Case Study - Prime Finding

This case study runs a (terrible) algorithm to find prime numbers by finding factors by repeated division

For the C calls, you will need a C compiler (e.g. gcc) and to build the C code as a "shared object". A simple build script for gcc is provided.

- prime_finder_serial.py
    - A serial version of the prime finder

Info:

The main code is a simple loop over a range of target numbers (an interval [lower, higher], checking each number for primality.

We swap in and out the various checkers by a conditional import. A command line parameter of 0, 1 or 2 will use the primary options of pure python, jitted pure python and external C. You can also enable/disable jitting of the main loop by adjusting the b parameter on line 6 in the script before running/importing

Try:

Run the various options (0, 1, 2) . To use default lower and upper bounds, enter a non-integer at the prompts (e.g just hit enter)

Turn on/off the jit of the main loop and recompare the timings

(Hard) Any guesses what's going on? On my machine the timings don't make much sense, with the JIT slowing down, not speeding up, the code

- prime_finder_parallel.py

- An example of how one might parallelise a task like this using MPI

Info:

This code WORKS but does not gain performance because the individual tasks are too easy for the comms overhead in Python

Try:

If you have mpi4py you can run this variant. Try adding timing information to show where the time is being spent

Why is this so bad? (Hint - one answer lies in the cost of the checking - especially when it becomes trivial)

If you can think of a more suitable task for this sort of task-based parallelism, try changing this code to do that instead. You want a self-contained, but long running, task for the workers.