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, 2\textsuperscript{nd} April 2013
background

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


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


*recognised ≠ adopted*
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
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
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**
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
we need software **maintainability** and **auditability**!


```
C:\lab>
f?? -o
data.exe
>
>
...ERROR

...why scientific programming does not compute
>
```

(climate modelling context)
Object-oriented implementations of the MPDATA advection equation solver in C++, Python and Fortran

Sylwester Arabas, Dorota Jarecka, Anna Jaruga, Maciej Fijalkowski

(Submitted on 7 Jan 2013 (v1), last revised 19 Mar 2013 (this version, v2))

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


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

- 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

- code: github.com/slayoo/mpdata/
Object-oriented implementations of the MPDATA advection equation solver in C++, Python and Fortran

Sylwester Arabas, Dorota Jarecka, Anna Jaruga, Maciej Fijalkowski

(Submitted on 7 Jan 2013 (v1), last revised 19 Mar 2013 (this version, v2))

- 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


- code: github.com/slayoo/mpdata/
Object-oriented implementations of the MPDATA advection equation solver in C++, Python and Fortran

Sylwester Arabas, Dorota Jarecka, Anna Jaruga, Maciej Fijalkowski
(Submitted on 7 Jan 2013 (v1), last revised 19 Mar 2013 (this version, v2))

- 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

- code: github.com/slayoo/mpdata/
Object-oriented implementations of the MPDATA advection equation solver in C++, Python and Fortran

Sylwester Arabas, Dorota Jarecka, Anna Jaruga, Maciej Fijalkowski

(Submitted on 7 Jan 2013 (v1), last revised 19 Mar 2013 (this version, v2))

- 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 (inline in the paper text)
  - compare language/library features and performance


- 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


- code: github.com/slayoo/mpdata/
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\textsuperscript{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

- 2nd-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

- 2nd-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\textsuperscript{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\textsuperscript{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

- 2nd-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

- 2nd-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\textsuperscript{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

- 2nd-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\textsuperscript{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

- 2nd-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

- 2nd-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\textsuperscript{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\textsuperscript{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

- 2nd-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

- 2nd-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\textsuperscript{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

- 2nd-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

- 2nd-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\textsuperscript{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

- 2nd-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

- 2nd-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

- 2nd-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

- 2nd-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 formulæ

- Advection equation: \( \partial_t \psi = -\nabla \cdot (\vec{v} \psi) \)

- Two time-level forward-in-time scheme:
  \[
  \psi^{[n+1]} = \psi^{[n]} - \sum_d \text{Adv}(\psi^{[n]}, \vec{C})
  \]
  \[
  \vec{C} = \vec{v} \cdot \frac{\Delta t}{\Delta x}
  \]

- 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:
- \( \psi \) - 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:
  \[
  \psi^{[n+1]} = \psi^{[n]} - \sum_d \text{Adv}(\psi^{[n]}, \tilde{C})
  \]
  \[
  \tilde{C} = \vec{v} \cdot \frac{\Delta t}{\Delta x}
  \]
- 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] + \pi^d_{1,0}}, \psi^{[n]}_{[i,j] + \pi^d_{1/2,0}}, C^{[d]}_{[i,j] + \pi^d_{1,0}} \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:
- \( \psi \) - conservative dependent variable (scalar field)
- \( \tilde{C} \) - Courant number (vector field)
- \( n \) - time level
- \( d \) - dimension
- \( i, j \) - grid indices
- \( \pi_{a,b}^d \) - permutation of \((a,b)\) of order \(d\)
MPDATA (Smolarkiewicz 1984) in formulæ

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

\[
C'[d]_{i,j} + \pi^d_{1/2,0} = \begin{vmatrix}
C[d]_{i,j} + \pi^d_{1/2,0} & 1 - C[d]_{i,j} + \pi^d_{1/2,0}
\end{vmatrix} \cdot A[d]_{i,j}(\psi)
\]

\[
- \sum_{q=0, q \neq d}^{N} C[d]_{i,j} + \pi^d_{1/2,0} \cdot \bar{C}[q]_{i,j} + \pi^d_{1/2,0} \cdot B[d]_{i,j}(\psi)
\]

\[
\bar{C}[q]_{i,j} + \pi^d_{1/2,0} = \frac{1}{4} \cdot \left( C[q]_{i,j} + \pi^d_{11/2} + C[q]_{i,j} + \pi^d_{01/2} + C[q]_{i,j} + \pi^d_{1-1/2} + C[q]_{i,j} + \pi^d_{0-1/2} \right)
\]

\[
A[d]_{i,j} = \frac{\psi[i,j] + \pi^d_{101} - \psi[i,j]}{\psi[i,j] + \pi^d_{101} + \psi[i,j]}
\]

\[
B[d]_{i,j} = \frac{1}{2} \cdot \frac{\psi[i,j] + \pi^d_{11} + \psi[i,j] + \pi^d_{01} - \psi[i,j] + \pi^d_{1-1} - \psi[i,j] + \pi^d_{0-1}}{\psi[i,j] + \pi^d_{11} + \psi[i,j] + \pi^d_{01} + \psi[i,j] + \pi^d_{1-1} + \psi[i,j] + \pi^d_{0-1}}
\]

symbols:

$\psi$ - conservative dependent variable (scalar field)

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

$n$ - time level

$d$ - dimension

$i, j$ - grid indices

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

altogether: 0.06 kLOC in \LaTeX
MPDATA (Smolarkiewicz 1984) in formulæ

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

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

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

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

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

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

- altogether: **0.06 kLOC in \LaTeX**

**Symbols:**
- $\psi -$ conservative dependent variable (scalar field)
- $\overline{C} -$ 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: $\psi$, $C[0]$, $C[1]$
  - integers: $n$, $d$, $i$, $j$
  - statements
  - ...
  - $#LOC \gg $#LOC in \LaTeX, human-readable?

- OO numerics:
  - instance of an array-of-floats class: $\psi$
  - 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:

- $\psi$ - 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: \( \psi \), \( C^0 \), \( C^1 \)
  ▶ integers: \( n \), \( d \), \( i \), \( j \)
  ▶ statements
  ▶ ...
  ▶ \#LOC \( \gg \) \#LOC in \LaTeX, human-readable?

▶ OO numerics:
  ▶ instance of an array-of-floats class: \( \psi \)
  ▶ 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:

\( \psi \) - 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: $\psi, C^0, C^1$
  - integers: $n, d, i, j$
  - statements
  - ...
  - #LOC $\gg$ #LOC in \LaTeX{}, human-readable?

- OO numerics:
  - instance of an array-of-floats class: $\psi$
  - 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:

- $\psi$ - 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: $\psi$, $C^0$, $C^1$
  ◀ integers: $n$, $d$, $i$, $j$
  ◀ statements
  ◀ ...
  ◀ #LOC $\gg$ #LOC in \LaTeX, human-readable?

▶ OO numerics:
  ◀ instance of an array-of-floats class: $\psi$
  ◀ 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:

$\psi$ - conservative dependent variable (scalar field)

$\bar{\mathbf{C}}$ - 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: $\psi$, $C[0]$, $C[1]$
  - integers: $n$, $d$, $i$, $j$
  - statements
  - ...
  - #LOC $\gg$ #LOC in \LaTeX{}, human-readable?

- OO numerics:
  - instance of an array-of-floats class: $\psi$
  - 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:
- $\psi$ - conservative dependent variable (scalar field)
- $\tilde{C}$ - Courant number (vector field)
- $n$ - time level
- $d$ - dimension
- $i$, $j$ - grid indices
- $\pi_{a,b}^d$ - permutation of (a,b) of order d
MPDATA (Smolarkiewicz 1984) in formulæ

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

- OO numerics:
  - instance of an array-of-floats class: $\psi$
  - 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:

$\psi$ - conservative dependent variable (scalar field)

$\tilde{C}$ - 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: $\psi$, $C^0$, $C^1$
   ▶ integers: $n$, $d$, $i$, $j$
   ▶ statements
   ▶ ...
   ▶ #LOC $\gg$ #LOC in \LaTeX, human-readable?

▶ OO numerics:
   ▶ instance of an array-of-floats class: $\psi$
   ▶ 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:

$\psi$ - conservative dependent variable (scalar field)

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

$n$ - time level

$d$ - dimension

$i, j$ - grid indices

$\pi^d_{a,b}$ - permutation of (a,b) of order d
OOP language/library set-ups for fast number crunching:
(in context of structured meshes, stencil-type algorithms)

- C++ & Blitz++
- Python & NumPy
- Fortran 200x

\[ \ldots \]¹

¹suggestions 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++:

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

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;
}
```

\[
\psi[i, j], \psi[i-1, j], \psi[i, j+1], C_x[i, j], C_x[i+1/2, j], C_x[i-1/2, j], C_y[i, j], C_y[i, j-1/2]
\]
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
        )

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

\[
\psi_{i,j+1}
\]

\[
C_{x}^{[i+1/2,j]}
\]

\[
C_{x}^{[i-1/2,j]}
\]

\[
\psi_{i-1,j}
\]

\[
\psi_{i,j}
\]

\[
C_{y}^{[i,j-1/2]}
\]
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

\[ \psi_{[i,j]}^{[n+1]} = \psi_{[i,j]}^{[n]} - \sum_{d=0}^{N-1} \left( F \left[ \psi_{[i,j]}^{[n]} , \psi_{[i,j]}^{[n]} + \pi_d^d , C_{[i,j]}^{[d]} \right] - F \left[ \psi_{[i,j]}^{[n]} - \pi_d^{-1,0} , \psi_{[i,j]}^{[n]} - \pi_d^{-1,0} , C_{[i,j]}^{[d]} \right] \right) \]

C++ / Blitz++:

```cpp
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))
    )
)
```

```cpp
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);
}
```

\( \text{pi}() \) returns an instance of \text{blitz::RectDomain}
example (2/3): dimensional indirection

\[ \psi_{[n+1]}[i,j] = \psi_{[n]}[i,j] - \sum_{d=0}^{N-1} \left( F \begin{bmatrix} \psi_{[n]}[i,j], \psi_{[n]}[i,j]+\pi_d, & C_{[d]}[i,j]+\pi_d \\ \psi_{[n]}[i,j]+\pi_d-1,0, & \psi_{[n]}[i,j], & C_{[d]}[i,j]+\pi_d \end{bmatrix} \right) \]

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++:

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

```cpp
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)
)
```

```cpp
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))
   )
)
```

return type: C++11’s auto ↦ 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
```

return type: "dimension" (array)
temporary objects: compiler-dependant
Is there any way to improve Python’s performance?
Is there any way to improve Python’s 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?
Is there any way to improve Python’s performance?
Is there any way to improve Python’s performance?
Is there any way to improve Python’s performance?
Is there any way to improve Python’s 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?
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?
Is there any way to improve Python’s 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?
Is there any way to improve Python’s performance?
Is there any way to improve Python’s performance?
Is there any way to improve Python’s 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
    → potential improvement in speed and memory consumption
  - switching to PyPy does not require code modifications!
    (in contrast to Numexpr, Cython, Numba, . . . )
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
  → potential improvement in speed and memory consumption
- switching to PyPy does not require code modifications!
  (in contrast to Numexpr, Cython, Numba, ... )
MPDATA in C++, Python and Fortran: performance

CPU time per timestep per grid point [µs]
grid size (nx=ny)

AMD Phenom™ II X6 1055T Processor

- C++ / GCC
- C++ / LLVM
- Fortran / GCC
- Python / CPython
- Python / PyPy

Intel® Core™ i5−2467M CPU 1.60GHz

- C++ / GCC
- C++ / LLVM
- Fortran / GCC
- Python / CPython
- Python / PyPy
MPDATA in C++, Python and Fortran: performance

The diagram shows the ratio of peak memory use (rss) to nominal data size for different grid sizes (nx=ny) and implementations:
- C++ / GCC
- C++ / LLVM
- Fortran / GCC
- Python / CPython
- Python / PyPy

The x-axis represents the grid size, and the y-axis shows the ratio of peak memory use to nominal data size.
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)
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?

<table>
<thead>
<tr>
<th>Language</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>ca. 200</td>
</tr>
<tr>
<td>C++</td>
<td>ca. 300</td>
</tr>
<tr>
<td>Fortran</td>
<td>ca. 500</td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>Language</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>ca. 200</td>
</tr>
<tr>
<td>C++</td>
<td>ca. 300</td>
</tr>
<tr>
<td>Fortran</td>
<td>ca. 500</td>
</tr>
</tbody>
</table>
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
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
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
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
  - trained personnel
  - reusable software components
  - 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 ≠ 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
  - trained personnel
  - reusable software components
  - 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 ≠ 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
  - trained personnel
  - reusable software components
  - 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 ≠ 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
  - trained personnel
  - reusable software components
  - 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 ≠ Blitz++ community
  - numerous alternative to Blitz++ (in contrast to NumPy)

- issues with Python:
  - tricky to leverage multi-core CPUs through shared-memory parallelisation
other issues:

- programmers’ community size: C++ and Python win
  - trained personnel
  - reusable software components
  - 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 ≠ Blitz++ community
  - numerous alternative to Blitz++ (in contrast to NumPy)

- issues with Python:
  - tricky to leverage multi-core CPUs through shared-memory parallelisation
other issues:

- programmers’ community size: C++ and Python win
  - trained personnel
  - reusable software components
  - 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
  - trained personnel
  - reusable software components
  - 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 ≠ 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](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](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 \(\mu\)-physics algorithms
  
  **deps:** Blitz++, Thrust (GPU/OpenMP)
  
  **use:** cloud/aerosol/precipitation \(\mu\)-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)

h @ t/dt=4
work in progress & future plans

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

h @ t/dt=8
2D shallow-water equations (OpenMP/Boost.Thread)
work in progress & future plans

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

h @ t/dt=16
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)

h @ t/dt=28
work in progress & future plans

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

h @ t/dt=32
work in progress & future plans

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

h @ t/dt=36
work in progress & future plans

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

h @ t/dt=40
work in progress & future plans

2D shallow-water equations (OpenMP/Boost.Thread)
work in progress & future plans

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

h @ t/dt=48
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)

h @ t/dt=64
work in progress & future plans

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

h @ t/dt=68
work in progress & future plans

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

$\text{h @ t/dt=72}$
work in progress & future plans

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

h @ t/dt=76
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](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](http://github.com/slayoo/advoocat)

**libcmphscs++**  C++ library of cloud $\mu$-physics algorithms
- **deps:** Blitz++, Thrust (GPU/OpenMP)
- **use:** cloud/aerosol/precipitation $\mu$-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 $\mu$-physics algorithms
  deps: Blitz++, Thrust (GPU/OpenMP)
  use: cloud/aerosol/precipitation $\mu$-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 \( \mu \)-physics (using Thrust)

- water vapour mixing ratio [g/kg]
- potential temperature [K]
- particle (> 1 um) concentration [1/cm\(^3\)]
- effective radius [um] (particles > 1 um)

`all GPL-licensed`
work in progress & future plans

**particle-based Monte-Carlo $\mu$-physics (using Thrust)**

- **water vapour mixing ratio [g/kg]**
- **potential temperature [K]**
- **particle (> 1 um) concentration [1/cm$^3$]**
- **effective radius [um] (particles > 1 um)**

---

All GPL-licensed.
work in progress & future plans

particle-based Monte-Carlo $\mu$-physics (using Thrust)

data showing water vapour mixing ratio, potential temperature, particle concentration, and effective radius over time and space.

all GPL-licensed
work in progress & future plans

particle-based Monte-Carlo $\mu$-physics (using Thrust)

all GPL-licensed
work in progress & future plans

particle-based Monte-Carlo $\mu$-physics (using Thrust)

- Water vapour mixing ratio [g/kg] at $t = 147$ s
- Potential temperature [K] at $t = 147$ s
- Particle (> 1 um) concentration [1/cm$^3$] at $t = 147$ s
- Effective radius [um] (particles > 1 um) at $t = 147$ s

All GPL-licensed
work in progress & future plans

particle-based Monte-Carlo $\mu$-physics (using Thrust)

1. **Water Vapour Mixing Ratio [g/kg]**
   - t = 177 s
   - X [km]: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4
   - Y [km]: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4
   - Colorbar: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4

2. **Potential Temperature [K]**
   - t = 177 s
   - X [km]: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4
   - Y [km]: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4
   - Colorbar: 288, 289, 290, 291, 292, 293

3. **Particle (> 1 um) Concentration [1/cm$^3$]**
   - X [km]: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4
   - Y [km]: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4
   - Colorbar: 0, 20, 40, 60, 80, 100, 120, 140

4. **Effective Radius [um] (Particles > 1 um)**
   - X [km]: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2, 1.4
   - Y [km]: 1, 10, 100
   - Colorbar: 1, 10, 100

All GPL-licensed
work in progress & future plans

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

water vapour mixing ratio [g/kg]

potential temperature [K]

t = 207 s

particle (> 1 um) concentration [1/cm³]

effective radius [um] (particles > 1 um)

all GPL-licensed
work in progress & future plans

particle-based Monte-Carlo $\mu$-physics (using Thrust)

---

**particle-based Monte-Carlo $\mu$-physics (using Thrust)**

- **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/mpdata](http://github.com/slayoo/mpdata)

- **libcmphscs++** C++ library of cloud $\mu$-physics algorithms
  - deps: Blitz++, Thrust (GPU/OpenMP)
  - use: cloud/aerosol/precipitation $\mu$-physics package
  - repo: [http://github.com/slayoo/advoocat](http://github.com/slayoo/advoocat)

- **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

---

**particle (> 1 um) concentration [1/cm$^3$]**

**effective radius [um] (particles > 1 um)**

---

**water vapour mixing ratio [g/kg]**

**potential temperature [K]**

---

**t = 237 s**

---

**potential temperature [K]**

---

**water vapour mixing ratio [g/kg]**
work in progress & future plans

**particle-based Monte-Carlo \(\mu\)-physics (using Thrust)**

- **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/mpdata](http://github.com/slayoo/mpdata)

- **libcmphscs++ C++ library of cloud \(\mu\)-physics algorithms**
  - deps: Blitz++, Thrust (GPU/OpenMP)
  - use: cloud/aerosol/precipitation \(\mu\)-physics package
  - repo: [http://github.com/slayoo/advoocat](http://github.com/slayoo/advoocat)

- **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

---

**Graphs:**

- Water vapour mixing ratio [g/kg] at \(t = 267\) s
- Potential temperature [K] at \(t = 267\) s
- Particle (> 1 um) concentration [1/cm\(^3\)] at \(t = 267\) s
- Effective radius [um] (particles > 1 um) at \(t = 267\) s
work in progress & future plans

particle-based Monte-Carlo \(\mu\)-physics (using Thrust)

- **water vapour mixing ratio [g/kg]**
- **potential temperature [K]**
- **particle (> 1 um) concentration [1/cm\(^3\)]**
- **effective radius [um] (particles > 1 um)**

All GPL-licensed.
work in progress & future plans

**particle-based Monte-Carlo \( \mu \)-physics (using Thrust)**

---

### Water Vapour Mixing Ratio [g/kg]

- **Figure:** Water vapour mixing ratio at \( t = 327 \) s.
- **Legend:**
  - Color bar from 6 to 8.
- **Axes:**
  - X [km]: 0 to 1.4
  - Y [km]: 0 to 1.4

### Potential Temperature [K]

- **Figure:** Potential temperature at \( t = 327 \) s.
- **Legend:**
  - Color bar from 288 to 293.
- **Axes:**
  - X [km]: 0 to 1.4
  - Y [km]: 0 to 1.4

---

### Particle Concentration [1/cm\(^3\)]

- **Figure:** Particle concentration at \( t = 327 \) s.
- **Legend:**
  - Color bar from 0 to 140.
- **Axes:**
  - X [km]: 0 to 1.4
  - Y [km]: 0 to 1.4

### Effective Radius [\( \mu \)-m] (particles > 1 \( \mu \)m)

- **Figure:** Effective radius at \( t = 327 \) s.
- **Legend:**
  - Color bar from 1 to 100.
- **Axes:**
  - X [km]: 0 to 1.4
  - Y [km]: 0 to 1.4

---

*all GPL-licensed*
work in progress & future plans

particle-based Monte-Carlo $\mu$-physics (using Thrust)

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 $\mu$-physics algorithms
deps: Blitz++, Thrust (GPU/OpenMP)
use: cloud/aerosol/precipitation $\mu$-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](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](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


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

CPU time per timestep per grid point [µs]

<table>
<thead>
<tr>
<th>grid size (nx=ny)</th>
<th>C++ / GCC</th>
<th>Fortran / Intel</th>
<th>Fortran / GCC</th>
<th>Python / CPython</th>
<th>Python / PyPy</th>
</tr>
</thead>
<tbody>
<tr>
<td>64^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
<tr>
<td>128^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
<tr>
<td>256^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
<tr>
<td>512^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
<tr>
<td>1024^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
<tr>
<td>2048^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
</tbody>
</table>

ratio of peak memory use (rss) to nominal data size

<table>
<thead>
<tr>
<th>grid size (nx=ny)</th>
<th>C++ / GCC</th>
<th>Fortran / Intel</th>
<th>Fortran / GCC</th>
<th>Python / CPython</th>
<th>Python / PyPy</th>
</tr>
</thead>
<tbody>
<tr>
<td>64^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
<tr>
<td>128^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
<tr>
<td>256^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
<tr>
<td>512^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
<tr>
<td>1024^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
<tr>
<td>2048^2</td>
<td>1.00</td>
<td>1.25</td>
<td>1.50</td>
<td>1.75</td>
<td>2.00</td>
</tr>
</tbody>
</table>

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

Thanks to Davide Del Vento!