Assignment 7 - Cython

This assignment is a simple series of steps that you need to follow to optimize matrix multiplication using Cython.

You need to submit a single Python notebooks (that can be run on the Jupyter server) with code for the problem as described below. No PDF document is needed - it should be possible to use the "Kernel -> Restart Kernel and Run All Cells" menu option, and get back the final results. Do not include any results from your runs when submitting the code.

The notebook should be clearly documented - the first cell should contain an overview of the problem, followed by the steps solving each of the problem statements below. The last cell should contain a Markdown cell with your final observations and conclusions.

NOTE: The readability and clarity of your document will play a role in the final marks. Unnecessarily long explanations are strongly discouraged - please be clear about your experiments and conclusions.

Problem

In class we already discussed the matrix multiplication operation, and found that the naive matrix multiplication (shown below) is significantly slower than the optimized numpy implementation. The naive algorithm is given below:

```
import numpy as np
def matrix_multiply(u, v):
    m, n = u.shape
    n, p = v.shape
    res = np.zeros((m, p))
    for i in range(m):
        for j in range(p):
            for k in range(n):
                res[i,j] += u[i,k] * v[k,j]
    return res
```

The other alternative way of multiplying matrices is using `np.matmul(u, v)` or `u @ v` (same result, same code will be run in both cases). You can measure the time required for either approach using the `%timeit` Jupyter helper function as shown in class.

You need to do each of the following steps, and report the results.

Step 1

- Construct two `numpy` matrices of size 10x10 each - the entries should be random numbers.

- Use the `%timeit` special command to find the time required for multiplying the matrices.
- Multiply the same matrices using the `u @ v` or `np.matmul(u, v)` notation and measure the time required.
- Estimate the total number of multiplications required for this matrix computation. Assuming that the total number of floating point operations is twice the number of multiplications (because we also need to add the numbers) estimate the FLOPS - floating point operations per second - that has been achieved on the system. (*NOTE*: it is preferable to express this in GFLOPS and it should not have more than 2 or 3 significant digits as it is not a very accurate measurement anyway).

Step 2

Use the `lscpu` command to find the maximum operating frequency of the CPU on your system (you are free to run all this on your own system and report results there).

Based on this, and assuming that your CPU can perform either one addition or multiplication per clock cycle, estimate the maximum FLOPS that may be achievable using a single processor core. Comment on how well this compares against the numbers you found in Step 1.

Step 3

Repeat step 1, but doubling the matrix sizes for each run. Note that you should be careful to first estimate how much your runtime is likely to be by looking at the time for the last run before doubling. When we evaluate your notebook, if it hangs for a long time, that will be forcibly stopped and will count against your marks. If you run this on a different machine you should specify till where you were able to take these measurements. Also, you may need to provide `-r` and `-n` parameters to `%timeit` (refer to the documentation to see what these are) to keep the total runtime controlled.

Plot the measured times for the above code as well as the numpy matmul and comment on the results: do they follow an expected path?

You should be able to run np.matmul for much higher matrix sizes than the above code. How high are you able to go with this?

Plot the estimated GFLOPS from both approaches to matrix multiplication and compare against the theoretical estimates.

You should find that the numpy code is higher than the theoretical estimate. Explain your observations briefly.

Step 4

Load the Cython extension - refer to the sample documentation for how to go about doing this. Convert the matrix_multiply function to cython and time it for 50x50 matrices. Compare with the original.

Step 5

Apply each of the following transformations, measure the running time. Comment on the observations. Your final conclusions should include a clear indication of which step resulted in the best improvement, and your thoughts on why this is the case.

- Declare each of the variables `i, j, k, m, n, p` as **`int`** types
- Use the decorator function `@cython.boundscheck(False)`. What does this do?

- Declare the input variables to be of type **`double`**`[:, :]`
- Declaring the output variable type is not easy to do. Instead, declare `res` also to be an argument to the function, of the same **`double`**`[:,:]` type, and make sure that `res` is initialized to a zero array before calling.

- Change the data type to **`float`**`[:,:]` and repeat the experiments. Does this change anything?

Compare your best result against the time of `u @ v`. What are your conclusions?

Expected work

This is a very straightforward assignment. The main work is going to be in running the experiments and tabulating your observations. It should not take more than 2 hours at most and can most likely be finished in under 1 hour.

IMPORTANT:

Given the nature of this assignment, the grading will be mostly based on the document you submit. There is sufficient work here that your documents should be quite unique and reflect the work you individually have done, and you should not need to consult with anyone else except for very minor clarifications.

If submissions from different people are found to be copied, all involved will get a penalty as it is impossible to distinguish who did the original work and who copied. So do not share your code or document with others.

Also, there may be random vivas conducted to ask for explanations of how you wrote the code: each of you needs to be able to answer for the code and document you submit.