ALPS チュートリアル – アプリケーションの ALPS 化

CMSI 神戸ハンズオン

ALPS Collaboration
http://alps.comp-phys.org/

ALPSize 全体の流れ

- 01-04 C++でのプログラミング
- 05-07 ALPS ライブラリ単体の利用
- 08 ALPS スケジューラ

ALPS スケジューラで単純な並列化と中断・再開

CMake

- **クロスプラットフォームなビルドツール**
  Windows **にも対応**
- Makefile **を生成**
- **高速な並列ビルド**
- **簡単なテスト**

alpsize の準備

```
$ mkdir -p alpsize/XX
$ cd alpsize/XX
```

ソースディレクトリとビルド・テストディレクトリは分けましょう

00 CMake 化 (1)

簡単なプログラムを CMake を利用してビルドする

```
$ cmake /opt/nano/alps/alpsize/00_cmake/
$ make
$ ctest
$ ./hello
```

00 CMake 化 (2)

```
$ cat /opt/nano/alps/alpsize/00_CMake/CMakeLists.txt
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
project(hello NONE)

# find ALPS Library
find_package(ALPS REQUIRED PATHS $ALPS_ROOT_DIR $ENVALPS_HOME NO_SYSTEM_ENVIRONMENT_PATH)
message(STATUS "Found ALPS: $ALPS_ROOT_DIR (revision: $ALPS_VERSION)")
include($ALPS_USE_FILE)

# enable C and C++ compilers
enable_language(CXX)

# rule for generating 'hello world' program
add_executable(hello hello.C)
add_alps_test(hello)
```

00 CMake 化 (2)

```
$ cat /opt/nano/alps/alpsize/00_CMake/CMakeLists.txt
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
project(hello NONE)

# find ALPS Library
find_package(ALPS REQUIRED PATHS $ALPS_ROOT_DIR $ENVALPS_HOME NO_SYSTEM_ENVIRONMENT_PATH)
message(STATUS "Found ALPS: $ALPS_ROOT_DIR (revision: $ALPS_VERSION)")
include($ALPS_USE_FILE)

# enable C and C++ compilers
enable_language(CXX)

# rule for generating 'hello world' program
add_executable(hello hello.C)
add_alps_test(hello)
```

00 CMake 化 (2)

```
$ cat /opt/nano/alps/alpsize/00_CMake/CMakeLists.txt
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
project(hello NONE)

# find ALPS Library
find_package(ALPS REQUIRED PATHS $ALPS_ROOT_DIR $ENVALPS_HOME NO_SYSTEM_ENVIRONMENT_PATH)
message(STATUS "Found ALPS: $ALPS_ROOT_DIR (revision: $ALPS_VERSION)")
include($ALPS_USE_FILE)

# enable C and C++ compilers
enable_language(CXX)

# rule for generating 'hello world' program
add_executable(hello hello.C)
add_alps_test(hello)
```

00 CMake 化 (2)

```
$ cat /opt/nano/alps/alpsize/00_CMake/CMakeLists.txt
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
project(hello NONE)

# find ALPS Library
find_package(ALPS REQUIRED PATHS $ALPS_ROOT_DIR $ENVALPS_HOME NO_SYSTEM_ENVIRONMENT_PATH)
message(STATUS "Found ALPS: $ALPS_ROOT_DIR (revision: $ALPS_VERSION)")
include($ALPS_USE_FILE)

# enable C and C++ compilers
enable_language(CXX)

# rule for generating 'hello world' program
add_executable(hello hello.C)
add_alps_test(hello)
```

00 CMake 化 (3)

add_alps_test は以下と同等

```
$ ./hello /opt/nano/alps/alpsize/00_CMake/hello.ip > hello.tmp
$ diff -u hello.tmp /opt/nano/alps/alpsize/00_CMake/hello.op
$ rm hello.tmp
```

実行バイナリを期待される出力と比較するテスト

```
$ cat /opt/nano/alps/alpsize/00_CMake/hello.op
hello, world
```

01 C プログラム

Wolff アルゴリズムのプログラム

```
$ cmake /opt/nano/alps/alpsize/01_original-c/
$ make
$ ctest
$ ./wolff
```

02 C++プログラム

Wolff アルゴリズムを C++で

```
$ cmake /opt/nano/alps/alpsize/02_basic-cpp/
$ make
$ ctest
$ ./wolff
```

03 STL の利用

STL を利用する

```
$ cmake /opt/nano/alps/alpsize/03_stl/
$ make
$ ctest
$ ./wolff
```

04 Boost の利用 (1)

Boost ライブラリを利用する

```
$ cmake /opt/nano/alps/alpsize/04_boost/
$ make
$ ctest
$ ./wolff
```

04 Boost の利用 (2)

```
...
#include <boost/random.hpp>
...
// random number generator
boost::mt19937 eng(SEED);
boost::variate_generator<boost::mt19937&, boost::uniform_real<> >
random_01(eng, boost::uniform_real<>());
```

05 ALPS Parameters

ALPS Parameter を利用する

```
$ cmake /opt/nano/alps/alpsize/05_parameters/
$ make
$ ctest
$ ./wolff
```

Parameter ライブラリ

プログラムに与えるパラメータを解釈するライブラリ

```
LATTICE = "chain lattice";
L = 16,

SEED = 2873
// C++ style comment
SWEEPS = 4096;
THERMALIZATION = SWEEPS/8;
/* C style comment */
{ T = 2; Sq = 2*PI/3; }
{ T = 1.8; }
```

- 四則演算可能
- π を文字で指定
- 空行も正しく認識
- C 風, C++風のコメント
- { } で囲んだ変数は異なるセット
- 空白を適切に認識
- 改行, セミコロン, コンマで変数を区別

Parameter ライブラリ

プログラムに与えるパラメータを解釈するライブラリ

```
LATTICE = "chain lattice";
L = 16,

SEED = 2873
// C++ style comment
SWEEPS = 4096;
THERMALIZATION = SWEEPS/8;
/* C style comment */
{ T = 2; Sq = 2*PI/3; }
{ T = 1.8; }
```

- 四則演算可能
- π を文字で指定
- 空行も正しく認識
- C 風, C++風のコメント
- { } で囲んだ変数は異なるセット
- 空白を適切に認識
- 改行, セミコロン, コンマで変数を区別

Parameter ライブラリの利用法

```
#include <boost/foreach.hpp>
#include <alps/parameter.h>
...
int main(int argc, char** argv) {
...
  alps::ParameterList plist = read_param(argc, argv);
  BOOST_FOREACH(alps::Parameters& p, plist) {
    double a = p["a"];
    double b = p.value_or_default("b", 0.5);
...
  }
...
}
```

Parameter ライブラリの利用法

```
#include <boost/foreach.hpp>
#include <alps/parameter.h>
...
int main(int argc, char** argv) {
...
  alps::ParameterList plist = read_param(argc, argv);
  BOOST_FOREACH(alps::Parameters& p, plist) {
    double a = p["a"];
    double b = p.value_or_default("b", 0.5);
...
  }
...
}
```

06 ALPS Alea

ALPS Alea を利用する

```
$ cmake /opt/nano/alps/alpsize/06_alea/
$ make
$ ctest
$ ./wolff
```

Alea ライブラリの利用法

マルコフ連鎖における平均値, 分散, 自己相関を計算するライブラリ

```
#include <alps/alea.h>
...
alps::ObservableSet obs;
...
obs << alps::RealObservable("Energy");
...
for (int i = 0; i != N; ++i) {
  double e = energy(); // calclulate something
  obs.reset(i == THERM); // when thermalized
  obs["Energy"] << e;
}
...
std::cout << obs["Energy"];
```

Tips: 出力を制御する

```
// Gnuplot 風出力
std::cout << 1./param["beta"] << " "
          << obs["Energy"].mean() << " "
          << obs["Energy"].variance() << std::endl;
```

07 ALPS Lattice

ALPS Lattice を利用する

```
$ cmake /opt/nano/alps/alpsize/07_lattice/
$ make
$ ctest
$ ./wolff
```

Lattice ライブラリの利用法

```
#include <alps/lattice.h>
...
class ising : public alps::graph_helper<> {
 public:
  ising(alps::Parameters const& p)
    : graph_helper<>(p) {}
...
};
```

Lattice の定義

```
<LATTICE name="square lattice" dimension="2">
  <PARAMETER name="a" default="1"/>
  <BASIS><VECTOR>a 0</VECTOR><VECTOR>0 a</VECTOR></BASIS>
  <RECIPROCALBASIS><VECTOR>2*pi/a 0</VECTOR><VECTOR>0 2*pi/a</VECTOR></RECIPROCALBASIS>
</LATTICE>
<UNITCELL name="simple2d" dimension="2">
  <VERTEX/>
  <EDGE><SOURCE vertex="1" offset="0 0"/><TARGET vertex="1" offset="0 1"/></EDGE>
  <EDGE><SOURCE vertex="1" offset="0 0"/><TARGET vertex="1" offset="1 0"/></EDGE>
</UNITCELL>
<LATTICEGRAPH name = "square lattice">
  <FINITELATTICE>
    <LATTICE ref="square lattice"/>
    <PARAMETER name="W" default="L"/>
    <EXTENT dimension="1" size="L"/>
    <EXTENT dimension="2" size="W"/>
    <BOUNDARY type="periodic"/>
  </FINITELATTICE>
  <UNITCELL ref="simple2d"/>
</LATTICEGRAPH>
```

Lattice ライブラリの便利な関数

```
std::vector<double> d(num_sites());
std::vector<double> J(num_bonds());
...
double energy = 0.;
BOOST_FOREACH(site_descriptor s, sites()) {
  BOOST_FOREACH(bond_descriptor b, neighbor_bonds(s)) {
    energy += J[b] * d[source(b)] * d[target(b)];
  }
}
...
boost::mt19937 rng;
boost::uniform_int<> dst(0, num_bonds()-1);
J[bond(dst(rng))] = 0;
Jtemp = param["J" + bond_type(b)];
```

- sites(), bonds() は BOOST_FOREACH と組み合わせる
- index(s) や index(b) で数字に変換
- bond(n) や site(n) で *_descriptor に変換

08 ALPS Scheduler (1)

ALPS Scheduler を利用する

```
$ cmake /opt/nano/alps/alpsize/08_scheduler/
$ make
$ ctest
$ ./hello
$ ./wolff
```

08 ALPS Scheduler (2)

```
class wolff_worker : public alps::parapack::lattice_mc_worker<> {
private:
  typedef alps::parapack::lattice_mc_worker<> super_type;

public:
  wolff_worker(alps::Parameters const& params) : super_type(params);
  virtual ~wolff_worker();

  void init_observables(alps::Parameters const&, alps::ObservableSet& obs);
  bool is_thermalized() const;
  double progress() const;

  void run(alps::ObservableSet& obs);
  void save(alps::ODump& dp) const;
  void load(alps::IDump& dp);
};
```

08 ALPS Scheduler (3)

```
class wolff_evaluator : public alps::parapack::simple_evaluator {
public:
  wolff_evaluator(alps::Parameters const&);
  void evaluate(alps::ObservableSet& obs) const;
};
```

08 ALPS Scheduler (4)

```
#include <alps/parapack/parapack.h>

int main(int argc, char** argv) {
  return alps::parapack::start(argc, argv);
}
```

ALPS Fortran

ALPS Scheduler を Fortran プログラムに