O wykorzystaniu programowania obiektowego do implementacji schematu MPDATA

Sylwester Arabas, Dorota Jarecka, Anna Jaruga
(grupa prof. Hanny Pawłowskiej)

Instytut Geofizyki, Wydział Fizyki, Uniwersytet Warszawski

9. października 2012 r.

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, ...):
  - schemat jego rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- ...
  - była wolnym i otwartym oprogramowaniem
  - posiadała dokumentację techniczną
  - była napisana obiektowo

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, . . .):
  - schemat 2-go rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- libmpdata++:
  - była wolnym i otwartym oprogramowaniem
  - posiadała dokumentację techniczną
  - była napisana obiektowo

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, . . .):
  - schemat 2-go rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- libmpdata++:
  - była wolnym i otwartym oprogramowaniem
  - posiadała dokumentację techniczną
  - była napisana obiektowo

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, . . .):
  - schemat 2-go rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- libmpdata++
  - była wolnym i otwartym oprogramowaniem
  - posiadała dokumentację techniczną
  - była napisana obiektowo

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, . . .):
  - schemat 2-go rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- libmpdata++:
  - była wolnym i otwartym oprogramowaniem
  - posiadała dokumentację techniczną
  - była napisana obiektowo

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, . . .):
  - schemat 2-go rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- libmpdata++:
  - była wolnym i otwartym oprogramowaniem
  - posiadała dokumentację techniczną
  - była napisana obiektowo

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, . . .):
  - schemat 2-go rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- wedle wiedzy autorów, nie istniała implementacja MPDATA, która:
  - [illegible]
  - posiadała dokumentację techniczną
  - była napisana obiektowo

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, . . .):
  - schemat 2-go rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- wedle wiedzy autorów, nie istniała implementacja MPDATA, która:
  - była wolnym i otwartym oprogramowaniem
  - posiadała dokumentację techniczną
  - była napisana obiektowo

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, . . .):
  - schemat 2-go rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- wedle wiedzy autorów, nie istniała implementacja MPDATA, która:
  - była wolnym i otwartym oprogramowaniem
  - posiadała dokumentację techniczną
  - była napisana obiektowo

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, . . .):
  - schemat 2-go rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- wedle wiedzy autorów, nie istniała implementacja MPDATA, która:
  - była wolnym i otwartym oprogramowaniem
  - posiadała dokumentację techniczną
  - była napisana obiektowo

tytułem wstępu...

$$\partial_t \psi_i + \nabla \cdot (\vec{v} \psi_i) = R_i$$

- rozwiązywanie układów równań transportu (ciągłości) jest podstawowym zadaniem wielu narzędzi do symulacji numerycznych w naukach o Ziemi
  $\rightsquigarrow$ zapotrzebowanie na dokładne i wydajne solwery
- schemat MPDATA (Smolarkiewicz 1983, . . .):
  - schemat 2-go rzędu w przestrzeni i czasie
  - z natury wielowymiarowy i zachowujący znak
  - opcja nie-oscylacyjna (flux-corrected transport)
  - iteracyjny $\rightsquigarrow$ łatwa współbieżność
  - zwyczajowy wybór w IGF UW :)
- wedle wiedzy autorów, nie istniała implementacja MPDATA, która:
  - była wolnym i otwartym oprogramowaniem
  - posiadała dokumentację techniczną
  - była napisana obiektowo

techniki programowania obiektowego ułatwiają tworzenie kodu:

- czytelnego dla człowieka $\neq$ otwartość $\rightsquigarrow$ możliwość recenzji kodu
- zwięzłego $\rightsquigarrow$ mniejsza podatność na błędy i łatwiejsze ich tropienie
- zdatnego do ponownego użycia $\rightsquigarrow$ biblioteki, nie kopiuj/wklej!
- modularnego $\rightsquigarrow$ pełen rozdział numeryki/fizyki/współbieżności/we-wy
- poddającego się optymalizacji (przez kompilator/autora biblioteki)

"[Object oriented programming] has become recognised as the almost unique successful paradigm for creating complex software"

NR: The Art of Scientific Computing (3rd ed., Press et al., 2007)

techniki programowania obiektowego ułatwiają tworzenie kodu:

- czytelnego dla człowieka + otwartość $\rightsquigarrow$ możliwość recenzji kodu
- zwięzłego $\rightsquigarrow$ mniejsza podatność na błędy i łatwiejsze ich tropienie
- zdatnego do ponownego użycia $\rightsquigarrow$ biblioteki, nie kopiuj/wklej!
- modularnego $\rightsquigarrow$ pełen rozdział numeryki/fizyki/współbieżności/we-wy
- poddającego się optymalizacji (przez kompilator/autora biblioteki)

"[Object oriented programming] has become recognised as the almost unique successful paradigm for creating complex software"

NR: The Art of Scientific Computing (3rd ed., Press et al., 2007)

techniki programowania obiektowego ułatwiają tworzenie kodu:

- czytelnego dla człowieka + otwartość $\rightsquigarrow$ możliwość recenzji kodu
- zwięzłego $\rightsquigarrow$ mniejsza podatność na błędy i łatwiejsze ich tropienie
- zdatnego do ponownego użycia $\rightsquigarrow$ biblioteki, nie kopiuj/wklej!
- modularnego $\rightsquigarrow$ pełen rozdział numeryki/fizyki/współbieżności/we-wy
- poddającego się optymalizacji (przez kompilator/autora biblioteki)

"[Object oriented programming] has become recognised as the almost unique successful paradigm for creating complex software"

NR: The Art of Scientific Computing (3rd ed., Preface, 2007)

techniki programowania obiektowego ułatwiają tworzenie kodu:

- czytelnego dla człowieka + otwartość $\rightsquigarrow$ możliwość recenzji kodu
- zwięzłego $\rightsquigarrow$ mniejsza podatność na błędy i łatwiejsze ich tropienie
- zdatnego do ponownego użycia $\rightsquigarrow$ biblioteki, nie kopiuj/wklej!
- modularnego $\rightsquigarrow$ pełen rozdział numeryki/fizyki/współbieżności/we-wy
- poddającego się optymalizacji (przez kompilator/autora biblioteki)

"[Object oriented programming] has become recognised as the almost unique successful paradigm for creating complex software"

NR: The Art of Scientific Computing
(3rd ed., Press et al., 2007)

techniki programowania obiektowego ułatwiają tworzenie kodu:

- czytelnego dla człowieka + otwartość $\rightsquigarrow$ możliwość recenzji kodu
- zwięzłego $\rightsquigarrow$ mniejsza podatność na błędy i łatwiejsze ich tropienie
- zdatnego do ponownego użycia $\rightsquigarrow$ biblioteki, nie kopiuj/wklej!
- modularnego $\rightsquigarrow$ pełen rozdział numeryki/fizyki/współbieżności/we-wy
- poddającego się optymalizacji (przez kompilator/autora biblioteki)

"[Object oriented programming] has become recognised as the almost unique successful paradigm for creating complex software"

NR: The Art of Scientific Computing (3rd ed., Press et al., 2007)

techniki programowania obiektowego ułatwiają tworzenie kodu:

- czytelnego dla człowieka + otwartość $\rightsquigarrow$ możliwość recenzji kodu
- zwięzłego $\rightsquigarrow$ mniejsza podatność na błędy i łatwiejsze ich tropienie
- zdatnego do ponownego użycia $\rightsquigarrow$ biblioteki, nie kopiuj/wklej!
- modularnego $\rightsquigarrow$ pełen rozdział numeryki/fizyki/współbieżności/we-wy
- poddającego się optymalizacji (przez kompilator/autora biblioteki)

"[Object oriented programming] has become recognised as the almost unique successful paradigm for creating complex software"

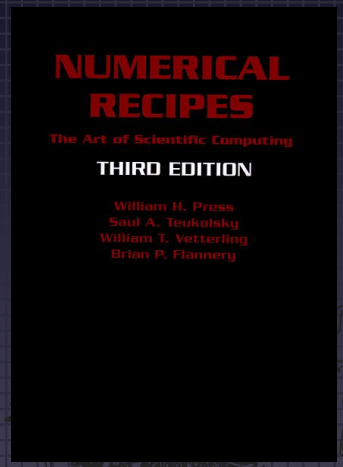
NR: The Art of Scientific Computing (3rd ed., Press et al., 2007)

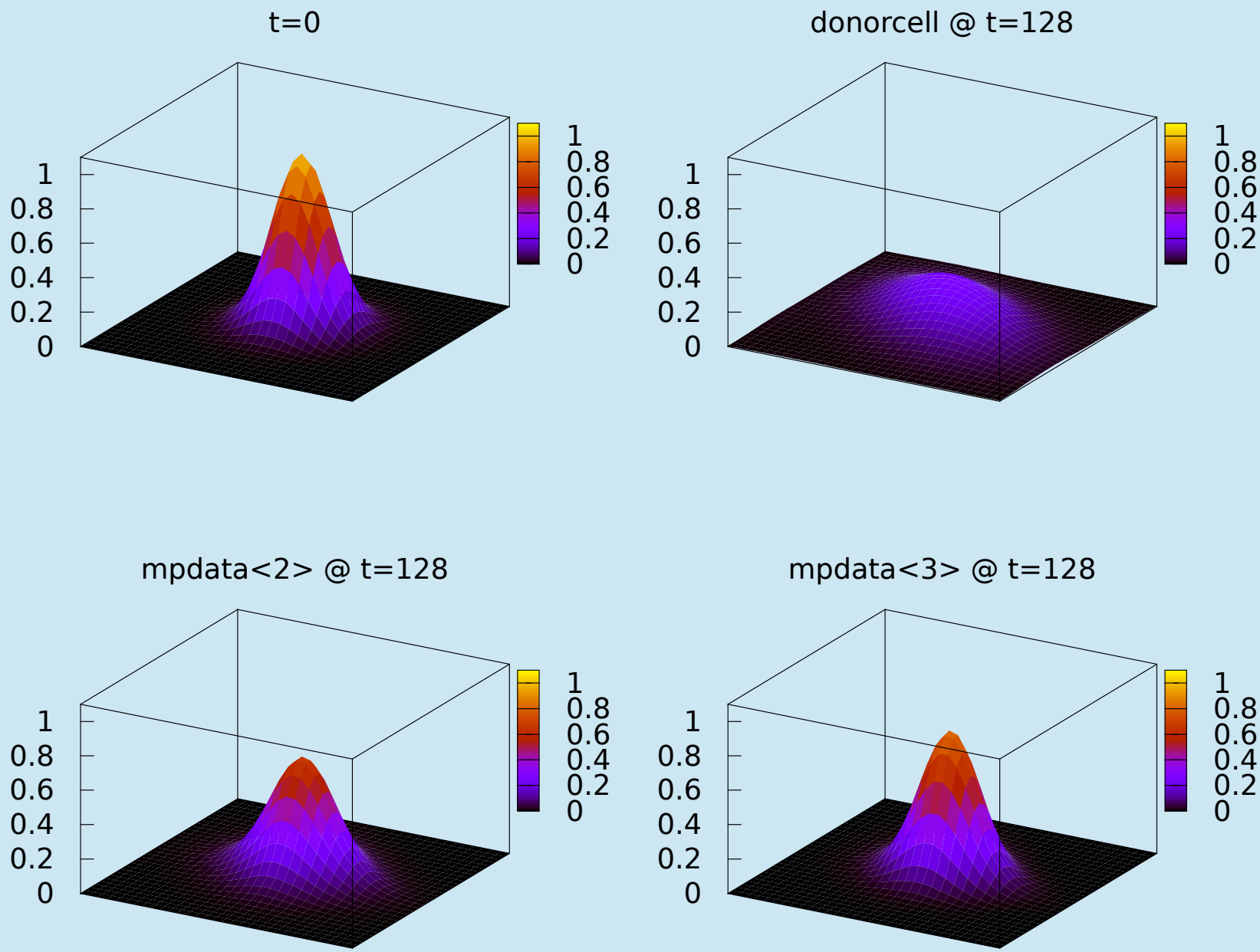
techniki programowania obiektowego ułatwiają tworzenie kodu:

- czytelnego dla człowieka + otwartość $\rightsquigarrow$ możliwość recenzji kodu
- zwięzłego $\rightsquigarrow$ mniejsza podatność na błędy i łatwiejsze ich tropienie
- zdatnego do ponownego użycia $\rightsquigarrow$ biblioteki, nie kopiuj/wklej!
- modularnego $\rightsquigarrow$ pełen rozdział numeryki/fizyki/współbieżności/we-wy
- poddającego się optymalizacji (przez kompilator/autora biblioteki)

"[Object oriented programming] has become recognised as the almost unique successful paradigm for creating complex software"

NR: The Art of Scientific Computing (3rd ed., Press et al. 2007)




t=0
donorcell @ t=128
mpdata<2> @ t=128
mpdata<3> @ t=128
0
0.2
0.4
0.6
0.8
1
nx=32, ny=32, C=(0.5, 0.25)

przykładowa implementacja obiektowa w C++:

tablice i operacje na tablicach	15 LOC
siatka Arakawa-C	15 LOC
prototypowy solwer	40 LOC
permutacje indeksów	15 LOC
periodyczne warunki brzegowe	20 LOC
schemat donor-cell	25 LOC
prędkości antydyfuzyjne	40 LOC
schemat MPDATA (dowolnego rzędu)	40 LOC
wygenerowanie wykresu z poprzedniego slajdu	50 LOC
	< 270 LOC

tablice i operacje na tablicach

biblioteka Blitz++ („expression templates”, ew. inna np. do GPU)

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

makro preprocesora ułatwiające wykorzystanie typu „auto” z C++11

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

automatyczne zwalnianie zaalokowanej pamięci + ujemne indeksy

```
8  #include <boost/ptr_container/ptr_vector.hpp>
9  struct arrvec_t : boost::ptr_vector<arr_t> {
10   const arr_t &operator[](const int i) const {
11     return this->at(
12       (i + this->size()) % this->size()
13     );
14   }
15 };
```

siatka Arakawa-C

instancja „połówki” i przeciążone operatory

```
16 struct hlf_t {} h;
17
18 inline rng_t operator+(const rng_t &i, const hlf_t &) {
19   return i + 1;
20 }
21
22 inline rng_t operator-(const rng_t &i, const hlf_t &) {
23   return i;
24 }
```

dla wygody: zdefiniowanie operacji $\pm$

```
25 template <class n_t>
26 inline rng_t operator^(const rng_t &r, const n_t &n) {
27   return rng_t(
28     (r - n).first(),
29     (r + n).last()
30   );
31 }
```

prototypowy solwer (1/2)

statyczny polimorfizm (adv_t, bcx_t, bcy_t), alokacja pamięci w konstruktorze

```
32  template <class adv_t, class bcx_t, class bcy_t>
33  struct solver_2D
34  {
35    arrvec_t psi, C;
36    int n;
37    rng_t i, j;
38    adv_t adv;
39    bcx_t bcx;
40    bcy_t bcy;
41
42    // ctor
43    solver_2D(int nx, int ny) :
44      n(0), i(0, nx-1), j(0, ny-1), adv(i, j),
45      bcx(i, j, adv_t::n_halos),
46      bcy(j, i, adv_t::n_halos)
47    {
48      const int hlo = adv.n_halos;
49      for (int l = 0; l < 2; ++l)
50        psi.push_back(new arr_t(i ^ hlo, j ^ hlo));
51      // redundant size (to make lower bounds of arrays equal)
52      C.push_back(new arr_t(i ^ h ^ hlo, j ^ hlo));
53      C.push_back(new arr_t(i ^ hlo, j ^ h ^ hlo));
54    }
```

prototypowy solwer (2/2)

gettery state(), Cx(), Cy() (zwracające „widoki” tablic bez halo)

```
55    arr_t state() {
56      return psi[n](i,j).reindex({0,0});
57    }
58    arr_t Cx() {
59      return C[0](i ^ h, j).reindex({0,0});
60    }
61    arr_t Cy() {
62      return C[1](i, j ^ h).reindex({0,0});
63    }
```

metoda solve() z pętlą po czasie (tu ew. OpenMP)

```
64    void solve(const int nt) {
65      for (int t = 0; t < nt; ++t) {
66        for (int s = 0; s < adv.n_steps; ++s) {
67          bcx.fill_halos(psi[n]);
68          bcy.fill_halos(psi[n]);
69          adv.op_2D(psi, n, C, i, j, s);
70          n = (n + 1) % 2;
71        }
72      }
73    }
74  };
```

permutacje indeksów

n.p. jedna funkcja fill_halos() czy donorcell() do zastosowania w dowolnym wymiarze

```
75  template <int d>
76  struct pi;
77
78  template <>
79  struct pi<0> : idx_t {
80    pi(const rng_t &i, const rng_t &j) :
81      idx_t({i,j})
82    {}
83  };
84
85  template <>
86  struct pi<1> : idx_t {
87    pi(const rng_t &j, const rng_t &i) :
88      idx_t({i,j})
89    {}
90  };
```

periodyczne warunki brzegowe

działa w dowolnym kierunku/wymiarze „d", separacja od numeryki (tu ew. MPI)

```
91  template <int d>
92  struct cyclic
93  {
94    // member fields
95    pi<d> left_halo, rght_edge, rght_halo, left_edge;
96
97    // ctor
98    cyclic(const rng_t &i, const rng_t &j, const int hlo) :
99      //        (   i.first    ...     i.last          ), (  j )
100     left_halo(rng_t(i.first()-hlo,  i.first()-1    ), j^hlo),
101     rght_edge(rng_t(i.last()-hlo+1, i.last()       ), j^hlo),
102     rght_halo(rng_t(i.last()+1,     i.last()+hlo   ), j^hlo),
103     left_edge(rng_t(i.first(),      i.first()+hlo-1), j^hlo)
104   {}
105
106   void fill_halos(const arr_t &psi)
107   {
108     psi(left_halo) = psi(rght_edge);
109     psi(rght_halo) = psi(left_edge);
110   }
111 };
```

schemat donor-cell (1/2)

$$F(\psi_L, \psi_R, C) = \max(C, 0) \cdot \psi_L + \min(C, 0) \cdot \psi_R$$

```
112 template <class T1, class T2, class T3>
113 inline auto F(
114   const T1 &psi_l, const T2 &psi_r, const T3 &C
115 ) return_macro(
116   .5 * (C + abs(C)) * psi_l +
117   .5 * (C - abs(C)) * psi_r
118 )
```

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

```
119 template <int d>
120 inline auto donorcell_1D(
121   const arr_t &psi, const arr_t &C,
122   const rng_t &i, const rng_t &j
123 ) return_macro(
124   F(psi(pi<d>(i,  j)), psi(pi<d>(i+1,j)), C(pi<d>(i+h,j))) -
125   F(psi(pi<d>(i-1,j)), psi(pi<d>(i,  j)), C(pi<d>(i-h,j)))
126 )
```

schemat donor-cell (2/2)

$$\psi_{i,j}^{[n+1]} = \psi_{i,j}^{[n]} - \sum_{d=0}^{N-1} \left(F\left[\psi_{i,j}^{[n]}, \psi_{i,j+\pi_{1,0}^d}^{[n]}, C_{i,j+\pi_{1/2,0}^d}^{[d]}\right] - F\left[\psi_{i,j+\pi_{-1,0}^d}^{[n]}, \psi_{i,j}^{[n]}, C_{i,j+\pi_{-1/2,0}^d}^{[d]}\right] \right)$$

```
127 inline void donorcell_2D(
128   const arrvec_t &psi, const int n,
129   const arrvec_t &C,
130   const rng_t &i, const rng_t &j
131 ) {
132   psi[n+1](i,j) = psi[n](i,j)
133     - donorcell_1D<0>(psi[n], C[0], i, j)
134     - donorcell_1D<1>(psi[n], C[1], j, i);
135 }
```

- uogólnienie do trzech wymiarów: 1 linijka więcej!
- brak pętli $\rightsquigarrow$ przejrzysty kod, ale też separacja od logiki halo
- mniej znaków niż na opisanie w LaTeXu

prędkości antydyfuzyjne (1/3)

dla wygody: definicja „ułamka”

```
136 template <class nom_t, class den_t>
137 static inline auto frac(
138   const nom_t &nom, const den_t &den
139 ) return_macro(
140   where(den > 0, nom / den, 0)
141 )
```

$$A_{i,j}^{[d]} = \frac{\psi_{i,j+\pi_{1,0}^d} - \psi_{i,j}}{\psi_{i,j+\pi_{1,0}^d} + \psi_{i,j}}$$

```
142 template <int d>
143 inline auto A(
144   const arr_t &psi, const rng_t &i, const rng_t &j
145 ) return_macro(
146   frac(
147     psi(pi<d>(i+1, j)) - psi(pi<d>(i,j)),
148     psi(pi<d>(i+1, j)) + psi(pi<d>(i,j))
149   )
150 )
```

prędkości antydyfuzyjne (2/3)

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

```
151 template <int d>
152 inline auto B(
153   const arr_t &psi, const rng_t &i, const rng_t &j
154 ) return_macro(
155   .5 * frac(
156     psi(pi<d>(i+1, j+1)) + psi(pi<d>(i, j+1)) -
157     psi(pi<d>(i+1, j-1)) - psi(pi<d>(i, j-1)),
158     psi(pi<d>(i+1, j+1)) + psi(pi<d>(i, j+1)) +
159     psi(pi<d>(i+1, j-1)) + psi(pi<d>(i, j-1))
160   )
161 )
```

- te funkcje zwracają części składowe wyrażeń macierzowych
- kompilator tworzy z nich jedną pętlę po elementach

prędkości antydyfuzyjne (3/3)

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

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

```
162 template <int d>
163 inline auto antidiff_2D(
164   const arr_t &psi,
165   const rng_t &i, const rng_t &j,
166   const arrvec_t &C
167 ) return_macro(
168   abs(C[d](pi<d>(i+h, j))) * (1 - abs(C[d](pi<d>(i+h, j))))
169   * A<d>(psi, i, j) - C[d](pi<d>(i+h, j)) * .25
170   * (
171     C[d-1](pi<d>(i+1, j+h)) +
172     C[d-1](pi<d>(i,   j+h)) +
173     C[d-1](pi<d>(i+1, j-h)) +
174     C[d-1](pi<d>(i,   j-h))
175   ) * B<d>(psi, i, j)
176 )
```

schemat MPDATA (dowolnego rzędu) (1/2)

liczba iteracji MPDATA jako parametr wzorca struktury

```
177 template <int n_iters>
178 struct mpdata
179 {
180   // member fields
181   arrvec_t tmp0, tmp1;
182   static const int n_steps = n_iters;
183   static const int n_halos = n_iters;
184
185   // ctor
186   mpdata(const rng_t &i, const rng_t &j) {
187     const int hlo = n_steps;
188     tmp0.push_back(new arr_t(i ^ h ^ hlo, j ^ hlo));
189     tmp0.push_back(new arr_t(i ^ hlo, j ^ h ^ hlo));
190     if (n_iters < 2) return;
191     tmp1.push_back(new arr_t(i ^ h ^ hlo, j ^ hlo));
192     tmp1.push_back(new arr_t(i ^ hlo, j ^ h ^ hlo));
193   }
```

schemat MPDATA (dowolnego rzędu) (2/2)

metoda op_2D() wywoływana przez solwer

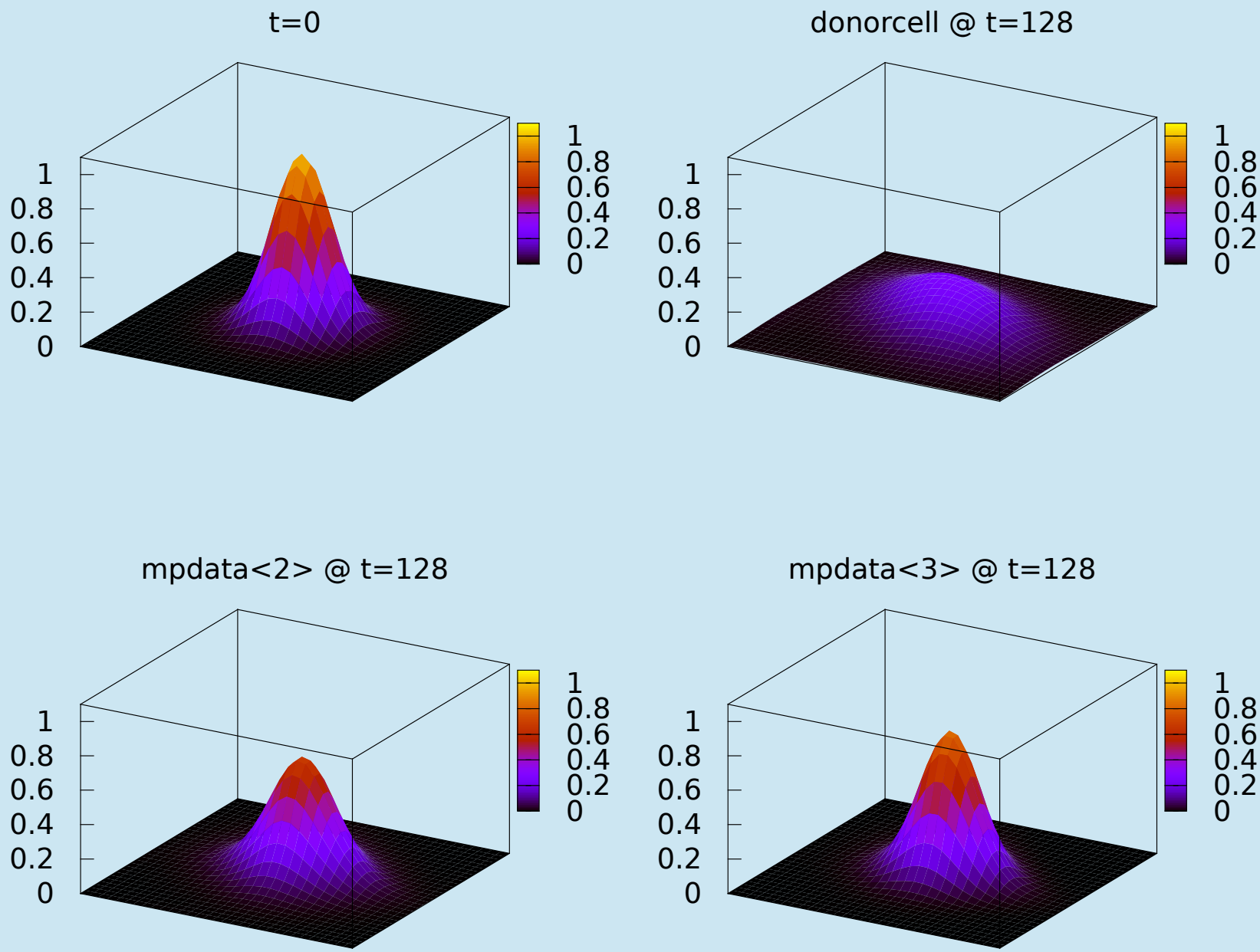
```
194   inline void op_2D(
195     const arrvec_t &psi, const int n, const arrvec_t &C,
196     const rng_t &i, const rng_t &j, const int step
197   ) {
198     // chosing input/output for antidiff. velocity
199     const arrvec_t
200       &C_unco = (step == 1) ? C : (step % 2) ? tmp0 : tmp1,
201       &C_corr = (step  % 2) ? tmp1 : tmp0;
202     const int hlo = n_steps - 1 - step;
203     // calculating the antidiffusive velocities
204     if (step > 0) {
205       const rng_t // ext. ranges as we only use C_+1/2
206         im(i.first() - 1, i.last()),
207         jm(j.first() - 1, j.last());
208       C_corr[0](im + h, j ^ hlo) =
209         antidiff_2D<0>(psi[n], im, j ^ hlo, C_unco);
210       C_corr[1](i ^ hlo, jm + h) =
211         antidiff_2D<1>(psi[n], jm, i ^ hlo, C_unco);
212     }
213     // performing a donor-cell step with C or C_corr
214     donorcell_2D(psi, n, step==0 ? C : C_corr, i, j);
215   }
216 };
```

Przykład użycia:

- zadanie warunku początkowego
- całkowanie z różną liczbą iteracji
- wykreślenie wyniku poprzez gnuplot-iostream

```
217 #include "listings.hpp"
218 #define GNUPLOT_ENABLE_BLITZ
219 #include <gnuplot-iostream/gnuplot-iostream.h>
220
221 template <class slv_t>
222 void init(slv_t &slv, const int nx, const int ny) {
223   blitz::firstIndex i;
224   blitz::secondIndex j;
225   slv.state() = exp(
226     -sqr(i-nx/2.) / (2.*pow(nx/10,2))
227     -sqr(j-ny/2.) / (2.*pow(ny/10,2))
228   );
229   slv.Cx() = .5;
230   slv.Cy() = .25;
231 }
232
233 int main() {
234   Gnuplot gp;
235   gp << "set term pdf size 13cm, 11cm\n"
236         "set output 'figure.pdf'\n"     "set nosurface\n"
237         "set border 4095\n"             "unset xtics\n"
238         "unset ytics\n"                 "set ticslevel 0\n"
239         "set cbrange [0:1.1]\n"         "set zrange [0:1.1]\n"
240         "set multiplot layout 2,2\n"    "set pm3d\n";
241   int nx = 32, ny = 32, nt = 128;
242   std::string fmt;
```

```
243   {
244     solver_2D<mpdata<1>, cyclic<0>, cyclic<1>> slv(nx, ny);
245     init(slv, nx, ny);
246     fmt = gp.binfmt(slv.state());
247     gp << "set title 't=0'\n"
248           "splot '-' binary" << fmt << " notitle\n";
249     gp.sendBinary(slv.state().copy());
250     slv.solve(nt);
251     gp << "set title 'donorcell @ t=" << nt << "'\n"
252           "splot '-' binary" << fmt << " notitle\n";
253     gp.sendBinary(slv.state().copy());
254   } {
255     solver_2D<mpdata<2>, cyclic<0>, cyclic<1>> slv(nx, ny);
256     init(slv, nx, ny);
257     slv.solve(nt);
258     gp << "set title 'mpdata<2> @ t=" << nt << "'\n"
259           "splot '-' binary" << fmt << " notitle\n";
260     gp.sendBinary(slv.state().copy());
261   } {
262     solver_2D<mpdata<3>, cyclic<0>, cyclic<1>> slv(nx, ny);
263     init(slv, nx, ny);
264     slv.solve(nt);
265     gp << "set title 'mpdata<3> @ t=" << nt << "'\n"
266           "splot '-' binary" << fmt << " notitle\n";
267     gp.sendBinary(slv.state().copy());
268   }
269 }
```


t=0
donorcell @ t=128
mpdata<2> @ t=128
mpdata<3> @ t=128
0
0.2
0.4
0.6
0.8
1
nx=32, ny=32, C=(0.5, 0.25)

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja MPDATA)
- możliwość tworzenia bibliotek wysokiego poziomu (konstrukcja libmpdata++)
- wysoka zgodność kompilatorów ze standardem

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja 1D MPDATA)
- możliwość obsługi wyjątków w przypadku błędów (konwergencja iteracji)
- wysoka zgodność kompilatorów ze standardem

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja 1D MPDATA)
- możliwość tworzenia wydajnych przykładów/testów jednostkowych
- wysoka zgodność kompilatorów ze standardem

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja 1D MPDATA)
- możliwość obsługi wyjątków we/wy (kontrola poprawności)
- wysoka zgodność kompilatorów ze standardem

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja MPDATA)
- możliwość obsługi wyjątków (przydatna przy [illegible])
- wysoka zgodność kompilatorów ze standardem

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja 1D MPDATA)
- możliwość wykorzystania wyrafinowanych technik kompilacji
- wysoka zgodność kompilatorów ze standardem

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja MPDATA)
- możliwość obsługi wyjątków we/wy (kontrola poprawności danych)
- wysoka zgodność kompilatorów ze standardem

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja 1D, 2D, 3D MPDATA)
- przenośność (brak wymuszonych przez kompilator zależności od platformy)
- wysoka zgodność kompilatorów ze standardem

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja MPDATA z Jakobianem lub bez, dla pól o zmiennym znaku lub o stałym)
- możliwość budowy wydajnych, przenośnych kontenerów i iteratorów
- wysoka zgodność kompilatorów ze standardem

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja MPDATA z Jakobianem lub bez, dla pól o zmiennym znaku lub o stałym)
- możliwość analizy wymiarowej podczas kompilacji (jednostki)
- wysoka zgodność kompilatorów ze standardem

podsumowanie

zastosowanie OOP do implementacji schematu MPDATA:

- umożliwia pełną separację numeryki, fizyki, współbieżności, we-wy
- skutkuje zwięzłością kodu
  (C++, Python, gorzej z FORTRANem 2008)
- pozwala na odwzorowanie notacji matematycznej w kodzie

wartość dodana w przypadku wyboru C++:

- wysoka wydajność ($\geqslant$ FORTRAN)
- duże środowisko (i rynek) programistów
- dostęp do bibliotek obiektowych (np. Boost.MPI, Thrust, ...)
- możliwość eliminacji duplikacji kodu bez utraty wydajności przy pomocy polimorfizmu statycznego (np. wersja MPDATA z Jakobianem lub bez, dla pól o zmiennym znaku lub o stałym)
- możliwość analizy wymiarowej podczas kompilacji (jednostki)
- wysoka zgodność kompilatorów ze standardem

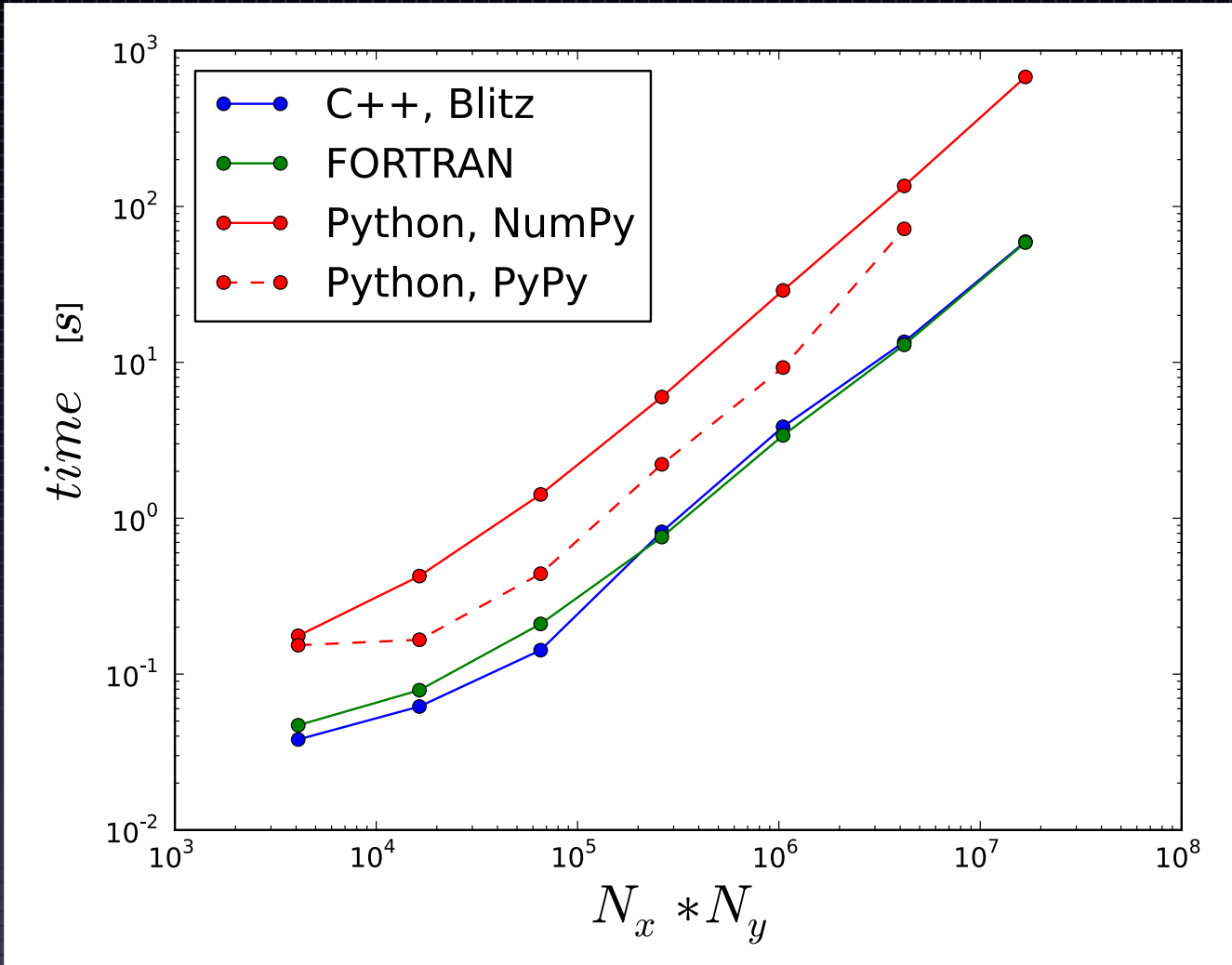
Dziękuję za uwagę!

podziękowania:
Piotr Smolarkiewicz
zespół Blitz++

wykorzystane biblioteki:
Blitz++
Boost.ptr_vector
gnuplot-iostream

finansowanie:
NCN (2011/01/N/ST10/01483)
FNP START

C++ vs. FORTRAN vs. Python: wydajność



dla donor-cell 2D (Jarecka et al. 2012 EGU, artykuł w przygotowaniu)

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

http://www.nature.com/news/2010/101013/full/467775a.html

C++ can check units for you (at no runtime cost!)

κ-Köhler parameterisation

```
  /// @brief activity of water in solution
  /// (eqs. 1,6) in @copydetails Petters_and_Kreidenweis_2007
  template <typename real_t>
  quantity<si::dimensionless, real_t> a_w(
    quantity<si::volume, real_t> rw3,
    quantity<si::volume, real_t> rd3,
    quantity<si::dimensionless, real_t> kappa
  )
  {
    return (rw3 - rd3) / (rw3 - rd3 * (real_t(1) - kappa));
  }
};
```

$\rightsquigarrow$ Boost.units

http://boost.org/doc/libs/release/libs/units/