**Parallel programming in Python:**

**What is the most efficient way to perform CPU-intensive multiprocessing operations?**

***A research proposal submitted by:  
Thomas Hickey***

***Proposed Supervisor:   
Dr Theodore Ward***

**Northeastern State University**

**January 2021**

**TABLE OF CONTENTS**

[1 INTRODUCTION 3](#_Toc192476462)

[1.1 Purpose of the study 3](#_Toc192476463)

[1.2 Context of the study 3](#_Toc192476464)

[1.3 Problem statement 4](#_Toc192476465)

[1.3.1 Main problem 4](#_Toc192476466)

[1.3.2 Sub-problems 4](#_Toc192476467)

[1.4 Significance of the study 4](#_Toc192476468)

[1.5 Delimitations of the study 4](#_Toc192476469)

[1.6 Assumptions 5](#_Toc192476471)

[2 Research methodology 6](#_Toc192476486)

[2.1 Research methodology / paradigm 6](#_Toc192476487)

[1.3.1 Main problem 6](#_Toc192476466)

[1.3.2 Sub-problems 6](#_Toc192476467)

[2.2 Research Design 6](#_Toc192476488)

[2.3 Procedure for data collection 7](#_Toc192476493)

[2.4 Data analysis and interpretation 7](#_Toc192476494)

[3 conclusion 8](#_Toc192476486)

# CHAPTER 1: INTRODUCTION

## Purpose of the study

The purpose of this research is to determine which method is the most efficient way to move data between the main process and subprocesses when utilizing the multiprocessing module in Python.

## Context of the study

With today’s computer systems containing CPU chipsets that have multiple processors a programmer has almost unlimited ways to run multiple processes concurrently. When using Python, the multiprocessing package allows the programmer to create subprocesses that can run in parallel with the main process. There are several different mechanisms available that can be used to move data from the main process to the subprocesses and back again, and each of these have a different methodology for implementation.

The main problem that I intend to address with this research is to determine which of three mechanisms available to a programmer for this purpose show the highest efficiency, as measured by overall processing time, when the subprocesses themselves are performing very CPU-intensive tasks.

## Problem statement

### Measuring the efficiency of different methods for data transfer available in the multiprocessing package in Python.

### Sub-problems

The first sub-problem is determining if using multiple simplex pipes is a more efficient method of moving data between subprocesses than using duplex pipes.

The second sub-problemis determining if the most efficient version of pipes that is found from examining the first sub-problem is a more efficient method of moving data between subprocesses than using the multiprocessing pool module.

## Significance of the study

The study fills a gap in that although there is plenty of documentation available to demonstrate how to implement these different methods of using the multiprocessing package in Python, there is almost no documentation to show what the most efficient choice would be for various situations.

The study will provide guidance to programmers who are trying to decide which implementation of the multiprocessing module to use when coding in parallel using Python.

## Delimitations of the study

The primary delimitation of this study will be that it will focus entirely on situations where subprocesses are running CPU-intensive tasks, and will only include two different tasks that are designed to test the efficiency of the channels between the main process and the subprocesses. Unfortunately, no two tasks could possibly encompass the entire scope of possible uses of the multiprocessing model.

Another delimitation will be that averages will have to be obtained over multiple subprocess iterations, which may serve to cover some situational differences that could otherwise be explored. I will attempt to offset this possibility as much as possible by making a sufficient number of runs through the task cycles to be representative of a true average.

The final delimitation that should be noted is that I only have access to one PC to use for testing this software, so there will be no way to determine with absolute certainty how much effect this will have on the efficiency of these different algorithms and what sort of results could be obtained using a different operating system or hardware setup.

## Assumptions

* The chosen CPU-intensive tasks will accurately represent a reasonable average of all CPU-intensive tasks that could be called upon to be performed with the multiprocessing module in Python.
* By averaging many different runs of the software algorithm any possible inconsistencies that could arise from operating system demands or other unpredictable background CPU usage will be accounted for.
* That my PC will demonstrate a realistic average performance that will remain reasonably accurate when scaled up or down to match which of the almost unlimited number and combinations of hardware and software are being used by another programmer.
* That the relative efficiency of each method of moving data between the main process and the subprocesses can be evaluated using the processing time required for each method performing the same task.

# CHAPTER 2: Research methodology

This section will outline a research hypothesis and break down the general design for a program that will examine the problem statement and test the research hypothesis.

## Research methodology / paradigm

This will be a quantitative experiment designed to perform CPU-intensive tasks in parallel using the multiprocessing module in Python. The research hypothesis in this case is the following:

### The simpler a solution in the module is in functionality the more efficient it will be.

### The simplest, and therefore most efficient, solution in the module would be simplex pipes, followed by duplex pipes. The most complex, and therefore least efficient, solution available in the module would be using a multiprocessing pool.

## Research Design

The design of the program used to test my theory is that there will be definitions in place to perform the following tasks:

* Simple encryption on a set of numbers using the SHA-256 hash function
* Generate a list of prime numbers from a set of numbers

The main part of this program will call these functions simultaneously and perform them using separate child processes. In one part of the main program it will be set to communicate the results back and forth from the subprocesses with simplex pipes, one part with duplex pipes, and then finally with the multiprocessing pool. In all cases these functions will be run in parallel using the same size sets of data. It will track the amount of time it takes to perform these functions each way and will output these results into a separate file so that the data can be processed and interpreted.

## Procedure for data collection

The program itself will capture all the data necessary to evaluate the efficiency of each algorithm by timing how long it takes to complete each cycle of each algorithm and storing this information for review.

## Data analysis and interpretation

After the data is stored it can be retrieved and processed into overall averages for each different iteration of the multiprocessing module. These averages can be graphed against each other and reviewed using independent software platforms, and if the hypothesis holds true then each method of using the multiprocessing module should show quantifiably different results in processing time. Since all the other factors have been equalized as much as possible, it can be reasonably assumed that the lower the processing time for each method the more efficient that method is at moving the data from the main method to the subprocesses and back again.

# Conclusion

When this experiment is concluded it should be possible to state unequivocally whether or not there is a statistically significant difference in efficiency as measured by processing time when using different implementations of parallel multiprocessing in Python.

Three different methods of moving data between subprocesses in Python will be examined in this experiment:

* Simplex pipes
* Duplex pipes
* Multiprocessing pools

The hypothesis of this experiment is that the determining factor will be the overhead required to operate each of the three different implementations of multiprocessing being tested. This overhead will determine which is most efficient when performing CPU-intensive tasks.

If this hypothesis is accurate, then the most efficient method of implementing multiprocessing would be simplex pipes, followed by duplex pipes, followed by using a multiprocessing pool. This would also mean that there is an inverse relationship between the amount of time and effort that goes into coding for multiprocessing in Python and the efficiency of the resulting code.

Should all that prove to be true, the recommendation of this study would be that parallel programs that are designed to accomplish CPU-intensive tasks should be set up to use simplex pipes if efficiency and processing time are a top-level concern in the run-time environment. If processing time is a concern, but not a critical one, then duplex pipes would be an adequate solution and would save some time in coding. If processing time is a low-level concern or not a concern at all then a multiprocessing pool will save a lot of time and effort in coding.