arm

Improving Python-based workloads in HPC

Florent.Lebeau@arm.com

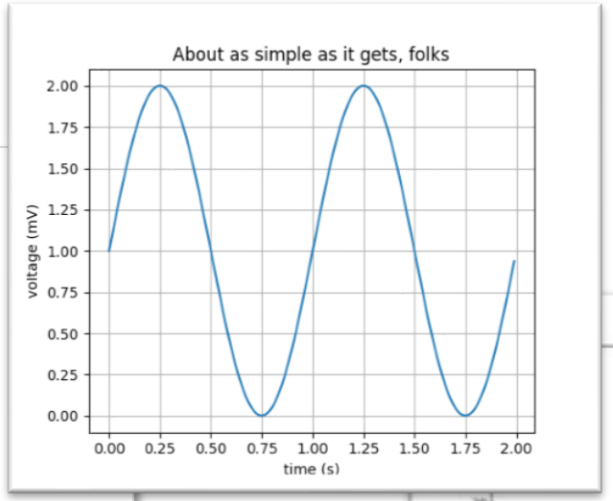
Outline

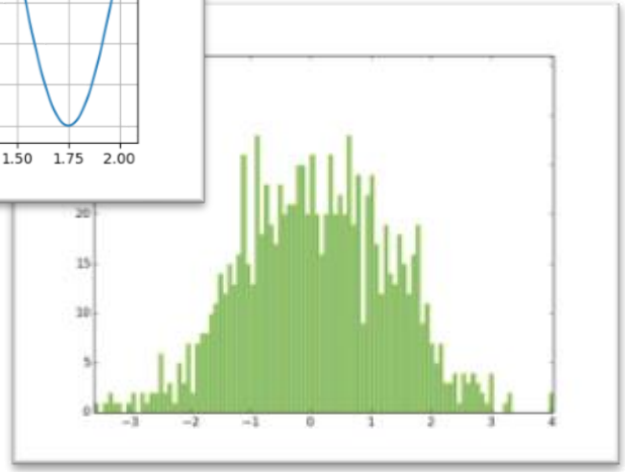
- Why using Python in HPC?
- Profiling Python
- Introduction to Arm MAP
- Optimizing Python-based workloads on single and many nodes

Python in HPC

- High-level
- Readable
- Free
- Cross-platform
- Extensible
  - $ pip install matplotlib
  - $ conda install matplotlib

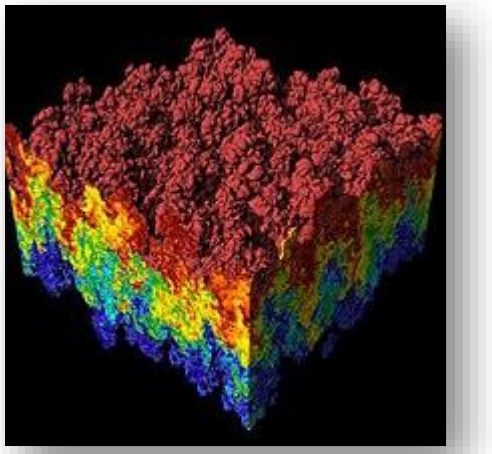
```
import matplotlib.pyplot as plt
import numpy as np

t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2*np.pi*t)
plt.plot(t, s)

plt.xlabel('time (s)')
plt.ylabel('voltage (mV)')
plt.title('About as simple as it gets, folks')
plt.grid(True)
plt.savefig("test.png")
plt.show()
```

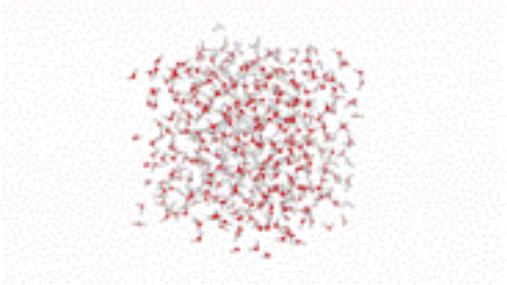




Python in HPC

- Essential modules:
  - **NumPy**: support of large multi-dimensional arrays and matrices
  - **SciPy**: support for linear algebra, integration, interpolation, FFT, …
  - **MPI4Py**: provides bindings of the MPI standard
- Rely on highly-optimized libraries
  - Written in lower-level languages: C, FORTRAN, …
  - BLAS, LAPACK, FFTW, …
- Can easily be interfaced with other languages





Profiling Python

- time

```
import time

[…]
t = time.clock()
[…]
print “time spent:“, time.clock() - t, “seconds”
```

- cProfile
  - $ python -m cProfile -s tottime ./laplace1.py slow 100 500

```
127857 function calls (124922 primitive calls) in 25.961 seconds
  Ordered by: internal time
  ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     500   25.568    0.051   25.631    0.051 laplace1.py:101(slowTimeStep)
   49706    0.064    0.000    0.064    0.000 {range}
       1    0.034    0.034   25.961   25.961 laplace1.py:23(<module>)
[…]
```

- Line_profiler
  - $ kernprof -l -v ./laplace1.py slow 100 500

```
Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
   99                                           @profile
  100                                           def slowTimeStep(self, dt=0.0):
[…]
  109     49500      35571.0      0.7      0.1          for i in range(1, nx-1):
  110   4851000    3698877.0      0.8      9.6              for j in range(1, ny-1):
  111   4802000    4984464.0      1.0     12.9                  tmp = u[i,j]
  112   4802000    8034699.0      1.7     20.8                  u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
  113   4802000   10799762.0      2.2     28.0                           (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
[…]
```

Profiling Python: challenges

Easy to use

- In complex workflow
- Without instrumenting the code
- Usable

Profile the whole application

- Large projects
- On many nodes

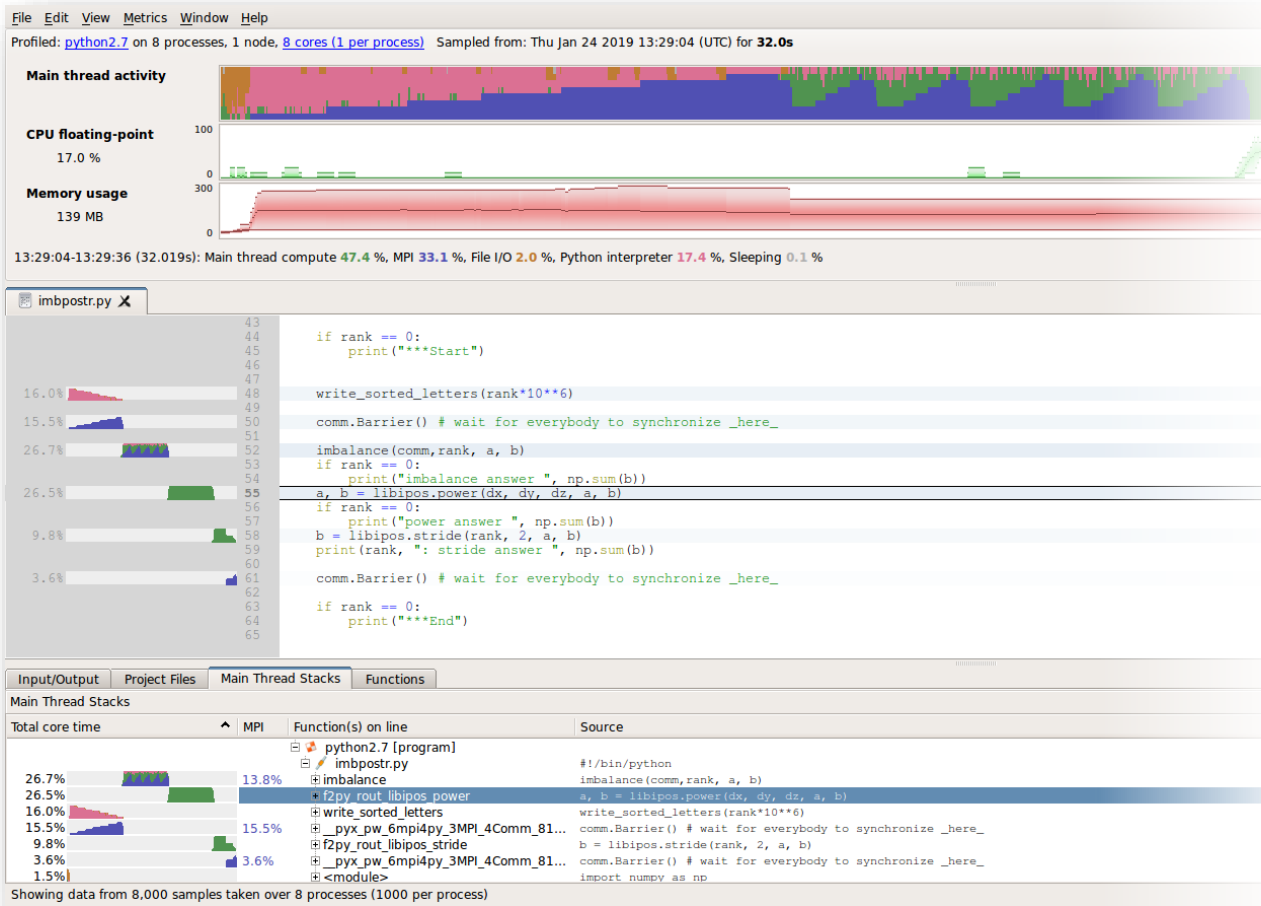
Little overhead

- Relevant profile at scale

Arm MAP profiler

arm MAP

- Arm MAP is a lightweight multi-node profiling tool
  - Shows processes and threads activity over time
  - Source code is annotated
  - Information aggregated by stacks and function
- **Compute**, **IO**, **MPI**, **Python interpreter**
- Part of Arm Forge



arm

Arm MAP: usage

- Launch command
  - $ **python** ./laplace1.py slow 100 100
- Profiling command
  - $ **map --profile python** ./laplace1.py slow 100 100
  - --profile: non-interactive mode
  - --output: name of output file
- Display profiling results
  - $ **map** laplace1.map

Laplace1.py

```
[…]
err = 0.0
for i in range(1, nx-1):
  for j in range(1, ny-1):
    tmp = u[i,j]
    u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
      (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
    diff = u[i,j] - tmp
    err += diff*diff
return numpy.sqrt(err)
[…]
```

Naïve Python loop

File Edit View Metrics Window Help

Profiled: python on 1 process, 1 node Sampled from: Thu Jan 24 2019 16:21:41 (UTC) for **27.7s**

Hide Metrics...

Main thread activity

CPU floating-point 0.3 %

CPU integer 10.0 %

CPU memory access 73.6 %

16:21:41-16:22:08 (27.717 s): Main thread compute **51.5** %, File I/O **0.1** %, Python interpreter **48.1** %

Zoom

laplace1.py

```
97      self.setTimeStepper(stepper)
98
99  def slowTimeStep(self, dt=0.0):
100     """Takes a time step using straight forward Python loops."""
101     g = self.grid
102     nx, ny = g.u.shape
103     dx2, dy2 = g.dx**2, g.dy**2
104     dnr_inv = 0.5/(dx2 + dy2)
105     u = g.u
106
107     err = 0.0
108     for i in range(1, nx-1):
109         for j in range(1, ny-1):
110             tmp = u[i,j]
111             u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
112                       (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
113             diff = u[i,j] - tmp
114             err += diff*diff
115
116     return numpy.sqrt(err)
117
118 def numericTimeStep(self, dt=0.0):
```

1.2% 6.3% 28.9% 44.3% 10.9% 7.4%

Time spent on line 112

Breakdown of the 44.3% time spent on this line:

Executing instructions	0.0%
Calling other functions	58.7%
Executing Python code	41.3%

Input/Output | Project Files | Main Thread Stacks | Functions

Main Thread Stacks

Total core time	Function(s) on line	Source	Position
	time_test	print "took", time_test(n, n, stepper=solver, n_iter=n_iter), "seconds"	laplace1.py:235
	solve	s.solve(n_iter=n_iter, eps=eps)	laplace1.py:228
	slowTimeStep	err = self.timeStep()	laplace1.py:195
44.3%	double_multiply, array_subscrip...	(u[i, j-1] + u[i, j+1])*dx2)*dnr_inv	laplace1.py:112
28.7%	array_subscript, double_multipl...	u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +	laplace1.py:111
10.9%	array_subscript	diff = u[i,j] - tmp	laplace1.py:113
7.4%	double_add, double_multiply, _...	err += diff*diff	laplace1.py:114

Showing data from 1,000 samples taken over 1 process (1000 per process)

Arm Forge 19.0.1 Main Thread View

Optimizing computation on NumPy arrays

Naïve Python loop

```
err = 0.0
for i in range(1, nx-1):
  for j in range(1, ny-1):
    tmp = u[i,j]
    u[i,j] = ((u[i-1, j] + u[i+1, j])*dy2 +
    (u[i, j-1] + u[i, j+1])*dx2)*dnr_inv
    diff = u[i,j] - tmp
    err += diff*diff
return numpy.sqrt(err)
```

NumPy loop

```
u[1:-1, 1:-1] =
    ((u[0:-2, 1:-1] + u[2:, 1:-1])*dy2 +
     (u[1:-1,0:-2] + u[1:-1, 2:])*dx2)*dnr_inv

return g.computeError()
```

NumPy array notation

File Edit View Metrics Window Help

Profiled: python on 1 process, 1 node Sampled from: Thu Jan 24 2019 16:17:48 (UTC) for **22.9s**

Hide Metrics...

Main thread activity

CPU floating-point 38.0 %

CPU memory access 90.9 %

CPU fp vector 38.0 %

16:17:48-16:18:10 (22.866 s): Main thread compute **97.1** %, Free I/O **1.0** %, Python interpreter **0.6** %, Sleeping **1.3** %

Zoom

laplace1.py

```
117
118     def numericTimeStep(self, dt=0.0):
119         """Takes a time step using a numeric expressions."""
120         g = self.grid
121         dx2, dy2 = g.dx**2, g.dy**2
122         dnr_inv = 0.5/(dx2 + dy2)
123         u = g.u
124         g.old_u = u.copy()
125
126         # The actual iteration
127         u[1:-1, 1:-1] = ((u[0:-2, 1:-1] + u[2:, 1:-1])*dy2 +
128                          (u[1:-1,0:-2] + u[1:-1, 2:])*dx2)*dnr_inv
129
130         return g.computeError()
131
132     def fortran77TimeStep(self, dt=0.0):
133         """Takes a time step using a simple fortran module that
134         implements the loop in fortran90 arrays.  Use f2py to compile
135         flaplace.f like so: f2py -c flaplace.f -m flaplace.  You need
136         the latest f2py version for this to work.  This Fortran
```

9.0% 23.0% 47.8% 19.0%

Time spent on line 128

Breakdown of the 47.0% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%
Executing Python code	0.0%

Input/Output | Project Files | Main Thread Stacks | Functions

Functions

Self	Total	Child	Function
31.1%	31.1%		__memmove_ssse3_back
29.2%	29.2%		_mm_add_pd [inlined]
13.4%	13.4%		_mm_sub_pd [inlined]
9.0%	9.0%		_mm_mul_pd [inlined]
6.6%	6.6%		_mm_store_pd [inlined]
4.5%	4.5%		ddot_kernel_8
1.3%	1.3%		brk
0.7%	0.7%		[PYTHON_IO_WRITE]
0.6%	0.6%		sse2_binary_add_DOUBLE [inlined]

Showing data from 1,000 samples taken over 1 process (1000 per process)

Arm Forge 19.0.1 Main Thread View

Use FORTRAN in Python applications

- F2PY: FORTRAN to Python interface generator
- Part of NumPy
- Compile with debugging flag for profiling
  - $ f2py **--debug** -c flaplace90_arrays.f90 -m flaplace90_arrays
  - Relies on underlying compiler: GCC, IFORT, PGI
  - Generates a *.so library imported in the Python script:

    ```
    import flaplace90_arrays
    ```
  - --debug: enables debug information

Use FORTRAN in Python applications

FORTRAN loop

```
subroutine timestep(u,n,m,dx,dy,error)
[…]
!f2py intent(in) :: dx,dy
!f2py intent(in,out) :: u
!f2py intent(out) :: error
!f2py intent(hide) :: n,m
[…]

u(1:n-2, 1:m-2)=((u(0:n-3, 1:m-2) + u(2:n-1, 1:m-2))*dy2 + &
       (u(1:n-2,0:m-3) + u(1:n-2, 2:m-1))*dx2)*dnr_inv

error=sqrt(sum((u-diff)**2))
end subroutine
```

Python script

```
import flaplace90_arrays

[…]

def fortran90TimeStep(self, dt=0.0):
  g = self.grid
  g.u,err =
    flaplace90_arrays.timestep(g.u, g.dx, g.dy)
  return err

[…]
```

FORTRAN code

File Edit View Metrics Window Help

Profiled: python on 1 process, 1 node Sampled from: Fri Jan 25 2019 13:33:46 (UTC) for **49.2s**

Hide Metrics...

Main thread activity

CPU floating-point 47.3 %

CPU memory access 52.0 %

CPU fp vector 28.4 %

13:33:46-13:34:35 (49.228 s): Main thread compute **98.0** %, File I/O **0.1** %, Python interpreter **0.5** %, Sleeping **1.4** %

Zoom

laplace1.py | flaplace90_arrays.f90

```
integer, intent(in) :: n,m
real (kind=8), dimension(0:n-1,0:m-1) :: diff
real (kind=8) :: dx2,dy2,dnr_inv

!f2py intent(in) :: dx,dy
!f2py intent(in,out) :: u
!f2py intent(out) :: error
!f2py intent(hide) :: n,m

dx2 = dx*dx
dy2 = dy*dy
dnr_inv = 0.5d0 / (dx2+dy2)

diff=u

u(1:n-2, 1:m-2) = ((u(0:n-3, 1:m-2) + u(2:n-1, 1:m-2))*dy2 + &
                   (u(1:n-2,0:m-3) + u(1:n-2, 2:m-1))*dx2)*dnr_inv

error=sqrt(sum((u-diff)**2))

end subroutine
```

28.1% 48.0% 21.9%

Time spent on line 25

Breakdown of the 47.9% time spent on this line:

Executing instructions	71.6%
Calling other functions	28.4%
Executing Python code	0.0%

Time in instructions executed:

Scalar floating-point	0.0%
Vector floating point	82.5%
Scalar integer	0.3%
Vector integer	0.0%
Memory access	16.9%
Branch	0.0%
Other instructions	0.3%

Input/Output | Project Files | Main Thread Stacks | Functions

Main Thread Stacks

Total core time	Child	Function(s) on line	Source	Position
		f2py_rout_flaplace90_arrays_ti...	g.u, err = flaplace90_arrays.timestep(g.u, g.dx, g.dy)	laplace1.py:150
		timestep, _int_free	source file not found: /tmp/tmpl_nWqZ/src.linux-x86_64-2.7/flaplace90_arraysmodule.c	...lace90_arraysmodule.c:262
48.0%	13.7%	memcpy_ssse3_back	(u(1:n-2,0:m-3) + u(1:n-2, 2:m-1))*dx2)*dnr_inv	flaplace90_arrays.f90:25
28.1%	28.0%	__memcpy_ssse3_back	diff=u	flaplace90_arrays.f90:22
21.9%			error=sqrt(sum((u-diff)**2))	flaplace90_arrays.f90:27
1.3%	1.3%	systrim.isra	_int_free (no debug info)	

Showing data from 1,000 samples taken over 1 process (1000 per process)

Arm Forge 19.0.1 Main Thread View

FORTRAN code

File Edit View Metrics Window Help

Profiled: python on 1 process, 1 node Sampled from: Fri Jan 25 2019 13:33:46 (UTC) for **49.2s**

Hide Metrics...

Main thread activity

CPU floating-point 47.3 %

CPU memory access 52.0 %

CPU fp vector 28.4 %

13:33:46-13:34:35 (49.228s): Main thread compute **98.0** %, File I/O **0.1** %, Python interpreter **0.5** %, Sleeping **1.4** %

Zoom

laplace1.py | flaplace90_arrays.f90

```
integer, intent(in) :: n,m
real (kind=8), dimension(0:n-1,0:m-1) :: diff
real (kind=8) :: dx2,dy2,dnr_inv

!f2py intent(in) :: dx,dy
!f2py intent(in,out) :: u
!f2py intent(out) :: error
!f2py intent(hide) :: n,m

dx2 = dx*dx
dy2 = dy*dy
dnr_inv = 0.5d0 / (dx2+dy2)

diff=u

u(1:n-2, 1:m-2) = ((u(0:n-3, 1:m-2) + u(2:n-1, 1:m-2))*dy2 + &
                   (u(1:n-2,0:m-3) + u(1:n-2, 2:m-1))*dx2)*dnr_inv

error=sqrt(sum((u-diff)**2))

end subroutine
```

28.1% 48.0% 21.9%

Time spent on line 27

Breakdown of the 21.9% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%
Executing Python code	0.0%

Time in instructions executed:

Scalar floating-point	85.8%
Vector floating point	0.0%
Scalar integer	4.6%
Vector integer	0.0%
Memory access*	24.2%
Branch	0.0%
Other instructions	0.0%

* 9.6% memory access instructions, 14.6% implicit memory accesses in other instructions, also counted in their categories

Input/Output | Project Files | Main Thread Stacks | Functions

Main Thread Stacks

Total core time	Child	Function(s) on line	Source
		f2py_rout_flaplace90_arrays_ti...	g.u, err = flaplace90_arrays.timestep(g.u, g.dx, g.dy)
		timestep, _int_free	source file not found: /tmp/tmpl_nWqZ/src.linux-x86_64-2.7/flaplace90_arraysmod
48.0%	13.7%	memcpy_ssse3_back	(u(1:n-2,0:m-3) + u(1:n-2, 2:m-1))*dx2)*dnr_inv
28.1%	28.0%	__memcpy_ssse3_back	diff=u
21.9%			error=sqrt(sum((u-diff)**2))
1.3%	1.3%	systrim.isra	_int_free (no debug info)

flaplace90_arrays.f90:27

Showing data from 1,000 samples taken over 1 process (1000 per process)

Arm Forge 19.0.1 Main Thread View

Recompile module with:

$ f2py --debug --opt="-O3 **-ffast-math**“ …

FORTRAN code fast-math

File Edit View Metrics Window Help

Profiled: python on 1 process, 1 node Sampled from: Mon Jan 28 2019 13:21:02 (UTC) for **46.4s**

Hide Metrics...

Main thread activity

CPU floating-point 38.4 %

CPU memory access 58.4 %

CPU fp vector 38.4 %

13:21:02-13:21:48 (46.442s): Main thread compute **98.7** %, File I/O **0.2** %, Python interpreter **0.2** %, Sleeping **0.9** %

Zoom

laplace1.py | flaplace90_arrays.f90

```
9   integer, intent(in) :: n,m
10  real (kind=8), dimension(0:n-1,0:m-1) :: diff
11  real (kind=8) :: dx2,dy2,dnr_inv
12
13  !f2py intent(in) :: dx,dy
14  !f2py intent(in,out) :: u
15  !f2py intent(out) :: error
16  !f2py intent(hide) :: n,m
17
18  dx2 = dx*dx
19  dy2 = dy*dy
20  dnr_inv = 0.5d0 / (dx2+dy2)
21
22  diff=u
23
24  u(1:n-2, 1:m-2) = ((u(0:n-3, 1:m-2) + u(2:n-1, 1:m-2))*dy2 + &
25                     (u(1:n-2,0:m-3) + u(1:n-2, 2:m-1))*dx2)*dnr_inv
26
27  error=sqrt(sum((u-diff)**2))
28
29  end subroutine
30
```

28.6% 56.2% 13.8%

Time spent on line 27

Breakdown of the 13.8% time spent on this line:

Executing instructions	100.0%
Calling other functions	0.0%
Executing Python code	0.0%

Time in instructions executed:

Scalar floating-point	0.0%
Vector floating point	52.9%
Scalar integer	0.7%
Vector integer	0.0%
Memory access	46.4%
Branch	0.0%
Other instructions	0.0%

Input/Output | Project Files | Main Thread Stacks | Functions

Main Thread Stacks

Total core time	Function(s) on line	Source	Position
	timestep	source file not found: /tmp/tmpYoiKbJ/src.linux-x86_64-2.7/flaplace90_arraysmodule.c	...lace90_arraysmodule.c:262
56.2%	memcpy_ssse3_back	(u(1:n-2,0:m-3) + u(1:n-2, 2:m-1))*dx2)*dnr_inv	flaplace90_arrays.f90:25
28.5%	memcpy_ssse3_back	diff=u	flaplace90_arrays.f90:22
13.8%		error=sqrt(sum((u-diff)**2))	flaplace90_arrays.f90:27
	1 other		
0.9%	systrim.isra	_int_free (no debug info)	

Showing data from 1,000 samples taken over 1 process (1000 per process)

Arm Forge 19.0.1 Connected to: mars Main Thread View

Multi-processing in Python

- MPI4Py
  - Provides Python bindings of the MPI standard
  - MPI: Message Passing Interface
- Rely on existing MPI infrastructure
  - MPICC must be in path when installing module
  - MPIRUN enables to launch the application
- Profiling command
  - $ **map --profile mpirun -n 8 python** ./mmprod.py

MPI4Py example

```
from mpi4py import MPI
import numpy

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

if rank == 0:
    data = {'a': 7, 'b': 3.14}
    comm.send(data, dest=1)
elif rank == 1:
    data = comm.recv(source=0)
    print('On process 1, data is ',data)
```

MPI-parallel matrix multiplication

File Edit View Metrics Window Help

Profiled: python2.7 on 8 processes, 1 node, 8 cores (1 per process) Sampled from: Mon Jan 28 2019 18:06:27 (UTC) for **36.8s**

Hide Metrics...

Main thread activity

CPU memory access 31.2 %

CPU fp vector 50.8 %

MPI call duration 0.27 s

18:06:27-18:07:03 (36.788s): Main thread compute **86.7** %, MPI **13.2** %, File I/O **0.0** %, Python interpreter **0.0** %

Zoom

mmprod.py

```
63    tile_B_ = np.empty_like(my_A)
64    req = [None, None, None, None]
65
66    for it in xrange(0, nit):
67      t0 = time()
68      for r in xrange(mpi_rows):
69          req[EAST]  = ccomm.Isend(tile_A , neigh[EAST])
70          req[WEST]  = ccomm.Irecv(tile_A_, neigh[WEST])
71          req[SOUTH] = ccomm.Isend(tile_B , neigh[SOUTH])
72          req[NORTH] = ccomm.Irecv(tile_B_, neigh[NORTH])
73
74          my_C += np.dot(tile_A, tile_B)
75
76          req[0].Waitall(req)
77
78      comm.barrier()
79      t_total = min(t_total, time()-t0)
80
81    t0 = time()
82    if comm.rank == 0:
83      np.dot(tile_A, tile_B)
84    comm.barrier()
85    t_serial = time()-t0
86
```

0.1% <0.1% 78.5% 1.4% 0.3% 1.6% 11.2%

Time spent on line 74

Breakdown of the 78.5% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%
Executing Python code	0.0%

Input/Output | Project Files | Main Thread Stacks | Functions

Functions

Self	Total	MPI	Child	Function
77.9%	77.9%			sgemm_kernel_SANDYBRIDGE
11.6%	11.6%	11.6%		MPI_Barrier
2.6%	2.6%			__ieee754_log_avx
1.6%	5.8%		4.2%	rk_gauss
1.4%	1.4%	1.4%		MPI_Waitall
1.0%	1.0%			rk_random

Showing data from 8,000 samples taken over 8 processes (1000 per process)

Arm Forge 19.0.1 Connected to: mars Main Thread View

MPI-parallel matrix multiplication

File Edit View Metrics Window Help

Profiled: python2.7 on 8 processes, 1 node, 8 cores (1 per process) Sampled from: Mon Jan 28 2019 18:06:27 (UTC) for **36.8s**

Hide Metrics...

Main thread activity

CPU memory access 31.2 %

CPU fp vector 50.8 %

MPI call duration 0.27 s

18:06:27-18:07:03 (36.788s): Main thread compute **86.7** %, MPI **13.2** %, File I/O **0.0** %, Python interpreter **0.0** %

Zoom

max
avg
min

mmprod.py

```
63    tile_B_ = np.empty_like(my_A)
64    req = [None, None, None, None]
65
66    for it in xrange(0, nit):
67      t0 = time()
68      for r in xrange(mpi_rows):
69          req[EAST]  = ccomm.Isend(tile_A , neigh[EAST])
70          req[WEST]  = ccomm.Irecv(tile_A_, neigh[WEST])
71          req[SOUTH] = ccomm.Isend(tile_B , neigh[SOUTH])
72          req[NORTH] = ccomm.Irecv(tile_B_, neigh[NORTH])
73
74          my_C += np.dot(tile_A, tile_B)
75
76          req[0].Waitall(req)
77
78      comm.barrier()
79      t_total = min(t_total, time()-t0)
80
81    t0 = time()
82    if comm.rank == 0:
83      np.dot(tile_A, tile_B)
84    comm.barrier()
85    t_serial = time()-t0
86
```

0.1% <0.1% 78.5% 1.4% 0.3% 1.6% 11.2%

Time spent on line 74

Breakdown of the 78.5% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%
Executing Python code	0.0%

Input/Output | Project Files | Main Thread Stacks | Functions

Functions

Self	Total	MPI	Child	Function
77.9%	77.9%			sgemm_kernel_SANDYBRIDGE
11.6%	11.6%	11.6%		MPI_Barrier
2.6%	2.6%			__ieee754_log_avx
1.6%	5.8%		4.2%	rk_gauss
1.4%	1.4%	1.4%		MPI_Waitall

Showing data from 8,000 samples taken over 8 processes (1000 per process)

Arm Forge 19.0.1 Connected to: mars Main Thread View

MPI-parallel matrix multiplication

File Edit View Metrics Window Help

Profiled: python2.7 on 8 processes, 1 node, 8 cores (1 per process) Sampled from: Mon Jan 28 2019 18:06:27 (UTC) for **36.8s**

Hide Metrics...

Main thread activity

CPU memory access 31.2 %

CPU fp vector 50.8 %

MPI call duration 0.27 s

18:06:27-18:07:03 (36.788s): Main thread compute **86.7** %, MPI **13.2** %, File I/O **0.0** %, Python interpreter **0.0** %

Zoom

mmprod.py

```
63    tile_B_ = np.empty_like(my_A)
64    req = [None, None, None, None]
65
66    for it in xrange(0, nit):
67      t0 = time()
68      for r in xrange(mpi_rows):
69          req[EAST]  = ccomm.Isend(tile_A , neigh[EAST])
70          req[WEST]  = ccomm.Irecv(tile_A_, neigh[WEST])
71          req[SOUTH] = ccomm.Isend(tile_B , neigh[SOUTH])
72          req[NORTH] = ccomm.Irecv(tile_B_, neigh[NORTH])
73
74          my_C += np.dot(tile_A, tile_B)
75
76          req[0].Waitall(req)
77
78      comm.barrier()
79      t_total = min(t_total, time()-t0)
80
81    t0 = time()
82    if comm.rank == 0:
83      np.dot(tile_A, tile_B)
84    comm.barrier()
85    t_serial = time()-t0
86
```

0.1% <0.1% 78.5% 1.4% 0.3% 1.6% 11.2%

Time spent on line 74

Breakdown of the 78.5% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%
Executing Python code	0.0%

Input/Output | Project Files | Main Thread Stacks | Functions

Functions

Self	Total	MPI	Child	Function
77.9%	77.9%			sgemm_kernel_SANDYBRIDGE
11.6%	11.6%	11.6%		MPI_Barrier
2.6%	2.6%			__ieee754_log_avx
1.6%	5.8%		4.2%	rk_gauss
1.4%	1.4%	1.4%		MPI_Waitall
1.0%	1.0%			rk_random

Showing data from 8,000 samples taken over 8 processes (1000 per process)

Arm Forge 19.0.1 Connected to: mars Main Thread View

MPI-parallel matrix multiplication: OpenBLAS

File Edit View Metrics Window Help

Profiled: python2.7 on 8 processes, 1 node, 8 cores (1 per process) Sampled from: Mon Jan 28 2019 18:06:27 (UTC) for **36.8s**

Hide Metrics...

Main thread activity

CPU memory access
35.8 %

CPU fp vector
63.1 %

MPI call duration
0 s

100
0
100
0
4.71
0

18:06:39-18:06:44 (4.635s, 12.6% of total): Main thread compute **100.0** %, MPI **0.0** %, File I/O **0.0** %, Python interpreter **0.0** %

Zoom

mmprod.py

```
63      tile_B_ = np.empty_like(my_A)
64      req = [None, None, None, None]
65
66      for it in xrange(0, nit):
67        t0 = time()
68        for r in xrange(mpi_rows):
69            req[EAST]  = ccomm.Isend(tile_A , neigh[EAST])
70            req[WEST]  = ccomm.Irecv(tile_A_, neigh[WEST])
71            req[SOUTH] = ccomm.Isend(tile_B , neigh[SOUTH])
72            req[NORTH] = ccomm.Irecv(tile_B_, neigh[NORTH])
73
74            my_C += np.dot(tile_A, tile_B)
75
76            req[0].Waitall(req)
77
78        comm.barrier()
79        t_total = min(t_total, time()-t0)
80
81      t0 = time()
82      if comm.rank == 0:
83        np.dot(tile_A, tile_B)
84      comm.barrier()
85      t_serial = time()-t0
86
```

100.0%

Time spent on line 74

Breakdown of the 100.0% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%
Executing Python code	0.0%

Input/Output | Project Files | Main Thread Stacks | Functions

Functions

Self	Total	MPI	Child	Function
98.8%	98.8%			sgemm_kernel_SANDYBRIDGE
0.8%	0.8%			sgemm_itcopy_SANDYBRIDGE
0.4%	0.4%			sgemm_oncopy_SANDYBRIDGE
<0.1%	100.0%		100.0%	<module>
<0.1%	100.0%		100.0%	main
<0.1%	1.2%		1.2%	array_matrixproduct

Showing data from 1,008 samples taken over 8 processes (126 per process)

Arm Forge 19.0.1 Main Thread View

Recompile NumPy module to use Intel MKL

MPI-parallel matrix multiplication: MKL

File Edit View Metrics Window Help

Profiled: python2.7 on 8 processes, 1 node, 8 cores (1 per process) Sampled from: Mon Jan 28 2019 17:55:28 (UTC) for **101.0s**

Hide Metrics...

Main thread activity

CPU memory access
7.6 %

CPU fp vector
93.4 %

MPI call duration
0 s

100
0
100
0
13.2
0

17:56:01-17:56:14 (13.034s, 12.9% of total): Main thread compute **100.0** %, MPI **0.0** %, OpenMP overhead **0.0** %

Zoom

mmprod.py

```
63      tile_B_ = np.empty_like(my_A)
64      req = [None, None, None, None]
65
66      for it in xrange(0, nit):
67        t0 = time()
68        for r in xrange(mpi_rows):
69            req[EAST]  = ccomm.Isend(tile_A , neigh[EAST])
70            req[WEST]  = ccomm.Irecv(tile_A_, neigh[WEST])
71            req[SOUTH] = ccomm.Isend(tile_B , neigh[SOUTH])
72            req[NORTH] = ccomm.Irecv(tile_B_, neigh[NORTH])
73
74            my_C += np.dot(tile_A, tile_B)
75
76            req[0].Waitall(req)
77
78        comm.barrier()
79        t_total = min(t_total, time()-t0)
80
81      t0 = time()
82      if comm.rank == 0:
83        np.dot(tile_A, tile_B)
84      comm.barrier()
85      t_serial = time()-t0
86
```

100.0%

Time spent on line 74

Breakdown of the 100.0% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%
Executing Python code	0.0%

Input/Output | Project Files | Main Thread Stacks | Functions

Functions

Self	Total	MPI	Child	Function
95.7%	95.7%			mkl_blas_avx_sgemm_kernel_0
3.0%	3.0%			mkl_blas_avx_sgemm_kernel_0_b0
0.9%	0.9%			mkl_blas_avx_sgemm_copyan
0.4%	0.4%			mkl_blas_avx_sgemm_copybn
<0.1%	100.0%		100.0%	<module>

Showing data from 1,032 samples taken over 8 processes (129 per process)

Arm Forge 19.0.1 Main Thread View

Multi-node matrix multiplication: MKL

File Edit View Metrics Window Help

Profiled: python2.7 on 4 processes, 4 nodes, 4 cores (1 per process) Sampled from: Mon Jan 28 2019 17:45:16 (UTC) for **116.7s**

Hide Metrics...

Main thread activity

CPU memory access 7.8 %

CPU fp vector 66.5 %

MPI call duration 0.80 s

17:45:16-17:47:12 (116.728s): Main thread compute **75.9** %, MPI **24.1** %, Python interpreter **0.0** %

Zoom

mmprod.py

```
63      tile_B_ = np.empty_like(my_A)
64      req = [None, None, None, None]
65
66      for it in xrange(0, nit):
67        t0 = time()
68        for r in xrange(mpi_rows):
69            req[EAST]  = ccomm.Isend(tile_A , neigh[EAST])
70            req[WEST]  = ccomm.Irecv(tile_A_, neigh[WEST])
71            req[SOUTH] = ccomm.Isend(tile_B , neigh[SOUTH])
72            req[NORTH] = ccomm.Irecv(tile_B_, neigh[NORTH])
73
74            my_C += np.dot(tile_A, tile_B)
75
76            req[0].Waitall(req)
77
78        comm.barrier()
79        t_total = min(t_total, time()-t0)
80
81      t0 = time()
82      if comm.rank == 0:
83        np.dot(tile_A, tile_B)
84      comm.barrier()
85      t_serial = time()-t0
86
```

68.6% 15.6% 2.8% 8.5%

Time spent on line 74

Breakdown of the 68.5% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%
Executing Python code	0.0%

Input/Output | Project Files | Main Thread Stacks | Functions

Functions

Self	Total	MPI	Child	Function
67.5%	67.5%			mkl_blas_avx_sgemm_kernel_0
15.6%	15.6%	15.6%		MPI_Waitall
8.5%	8.5%	8.5%		MPI_Barrier
2.8%	2.8%			mkl_blas_avx_sgemm_kernel_0_b0
1.5%	1.4%			__ieee754_log_avx

Showing data from 4,000 samples taken over 4 processes (1000 per process)

Enable multi-threading with MKL_NUM_THREADS=8

Hybrid OMP/MPI matrix multiplication

Multithreaded/OpenMP MKL

File Edit View Metrics Window Help

Profiled: python2.7 on 4 processes, 4 nodes, 32 cores (8 per process) Sampled from: Mon Jan 28 2019 17:33:55 (UTC) for **35.9s**

Hide Metrics...

Application activity

CPU memory access 10.7 %

CPU fp vector 29.5 %

MPI call duration 0.77 s

17:33:55-17:34:30 (35.870s): Main thread compute **15.9** %, OpenMP **29.6** %, MPI **53.8** %, OpenMP overhead **0.4** %, Sleeping **0.0** %

Zoom

mmprod.py

```
63    tile_B_ = np.empty_like(my_A)
64    req = [None, None, None, None]
65
66    for it in xrange(0, nit):
67      t0 = time()
68      for r in xrange(mpi_rows):
69          req[EAST]  = ccomm.Isend(tile_A , neigh[EAST])
70          req[WEST]  = ccomm.Irecv(tile_A_, neigh[WEST])
71          req[SOUTH] = ccomm.Isend(tile_B , neigh[SOUTH])
72          req[NORTH] = ccomm.Irecv(tile_B_, neigh[NORTH])
73
74          my_C += np.dot(tile_A, tile_B)
75
76          req[0].Waitall(req)
77
78      comm.barrier()
79      t_total = min(t_total, time()-t0)
80
81    t0 = time()
82    if comm.rank == 0:
83      np.dot(tile_A, tile_B)
84    comm.barrier()
85    t_serial = time()-t0
86
```

29.7% 49.6% 0.1% 1.2% 3.7%

Time spent on line 74

Breakdown of the 28.6% time spent on this line:

Executing instructions	0.0%
Calling other functions	100.0%
Executing Python code	0.0%

Input/Output | Project Files | OpenMP Stacks | OpenMP Regions | Functions

Functions

Self	Total	MPI	Child	Overhead	Function
49.6%	49.6%	49.6%			MPI_Waitall
27.5%	27.5%				mkl_blas_avx_sgemm_kernel_0
6.0%	6.0%				__ieee754_log_avx
3.8%	3.8%	3.8%			MPI_Barrier
3.2%	12.7%		9.5%		rk_gauss
2.4%	2.4%				rk_random

Showing data from 3,640 samples taken over 4 processes (910 per process)

Arm Forge 19.0.1 OpenMP View

Conclusion: topics covered

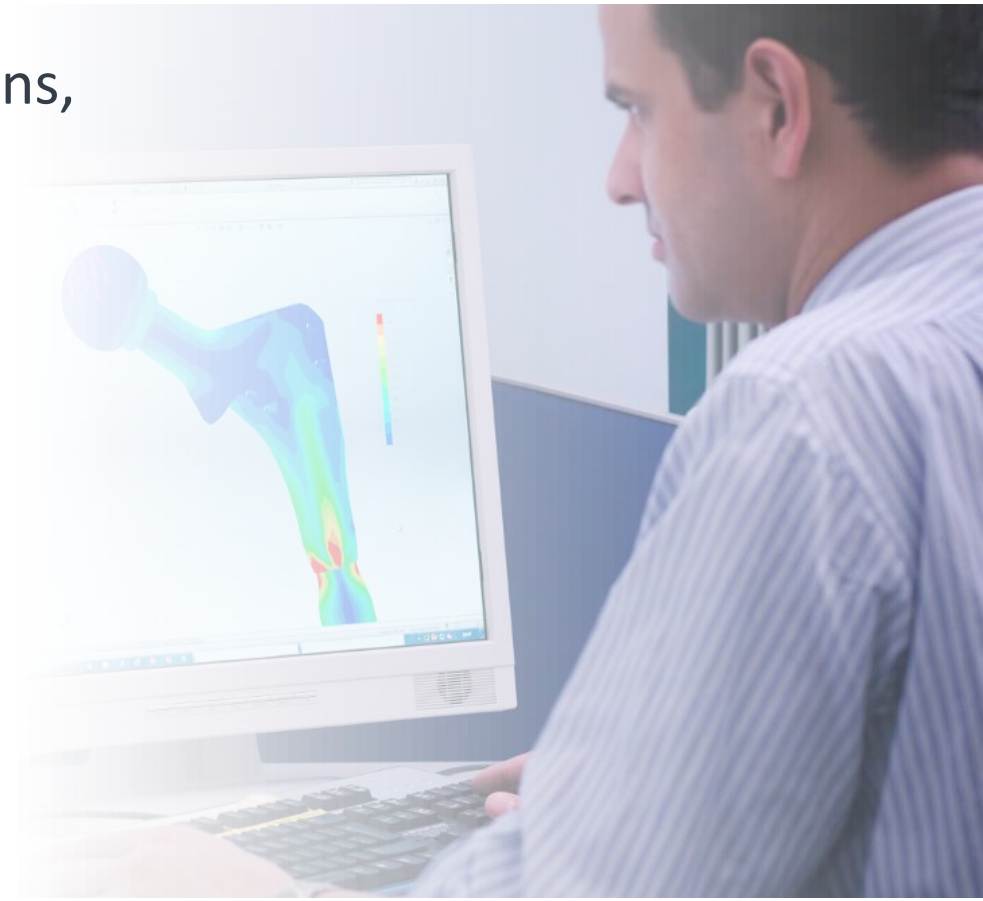
- Profiling Python applications on HPC systems easily with Arm MAP
- How to analyze MAP profiling results
- How to optimize Python
  - Using NumPy array notation and minimizing time spent in interpreter
  - Integrating FORTRAN code
  - Maximizing HPC library efficiency
  - Using multi-threading

Request a trial license

- If you would like to profile your own Python applications, visit our website

- A trial licence of Arm Forge enables:
  - Multi-node and multi-threaded profiling with Arm MAP
  - On all HPC architectures
  - On C/C++, FORTRAN and Python codes
  - Debugging with Arm DDT
  - Access to our support

- Available resources online for your trial



Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
תודה

arm

Questions

arm