機器學習於材料資訊的應用
Machine Learning on Material Informatics

陳南佑(NAN-YOW CHEN)

nanyow@narlabs.org.tw

楊安正(AN-CHENG YANG)

acyang@narlabs.org.tw

善用C-Extensions &JIT來加速你的python code

常見加速手段

1. 使用multiprocessing來使用所有的 cores，要改寫程式，速度提升要看工作可平行部分占比。
2. 使用 NumPy、Pandas，或是 Scikit-Learn 庫，透過環境變數(OMP_NUM_THREADS，OPENBLAS_NUM_THREADS，MKL_NUM_THREADS，)讓使用這些件自動平行化，。
3. 在GPU環境中使用Rapids 來進一步提高處理速度。
4. 利用其他編譯器對python程式碼加速(小小小部分修改)

OPENMP

- OpenMP（Open Multi-Processing）是一套支援跨平台共享記憶體方式的多執行緒平行的編程API。
- 用於共享記憶體平行系統上的一種指導性注釋（Compiler Directive）多執行緒程式設計。OpenMP支援的程式語言包括C語言、C++和Fortran。
- 透過在原始碼中加入專用的pragma來指明自己的意圖，由此編譯器可以自動將程式進行平行化，並在必要之處加入同步以及交換資訊。
- 當選擇忽略這些pragma(編譯時未指定openmp)，或者編譯器不支援OpenMP時，程式又可退化為通常的程式（序列模式），程式碼仍然可以正常運作，只是不能利用多執行緒來加速程式執行。
- 使用方式。
  - C
  - Fortran
- 執行方式
  - export OMP_NUM_THREADS=4; python openmp.py
  - Add os.environ["OMP_NUM_THREADS"] = ‘8’, before import numpy

OPENMP在各種語言的實作

C

```c
#include <stdio.h>

#include <omp.h>


int main(int argc, char** argv){

    int thread_id;


    #pragma omp parallel

    {

        printf("Hello from process: %d\n",
omp_get_thread_num());

    }

    return 0;

}
```

FORTRAN

```fortran
PROGRAM Parallel_Hello_World

USE OMP_LIB


!$OMP PARALLEL


    PRINT *, "Hello from process: ",
OMP_GET_THREAD_NUM()


!$OMP END PARALLEL


END
```

PYTHON

```python
import os
#must set these before loading numpy:
os.environ["OMP_NUM_THREADS"] = '4' # export
OMP_NUM_THREADS=4
os.environ["OPENBLAS_NUM_THREADS"] = '4' #
export OPENBLAS_NUM_THREADS=4
os.environ["MKL_NUM_THREADS"] = '4' # export
MKL_NUM_THREADS=6
```

OPENMP results

OMP_NUM_THREADS	matrix generation	Dot production
1	9.78348922729	44.6111421585
2	9.66711163520	25.1156024932
4	9.91228699684	19.0398018360

```
import os
#must set these before loading numpy:
os.environ["OMP_NUM_THREADS"] = '4' # export
OMP_NUM_THREADS=4
os.environ["OPENBLAS_NUM_THREADS"] = '4' # export
OPENBLAS_NUM_THREADS=4
os.environ["MKL_NUM_THREADS"] = '4' # export
MKL_NUM_THREADS=6
#os.environ["VECLIB_MAXIMUM_THREADS"] = '4' # export
VECLIB_MAXIMUM_THREADS=4
#os.environ["NUMEXPR_NUM_THREADS"] = '4' # export
NUMEXPR_NUM_THREADS=6

import numpy as np
import time

#np.__config__.show() #looks like I have MKL and blas
np.show_config()

start_time=time.time()
#test script:
a = np.random.randn(5000, 50000)
b = np.random.randn(50000, 5000)
ran_time=time.time()-start_time
print("time to complete random matrix generation was %s
seconds" % ran_time)
np.dot(a, b) #this line should be multi-threaded
print("time to complete dot was %s seconds" % (time.time() -
start_time - ran_time))
```

C-Extensions(Cython)

- C/C++面向電腦底層，和組合語言有很好的互動。
- Python符合人的思維方式，是容易使用的語言。
- 效率和易用性需要取得妥協。有多種方式及層次可以達到這目的
- Python提供了與C/C++Lib相互呼叫的機制，能透過import作進行呼叫的C/C++的Lib，就是C-Extensions，有很多選擇可以實現C-Extensions，例如Swig、Cython等。
- Cython的基本精神是，將原本python程式碼擴展為Cython語言（副檔名通常為pyx文件）直接編譯成C-Extensions。
- 安裝方式。
  - pip install cython
  - conda install cython

使用步驟

- 按照cython規範改動程式(變數及函式)。
- 存檔為.pyx副檔名作為識別。
- 準備setup.py說明如何編譯程式。
- python setup.py build_ext --inplace
- 在python引入編譯後的程式。

變數宣告的差異

PYTHON

與某些程式語言（如：C/C++, Java 等）不同的是，在Python 中使用變數不需要事先宣告，或是指定資料型態。

x = 0.5

CYTHON

cdef int a, b, c

cdef char *s

cdef float x = 0.5 (single precision)

cdef double x = 63.4 (double precision)

cdef list names

cdef dict goals_for_each_play

cdef object card_deck

函示宣告的差異

PYTHON

def – 普通的 Python 函式。

CYTHON

cdef – Cython 函式，不能透過純 Python 程式碼使用該函式，必須在 Cython 內使用

cpdef – C 和 Python 共用，可以透過 C 或者 Python 程式碼使用該函式。

Demo case: 階乗

run_python.py

```
def test(x):
    y = 1
    for i in range(1, x+1):
        y *= i
    return y
```

run_cython.pyx

```
cpdef int test(int x):
    cdef int y = 1
    cdef int i
    for i in range(1, x+1):
        y *= i
    return y
```

setup.py

```
import setuptools
from distutils.core import setup
from Cython.Build import cythonize


setup(ext_modules = cythonize('run_cython.pyx'))
```

(py39-tf2-gpu) C:\Users\ac_ya\Dropbox\機器學習於材料資訊的應用\111\src>python setup.py build_ext
running build_ext
building 'run_cython' extension
error: Unable to find vcvarsall.bat

沒有找到vcvarsall.bat指定的vc++編譯器

Windows要額外做的事

在windows平臺下，cython要調用vc++編譯器對生成的c文件進行編譯，才能產生pyd文件。

(py39-tf2-gpu) C:\Users\ac_ya\Dropbox\機器學習於材料資訊的應用\111\src>python

Python 3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32

Type "help", "copyright", "credits" or "license" for more information.

>>>

對應的VC版本

Visual C++	CPython
14.x	3.5 - 3.10
10.0	3.3 - 3.4
9.0	2.6 - 2.7, 3.0 - 3.2

Visual Studio 2017 version 15.5	1912
Visual Studio 2017 version 15.6	1913
Visual Studio 2017 version 15.7	1914
Visual Studio 2017 version 15.8	1915
Visual Studio 2017 version 15.9	1916
Visual Studio 2019 RTW (16.0)	1920

對應的VS版本

要多裝Microsoft Build Tools 2015

https://wiki.python.org/moin/WindowsCompilers https://www.microsoft.com/zh-TW/download/details.aspx?id=48159

https://docs.microsoft.com/en-us/cpp/preprocessor/predefined-macros?view=msvc-170

Windows還是有問題

(py39-tf2-gpu) C:\Users\ac_ya\Dropbox\機器學習於材料資訊的應用\111\src>python setup.py build_ext
running build_ext
building 'run_cython' extension
error: Microsoft Visual C++ 14.0 or greater is required. Get it with "Microsoft C++ Build Tools": https://visualstudio.microsoft.com/visual-cpp-build-tools/

改到其他平台(colab)

- 2023-ML@MGI-Week05-imp.ipynb
  1. !pip install cython
  2. !python setup.py build_ext --inplace
  3. !python test_code.py 10 100 1000 10000

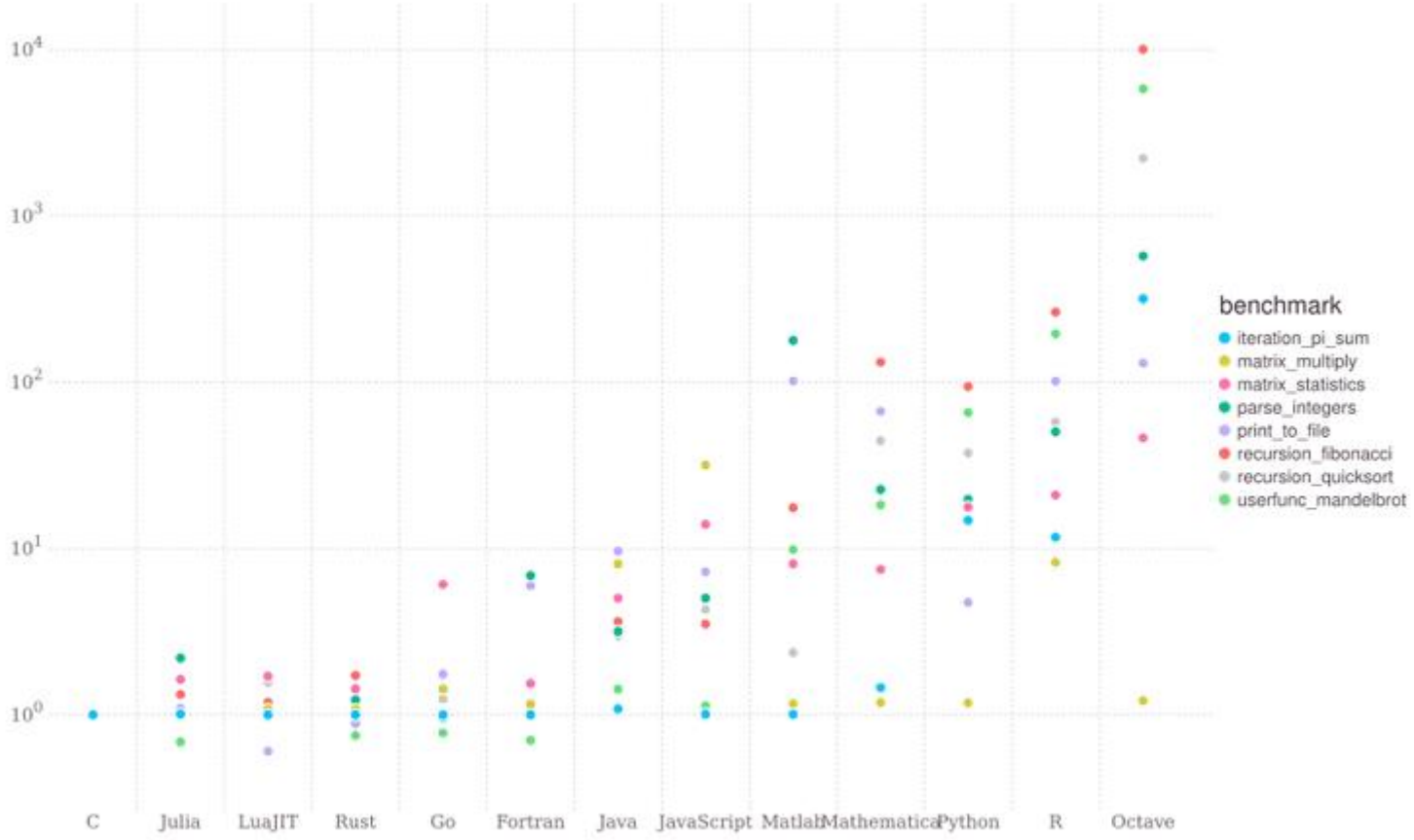
```
1 !python test_code.py 10 100 1000 10000

[10, 100, 1000, 10000]
Python time = 4.76837158203125e-06
Cython time = 2.1457672119140625e-06
Speedup = 2.2222222222222223
Python time = 1.6450881958007812e-05
Cython time = 9.5367431640625e-07
Speedup = 17.25
Python time = 0.0002663135528564453
Cython time = 1.430511474609375e-06
Speedup = 186.16666666666666
Python time = 0.020389318466186523
Cython time = 1.4781951904296875e-05
Speedup = 1379.3387096774193
```

results.csv — Raw

Search this file...

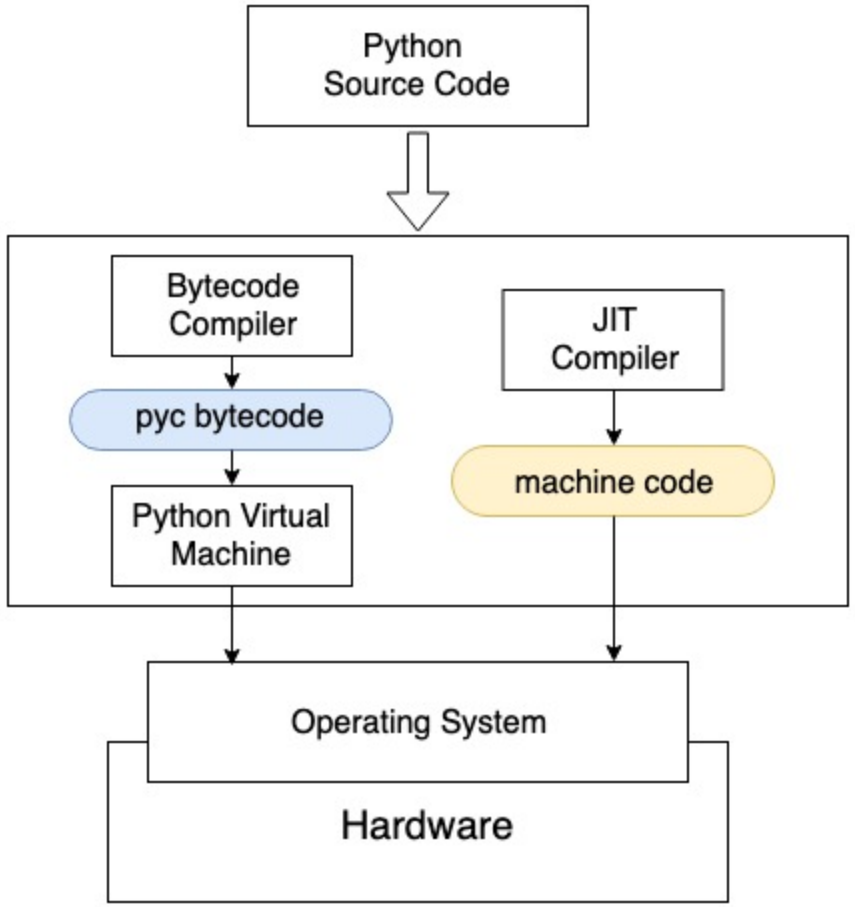
	Number	Python Time	Cython Time	Speedup
2	10	1.6689300537109375e-06	4.76837158203125e-07	3.5
3	100	3.337860107421875e-06	4.76837158203125e-07	7.0
4	1000	2.193450927734375e-05	9.5367431640625e-07	23.0
5	10000	0.0002090930938720703	6.4373016357421875e-06	32.481
6	100000	0.002156257629394531 2	6.008148193359375e-05	35.89
7	1000000	0.02128767967224121	0.0005953311920166016	35.75
8	10000000	0.2148280143737793	0.00594782829284668	36.1187317112278

JIT(Numba)



- 電腦只認得二進位的機器碼，C/C++靠編譯器(compiler)將原始碼編譯成可執行檔。
- Python或Java沒有編譯的步驟，直接透過直譯器(interpreter)把程式碼翻譯成位元組碼(bytecode)後，位元組碼是一種與機器硬體無關的中間代碼，再丟到虛擬機上執行。

Numba



- Numba是一個JIT編譯器實做，透過在程式碼加入簡單的裝飾(@jit)，使用LLVM編譯產生直接的機器碼。
- Just-In-Time（JIT）提升程式碼執行的速度，同時保留python的易用性。使用JIT技術時，JIT編譯器將程式碼編譯成機器碼，可以直接在CPU上執行，跳過了虛擬機，犧牲了平台間的可移植性，換回效率，執行速度和原生的C程式幾乎一樣。
- 安裝方式
  - pip install numba
  - conda install numba

Numba

- (py310-tf2) \python jit_test.py

499999950000000

Time used: 4.799429416656494 sec

23.5X

499999950000000

Time used: 0.2655847072601318 4 sec

```
import time
from numba import jit

def sum_cal(x,y):
    sum_num = 0
    for i in range(x,y):
        sum_num += i
    return sum_num
start_time = time.time()
print(sum_cal(1,100000000))
print('Time used: {}
sec'.format(time.time()-
start_time))

@jit
def sum_cal_jit(x,y):

    sum_num = 0
    for i in range(x,y):
        sum_num += i
    return sum_num

start_time = time.time()
print(sum_cal_jit(1,100000000))
print('Time used: {}
sec'.format(time.time()-
start_time))
```

適用和不適用的場景

適用

Python原生函式和部分NumPy函式

不適用

try…except 異常處理

with…

yield from

scikit-learn、tensorflow、pytorch、pandas