Matrix Multiplication: Iterative vs Thread Based Algorithms

About

This repository contains the source code needed to run a simple program that compares two approaches to matrix multiplication.

Running The Program

- Dependencies
1. Python3.6+
2. Git CLI tool
3. Python library: NumPy

Steps to run

1. Clone this repo to your local machine:
   - git clone https://github.com/hassamsolanomorel/CSIS612-Project1.git

2. Ensure you're local machine has python3.6 or above:
   - python --version' or 'python3 —version

3. Install numpy:
   - pip3 install --user numpy

4. CD to the root directory of the project:
   - cd path/to/project

5. Run the project by replacing 'INT_SIZE' with the size of the square matrix you would like to compute:
   - python3 matrixmultiply.py <INT_SIZE>

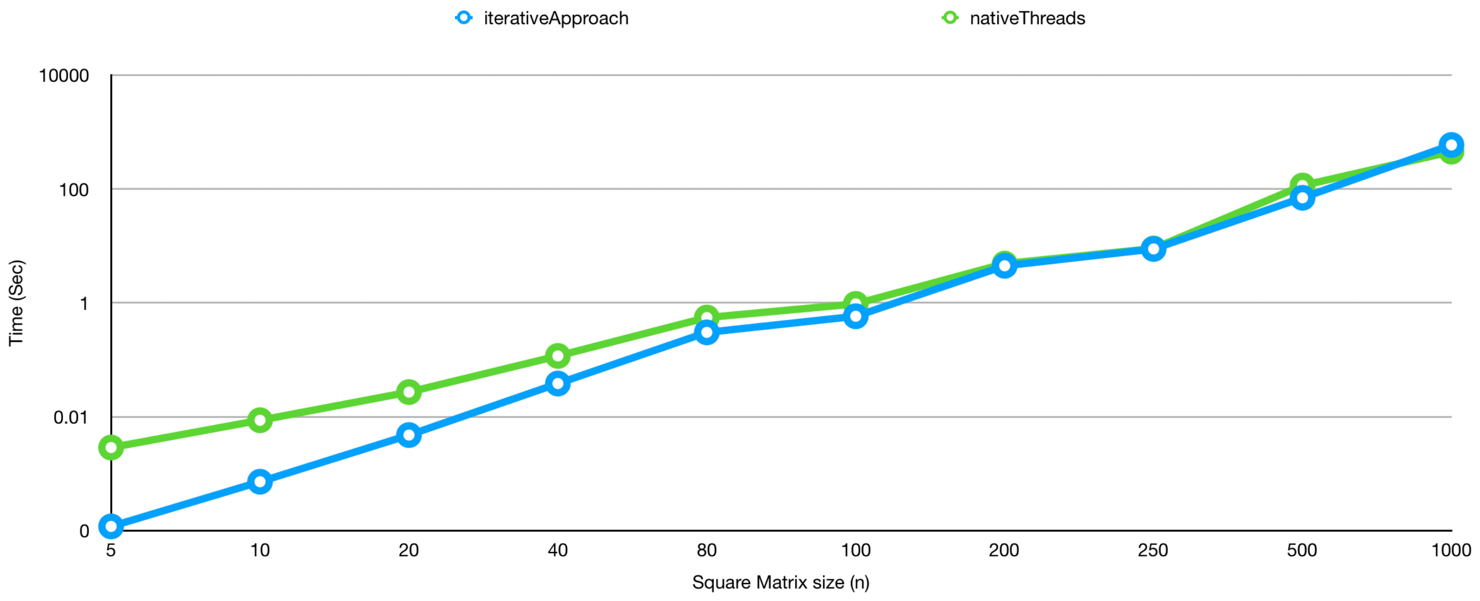
Results

The program was tested by running it against an increasing INT_SIZE. The values passed in and calculated times are presented below.

Time To Complete Multiplication

Matrix Size (n x n)	iterativeApproach(secs)	nativeThreads(secs)
5	0.00011802	0.00286794
10	0.00071907	0.00865197
20	0.00475192	0.02702904
40	0.03820086	0.11665988
80	0.30257416	0.55097771
100	0.58022404	0.95731878
200	4.46881604	4.88616800
250	8.84446096	9.00189185
500	69.97950411	114.93207502
1000	592.63208199	452.60761786

A graphic representation of the results are presented below.



Discussion

The results show that the iterative approach actually out performs the threaded solution up to a matrix size beyond ~200. These results were not expected and encouraged a deeper dive to understand these results.

Deeper Dive

While python is a modern language, its creation actually pre-dates the implementation of threading. For this reason, the python language supports native threads but they do not function like threads in other languages such as C. In python every process is run using an interpreter. One of the many features in this interpreter is called the global interpreter lock or GIL as its better known in the community. The GIL is a feature implemented to help avoid concurrency issues within the python language. While an entire book could (and has been) dedicated to the GIL, the basic idea behind this feature is to lock the interpreter into running only one thread at a time as a method of avoiding object access conflicts. In practice this means that using python's native "threading" library will in effect still create single threaded applications.

This understanding of python's GIL feature helps us explain the results above. In the results we see that the iterative approach (true single-thread algorithm) actually out performs the multi-threaded approach (pseudo-threading algorithm) up to a matrix size of ~200x200. This difference in performance is likely explained by the overhead required to setup the "threads". We do observe the native threads approach beginning to outperform the iterative approach after this point. This likely indicates that the small performance gains of the multi-threaded approach begin to over take the overhead required to set up the threads around a matrix size of ~200x200.