Python tools supporting development and data analysis
PROPOZE project: Numerical Weather Prediction for sustainable Europe

Marcin Polkowski

Institute of Meteorology and Water Management

29 May 2018

Introduction

1. Python in simple, reliable and easy to port cross platform
2. Python is available on most supercomputers
3. Python allows easy automation of testing process
4. Python allows easy data visualization

Introduction

User **FiletOfFish1066**:
From around 6 years ago up until now, I have done nothing at work. I am not joking. For 40 hours each week I go to work, play League of Legends in my office, browse reddit, and do whatever I feel like. In the past 6 years I have maybe done 50 hours of real work. So basically nothing. (...) I explained I had automated my own job (...). Anyway, I was fired.

Python scripts in PROPOZE project

1. Testing and comparing result from different versions of the dwarf
2. Testing performance on supercomputer
3. Comparing weather forecasts from different versions of model

Testing and comparing result from different versions of the dwarf

1. During test single process run signal variable is dumped to binary file every n time steps
2. Same data dumping can be achieved during MPI multi process runs
3. Ideally both single and multiprocess output should be exactly same
4. Python script compiles code for each run (different versions, different compiler options, different CPU config [for static memory], mpi / no mpi), runs the executable, compare results.

Testing and comparing result from different versions of the dwarf

Terminal

```
goto@IMGW:-/IMGW … /mpdata-fortran/manualbuild_diffusion feature/DIFFUSION(+2/-2) ± python3 mpi_test.py
Diffusion Gaussian Test MPI vs NOMPI
by Marcin Polkowski

Test debug: 0
Make style: gnutest

Cleaning previous results:

Starting test.

NOMPI (x:1, y:1, z:1) Compile... finished in  9.044s with 0 errors and 7 warnings
Executing...  finished in  4.863s, result moved to nompi111.nc

MPI (x:1, y:1, z:1) Compile:... finished in  8.901s with 0 errors and 13 warnings
Executing...  finished in  4.766s, result moved to mpi111.nc
Comparing: no difference between nompi111.nc and mpi111.nc

MPI (x:2, y:1, z:1) Compile:... finished in  9.381s with 0 errors and 13 warnings
Executing...  finished in  2.936s, result moved to mpi211.nc
Comparing: difference between nompi111.nc and mpi211.nc

MPI (x:1, y:2, z:1) Compile:... finished in 10.751s with 0 errors and 13 warnings
Executing...  finished in  2.790s, result moved to mpi121.nc
Comparing: no difference between nompi111.nc and mpi121.nc

MPI (x:1, y:1, z:2) Compile:... finished in  8.986s with 0 errors and 13 warnings
Executing...  finished in  2.614s, result moved to mpi112.nc
Comparing: no difference between nompi111.nc and mpi112.nc

Test summary:
(x:1, y:1, z:1): OK     time: 4.766     speedup: 1.02   cpus: 1
(x:2, y:1, z:1): FAIL   time: 2.936     speedup: 1.66   cpus: 2
(x:1, y:2, z:1): OK     time: 2.790     speedup: 1.74   cpus: 2
(x:1, y:1, z:2): OK     time: 2.614     speedup: 1.86   cpus: 2
goto@IMGW:-/IMGW … /mpdata-fortran/manualbuild_diffusion feature/DIFFUSION(+2/-2) 1m5s ±
```

Testing performance on supercomputer

1. We want to check how computation scales up with number of CPUs used
2. We want to check how performance differs depending on CPU distribution along 3 axis of the model
3. We want this testing process fully automatic
4. We use Python to work for us!

Testing performance on supercomputer: scalability

1. Tested executables (multiple are supported) need to support dynamic memory allocation and feature setting number of CPUs in x, y and z-direction as command line parameters
2. Tested executables need to support execution timer (MPI or CPU time)
3. Python testing framework requires just few parameters to run

Testing performance on supercomputer: scalability

```
mode = "group" # or "single"
T = Tester()
T.SetTimer("diff_fckflxdv")

T.AddDomain( 512,  256, 128, color='#800000')
T.AddDomain(1024,  512, 128, color='#000080')
T.AddDomain(2048, 1024, 128, color='#008000')

T.AddIterations(0,1500,100)

T.SetTemplate('GRAD.tpl')
T.SetOutputSuffix("logout")
T.AddExecutable('./test.out', 'test', symbol='o')


T.SetCpuConfig([2,4,8,16,32,64,128], [2,4,8,16,32,64], [1,2,4,8,16,32])
T.GenerateJobs()
```

Testing performance on supercomputer: scalability

marcin — ssh goto@SR101661 — 173×48

```
[mpolkowski@dn:~/IMGW/HPC] master(+41/-21)* 35s 1 ± python3 run2.py test
Processing outfiles...  done
Reading timers...       done
Cleaning up files...    done

Processing Domain [512, 256, 128] for test
        Processing cpu config 4
```

result	executable	domain size	cpu x	cpu y	cpu z	timesteps	time / iter	to fastest	time / iter / core
OK	test	512 x 256 x 128	2	2	1	100	2.3567 s	1.00 x	9.4266080 s

```
        Processing cpu config 8
```

result	executable	domain size	cpu x	cpu y	cpu z	timesteps	time / iter	to fastest	time / iter / core
OK	test	512 x 256 x 128	2	2	2	100	1.1900 s	1.00 x	9.5197520 s
FAIL	test	512 x 256 x 128	2	4	1	-	-	-	-
FAIL	test	512 x 256 x 128	4	2	1	-	-	-	-

```
        Processing cpu config 16
```

result	executable	domain size	cpu x	cpu y	cpu z	timesteps	time / iter	to fastest	time / iter / core
OK	test	512 x 256 x 128	4	2	2	100	0.6017 s	1.00 x	9.6272960 s
OK	test	512 x 256 x 128	4	4	1	100	0.6029 s	1.00 x	9.6461760 s
OK	test	512 x 256 x 128	2	4	2	100	0.6054 s	1.01 x	9.6869600 s
OK	test	512 x 256 x 128	8	2	1	100	0.6088 s	1.01 x	9.7412320 s
OK	test	512 x 256 x 128	2	8	1	100	0.6160 s	1.02 x	9.8556320 s
OK	test	512 x 256 x 128	2	2	4	100	0.6161 s	1.02 x	9.8576960 s

```
        Processing cpu config 32
```

result	executable	domain size	cpu x	cpu y	cpu z	timesteps	time / iter	to fastest	time / iter / core
OK	test	512 x 256 x 128	4	4	2	100	0.3086 s	1.00 x	9.8749760 s
OK	test	512 x 256 x 128	8	2	2	100	0.3105 s	1.01 x	9.9362880 s
OK	test	512 x 256 x 128	8	4	1	100	0.3107 s	1.01 x	9.9434880 s

Figure: Example of scalability test result

Testing performance on supercomputer: scalability

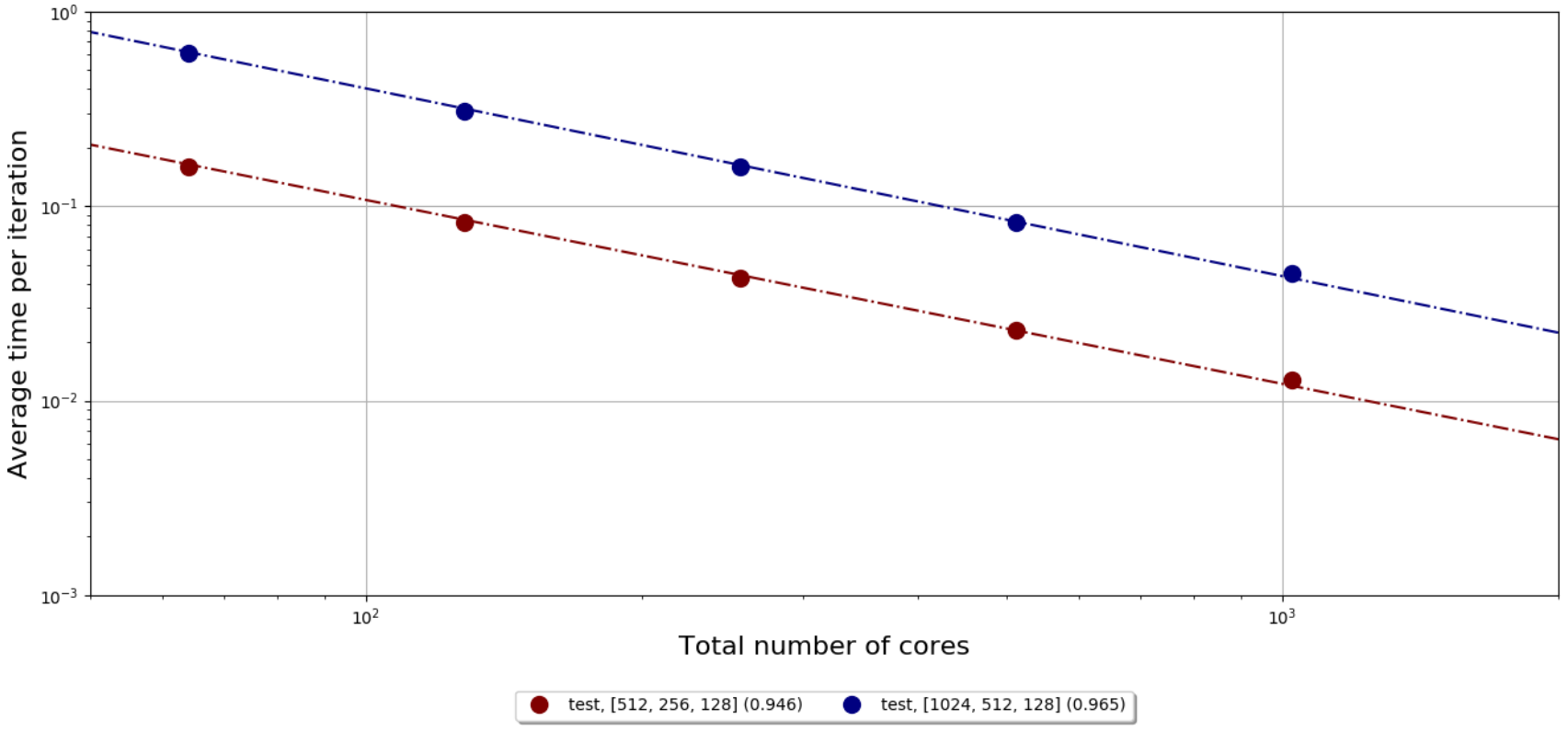


Figure: Example of scalability test result

Testing performance on supercomputer: scalability

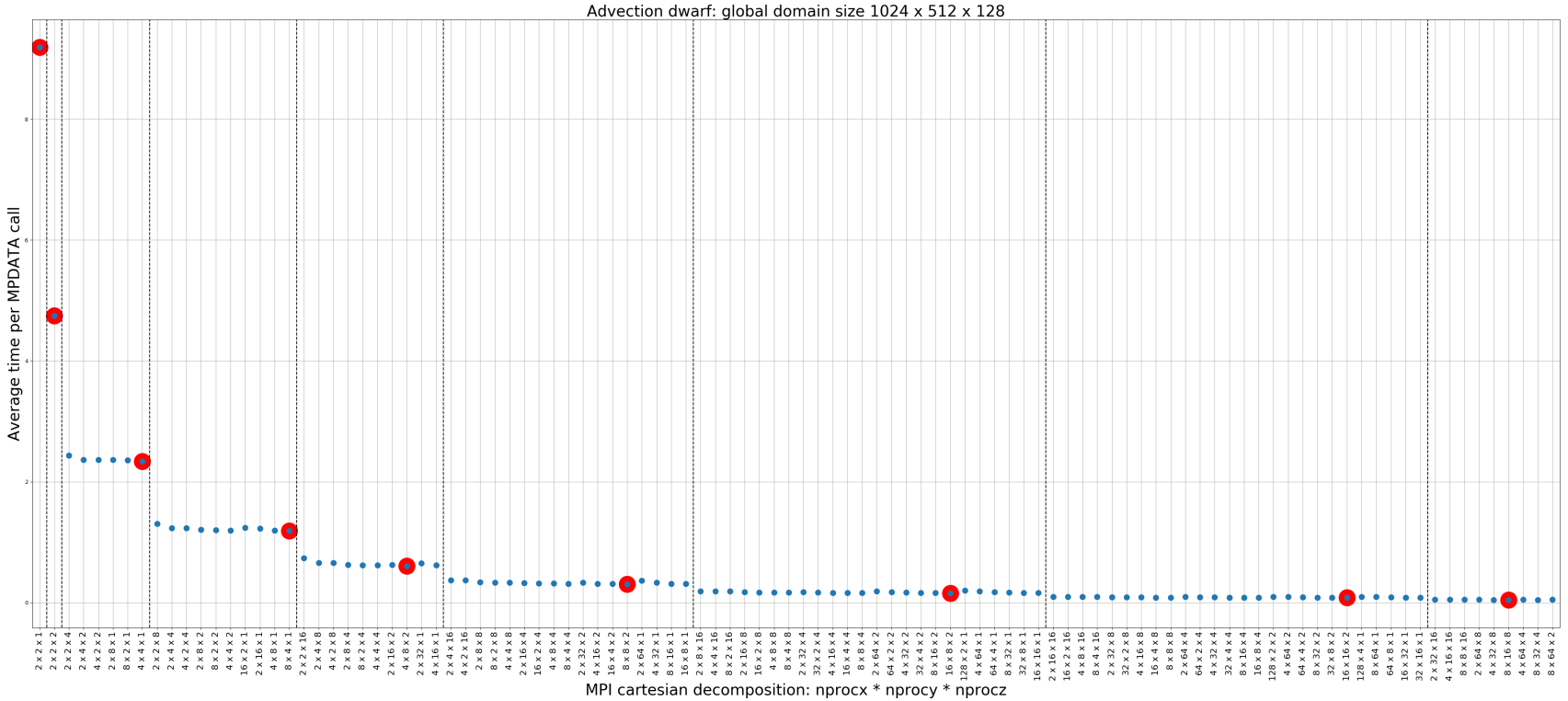


Figure: Example of CPU config test result

Testing performance on supercomputer: scalability

```
mode = "group" # or "single"
T = Tester()
T.SetTimer("diff_fckflxdv")

T.AddDomain( 525,  437,  128, color='#058062')

T.AddIterations(0,1500,100)
T.SetTemplate('GRAD.tpl')
T.SetOutputSuffix("logout")
T.AddExecutable('./test.out', 'test', symbol='o')

T.GenerateJobsTotalCPU(240,50,50,1)
T.GenerateJobsTotalCPU(480,50,50,1)
```

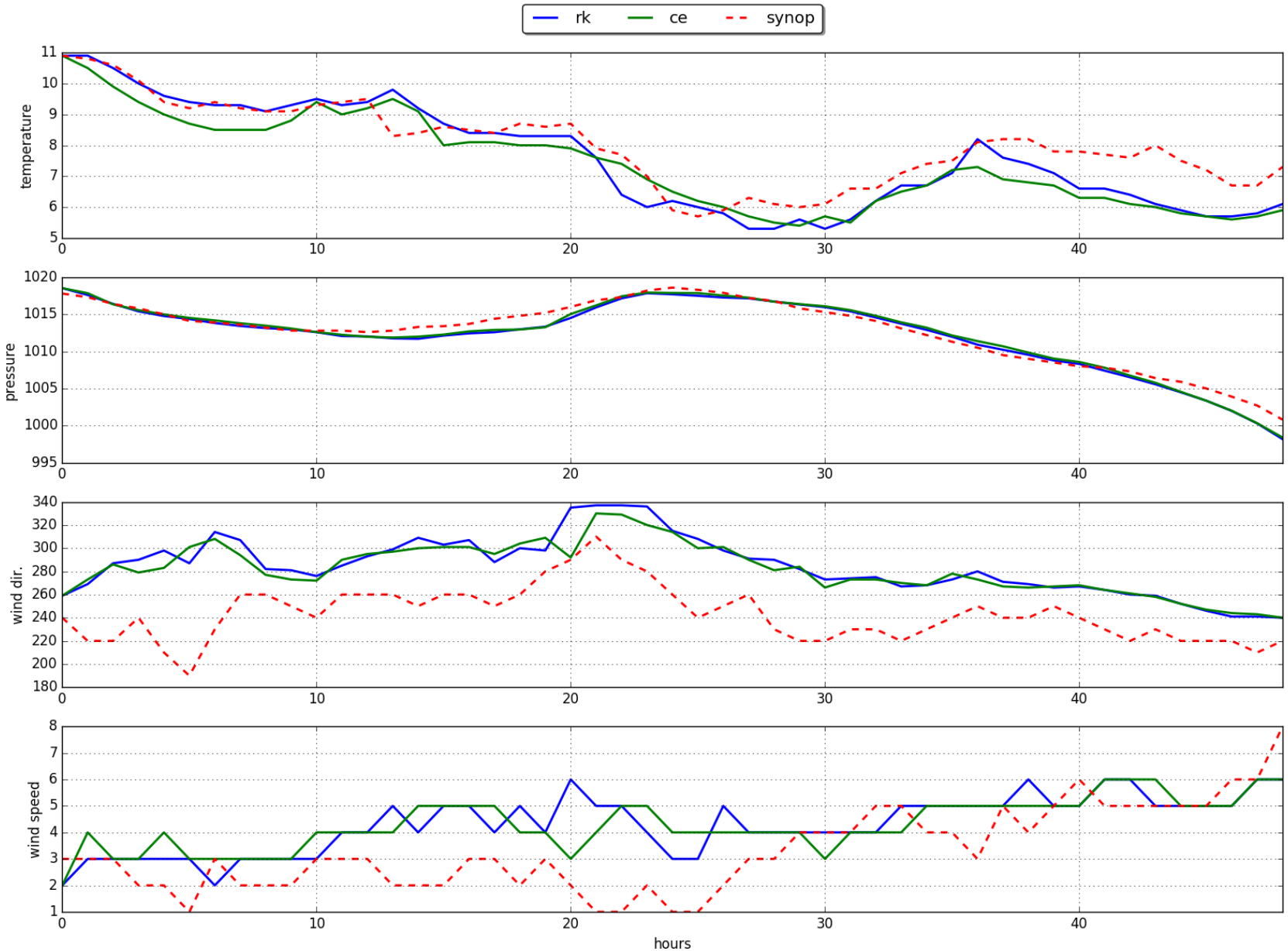
Testing performance on supercomputer: scalability

1. This Python framework is under constant development
2. It is available on github:
   https://github.com/gozwei/HPC-performance-tester
3. Documentation is still on to do list

Comparing weather forecasts from different versions of model

1. One Python scripts reads SYNOP weather reports and saves data into easy to use SQL database (this is done for efficiency)
2. Second script allows comparing real measurements with forecasts from different model versions for selected station
3. Third script allows computing and visualizing mean error between real world data and model forecast over selected set of stations

Testing performance on supercomputer: scalability



Summary

Python makes our lives easier :)