Lecture 19 (2-27-26)

Parallel Computing

Motivating Questions

1. What is the difference between **concurrency** and **parallelism?**
2. What is **data parallelism?**
3. What is **task parallelism?**
4. Are **more cores** always better?

Concurrency and Parallelism:

- **Concurrency:**
  - **composition** of independent computations
  - Concerned about **structure** (division of labor)
  - "how do I break up a problem in different tasks?"
- **Parallelism:**
  - **simultaneous** execution of (possibly related) computations
  - Concerned with **execution (with hardware resources)**
- **Concurrency** provides a way to structure a solution to solve a problem that may (but not necessarily) be **parallelizable**
- You can have **concurrency** without **parallelism**
- Very rarely do you see **parallelism** without **concurrency**

- **More parallelism** or **more concurrency** isn't always a good thing
  - Bui demonstrated this concept by asking athlete and a non athlete to come up to the front
  - He tells both of them to grab the microphone (there's only one microphone)
    - this is a **race condition:** multiple tasks compete for the same resource in an unpredictable manner
  - Now he says suppose in order to talk one person needs the water bottle and the microphone
    - but the two volunteers grab one or the other (no person has both)
    - This is a **deadlock:** multiple tasks are stuck waiting for each other

- **race conditions and deadlock** are common pitfalls of concurrency

---

- We are going to bypass this by having little to no communication between our tasks
  - **Embarrassingly Parallel:**
    - Some problems are so **naturally or obviously parallel,** that is exhibit natural **concurrency,** and we label them as **embarrassingly parallel**
      - Little or no **dependency** between tasks
      - Little or no need to **communicate** between tasks
        -Note: the GPU is a parallel supercomputer that puts graphics on the screen
- Walking through HULK (homework assignment)
  - first step is generating permutations given the alphabet and the length
    - you will have 36^8 different permutation on the homework
      - so you **do not want to store these in some data structure**
    - this is why we will use a generator to produce one by one
  - Then we move to the second step: crack
    - computing the sha1sum of the permutation
    - and see if its in the existing collections
      - if its in the existing collection then you've found the password
    - remember to use a set so that you use constant time look up
    - This step is actually one you can do in **parallel**
      - bui demonstrates this by having one person generate permutations:
        - but this person hands off different passwords to three different people
        - now three passwords can be checked if they match at the same time
          - there is no need for these three people to talk to each other
  - The last stage : print once you have found a match

Data parallelism

- **Functional programming** maps well to problems that exhibit **data parallelism**
  -**concurrent execution** of the **same task** across the elements of a **dataset**
- In such situations, the **dataset** normally exhibits **data independence,** meaning each element can be processed **independent** of the other

---

Task Parallelism

- **Generators** support problems that exhibit **task parallelism**
  - **concurrent execution** of the **different tasks** on same or different **datasets**
- In such situations, the **different tasks** usually must **communicate** and **coordinate** with each other. The **concurrent execution** of such **tasks** are **interleaved** throughout the running of the application.

`slurp.py`

```
#!/usr/bin/env python3

import concurrent.futures
import os
import re
import sys

import requests

# Functions

def wget(url):
    p = os.path.basename(url)           # Review: os.path.basename

    print('Downloading {} to {}'.format(url, p))
    r = requests.get(url)
    with open(p, 'wb') as fs:           # Review: Writing to a file, with
statement
        fs.write(r.content)

    return p

def flatten(sequence):
    for iterable in sequence:
        yield from iterable

# Main Execution

def main():
    # Download images from Flickr
    # https://www.flickr.com/photos/indianafirst/albums/72177720332075049/
    #
https://www.flickr.com/photos/indianafirst/albums/72177720332075049/page2
    #
```

```
https://www.flickr.com/photos/indianafirst/albums/72177720332075049/page3
    pages  = (requests.get(argument) for argument in sys.argv[1:])
    assets = flatten(
        re.findall(r'src="//(.*jpg)"', page.text) for page in pages
    )
    urls   = ('https://' + asset for asset in assets)

    # Sequential
    list(map(wget, urls))               # Discuss: Why list?

    # Parallel                          # Discuss: concurrent.futures
    '''
    with concurrent.futures.ProcessPoolExecutor() as executor:
        executor.map(wget, urls)        # Discuss: Timing
    '''

if __name__ == '__main__':
    main()
```