Productivity-oriented software design for geoscientific modelling

Dorota Jarecka, Sylwester Arabas, Anna Jaruga

University of Warsaw

2014 April 7th
UCAR SEA Software Engineering Conference 2014
Boulder, CO, USA
1. Introduction

2. Case study: **libmpdata++ & libcloudph++**
   developed at the University of Warsaw

3. Future plans: (my upcoming 2-year **postdoc @NCAR**)
1 Introduction

2 Case study: libmpdata++ & libcloudph++
   developed at the University of Warsaw

3 Future plans: (my upcoming 2-year postdoc @NCAR)
personal background

MSc theoretical elementary particle physics

PhD computational (F77) atmospheric physics

lessons learnt:

- prefer Python/NumPy to F77
- my research productivity can be improved!
  ↳ software design does matter!
lesson learnt - common perspective

Merali 2010 (Nature 467)

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

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

Well, I finished the computer program I needed for my research.

It's not pretty but it works.

It's inelegant, messy, inefficient, and if you try to change anything about it, it will probably explode.

Sounds like every academic I've ever met.

Great! Can I get my Ph.D. now?

www.phdcomics.com
software design and researchers’ productivity

users’ perspective
- ease of use
- robustness
- result reproducibility

developers’ perspective
- extendability
- maintainability

researcher = user & developer
researcher = user & developer

Merali 2010 (Nature 467)

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

...why scientific programming does not compute

>

...SCIENTISTS AND THEIR SOFTWARE

A survey of nearly 2,000 researchers showed how coding has become an important part of the research toolkit, but it also revealed some potential problems.

> 45% said scientists spend more time today developing software than five years ago."

> 38% of scientists spend at least one fifth of their time developing software.
this talk: lessons learnt in a joint U. Warsaw/NCAR project


the aim
develop a productivity-oriented open-source software suite for aerosol/cloud-microphysics research

the team

NCAR: W. Grabowski & D. Jarecka
ECMWF: P. Smolarkiewicz (previously @NCAR)
talk outline

1. Introduction

2. Case study: libmpdata++ & libcloudph++
devolved at the University of Warsaw

3. Future plans: (my upcoming 2-year postdoc @NCAR)
libmpdata++ & libcloudph++

libmpdata++ parallel solvers for systems of transport equations
- http://libmpdataxx.igf.fuw.edu.pl/

libcloudph++ aerosol/cloud-microphysics algorithm collection
- http://libcloudphxx.igf.fuw.edu.pl/
a few words on first design choices

- structure the code into “standalone” libraries
  - easier to document, to test and to contribute to
  - easier to use in various contexts
- leverage existing **reusable** software
  - save time, better test coverage

```plaintext
icicle

libcloudph++
  - Thrust
  - CUDA or OpenMP

libmpdata++
  - Boost (Units, odeint, ...
  - Blitz++
  - HDF5
  - OpenMP (or Boost.Thread)
  - Boost (...

Boost
  - (program_options, Spirit, ...
```

Dorota Jarecka: Productivity-oriented software design for geoscientific modelling 9 / 30
software design and researchers’ productivity

**users’ perspective**
- ease of use
- robustness
- result reproducibility

**developers’ perspective**
- extendability
- maintainability

researcher = user & developer
ease of use (and misuse) - model users’ perspective

• understandable output
  ~ docs, open data format

• easy way of providing input and “setup”
  (setups are much more than “parameters”!)
  ~ docs, setup/solver separation!

• setup implementation out of tree
  (structuring into libraries helps!)

• as few constraints as possible
  (e.g. dimensionality, data types ~ C++ templates)

• library API: flexible and documented
complete API docs on arXiv

libcloudph++ 0.1: single-moment bulk, double-moment bulk, and particle-based warm-rain microphysics library in C++

Sylwester Arabas, Anna Jaruga, Hanna Pawlowska, Wojciech W. Grabowski

(Submitted on 7 Oct 2013)

code part of the paper!

tion, and are grouped into a structure named lgrngn::opts_init_t (Listing 5.2). The initial

template<typename real_t>
struct opts_init_t
{
  // initial dry sizes of aerosol
  typedef boost::ptr_unordered_map<
    real_t, // kappa
    unary_function<real_t> // n(ln(rd)) @ STP
  > dry_distros_t;
  dry_distros_t dry_distros;

  // Eulerian component parameters
  int nx, ny, nz;
  real_t dx, dy, dz, dt;

  // mean no. of super-droplets per cell
  real_t sd_conc_mean;

  // coalescence Kernel type
  kernel_t kernel;

  // ctor with defaults (C++03 compliant) ...

Listing 5.2: lgrngn::opts_init_t structure definition

dry size spectrum of aerosol is represented with a map associating values of the solubility parameter \( \kappa \) with pointers to functors returning con-
software design and researchers’ productivity

users’ perspective
- ease of use
- robustness
- result reproducibility

developers’ perspective
- extendability
- maintainability

researcher = user & developer
robustness - model users’ perspective

- detect and report out-of-range model parameters, e.g.
  - warm-rain microphysics below a freezing point
  - numerical stability criteria,
  - ill-posed initial conditions.

~~ numerous asserts in ibmpdata++ & libcloudph++
(can be off for production runs - CMake’s Release/Debug modes)

- sane error handling:
  - let the user choose what to ignore (do not ignore errors by default!),
  - propagate system/library errors (e.g. i/o, numerics),

~~ take advantage of C++/Python exceptions

- mention all above in the docs...
software design and researchers’ productivity

users’ perspective
• ease of use
• robustness
• result reproducibility

developers’ perspective
• extendability
• maintainability

researcher = user & developer
result reproducibility
- model users’/reviewers’ perspective

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

- “paper must be accompanied by the code, or means of accessing the code, for the purpose of peer-review”

- “we strongly encourage referees to compile the code, and run test cases supplied by the authors”

- ...
result reproducibility
- model users’/reviewers’ perspective

- access to software and correct version of the code
  \(\leadsto\) free/libre & open code and version history

- avoid vendor-specific hardware requirements
  \(\leadsto\) e.g., use GPU/CUDA but offer a fallback option

---

libcloudph++

compile-time CUDA / OpenMP choice

99% common code \(\leadsto\) implemented using **Thrust**
**users’ perspective**  
- ease of use  
- robustness  
- result reproducibility  

**developers’ perspective**  
- extendability  
- maintainability  

**researcher = user & developer**
extendability - model developers’ perspective

- language with a market of trained personnel (Python, C++, ...)

- separation of concerns (cloud physicist does cloud physics, etc.)
  \[ \Rightarrow \text{structuring into libraries helps again!} \]

- human-readable code (code vs. paper – they both describe the same algorithm)
  \[ \Rightarrow \text{dimensional analysis} \]
// Reynolds number for a particles falling with terminal velocity
// see e.g. section 4 in Smolik et al 2001, Aerosol Sci.

template<typename real_t>
BOOST_GPU_ENABLED
quantity<si::dimensionless, real_t> Re(
    const quantity<si::velocity, real_t> v_term,    // particle terminal velocity
    const quantity<si::length, real_t> r_w,        // particle wet radius
    const quantity<si::mass_density, real_t> rho,  // air density
    const quantity<si::dynamic_viscosity, real_t> eta // air viscosity
)
{
    return v_term * (2 * r_w) * rho / eta;
}
extendability - model developers’ perspective

• language with a market of trained personnel
  (Python, C++, . . . )

• separation of concerns
  (cloud physicist does cloud physics, etc.)
  \[\Rightarrow\] structuring into libraries helps again!

• human-readable code
  (code vs. paper – they both describe the same algorithm)
  \[\Rightarrow\] dimensional analysis
  \[\Rightarrow\] “blackboard abstractions”
OOP formula translation: C++, Fortran or Python?

Arabas, Jarecka et al. 2014

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

Sylwester Arabas¹, Dorota Jarecka¹, Anna Jaruga¹, Maciej Fijałkowski²

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

Journal: Scientific Programming
DOI: 10.3233/SPR-140379
Online Date: Monday, March 24, 2014
\[
\psi_{i,j}^{n+1} = \psi_{i,j}^{n} - (F[\psi_{i,j}^{n}, \psi_{i+1,j}^{n}, C_{i+1/2}] - F[\psi_{i-1,j}^{n}, \psi_{i,j}^{n}, C_{i-1/2}]) \\
F(\psi_L, \psi_R, C) = \frac{C + |C|}{2} \cdot \psi_L + \frac{C - |C|}{2} \cdot \psi_R
\]

```python
def adv_op(psi, C, i):
    return 
    f(psi[i], psi[i+one], C[i+hlf]) - 
    f(psi[i-one], psi[i], C[i-hlf])

def scalar_advection(psi, n, C, i):
    psi[n+1][i] = psi[n][i] - adv_op(psi[n], C, i)

def f(psi_l, psi_r, C):
    return (C + abs(C))/2 * psi_l + (C - abs(C))/2 * psi_r```

human-readable array expressions!
software design and researchers’ productivity

**users’ perspective**

- ease of use
- robustness
- result reproducibility

**developers’ perspective**

- extendability
- maintainability

researcher = user & developer
good test coverage

separations of concerns requires testing!

⇝ structuring into libraries helps again!
example: libmpdata++ tests

Jaruga et al. 2014 (in preparation)

- unmodified code, out-of-tree setups
- but these are not unit tests...
1 Introduction

2 Case study: libmpdata++ & libcloudph++
developed at the University of Warsaw

3 Future plans: (my upcoming 2-year postdoc @NCAR)
my upcoming 2-year postdoc @NCAR

funding (2014–2016)

Ministry of Science and Higher Education
Republic of Poland

the aim

- development of a new parameterization of processes related to droplet growth and rain formation in numerical cloud models
- development of a testing protocol for microphysical schemes

collaboration with

NCAR: W. Grabowski (MMM), D. Del Vento (CISL)
project’s approach

- reuse reliable existing code whenever possible
- code in Python/NumPy whenever possible

\[\Rightarrow \text{libmpdata++ and libcloudph++ do allow it!}\]
project’s methodology

• writing automated tests during code development
  ⇝ verify behaviors of schemes during the development

• using technics of object-oriented programming
  ⇝ help to maintain modularity and improve readability

• using the Python/Numpy language/library with solutions for improving the relatively poor performance
  ⇝ benefit from Python’s ease of use
OOP formula translation: C++, Fortran or Python?

Arabas, Jarecka et al. 2014

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

Sylwester Arabas¹, Dorota Jarecka¹, Anna Jaruga¹, Maciej Fijałkowski²

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

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

• Python is definitely the easiest language to use and debug among the three languages!

• PyPy performance is promising!
a take-home message

productivity-oriented design choices
⇝ investment that does pay back

further reading:
• libcloudph++: [http://libcloudphxx.igf.fuw.edu.pl/](http://libcloudphxx.igf.fuw.edu.pl/)
• libmpdata++: [http://libmpdataxx.igf.fuw.edu.pl/](http://libmpdataxx.igf.fuw.edu.pl/)

Thanks for your attention!
Suggestions are very welcome!