computing applications
using object-oriented programming

all are available as free & open-source solutions

cleverly used may significantly improve
code auditability and maintainability

Thanks for your attention!

Contact: Sylwester Arabas / sarabas@igf.fuw.edu.pl

Paper: `http://arxiv.org/abs/1301.1334`

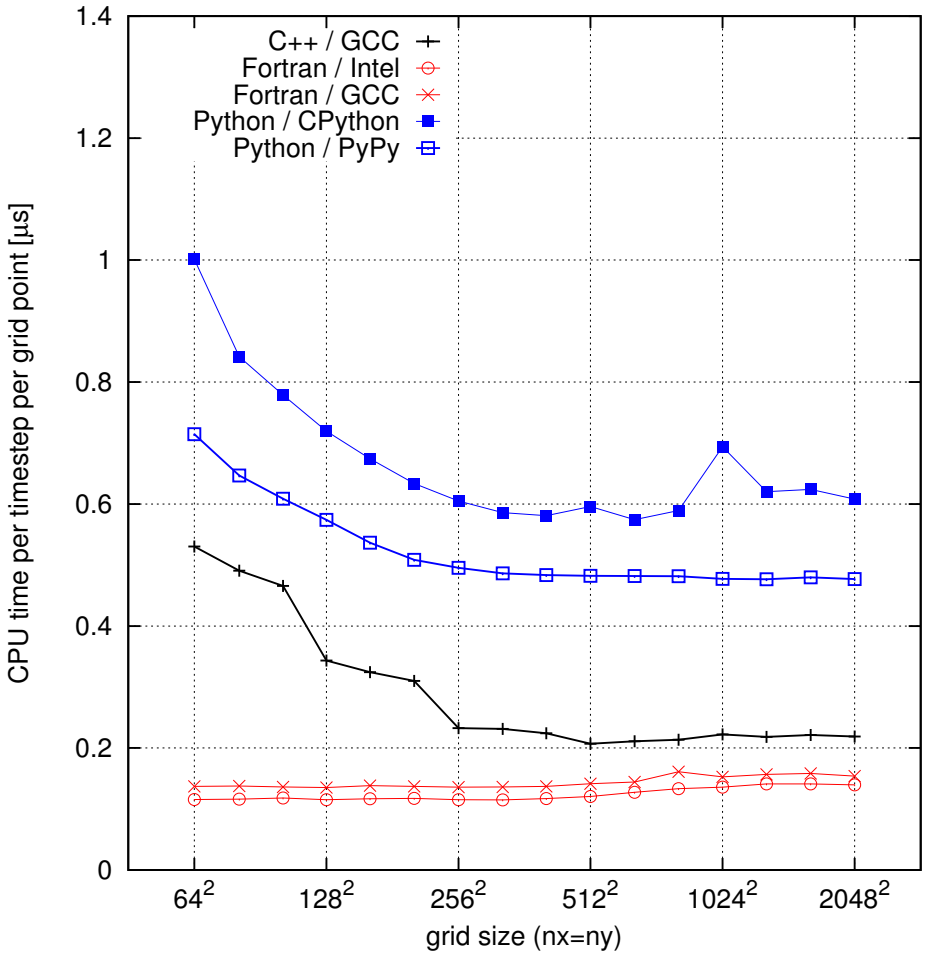
Co-authors:

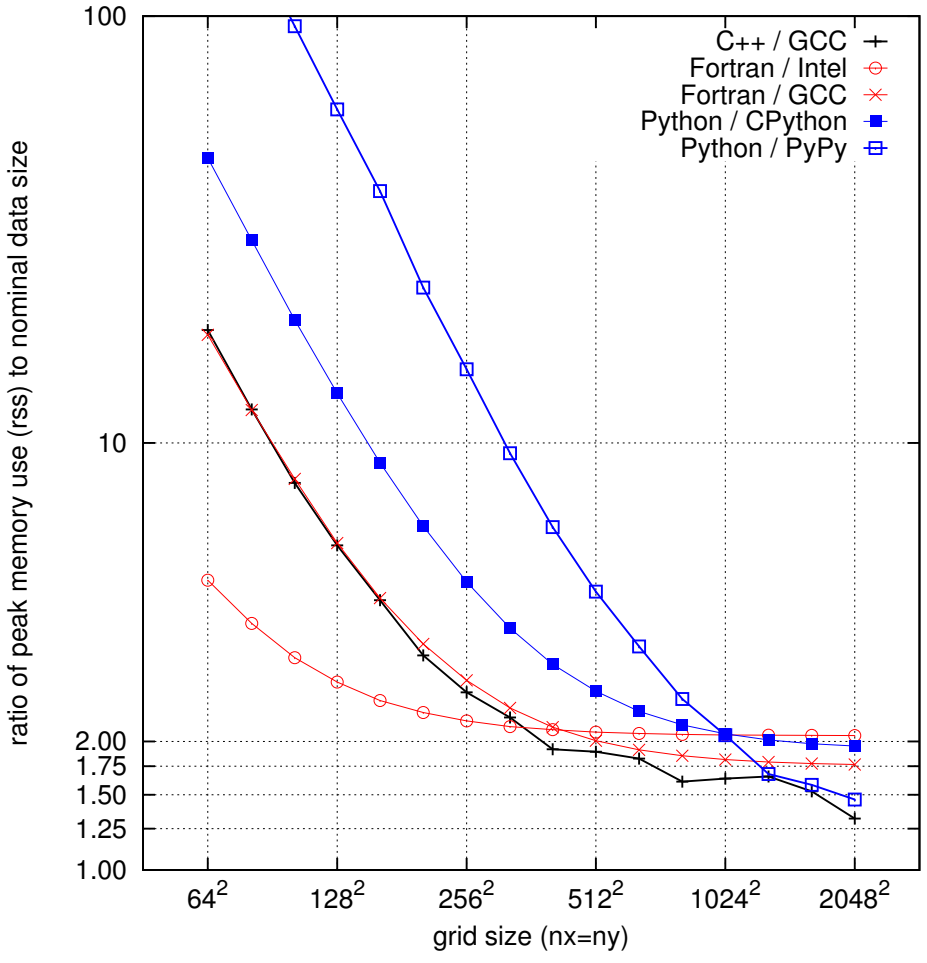
- Dorota Jarecka & Anna Jaruga (Univ. Warsaw)
- Maciej Fijałkowski (PyPy team)

Thanks to:

- UCAR/SEA for supporting my travel to US
- Piotr Smolarkiewicz for tutoring
- Hanna Pawłowska - head of our group @ Univ. Warsaw
- Polish National Science Centre for funding (2011/01/N/ST10/01483)
- Authors of free/libre open-source software used in the project

performance tests on yellowstone





g++/gfortran: -Ofast -march=native
ifort: -fast -axAVX

Thanks to Davide Del Vento!