HPC on Python

Ramakrishnan Kannan, Shruti Shivakumar

Agenda

- Introduction
- HPC on Python
  - Multiprocessing and joblib
  - Numba and shared memory
  - Numba and cupy
  - MPI4PY

Programming Languages

- Interpreter
  - Platform independence and easy scripting
- Compiler
  - Compiler Optimizations
- Virtual Machines -- Javascript and Java
  - Platform independence vs compilation

Python and performance challenges

- Interpreted – Most of the things are known during runtime
- Type free language
- Parallel Processing
  - Global Interpreter Lock (GIL)
    - Python Global Interpreter Lock or GIL, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter.

Multiprocessing

- multiprocessing.Pool() is the number of processes to create in the pool.
- apply_async() function to pass the arguments to the function cube as list.
- Asynchronous call and do not wait for the function to finish
- get() waits for the task to finish and retrieve the result.

```
import multiprocessing
import time

def cube(x):
    return x**3

if __name__ == "__main__":
    pool = multiprocessing.Pool(3)
    start_time = time.perf_counter()
    processes = [pool.apply_async(cube, args=(x,)) for x in range(1,1000)]
    result = [p.get() for p in processes]
    finish_time = time.perf_counter()
    print(f"Program finished in {finish_time-start_time} seconds")
    print(result)
```

joblib

- Simpler interface than multiprocessing
- Can be used for spawning multithreads

```
import time
from joblib import Parallel, delayed

def cube(x):
    return x**3

start_time = time.perf_counter()
result = Parallel(n_jobs=3)(delayed(cube)(i) for i in range(1,1000))
finish_time = time.perf_counter()
print(f"Program finished in {finish_time-start_time} seconds")
print(result)
```

Numba - Motivation

```
for i in xrange(4096):
    for j in xrange(4096):
        for k in xrange(4096):
            C[i][j] += A[i][k] * B[k][j]
```

Table 1. Speedups from performance engineering a program that multiplies two 4096-by-4096 matrices. Each version represents a successive refinement of the original Python code. "Running time" is the running time of the version. "GFLOPS" is the billions of 64-bit floating-point operations per second that the version executes. "Absolute speedup" is time relative to Python, and "relative speedup," which we show with an additional digit of precision, is time relative to the preceding line. "Fraction of peak" is GFLOPS relative to the computer's peak 835 GFLOPS. See Methods for more details.

Version	Implementation	Running time (s)	GFLOPS	Absolute speedup	Relative speedup	Fraction of peak (%)
1	Python	25,552.48	0.005	1	—	0.00
2	Java	2,372.68	0.058	11	10.8	0.01
3	C	542.67	0.253	47	4.4	0.03
4	Parallel loops	69.80	1.969	366	7.8	0.24
5	Parallel divide and conquer	3.80	36.180	6,727	18.4	4.33
6	plus vectorization	1.10	124.914	23,224	3.5	14.96
7	plus AVX intrinsics	0.41	337.812	62,806	2.7	40.45

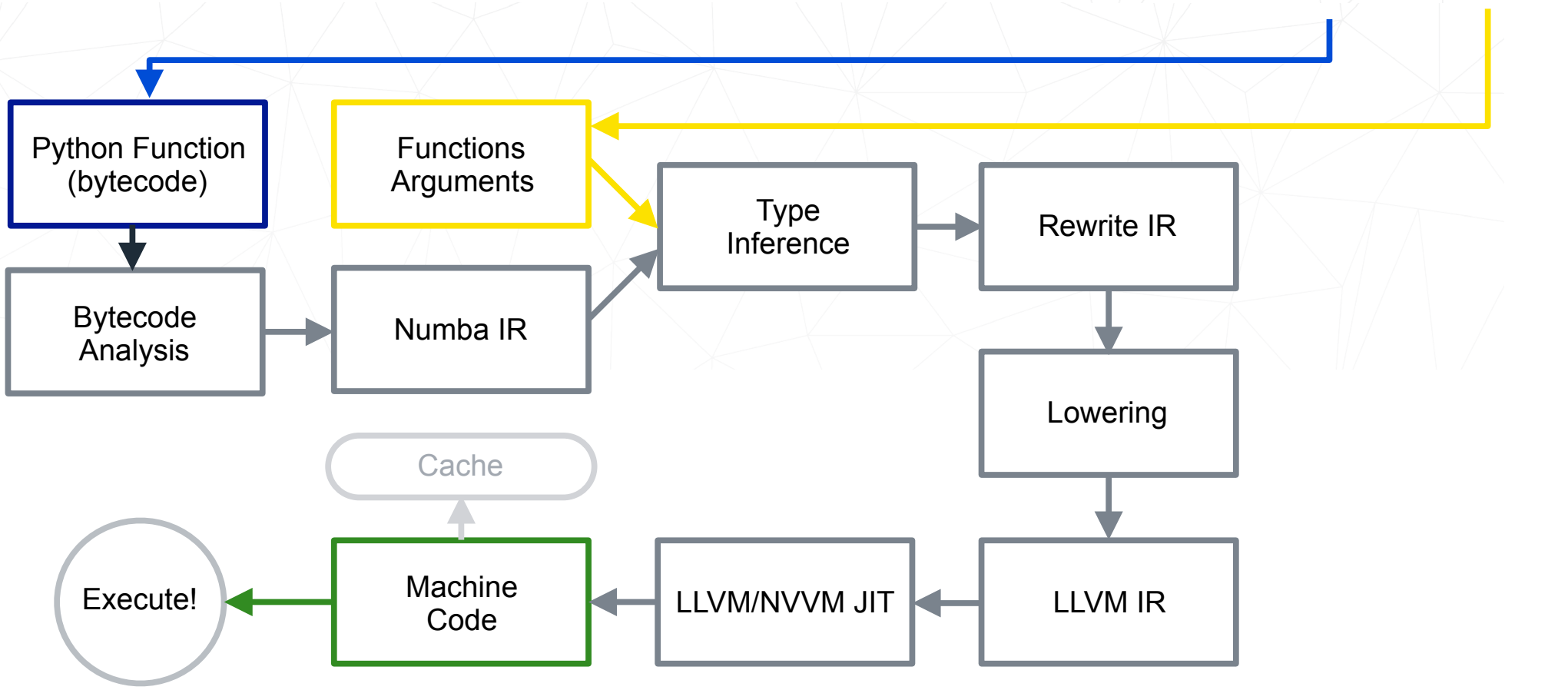
Leiserson, Charles E., Neil C. Thompson, Joel S. Emer, Bradley C. Kuszmaul, Butler W. Lampson, Daniel Sanchez, and Tao B. Schardl. "There's plenty of room at the Top: What will drive computer performance after Moore's law?." *Science* 368, no. 6495 (2020): eaam9744.

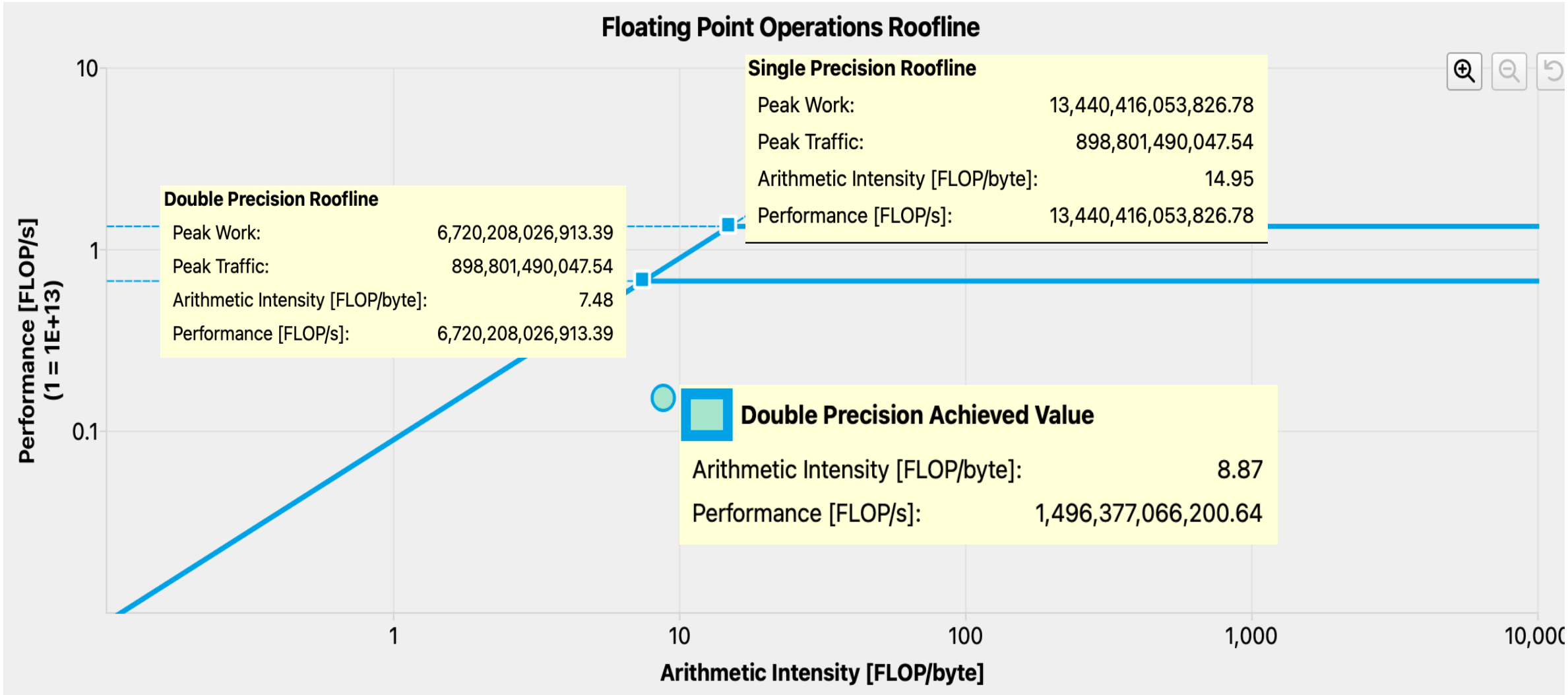
Numba: A JIT Compiler for Python

- An open-source, function-at-a-time compiler library for Python
- Compiler toolbox for different targets and execution models:
- Single-threaded CPU, multi-threaded CPU, GPU – (Openmp and CUDA)
- regular functions, “universal functions” (array functions), etc
- Combine ease of writing Python with speeds of Native code
- Opensource - BSD licensed (including GPU compiler) -- https://github.com/numba/numba
- Goal is to empower scientists who make tools for themselves and other scientists

Numba

```
@jit
def do_math(a, b):
    ...
>>> do_math(x, y)
```




Floating Point Operations Roofline
Single Precision Roofline
Peak Work: 13,440,416,053,826.78
Peak Traffic: 898,801,490,047.54
Arithmetic Intensity [FLOP/byte]: 14.95
Performance [FLOP/s]: 13,440,416,053,826.78
Double Precision Roofline
Peak Work: 6,720,208,026,913.39
Peak Traffic: 898,801,490,047.54
Arithmetic Intensity [FLOP/byte]: 7.48
Performance [FLOP/s]: 6,720,208,026,913.39
Double Precision Achieved Value
Arithmetic Intensity [FLOP/byte]: 8.87
Performance [FLOP/s]: 1,496,377,066,200.64
Performance [FLOP/s] (1 = 1E+13)
10
1
0.1
1
10
100
1,000
10,000
Arithmetic Intensity [FLOP/byte]