Fortran+Pythonによるアプリケーション開発

中尾 昌広（理化学研究所 計算科学研究センター）

もくじ

- Pythonの概要
- FortranとPythonとの連携方法
- FortranとPython MPIプログラムとの連携方法
- Fortranと並列言語XcalableMPとの連携方法

Pythonの特徴

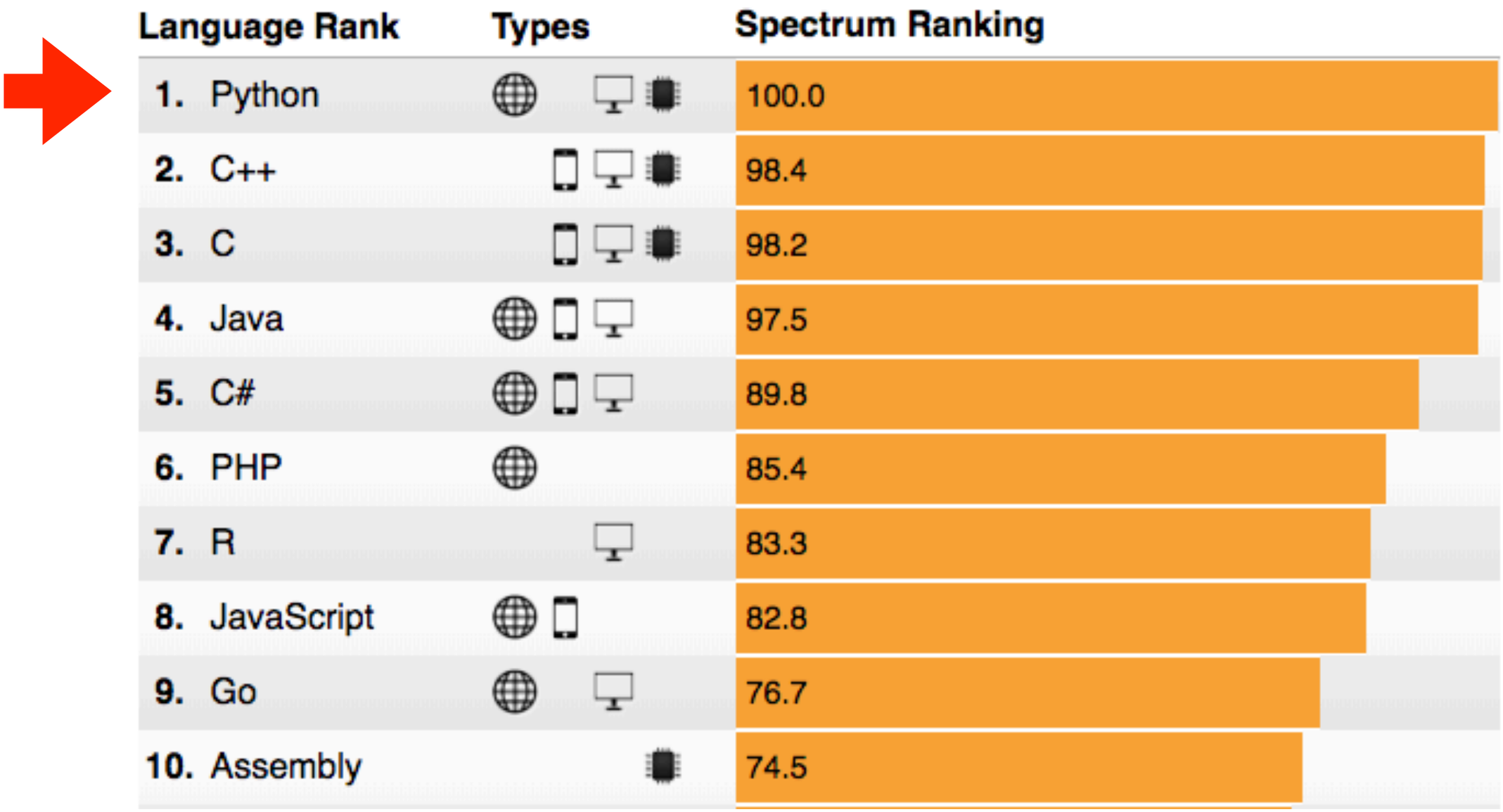
- 文法がシンプル
  - 学習コストが低い
  - 誰が書いても同じようなコードになるため，可読性が高い

- 複数のプログラミングパラダイム
  - オブジェクト指向，手続き型，関数型
  - 動的型付け，ガベージコレクション

- 多種多様なライブラリが提供されている
  - 標準ライブラリとサードパーティ製のライブラリがある
  - 人工知能，数値計算，組み込み，Web構築，など
  - CやCUDAなどで記述されたライブラリは高速

Pythonとは

- 最も人気のあるプログラミング言語（2018年IEEE spectrum調べ）



Language Rank	Types	Spectrum Ranking
1. Python		100.0
2. C++		98.4
3. C		98.2
4. Java		97.5
5. C#		89.8
6. PHP		85.4
7. R		83.3
8. JavaScript		82.8
9. Go		76.7
10. Assembly		74.5

https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018

Pythonのプログラミング例

```
print("Hello World!")
```

```
if x == 10:
    print ("AAA")
    print ("BBB")
else
    print ("CCC")
    print ("DDD")
```

同じ数の空白でインデントされた文がブロックとみなされる

```
for x in range(3):
    print(x)
```

0, 1, 2を出力

```
names = ["Taro", "Jiro", "Saburo"]
for name in names:
    print(name)
```

リストを宣言し，それをfor文に指定できる

Pythonの中の高速化関連機能

- NumPy, SciPy
  - 数値計算のためのライブラリ集
  - FortranやCに引数渡し可能な配列・構造体
  - BLAS，LAPACKとの連携
- mpi4py
  - 並列Pythonプログラムを作成できる（mpiexecで実行）
  - PythonからMPI関数を呼び出すためのラッパー
- Cython
  - PythonコードをCのコードに変換するトランスレータ
  - 変換後のコードは，Cコンパイラが機械語に翻訳する

Numpyを使ったPythonプログラム例

- 配列クラス（numpy.array）には配列や行列のためのメソッドが多数提供されている

```
import numpy as np

A = np.array([1, 2, 3])
B = np.array([4, 5, 6])

print (A + B)
print (A * B)
print (A * 2)
print (np.dot(A, B))

C = np.array([[1,2], [3,4]])
D = np.array([[4,3], [2,1]])
print (np.dot(C, D))
```

numpyというライブラリをnpという名前でインポートする

1次元配列AとBの定義

```
[5 7 9]
[4 10 18]
[2 4 6]
32 = (4+10+18)
```

要素毎の演算

dot積

2次元配列CとDの定義

```
[[ 8  5]
 [20 13]]
```

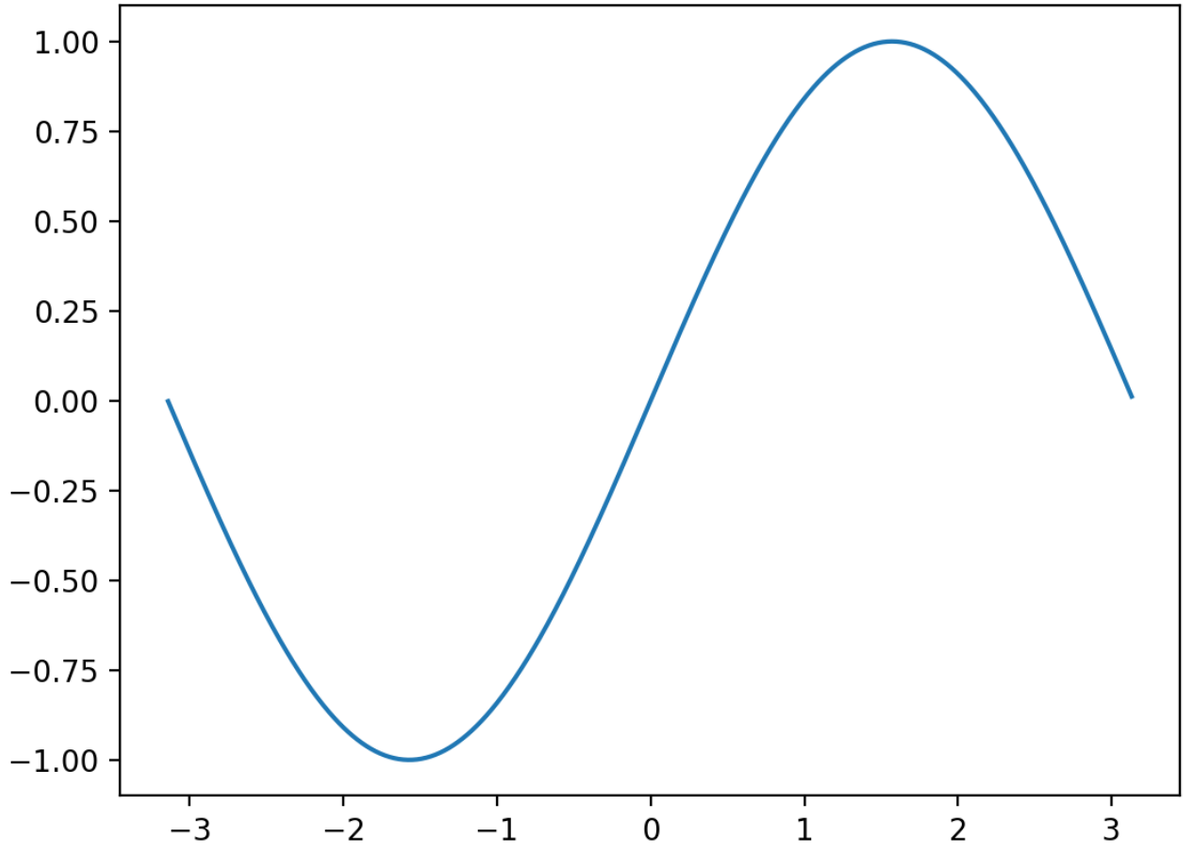
行列積

グラフ表示ライブラリ

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-3.14, 3.14, 0.01)
y = np.sin(x)
plt.plot(x, y)
```

matplotlibはPythonでグラフを描画するときなどに使われるライブラリ．アニメーションも可能

-3.14～3.14まで0.01刻みの等差数列を生成



もくじ

- Pythonの概要
- FortranとPythonとの連携方法
- FortranとPython MPIプログラムとの連携方法
- Fortranと並列言語XcalableMPとの連携方法

FortranとPythonの比較

- アプリケーションサイクル（多くの場合，並行して行われる）
  1. 開発
  2. 実行
  3. メンテナンス
     - 新機能の追加・バグ取り
     - 他マシンへのポーティング

総時間を小さくしたい

下表は，あくまで個人的な感想です。

	開発/メンテ時間	実行時間
Fortran	△	◎
Python	◎	△ ～ ○※

※ ○ は，高速なライブラリやCythonなどのツールを使った場合.
特定のアーキテクチャに最適化したFortranコードには勝てない.

Fortran+Pythonという選択肢

- Pythonには数値計算のライブラリが揃っているので,
  実行時間が短い数値計算ならPythonだけでOK

- 実行時間が長い数値計算の場合はFortranで書きたい
- Pythonのライブラリも使いたい（グラフ表示や統計処理など）

- 高速なFortranのコードをPythonから呼び出す

	開発/メンテ時間	実行時間
Fortran	△	◎
Python	◎	△～○
Fortran+Python	○	◎

Fortran+Pythonのための機能

- ctypes
  - Pythonの標準ライブラリ（Numpy中にもある）
  - Cと互換性のあるデータ型を提供し，動的リンク/共有ライブラリ内の関数呼び出しを可能にする
  - Fortran2003の機能により，Fortranのサブルーチンや関数はCの関数と同等に扱える
- f2py
  - FortranコードからPythonモジュールを作成するツール
  - Numpyに同梱されている

本発表では，標準ライブラリにあるctypesについて紹介する.

Pythonの処理系

非常に沢山あるが，その一部を紹介

- CPython：本発表ではこれを用います.
  - C言語で実装
  - 最も広く用いられている
- Jython
  - Javaで実装
  - Javaクラスライブラリを Pythonで利用できる
- IronPython
  - C#で実装
  - .NETの基本クラスライブラリが利用できる
- PyPy
  - Pythonで実装（RPythonで記述されたPythonの処理系）
  - JIT機能により，CPythonよりも高速

動作環境

- Debian GNU/Linux 9.4
- gfortran 6.3.0
- CPython 3.5.3
- OpenMPI 2.0.2

環境依存がかなり強いので，ご注意ください。

サンプルプログラムのいくつかは下記ページを参考に作成しています。

http://d.hatena.ne.jp/ignisan/20121017/p1

本スライドとサンプルプログラムはHPFPCのサイトで公開します。

型について

https://docs.python.jp/3/library/ctypes.html

ctypes の型	C の型	Python の型
c_bool	_Bool	bool (1)
c_char	char	1文字のバイト列オブジェクト
c_wchar	wchar_t	1文字の文字列
c_byte	char	int
c_ubyte	unsigned char	int
c_short	short	int
c_ushort	unsigned short	int
c_int	int	int
c_uint	unsigned int	int
c_long	long	int
c_ulong	unsigned long	int
c_longlong	__int64 または long long	int
c_ulonglong	unsigned __int64 または unsigned long long	int
c_size_t	size_t	int
c_ssize_t	ssize_t または Py_ssize_t	int
c_float	float	浮動小数点数
c_double	double	浮動小数点数
c_longdouble	long double	浮動小数点数
c_char_p	char * (NUL 終端)	バイト列オブジェクトまたは None
c_wchar_p	wchar_t * (NUL 終端)	文字列または None
c_void_p	void *	整数または None

例1：スカラを引数に渡す

fmath.f90

```
subroutine add(a,b) bind(C)
  implicit none
  integer(8),intent(in) :: a
  integer(8),intent(inout):: b
  b = a + b
end subroutine add

subroutine mult(a,b) bind(C)
  implicit none
  integer(8),intent(in) :: a
  integer(8),intent(inout):: b
  b = a * b
end subroutine mult
```

Pythonから呼び出したい
サブルーチンや関数の宣言の
最後に**bind(C)**を加える

calc.py

```
from ctypes import *

fmath = cdll.LoadLibrary("fmath.so")
fmath.add.argtypes  = [ POINTER(c_int64), POINTER(c_int64) ]
fmath.add.restype   = c_void_p
fmath.mult.argtypes = [ POINTER(c_int64), POINTER(c_int64) ]
fmath.mult.restype  = c_void_p

a = c_int64(2)
b = c_int64(3)
fmath.add(a, b)
print ("2 + 3 = ", b.value) # 5

a = c_int64(2)
b = c_int64(3)
fmath.mult(a, b)
print ("2 * 3 = ", b.value) # 6
```

本来はfmath.add(byref(a), byref(b))というように，byref()を
使ってポインタにしないといけないが，何故かなくても動作する

コンパイルと実行

```
% gfortran -shared -fPIC -o fmath.so fmath.f90
% export LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH
% python3 calc.py
2 + 3 = 5
2 * 3 = 6
%
```

共有ライブラリの生成

環境変数

Pythonの実行

例2：Numpyを使って配列を引数に渡す

fmath.f90

```
subroutine add_one(a,b,N) bind(C)
  implicit none
  integer(8),intent(in) :: N
  real(8),dimension(N),intent(in) :: a
  real(8),dimension(N),intent(inout):: b

  b(:) = a(:) + 1
end subroutine add_one
```

pythonの配列は0-originなので,
上のコードのdimension(N)を
dimension(0:N-1)としても良い.

calc.py

```
from ctypes import *
import numpy as np

fmath = np.ctypeslib.load_library("fmath.so",".")
fmath.add_one.argtypes = [
            np.ctypeslib.ndpointer(dtype=np.float64),
            np.ctypeslib.ndpointer(dtype=np.float64),
            POINTER(c_int64) ]
fmath.add_one.restype = c_void_p

inArray  = np.arange(0.,10.,dtype=np.float64)
outArray = np.empty_like(inArray)
size = byref(c_int64(inArray.size))
fmath.add_one(inArray, outArray, size)
print (inArray)  # 0,1,2,..,9
print (outArray) # 1,2,3,..,10
```

Numpyを使うため， numpyのload_library()を使って
共有ファイルライブラリをロードする.

コンパイルと実行

```
% gfortran -shared -fPIC -o fmath.so fmath.f90
% python3 calc.py
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
[ 1.   2.   3.   4.   5.   6.   7.   8.   9.  10.]
%
```

共有ライブラリへのパスはnp.ctypeslib.load_library()で指定しているため,
環境変数LD_LIBRARY_PATHの設定は不要

Pythonで作成した多次元配列や構造体も，Fortranに渡すことができる.
ただし，多次元配列は次元の順が逆になるので注意が必要.

例3：関数の例（配列の総和を返す）

fmath.f90

```
integer(8) function sum(a,N) bind(C)
  implicit none
  integer(8),intent(in) :: N
  integer(8),dimension(N),intent(in) :: a
  integer :: i

  sum = 0
  do i=1, N
     sum = sum + a(i)
  end do
end function sum
```

calc.py

```
from ctypes import *
import numpy as np

fmath = np.ctypeslib.load_library("fmath.so",".")
fmath.sum.argtypes = [
                np.ctypeslib.ndpointer(dtype=np.int64),
                POINTER(c_int64) ]
fmath.sum.restype = c_int64

inArray  = np.arange(0.,10.,dtype=np.int64)
size = byref(c_int64(inArray.size))
print (inArray)
print (fmath.sum(inArray, size)) # 45
```

コンパイルと実行

```
% gfortran -shared -fPIC -o fmath.so fmath.f90
% python3 calc.py
[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
45
%
```

例4：結果をplotする（ルンゲクッタ法）

fmath.f90

```
subroutine rungekutta(dt, nend, values) bind(C)
  implicit none
  real(8), intent(in) :: dt
  integer(8), intent(in)  :: nend
  double precision, intent(out) :: values(nend)
  double precision :: t, p1, p2, p3, p4, u
  integer(8) :: n

  u = 1d0
  do n = 1, nend
    t = dt * real(n)
    p1 = u
    p2 = u + 0.5d0 * dt * p1
    p3 = u + 0.5d0 * dt * p2
    p4 = u + dt * p3
    u = u + dt * 1/6d0 * (p1 + 2 * p2 + 2 * p3 + p4)
    values(n) = u
  end do
end subroutine rungekutta
```

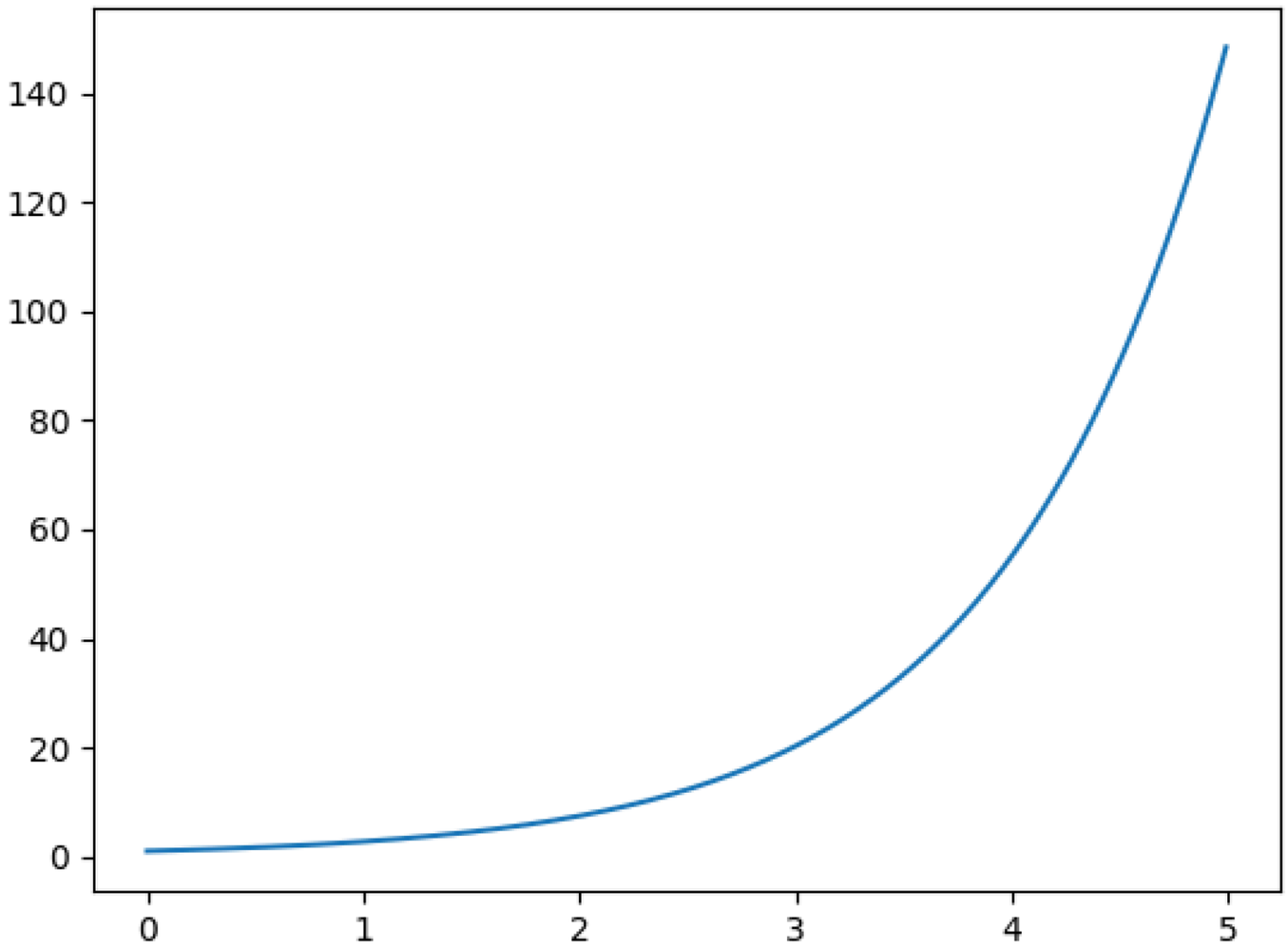
calc.py

```
from ctypes import *
import matplotlib.pyplot as plt
import numpy as np

fmath = np.ctypeslib.load_library("fmath.so",".")
fmath.rungekutta.argtypes = [
    POINTER(c_double), POINTER(c_int64),
    np.ctypeslib.ndpointer(dtype=np.float64)]
fmath.rungekutta.restype = c_void_p

n = 500
a   = np.zeros(n)
dt  = c_double(0.01)
len = c_int64(a.size)

fmath.rungekutta(dt, len, a)

plt.plot(np.arange(0,5,0.01), a)
plt.show()
```

下記ページを参考に作成
https://kyoto-geopython.github.io/Kyoto-GeoPython/html/応用編/Fortran,%20C言語%20との連携.html

コンパイルと実行（デモ）

```
% gfortran -shared -fPIC -o fmath.so fmath.f90
% python3 calc.py
```



例5：速度評価比較（scal）

fmath.f90

```
subroutine scal(x, alpha, N) bind(C)
  implicit none
  integer(8), intent(in) :: N
  real(8), dimension(N), intent(inout) :: x
  real(8), intent(in) :: alpha

  x(:) = x(:) * alpha
end subroutine scal
```

calc.py

```
from ctypes import *
import numpy as np
import time

fmath = np.ctypeslib.load_library("fmath.so",".")
fmath.scal.argtypes = [
    np.ctypeslib.ndpointer(dtype=np.float64),
    POINTER(c_double), POINTER(c_int64)]
fmath.scal.restype = c_void_p

n     = 10000000
x     = np.ones(n, dtype=np.float64)
alpha = c_double(0.01)
len = c_int64(x.size)

start_time = time.perf_counter()
fmath.scal(x, alpha, len)
elapsed_time = time.perf_counter() - start_time
print ("elapsed_time:{0}".format(elapsed_time))
```

例5：速度評価比較（scal）

Fortran+Python v.s. Fortranのみ v.s. Pythonのみ　の3種類で比較

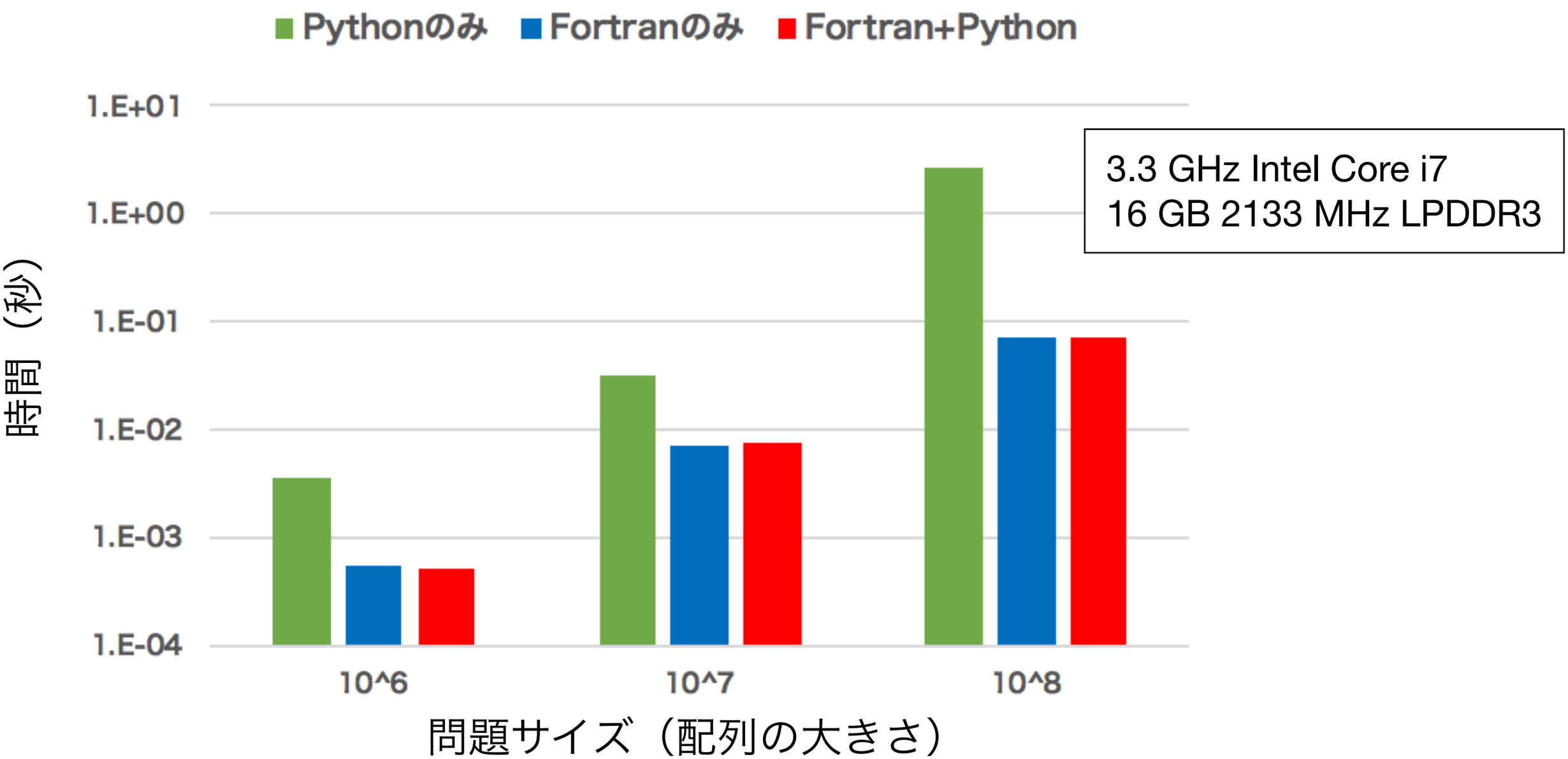
Fortranのみ

```
program main
  implicit none
  integer(8), parameter :: N = 10000000
  real(8), dimension(N) :: x
  real(8),parameter :: alpha = 0.01
  integer(8) :: t1, t2, t_rate, t_max

  x(:) = 1.0
  call system_clock(t1)
  call scal(x, alpha, N)
  call system_clock(t2, t_rate, t_max)

  write(*,*) (t2-t1)/dble(t_rate)

end program main
```

Pythonのみ

```
import numpy as np
import time

n = 10000000
x = np.ones(n)
start_time = time.perf_counter()
x = x * alpha
elapsed_time = time.perf_counter() - start_time
print ("elapsed_time:{0}".format(elapsed_time))
```

例5：速度評価比較（scal）：結果



Fortran+PythonとFortranの間は差はほぼない.

FortranとPythonは，問題サイズが最大のときは30倍以上の差.

もくじ

- Pythonの概要
- FortranとPythonとの連携方法
- FortranとPython MPIプログラムとの連携方法
- Fortranと並列言語XcalableMPとの連携方法

mpi4pyパッケージ

- https://github.com/mpi4py/mpi4py

```
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

print("Hello World at rank {0}/{1}".format(rank, size))
```

MPI_Init()とMPI_Finalize()は暗黙的に呼ばれる

```
% mpiexec -n 4 python3 ./hello.py | sort
Hello World at rank 0/4
Hello World at rank 1/4
Hello World at rank 2/4
Hello World at rank 3/4
```

mpi4pyパッケージ + numpy

- 参考：https://keichi.net/post/mpi4py/

```
from mpi4py import MPI
import numpy as np

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    buf = np.arange(100, dtype=np.float64)
    req = comm.Isend(buf, dest=1, tag=0)
elif rank == 1:
    buf = np.empty(100 ,dtype=np.float64)
    req = comm.Irecv(buf, source=0, tag=0)

req.wait()
```

FortranやCのMPIよりシンプルな気がする

MPI+PythonからSerialのFortranを呼ぶ

fmath.f90（変更なし）

```
subroutine scal(x, alpha, N) bind(C)
  implicit none
  integer(8), intent(in) :: N
  real(8), dimension(N), intent(inout) :: x
  real(8), intent(in) :: alpha

  x(:) = x(:) * alpha
end subroutine scal
```

例5の速度評価に用いたscalプログラム.

赤太字は変更点.

calc.py

```
from ctypes import *
import numpy as np
from mpi4py import MPI

fmath = np.ctypeslib.load_library("fmath.so",".")
fmath.scal.argtypes = [
    np.ctypeslib.ndpointer(dtype=np.float64),
    POINTER(c_double), POINTER(c_int64)]
fmath.scal.restype = c_void_p

comm = MPI.COMM_WORLD
size = comm.Get_size()
n    = int(10000000/size)
x    = np.ones(n, dtype=np.float64)
alpha = c_double(0.01)
len   =    c_int64(x.size)

fmath.scal(x, alpha, len)
```

※10000000は整数sizeで割り切れると仮定

Python+MPIとFortran+MPIとの連携1

fmath.f90

```
subroutine hello(comm) bind(C)
  implicit none
  include "mpif.h"
  integer(4) :: comm, size, rank, ierr

  call MPI_Comm_size(comm, size, ierr)
  call MPI_Comm_rank(comm, rank, ierr)
  print *, "Hello World at rank ", rank, "/", size
end subroutine hello
```

hello.py

```
from ctypes import *
import numpy as np
from mpi4py import MPI

fmath = np.ctypeslib.load_library("fmath.so",".")
fmath.hello.argtypes = [ POINTER(c_int32) ]
fmath.hello.restype = c_void_p

comm = MPI.COMM_WORLD
comm = comm.py2f()
fmath.hello(c_int32(comm))
```

PythonのMPIコミュニケータはpy2f()メソッドでFortranのMPIコミュニケータに変換する.

FortranのMPIコミュニケータはInteger型なのでc_int32に変換してからFortranのサブルーチンに渡す（このフローはおそらく実装依存）.

実行結果

```
% mpiexec -n 4 python3 ./hello.py | sort
Hello World at rank 0 / 4
Hello World at rank 1 / 4
Hello World at rank 2 / 4
Hello World at rank 3 / 4
```

Python+MPIとFortran+MPIとの連携2

fmath.f90

```
subroutine scal(x, alpha, N, comm) bind(C)
  implicit none
  integer(8), intent(in) :: N
  real(8), dimension(N), intent(inout) :: x
  real(8), intent(in) :: alpha
  real(8), dimension(N) :: y
  include "mpif.h"
  integer(4) :: comm, size, rank, ierr, len
  integer(4) :: start, end

  call MPI_Comm_size(comm, size, ierr)
  call MPI_Comm_rank(comm, rank, ierr)
  len = N/size
  start = rank * len + 1
  end = start + len - 1
  x(start:end) = x(start:end) * alpha
  call MPI_Allgather(x(start), len, MPI_REAL8,
  y, len, MPI_REAL8, comm, ierr)
  x(:) = y(:)
end subroutine scal
```

calc.py

```
from ctypes import *
import numpy as np
from mpi4py import MPI

fmath = np.ctypeslib.load_library("fmath.so",".")
fmath.scal.argtypes = [
    np.ctypeslib.ndpointer(dtype=np.float64),
    POINTER(c_double), POINTER(c_int64),
    POINTER(c_int32)]
fmath.scal.restype = c_void_p

comm = MPI.COMM_WORLD
comm  = comm.py2f()
n     = 10000000
x     = np.ones(n, dtype=np.float64)
alpha = c_double(0.01)
len = c_int64(x.size)

fmath.scal(x, alpha, len, c_int32(comm))
```

※10000000は整数sizeで割り切れると仮定

もくじ

- Pythonの概要
- FortranとPythonとの連携方法
- FortranとPython MPIプログラムとの連携方法
- Fortranと並列言語XcalableMPとの連携方法

XcalableMP（XMP）とは（1/2）

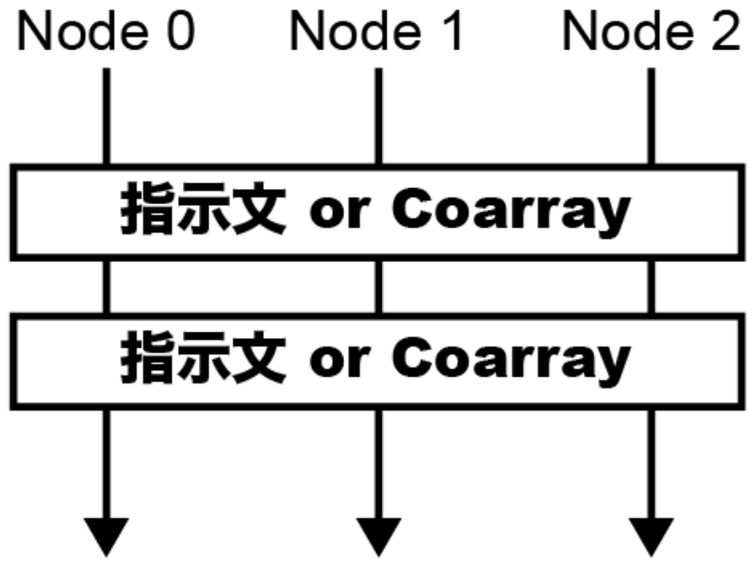
- High Performance Fortran (HPF)のように，指示文を用いて並列化
- MPIに代わる並列プログラミングモデルとしてPCクラスタコンソーシアム・XMP規格部会が仕様を策定している
- FortranとCに対して，指示文とCoarray記法を用いた並列拡張
  - 指示文なので，既存の逐次コードからの移行が容易
  - MPI/OpenMP/Pythonとの連携機能
    - 既存のコードの部分的なXMP化も可能

第1回 並列Fortranに
関するシンポジウムでも
発表しています。

http://xcalablemp.org

XcalableMPとは（2/2）

- SPMD (Single Program Multiple Data)
  - MPIと同じ
  - 各実行主体（XMPではノードと呼称, MPIのランクのようなもの）が同じコードを独立に重複して実行する
  - 通信は指示文やCoarray記法によって明示的に行われる
    - 通信箇所が明確なので，性能チューニングが容易（HPFの教訓）



- 2つのメモリモデル
  - 指示文によるグローバルビュー
  - Coarray記法によるローカルビュー

Python+MPIとXMPとの連携

fmath.f90

```
subroutine scal(x, N, alpha) bind(C)
!$xmp nodes p(*)
!$xmp template t(N)
!$xmp distribute t(block) onto p
  real(8), dimension(N), intent(inout) :: x
  real(8), intent(in) :: alpha
!$xmp align x(i) with t(i)

!$xmp array on t(:)
  x(:) = x(:) * alpha
end subroutine scal
```

calc.py

```
from mpi4py import MPI
from ctypes import *
import numpy as np
import xmp

lib = xmp.Lib("fmath.so")
comm = MPI.COMM_WORLD
size = comm.Get_size()

n     = 1000000
m     = int(n/size)
x     = np.ones(m, dtype=np.float64)
alpha = c_double(0.01)

job = lib.call(comm, "scal", (x, n, alpha))
if comm.Get_rank() == 0:
    print (job.elapsed_time())  // second
```

※10000000は整数sizeで割り切れると仮定

まとめ

- 目的
  - Fortranの高速性とPythonの高生産性の両方を使う（適材適所）
  - 処理の重たい箇所はFortranを用いて，処理の軽い箇所はPythonを用いる
- 発表内容
  - Pythonの概要
  - FortranとPythonとの連携方法
  - FortranとPython MPIプログラムとの連携方法
  - Fortranと並列言語XcalableMPとの連携方法