ALPSCore tutorial

Alexander Gaenko

Simons Foundation Summer School Workshop
January 27, 2017

Overview

This tutorial contains:

- General overview of ALPSCore.
- Exercises:
  - Installation of ALPSCore.
  - Use of ALPSCore to gradually build a feature-rich simulation code.
- The hand-out materials contain:
  1. Useful links.
  2. Hands-on exercises.
  3. Diff files highlighting changes as we put in new features.
  4. Diff files that define exercises solutions.

The solutions to the exercises are available online:
`https://git.io/alpstut2_solutions`

INTRODUCTION

What it is about

ALPSCore originated from Algorithms and Libraries for Physics Simulations (ALPS) `http://alps.comp-phys.org`

The grand idea: Make the library code from ALPS available with shorter development cycle and decent documentation.

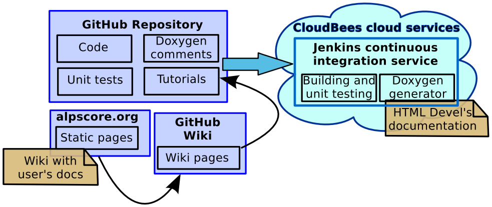
- What is the library code?
  - That will most probably useful for many applications.
  - However, ALPSCore is oriented towards Monte Carlo.
- What is the development cycle?
  - Introducing features (by request?)
  - Testing the features
  - Fixing bugs
  - Documenting
- What is decent documentation?
  - User's documentations
  - Tutorials (like this one!)
  - Developer's Doxygen-generated reference.

Web sites

Contributors:

- Emanuel Gull's group, University of Michigan (USA);
- Lukas Gamper, ETH Zurich (Switzerland);
- ... and many other ALPS contributors.

- Source code: `https://github.com/ALPSCore/ALPSCore`
- Documentation & tutorials: `http://alpscore.org`
- CloudBees for Continuous Delivery.



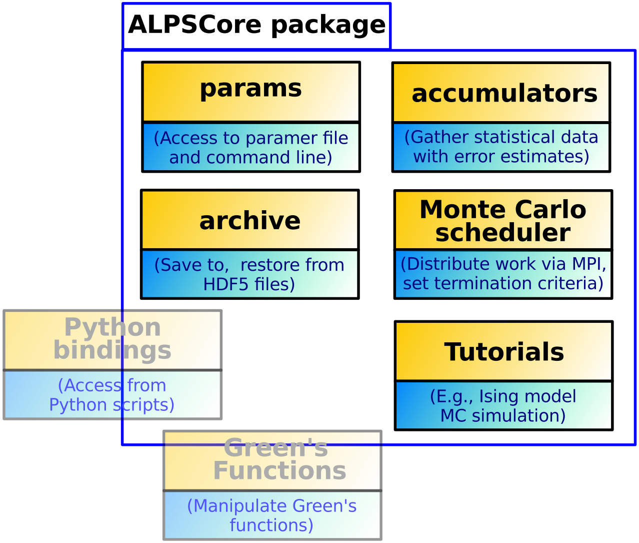
Why to use ALPSCore?

Think of a typical MC simulation to-do:

1. Read the simulation parameters.
2. Decide the step in the phase space.
3. Calculate (”measure”) the values of interest.
4. Collect statistics properly, taking into account autocorrelation.
5. Compute derived values with proper error propagation.
6. Save intermediate results regularly.
7. Set the step or time limit on the simulation.
8. Parallelize the whole thing.

Except the highlighted items, everything else is ”boilerplate”.
Sometimes non-trivial one!
Use ALPSCore to minimize boilerplate programming and concentrate on relevant science.

Overview


ALPSCore package
params
(Access to paramer file and command line)
accumulators
(Gather statistical data with error estimates)
archive
(Save to, restore from HDF5 files)
Monte Carlo scheduler
(Distribute work via MPI, set termination criteria)
Tutorials
(E.g., Ising model MC simulation)
Python bindings
(Access from Python scripts)
Green's Functions
(Manipulate Green's functions)

INSTALLING ALPSCore

Tutorial prerequisites

What do you need to *know?*

1. Basic Unix command-line operations
2. Basic C++
3. Optional: git
4. Optional: installing packages for your OS
5. Optional: CMake

What do you need to *have?*

1. Git
2. CMake
3. C++ compiler
4. HDF5 package
5. Boost package
6. MPI package

Obtainig ALPSCore

How to get ALPSCore?

- Downloading release:

`https://github.com/ALPSCore/ALPSCore/releases`

- Getting from GitHub:

```
git clone https://github.com/ALPSCore/ALPSCore.git
```

- Installing an ALPSCore package:
  - Via `macports` or `homebrew` on MacOS X.
  - Via `portage` on Gentoo Linux.
  - Debian or RedHat binary packages may be created if demand arises.

Building and compilation

There are (of course!) prerequisites.

- Tools:
  - C++03 (not `C++1x` — by policy).
    Tested with GCC 4.2+, Intel 10.0+, Clang 3.2+
  - CMake 2.8.12+
- Libraries:
  - HDF5 1.8+
  - Boost 1.54.0+
  - MPI (technically, optional)
  - GoogleTest (included!)

Building and compilation

- ALPSCore will try to find Boost, HDF5 and MPI.
- Usually, if you can run `mpicc`, ALPSCore will also find MPI.
- Building using CMake command line:

```
$ cmake -DBOOST_ROOT=/path/to/boost \
        -DHDF5_ROOT=/path/to/hdf5 \
        -DCMAKE_INSTALL_PREFIX=/usr/local/ALPSCore \
        /path/to/alpscore/sources
$ make
$ make install
```

- There are some other CMake variables, less frequently used

Exercise 1

Exercise 1: Download and install this tutorial

Open a terminal. Then, enter the following commands.

```
$ cd ~
$ mkdir alpstut
$ cd alpstut
$ git clone https://github.com/ALPSCore/Tutorial2.git
$ cd Tutorial2
$ tutorial=$PWD
$ ls -l
```

You should see a list of files and no error messages.

Exercise 2

Exercise 2: Download/install prerequisites

Ubuntu Linux:

```
$ sudo apt-get install cmake
$ sudo apt-get install libhdf5-dev
$ sudo apt-get install libboost-all-dev
$ sudo apt-get install mpi-default-dev
```

Mac OS X, port system:

```
$ sudo port install alpscore
```

Mac OS X, HomeBrew system:

```
$ brew update
$ brew tap homebrew/science
$ brew install alpscore
```

This will install the latest ALPSCore release (we don't need it!) and prerequisites.

More about Homebrew

```
1 $ xcode-select --install
2 $ /usr/bin/ruby -e \
3   "$(curl -fsSL \
4      https://raw.githubusercontent.com/\
5      Homebrew/install/master/install)"
6 $ brew update
7 $ brew tap homebrew/science
8 $ brew install alpscore
9 $ brew uninstall alpscore
```

This code downloads HomeBrew (lines 1–6), declares that we are interested in "science" section (line 7), installs `ALPSCore` with prerequisites (line 8), then uninstalls `ALPSCore` (but not the prerequisites) (line 9).

Exercise 2

Exercise 2 (cont): Download/install prerequisites

Test that you indeed have them:

```
$ cmake --version
$ g++ --version
$ h5cc --version
$ mpicxx --version
```

Cross-check:

- CMake 2.8.12+
- HDF5 1.8+
- MPI available

Exercise 3

Exercise 3: Download and install ALPSCore.

```
1  $ git clone https://github.com/ALPSCore/ALPSCore
2  $ cd ALPSCore
3  $ mkdir build
4  $ cd build
5  $ export ALPSCore_DIR=$PWD/install
6  $ cmake -DCMAKE_INSTALL_PREFIX=$ALPSCore_DIR ..
7  $ make
8  $ make test
9  $ make install
```

1: get ALPSCore from GitHub repository.

2–4: create a directory for the build

5: denote where it will be installed

6: generate the build

7–9: do the build, run the tests, and install

USING ALPSCore

Using ALPSCore in your program

Note: ALPSCore_DIR is pointing to the installation directory.

- How CMakeLists.txt should look to use ALPSCore: https://git.io/alpstut2_s1_cmake
- Catch: compilers!
- In-source builds are messy

```
$ export CXX=$(which needed_cpp)
$ export CC=$(which needed_cc)
$ mkdir 000build
$ cd 000build
$ cmake ..
```

Using ALPSCore in your program

A few tips and possible catches:

- Most of times, it is enough to remake after any file changes:
  ```
  $ make
  ```
- To speed up on 4 cores:
  ```
  $ make -j4
  ```
- Catch: if CMake files change:
  ```
  $ cmake .
  ```
  in the build directory
- Catch: if compilers change:
  ```
  $ rm -rf CMake*
  ```
- Catch: after accidental in-source build,
  remove the generated files.
- Catch: if `ALSPCore` itself is updated, regenerate:
  ```
  $ cmake .
  ```
  in your build directory

Exercise 4

Exercise 4: Build and run a dummy program
that uses ALPSCore and does nothing.
The code is at `$tutorial/step1_trivial`.
CMake file online: `https://git.io/alpstut2_s1_cmake`
Source file online: `https://git.io/alpstut2_s1_main`

Exercise 4

Exercise 4: Build and run a dummy program
that uses ALPSCore and does nothing.
The code is at $tutorial/step1_trivial.
CMake file online: https://git.io/alpstut2_s1_cmake
Source file online: https://git.io/alpstut2_s1_main

```
$ cd $tutorial/step1_trivial
$ mkdir 000build
$ cd 000build
$ cmake ..
$ make
$ ./alpsdemo
```

Parameters

- `alps::params` class is responsible for parameter parsing.
- `boost::program_options` is the engine.
- See Doxygen documentation (link from `http://alpscore.org/`) for detailed info.

Features:

- One can use input file, override with command line.
- Input file may contain sections: `[title]`
- A parameter must be defined to make it known.
- Unknown parameters are silently ignored.
- Auto-generated help message.
- Accessing an undefined parameter throws an exception.
- You can assign to parameters, which makes them defined.
- Potential information loss $\Rightarrow$ exception.

Exercise 5

Exercise 5: Build and run a program that uses parameters.
The code is at `$tutorial/step2_params`.
Online: `https://git.io/alpstut2_s2`

- Play with the different values of parameters.
- Try to override them from the command line.
- Change the program to make "`--loud`" parameter an integer, with 0 meaning "be quiet".

Exercise 5

Exercise 5: Build and run a program that uses parameters.
The code is at `$tutorial/step2_params`.
Online: `https://git.io/alpstut2_s2`

- Play with the different values of parameters.
- Try to override them from the command line.
- Change the program to make "`--loud`" parameter an integer, with 0 meaning "be quiet".

```
$ cd $tutorial/step2_params
$ mkdir 000build
$ cd 000build
$ cmake ..
$ make
$ ./alpsdemo
$ ./alpsdemo --help
$ ./alpsdemo ../params.ini
$ ./alpsdemo ../params.ini --count=3
....
```

Simple simulation class (doing nothing)

- Simulation that just says "I am running."
- Derived from `alps::mc_base` class.
- Must define virtual methods:
  - `void measure()`
  - `void update()`
  - `double fraction_completed()`
- Should define static method:
  - `static parameters_type& define_parameters(parameters_type&)`
- Passes to the base class constructor:
  - parameters object
  - a PRNG seed offset (will be needed for parallel simulations)

Exercise 6

Exercise 6: Build and run a trivial MC program.
The code is at `$tutorial/step3_trivial_mc`.
Online: `https://git.io/alpstut2_s3`
Note: the simulation code is split into 2 files.

- Build and run.
- Run with small counts.
- Run with large count and small timelimit; time the execution:
  `$ time -p your_command`
- Set large time limit and interrupt the program (via `Ctrl-C`).
- Change `fraction_completed()` so that `--count=0` would mean "till timeout".
- Change the name of the `update()` method and see it does not compile any more.

Exercise 6

Exercise 6: Build and run a trivial MC program.
The code is at `$tutorial/step3_trivial_mc`.
Online: `https://git.io/alpstut2_s3`
Note: the simulation code is split into 2 files.

- Build and run.
- Run with small counts.
- Run with large count and small timelimit; time the execution:
  `$ time -p` *`your_command`*

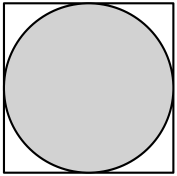
```
$ time -p ./alpsdemo --count=10000000 --timelimit=1
```

- Set large time limit and interrupt the program (via `Ctrl-C`).
- Change `fraction_completed()` so that `--count=0` would mean "till timeout".
- Change the name of the `update()` method and see it does not compile any more.

Compute π by Markov chain MC

The problem:

- Integral of objective function over an area.
- Trivial Metropolis step to stay inside the area.
- Area: unit square; any step outside is rejected.
- Objective function:
  1 if inside an inscribed circle, 0 otherwise.
- Expected result: π (if multiplied by 4).



We need to use:

- **Accumulators**: named observable to gather statistics
- **Results**: named as accumulators, allow arithmetic operations (with error propagation!)
- Accumulators & Results can hold a vector (e.g., for vector-valued or parametrized objective function)

Types of accumulators

Types of accumulators:

1. Mean only (cheapest, least useful):
   `MeanAccumulator<double>`
2. No binning (cheap, no autocorrelation info):
   `NoBinningAccumulator<double>`
3. Full binning (most expensive, autocorrelation, error propagation):
   `FullBinningAccumulator<double>`
4. Log binning (less memory demanding, no error propagation):
   `LogBinningAccumulator<double>`

If a method is not available for the given accumulator type, it throws!

Exercise 7

Exercise 7: Compute π by Markov chain MC.
The code is at `$tutorial/step4_pi`.
Online: `https://git.io/alpstut2_s4`

- Build and run the program.
- Run with various time limits.
- Run with different step sizes, compare autocorrelation lengths.
- Replace `FullBinningAccumulator` to `NoBinningAccumulator`
- Run with very low or high `--step`
  and see the underestimated error bars.

Checkpointing the simulation

- Checkpoint: save intermediate results, load to resume
- ALPSCore utilizes HDF5 format
  - Cross-platform
  - Hierarchical structure: Groups ($\sim$directories), Data ($\sim$files)
- ALPSCore can save/load:
  - Basic types (`int`, `double` *etc.*)
  - Vectors of basic types and of vectors of basic types *etc.*
  - Accumulators and parameters
  - Any user-defined class
    - Via `save()`/`load()` class members
    - Via traits (harder to do — more complex code)
- Parameters can be constructed from HDF5 file too
  - Should not try to define them again in this case!
    Use `par.is_restored()`.

Exercise 8

Exercise 8: Running and resuming.

The code is in `$tutorial/step5_pi_checkpoint`.

Online: `https://git.io/alpstut2_s5`

- Build and run the code. There is an error: find and fix it!
- Build, run the corrected code (note more options available!).

```
$ ./alpsdemo --help
$ # Run for 5 sec
$ ./alpsdemo --step 1 --timelimit 5
```

- Note new files appear:
  - "`*.out`" file contains simulation results
  - "`*.clone.h5`" file contains checkpoint
- Restore the checkpoint:

```
$ # Run for 10 more sec:
$ ./alpsdemo alpsdemo.clone.h5 --timelimit 10
```

- Note:
  - compulsory `--step` is read from the checkpoint
  - parameters can be overridden (like `--timelimit`)

How to use MPI?

- Not many changes compared to the sequential version.
  1. Use `alps::mcmpiadapter`<*SequentialSimulationClass*> as your simulation class.
  2. Initialize MPI environment
  3. Make sure that the parallel processes do not conflict for input/output
  4. Use special constructor for parameters
- Note that the completion is checked only at certain intervals (1 sec minimum)
- Look at the code changes in your handouts.

Exercise 9

Exercise 9: Parallel runs.

The code is in `$tutorial/step6_pi_mpi`.
Online: `https://git.io/alpstut2_s6`

- Build the MPI-parallelized program.
- Do timed runs with different number of processes.
- Observe checkpoint names.
- Try to restore from checkpoints, see how statistics builds up.

"Real-world" application: 2D Ising simulation

- The same principle as any other MC simulation:
  - Constructor: generated random spin population.
  - Update step: try to flip a random spin; compute energy change.
  - Measurements: energy, magnetization, magnetization squared.
- Performs arithmetics on results.
- For the sake of simplicity and clarity,
  a few optimization opportunities missed.
- The program uses a user-defined datatype,
  therefore needs loading/saving for it.

Exercise 10

Exercise 10: Parallelize the 2D Ising code.

The code is in `$tutorial/step7_ising`.
Online: `https://git.io/alpstut2_s7`

Steps:

1. Initialize MPI environment.
2. Use `alps::mcmpiadapter` template.
3. Use the parallel parameter constructor.
4. Make sure each rank has its own checkpoint file.
5. Make sure only the master process outputs the results.