Manual for Parallel Computing in Python

Introduction

With the rapid increase of multicore processors, each core with its own supply of threads, the ability to parallelize code has become a necessary tool to any aspiring developer. This manual seeks to convey the knowledge of how to utilize parallel programming using the standard Python 3.9 modules. Parallel computing or parallel programming is a type of computing architecture in which several processors execute an application or computation simultaneously. (1) This "parallelizing" of the code allows for more complex applications and/or a greater amount of computational power than can be accomplished with a single thread of code execution.

Commented [LB1]: Hello! I am Lilac and I will be looking over your rough draft today. You mention that you want to make sure that your writing is professional, and you want to achieve clarity, improve style/wording, & focus on grammar/punctuation. I am happy to look over this draft and give you feedback!

Commented [LB2]: Sentences (and clauses) that begin with *with* require a comma in the middle of the sentence. To strengthen this sentence, simply delete "with", slightly alter the main clause and present it as a parenthetical phrase, and close the sentence with a verb to transform what was originally a subordinate clause into the main clause:
"The ability to parallelize code has become a necessary tool to any aspiring developer due to…"

Commented [LB3]: I'm not sure what this (1) means.

Commented [LB4R3]: If it is a citation, it is best to avoid using citations in your first paragraph. You want the reader to know that the information you're giving is your own, using citations/research as support to your own point.

SECTION 2: DETAIL ABOUT THREADS, PROCESSES. WHO IS INTENDED AUDIENCE

Intended Audience

This manual assumes that the reader has prior programming experience and a familiarity with Python. Having said this, there are some topics that should be summarized before beginning with the procedure. To reiterate, parallel computing is a type of programming that utilizes multiple sequences of code execution simultaneously to perform more machine instructions. In regards to this text, a thread is a single sequence of instructions that can be managed independently. The majority of projects that a beginning programmer studies are done with a single thread and can be considered serial programs. In addition to this, a process is an instance that is being executed by one or more threads. Multi-processing is, as its name implies, the ability to access multiple processes each with their own supply of threads to execute instructions within an application. With this knowledge, the progression from lowest complexity to highest for parallel programming is: serial process, multi-threaded process, multi-process application.

NOTE: (Add section about GIL here?)

SECTION 3: PROCEDURE FOR PARALLELIZING CODE

Preface

This section will go into detail on a general approach to parallelizing an I/O bound or CPU bound process using Python and its standard modules. The examples used may have modules that are not included with the current version of Python (Python 3.9.0) used by this manual. This text will include one example of an I/O bound process and one example of a CPU bound process.

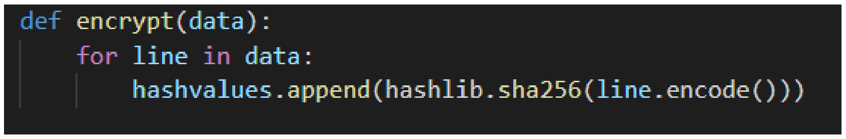
1. To begin parallelizing I/O and CPU bound programs, you must first determine the “driver” function. This “driver” function is the function that will be performing the specific instructions that we want to parallelize. This function typically will be called many times by the program that you are working with.
2. Once you have determined the driver function, you must then process the input that is being fed to the function. You must partition the data appropriately in order to parallelize the code correctly. The easiest way to accomplish this is to create a list object with all of your data stored inside.
3. After the list object is created, we can then begin operating over the list object with a simple for loop which will pass this data into the appropriate thread or process object which will parallelize the program.

Commented [LB5]: This section you use the word “you” a lot. Think about instruction manuals you have read. Do they say “Take your screwdriver and you’ll need to screw a fastener to your bed frame”? Rather, manuals say “Screw fastener to the bed frame”.

Replace instances of "you" in your essay either by using "individual" or "one" to refer to a single hypothetical person and using "people" to refer to a large group to whom something you're saying applies. Or rearrange your sentence to avoid using “you”. For example, your first sentence can be rephrased to say “Determine the “driver” function to begin parallelizing I/O and CPU bound programs.”

Example of Multithreading (CPU bound process):

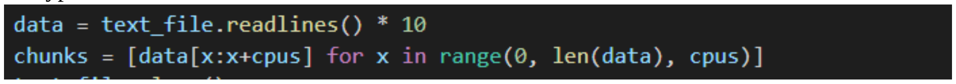
This program’s purpose is to encrypt a list of strings.



```python
def encrypt(data):
    for line in data:
        hashvalues.append(hashlib.sha256(line.encode()))
```

The list of strings is formed from a text file located within the same directory as the script. This list must then be split into a list of chunks. Each chunk can be fed to the appropriate thread for encryption.

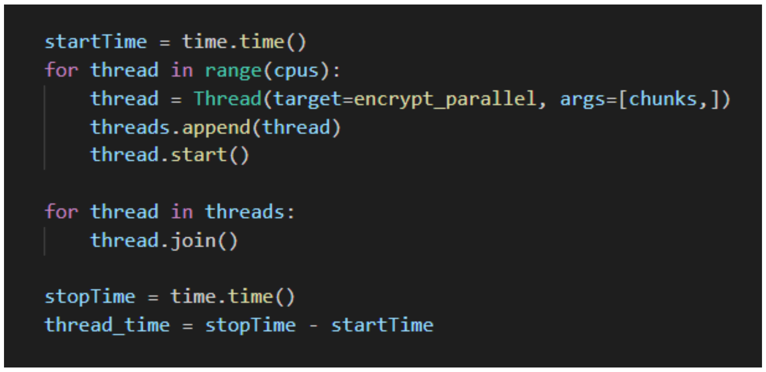
Commented [LB6]: Is chunk the proper term for this? I’m unfamiliar with the lingo here, but I would probably use the word “section” over “chunk” if it doesn’t change the meaning.



```python
data = text_file.readlines() * 10
chunks = [data[x:x+cpus] for x in range(0, len(data), cpus)]
```

Once the data is split into chunks, we can begin feeding it to the threads. The threads are stored into a list and then are then iterated over using a for loop as shown below. In the context of this program, the number of threads is equal to the number of cores on the CPU.

Commented [LB7]: Generally, we don’t use the term “we” in academic papers, just like I suggested you avoid “you” terms as well.

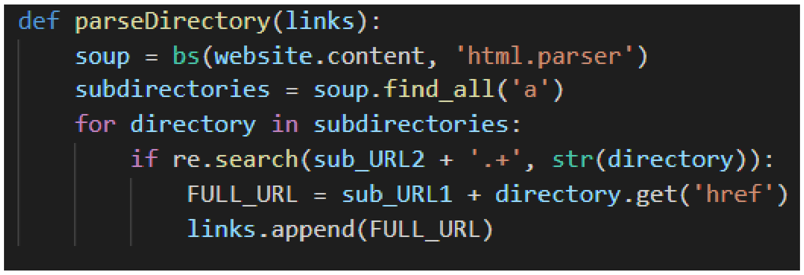


```python
startTime = time.time()
for thread in range(cpus):
    thread = Thread(target=encrypt_parallel, args=[chunks,])
    threads.append(thread)
    thread.start()

for thread in threads:
    thread.join()

stopTime = time.time()
thread_time = stopTime - startTime
```

Example of Multiprocessing (I/O bound process):

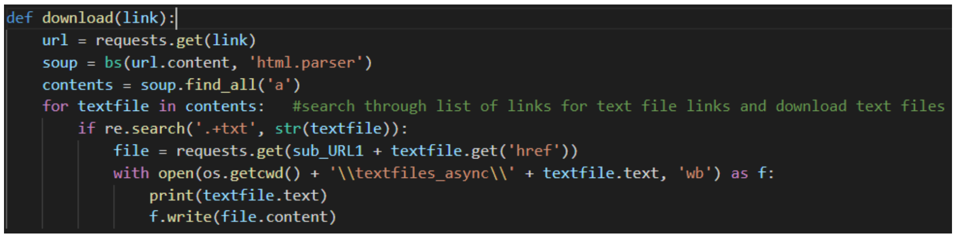
This program's purpose is to download a series of text files from the website https://www.sec.gov/Archives/edgar/data/20. It accomplishes this by parsing through the HTML code of the given URL and building a list of string objects named "links" which is filled with the URLs of all of the subdirectories contained within. This is implemented by a function named "parseDirectory" which takes a list for its single argument.

Commented [LB8]: Do you need to have the exact link here? Could you say "This program's purpose is to download a series of text files from U.S. Securities and Exchange Commission"?



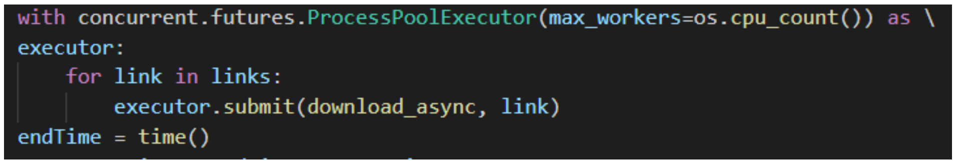
```python
def parseDirectory(links):
    soup = bs(website.content, 'html.parser')
    subdirectories = soup.find_all('a')
    for directory in subdirectories:
        if re.search(sub_URL2 + '.+', str(directory)):
            FULL_URL = sub_URL1 + directory.get('href')
            links.append(FULL_URL)
```

This list of links is then passed into the driver function named "download".



```python
def download(link):
    url = requests.get(link)
    soup = bs(url.content, 'html.parser')
    contents = soup.find_all('a')
    for textfile in contents:   #search through list of links for text file links and download text files
        if re.search('.+txt', str(textfile)):
            file = requests.get(sub_URL1 + textfile.get('href'))
            with open(os.getcwd() + '\\textfiles_async\\' + textfile.text, 'wb') as f:
                print(textfile.text)
                f.write(file.content)
```

This function takes in a singular link (string) as its argument in order to coordinate with Python's process pool executor.



```python
with concurrent.futures.ProcessPoolExecutor(max_workers=os.cpu_count()) as \
executor:
    for link in links:
        executor.submit(download_async, link)
endTime = time()
```

SECTION 4: CONCLUSION

Parallel computing, when done correctly, can be a gamechanger for any developer looking to add to their skillset. This manual covered only a fraction of potential methods that can be used for parallelizing the Python code. For any additional details regarding some of the material used in the manual, please see the attached appendix.