FUTILE

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

Dictionaries

Structured I/O

YAML

Code Profiling

Outlook

FORTRANCON 2020

The FUTILE project: an embedded DSL to simplify the treatment of low-level operation in large Fortran programs

Luigi Genovese

L_Sim – CEA Grenoble

July 3, 2020

University of Zurich (Virtually)

BigDFT: DFT with wavelets (www.bigdft.org)

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

Dictionaries

Structured I/O

YAML

Code Profiling

Outlook

A Density Functional Theory code conceived for HPC

- DFT calculations with many thousands atoms
- ☛ Simulation setups (input params.) become system dependent
- The application has to be composed in various ways (MPI + OpenMP + GPU)
- ☛ Datas are more difficult to manage for reproducibility

Solutions?

- Have manageable Input/Output structure
- Task scheduling: resource-balancing at the runtime
- Performance instrumentation and prediction
- ☛ Identify appropriate coding paradigms
  Conceive a DSL for *development in HPC*
- "*Offline*" postprocessing data treatment
  Facilitate the *continuity* of scientific research

Code release and distribution

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

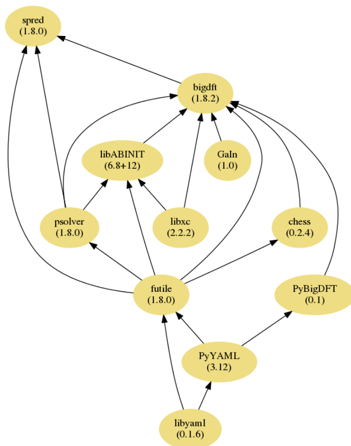
Dictionaries

Structured I/O

YAML

Code Profiling

Outlook



Modularity first

Each section of BigDFT is, when appropriate, defined as a module with its own build system and compilation instructions. At present:

- PSolver 1.8
- PyBigDFT 0.1
- FUTILE 1.0

Contain*ered* ! `hub.docker.com/r/bigdft`

BigDFT suite is relased as a `docker` container (GPU-direct ready) ☛(also on NVidia NGC repository)

Real-world problem: Dealing with Legacy codes

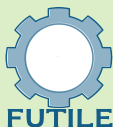
BigDFT experience
The FUTILE Project
Memory Allocation
Code restructuring
Dictionaries
Structured I/O
YAML
Code Profiling
Outlook

How to implement modern ideas in computer physics development? ☛ Mainly a *cultural* problem in the community.
Several problems/questions:

- Portability and Maintainance: if new ideas are implemented in existing codes, how to disseminate them? Will this lead to true improvements or "just" homebrew solutions?
- Diffusion of disruptive solutions: if new libraries/modules are written from scratch, how to make them compatible with existing codes?

FUTILE: an opportunity to move forward

- An embedded DSL written for fortran codes
- Suitably extended to C, improves interoperability
- Separate low-level concepts from their implementations.



What is an embedded DSL

FUTILE
Reinventing the wheel

BigDFT experience

The FUTILE Project
Memory Allocation
Code restructuring
Dictionaries
Structured I/O
YAML
Code Profiling

Outlook

Employs Fortran standards to generalize low-level operations

```fortran
subroutine test_routine(i2)
 use futile
 integer, dimension(:,:), intent(in) :: i2
 integer, allocatable :: i1(:),j1(:),i3(:,:,:)
 real(f_double),allocatable :: d1(:),d2(:,:)
 character(len=256), allocatable :: c1(:)
 call f_routine(id='test_routine') !body
 d1 = f_malloc0(17,id='d1') ! (1:17) set to 0
 d2 = f_malloc([0.to.5,-1.to.1],id='d2')
 i3 = f_malloc([10,3,2],id='i3') !(1:10,1:3,1:2)
 j2 = f_malloc(src=i2,id='j2') !all. + copy
 c1 = f_malloc_str(256,5,id='c1') !(w/ len)
 call f_free(d1) !deallocate
 ! ...
 call f_release_routine()
end subroutine test_routine
```

Usage of database in the libraries API

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

Dictionaries

Structured I/O

YAML

Code Profiling

Outlook

Imagine we want to write a module that might be used in different codes

A very large set of options: which API?

- Traditional functionalities easy-to-use (default parameters)
- Developers should not change API while introducing new functionalities
- ☛ Integration must be done "once"!

The database as an I/O of options for integrations

Putting the library options as a dictionary solves *a lot* or problems

- Reduce *intrusivity* in the host code
- Allow *extension* of the used library at no cost
- Make the library accessible by other developers

Serialization example: Dictionaries in legacy codes

Reinventing the wheel

BigDFT experience
The FUTILE Project
Memory Allocation
Code restructuring
Dictionaries
Structured I/O
YAML
Code Profiling
Outlook

Python traditional

```
wc={}
wc['brazil'] = 5 # update...
wc['italy'] = 4   wc['france'] += 1 #increment
wc['france'] = 1
```

```
type(dictionary), pointer :: wc
call dict_init(wc) ! or also wc => dict_new()
call dict_set(wc//'brazil',5)
call dict_set(wc//'italy',4)
call dict_set(wc//'france',1)
!or also
wc => yaml_load('{ brazil: 5, italy: 4, france: 1}')
! update...
icups = wc // 'france' !store
call dict_set(wc // 'france', icups+1) !increment
```

Example: API of ISF Poisson Solver v 1.8

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

Dictionaries

Structured I/O

YAML

Code Profiling

Outlook

```
call f_lib_initialize() !initialize futile modules
call dict_init(inputs) !default values (inputs={})
!override if willing (example of GPU)
if (gpu) &!in python inputs['setup']['accel']='CUDA'
  &call dict_set(inputs// 'setup'// 'accel', 'CUDA')
!set the kind of solver and communicator
kernel=pkernel_init(mpirank,mpisize,inputs, & !setup
  & 'F',ndims,hgrids) !geometry
!free input variables if needed
call dict_free(inputs)
call pkernel_set(kernel) !allocate buffers
![...]  loop of DFT code
!transform density in potential
call Electrostatic_Solver(kernel,rhoV,energies)
print *,energies%hartree
![...]  end DFT code loop
call pkernel_free(kernel) !release buffers
call f_lib_finalize()
```

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

Dictionaries

Structured I/O

YAML

Code Profiling

Outlook

A very large set of options: input file

Fundamental requirements

- Traditional functionalities should be easy-to-use (default parameters)
- New functionalities should not interfere with previous keywords
- ☛ Unrelated keyword should be allowed in the input file

"FUTILE" approach to input variables and options

- Markup languages offer the good flexibility (`key` $\rightarrow$ `value`)
- Evident problems of portability with homebrew solutions or Fortran `NAMELIST` approach
- Options might be "polymorphic" (scalars or arrays, lists)
- ☛ YAML Markup *dialect* presents ideal features

Example of an input file (NAMELIST format)

FUTILE

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

Dictionaries

Structured I/O

YAML

Code Profiling

Outlook

```
&setup:
   taskgroup_size= 0 !  Size of the taskgroups
   accel= 0 ! Acceleration
   global_data    = .false.
   verbose        = .true. !Verbosity switch
   output         = 0 !Quantities to be plotted
&end
&kernel
   screening      = 0 !Mu screening parameter
   isf_order      = 16 !Order of the ISF family
   stress_tensor  = .true. !Extract stress tensor
&end
&environment
   cavity         = .false. !Type of the cavity
   input_guess    = .true. !Input guess procedure
   fd_order       = 16 !Order of FD derivatives
&end
```

Example of an input file (XML format)

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

Dictionaries

Structured I/O

YAML

Code Profiling

Outlook

```xml
<setup>
    <taskgroup_size>0</taskgroup_size>
    <accel>none</accel>
    <global_data>No</global_data>
    <verbose>Yes</verbose>
    <output>none</output>
</setup>
<kernel>
    <screening>0</screening>
    <isf_order>16</isf_order>
    <stress_tensor>Yes</stress_tensor>
</kernel>
<environment>
    <cavity>none</cavity>
    <input_guess>Yes</input_guess>
    <fd_order>16</fd_order>
</environment>
```

Example of an input file (YAML format)

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

Dictionaries

Structured I/O

YAML

Code Profiling

Outlook


```
setup:
    taskgroup_size: 0 #  Size of the taskgroups
    accel: none # Acceleration
    global_data :    No
    verbose :        Yes #Verbosity switch
    output :         none #Quantities to be plotted
kernel:
    screening :      0 #Mu screening parameter
    isf_order :      16 #Order of the ISF family
    stress_tensor :  Yes #Extract stress tensor
environment:
    cavity :         none #Type of the cavity
    input_guess :    Yes #Input guess procedure
    fd_order :       16 #Order of FD derivatives
    itermax :        50 #Max. No.of GPS iterations
    minres :         1.e-6 #Convergence threshold
```

YAML ads: ideal for electronic structure codes

Markup Language has fundamental advantages

- Creating Databases of calculations (e.g. high-troughput, data mining)
- Driving Workflows automatically

- Link Differences bw YAML, JSON, XML
- Link A serialization of a python dictionary
- Can be straightforwardly used for the input file
- Used for the logfile of BigDFT
- ☛ A logfile may become an input file!

Both for the Human and the Robot

Provides two *views* of the file: text and/or dictionary

YAML Emitter and parser

FUTILE

Reinventing the wheel

BigDFT experience
The FUTILE Project
Memory Allocation
Code restructuring
Dictionaries
Structured I/O
YAML
Code Profiling
Outlook

```
call yaml_mapping_open('Invoice')
  call yaml_map('sku','BL394D')
  call yaml_map('quantity',4)
  call yaml_map('description','Basketball')
  call yaml_map('price',450.0,fmt='(f8.2)')
  call yaml_map('parcel dimensions',[30,32,35])
call yaml_mapping_close()
```

```
Invoice:
  sku :                 BL394D
  quantity :            4
  description :         Basketball
  price :               450.00
  parcel dimensions :   [30, 32, 35]
```

Parsing can be executed via libYaml

Dictionaries enables to *store* ML information in a low-level language code!

Profiling

FUTILE
Reinventing the wheel

BigDFT experience

The FUTILE Project
Memory Allocation
Code restructuring
Dictionaries
Structured I/O
YAML
Code Profiling

Outlook

Timing

- f_timing("category", ON/OFF) : hierarchical, aggregated with others of the same kind
- f_routine("myroutinename") : meant for routine profiling, keeps the tree of callers

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation
Code restructuring
Dictionaries
Structured I/O
YAML
Code Profiling

Outlook

```
WFN_OPT: # % ,  Time (s), Load per MPI proc (relati
 Classes:
   Flib LowLevel :        [ 2.9,  18., ...]
   Communications :       [ 5.5,  34., ...]
 Categories: #Ordered by time consumption
  Precondition:
    Data :               [ 65.0,  4.01E+02, ..]
    Class :              Convolutions

 - call_bigdft: [ 679.,  1,  99.76%*]
   - cluster: [ 679.,  1,  100.04%]
     - preconditionall2: [ 438.,  4,  64.57%]
       - segment_invert: [ 6.46,  656,  1.48%]
     - LocalHamiltonianApplication: [ 156.,  5,
23.00%]
```

Merge development information with profiling

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

Dictionaries

Structured I/O

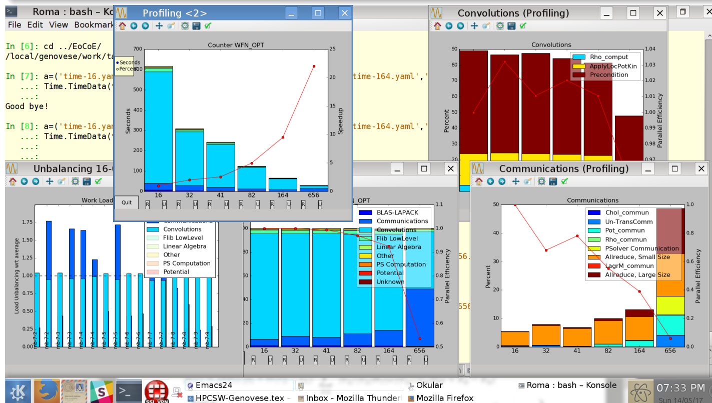
YAML

Code Profiling

Outlook

Typical examples when processing *crunched* data in ML

- Understanding code behaviour on a new architecture
- Identify optimization strategies *for the end-user* (modification of the input file)
- Small demo, notebook-based



Wrap up: what tools like that enable

- Treat the code from scripting and postprocess the logfile

```
from BigDFT import Calculators as C
single_point=C.SystemCalculator()
inp = {}
inp['dft'] = { 'ixc': 'LDA' }
inp['output'] = { 'orbitals': 'binary' }
log=single_point.run(input=inp,
                     posinp='molecule.xyz')
print log['energy']
```

- Create databases at runtime to store appropriate informations
- Analyze and profile the code behaviour and performance
- Restructure the code, enhance modularity and portability

FUTILE

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation

Code restructuring

Dictionaries

Structured I/O

YAML

Code Profiling

Outlook

Utilities library for Fortran codes

- Input/output handling (YAML)
- Code profiling
- Memory handling and profiling
- Analysis tools
- And much more: precisions, profiled wrappers for MPI, FFT, linear algebra calls
- Dictionaries

- https://gitlab.com/l_sim/futile/
- (ongoing) Documentation: https://bigdft-suite.readthedocs.io/projects/futile/

Outlook and Messages

Reinventing the wheel

BigDFT experience

The FUTILE Project

Memory Allocation
Code restructuring
Dictionaries
Structured I/O
YAML
Code Profiling

Outlook

We have tried to identify and modularize some low-level operations for fortran legacy codes (similar spirit of `glib` library for C). Objectives:

- Benefit from existing standards
- Make the code usage interoperable
- Enable new complicated approaches (robots help humans, and not viceversa)
- Introduce a different way of thinking when writing routines

The FUTILE library is one attempt to *modernize* the approach from the low-level. An idea to start “think” differently.

Main message: intentional (HPC) programming easier

Things that can be done are less dependent on the programming language adopted