Cython: A First Look

SHARCNET: General Interest Webinar

Tyler Collins
HPC Analyst, Brock University

Outline: Today's Aim

- Introduce Cython
- Get everyone on the same page and explain some core concepts
- Live demo
- Quick recap
- Question period

Hopefully at the end of this talk, you will use Cython in your own projects!

This webinar and its materials can be found on GitHub, here:

https://github.com/Andesha/sharcnet-cython

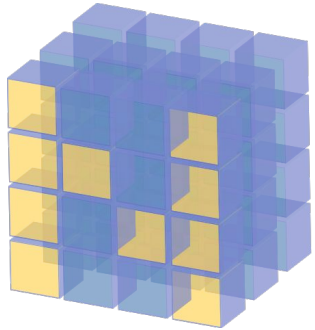
Some Python Commentary

- Python sure is awesome - but awesome isn't free!
- Even other languages have this narrative:
  - "There are no zero cost abstractions!" - Chandler Carruth, 2019
- "Each abstraction must provide more benefit than cost"
  - From the same talk as above
- What happens when you really need your Python to be **faster**?
  - You suffer presumably...

What is Cython?

- Superset of Python
- Recover the C-like performance we gave away
- Python is compiled into C/C++, and then called as per usual
    - No barriers

By using annotations and hints - we can generate good C/C++ code to call **from** Python that is orders of magnitude faster than vanilla Python!



NumPy

pandas

Getting on the Same Page

Just for so we are all working from similar assumptions there will be some quick notes on the following:

- Types and type systems
- Compiled vs Interpreted
- Complexity

We'll be doing a bunch of sweeping generalizations - try not to cringe if you're an expert!

Useful Definitions: Types and Type Systems

- What's a type?
  - A known representation of data that has associated operations
  - Integer, string, boolean, etc
- Dynamically typed
  - Verify at run time (on the fly)
  - Can see a lot of runtime errors
  - Runtime overhead due to figuring things out
  - Pretty easy to write though!
- Statically typed
  - Variables and functions have "signatures" which define what types they operate on
  - Mixing and matching between types is not strictly allowed (mostly)
  - Advantages include things like syntax/grammar checking and error catching

Useful Definitions: Compiled vs Interpreted

We all *know* compiled is faster… but why?

Short answer:

- Compiled code is running natively on a machine from a static source (perhaps a binary)
  - Can be highly optimized for known patterns or systems
- Interpreters require layers of execution before results are seen
  - Interpret to some bytecode, possibly more steps
  - Overhead for access variables
  - All of this must be done on the fly - slowly!

Useful Definitions: Complexity

- Not comparing formally as something like: *O(n)*
- However… what's list access look like in Python?
  - Make sure the variable indexing the list is numeric
  - Determine if it's within bounds
  - If negative, do some wrap around magic
  - Sometimes even more!
- What does array access look like in something like C?
  - Read the memory based on some offset
  - … that's it (mostly)
- Often there is large complexity overhead that is abstracted away for you in Python

How Does Cython Work?

We give back some of our abstractions!

- Compilation
  - Sometimes we can take advantage of specific system optimizations too!
- Annotation of types
  - You'll be surprised how much this helps
- Complexity
  - Recall: what's in a list anyway? Shouldn't there be something better?

Live Demo

Some quick details

- Reference material is on GitHub here:
  - https://github.com/Andesha/sharcnet-cython
- We will be using Jupyter lab
  - The starting notebook and a completed notebook are on GitHub
- If we have time, we'll explore compiling code on Compute Canada systems
- Our test case will be a prime sieve!

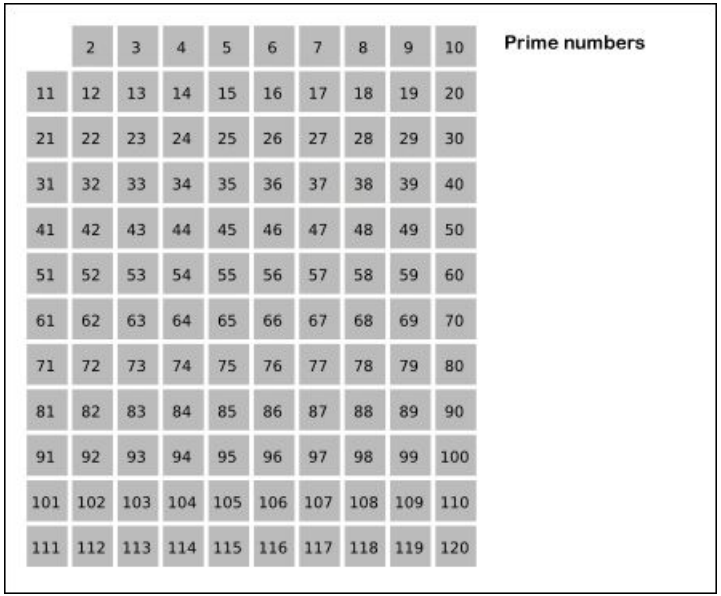
Practical Example: Prime Sieving

Canonical example is the Sieve of Eratosthenes

Example procedure:

1. Create a list of integers 2 -> N
2. Start at 2, all factors of it are marked in the list as non-prime (false)
3. Go to next true index
4. Mark all factors of it in the list as false
5. Go to step 3
6. All remaining true indices are prime numbers

Practical Example: Prime Sieving



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110
111	112	113	114	115	116	117	118	119	120

Taken from: https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

DEMO TIME - WHAT COULD GO WRONG? :)

Post Demo Discussion

Hopefully at this point, you are convinced!

Some external links for standard questions:

- Main documentation, here
  - This is where you find your type definitions and more
- Another Cython example using prime numbers, here
- Type memory views, here
  - Includes details on NxN arrays and different kinds of numpy interactions
- Compute Canada Python documentation, here
  - Remember our systems are slightly different!

Takeaways

- Python is super convenient, but sacrifices speed to get there
- We can recover a lot of this speed with Cython
- There's some awesome tools out there to help you profile your code
- Compiling for use on the Compute Canada systems is easy

Thanks very much!

Questions?