02613 Python and High Performance Computing

Exercises for Week 6

---

1. Mean Faces

In this exercise we will use the CelebA dataset of celebrity images as our data source. Examples are shown above.

You can find it at: `/dtu/projects/02613_2025/data/celeba/`. We have also created smaller subsets for fast testing. To begin with use the 'celeba_200.npy' file for testing and then work your way up to at least the 'celeba_100K.npy' version.

Each image is 128×128 RGB pixels. They are stored in an $N \times 128 \times 128 \times 3$ array, such that `array[i]` is a $128 \times 128 \times 3$ image. The goal is to compute the mean (i.e., average) face by summing all the images and then dividing by the number of images. Since CelebA contains over 200,0000 images, we want the sum to be parallel, i.e., we want a parallel reduction.

In order to get good performance, we need to minimize data transferred between processes. We can do this by creating an array of shared memory and then performing the reduction in place. The template below sets up a shared array and performs a 'dummy' operation. Adapt it for this exercise.

```python
import ctypes
import multiprocessing as mp
import sys
from time import perf_counter as time
import numpy as np
from PIL import Image


def init(shared_arr_):
    global shared_arr
    shared_arr = shared_arr_
```

```python


def tonumpyarray(mp_arr):
    return np.frombuffer(mp_arr, dtype='float32')


def reduce_step(args):
    b, e, s, elemshape = args
    arr = tonumpyarray(shared_arr).reshape((-1,) + elemshape)
    # Change the code below to compute a step of the reduction
    # ---------------------------8<---------------------------
    arr[b:e:s] = 1.0 - arr[b:e:s]  # <-- Dummy op. Replace with correct


if __name__ == '__main__':
    n_processes = 1
    chunk = 2

    # Create shared array
    data = np.load(sys.argv[1])
    elemshape = data.shape[1:]
    shared_arr = mp.RawArray(ctypes.c_float, data.size)
    arr = tonumpyarray(shared_arr).reshape(data.shape)
    np.copyto(arr, data)
    del data

    # Run parallel sum
    t = time()
    pool = mp.Pool(n_processes, initializer=init, initargs=(shared_arr
        ,))

    # Change the code below to compute a step of the reduction
    # ---------------------------8<---------------------------
    pool.map(reduce_step,
             [(i, i + chunk, 1, elemshape) for i in range(0, len(arr),
                 chunk)],
             chunksize=1)

    # Write output
    print(time() - t)
    final_image = arr[0]
    # final_image /= len(arr) # For mean
    Image.fromarray(
        (255 * final_image.astype(float)).astype('uint8')
    ).save('result.png')
```

1. **Autolab** Modify the provided example to compute the first step of a parallel reduction, i.e., every second element is summed with its neighbor. For example, after this step, an input of $[1, 2, 3, 4]$ should be $[1 + 2, 2, 3 + 4, 4] = [3, 2, 7, 4]$.
   Hint: for testing, make some dummy data like:

```python
import numpy as np
arr = np.arange(10)  # 0 to 9
arr = arr.astype('float32')
arr = arr[:, None, None, None]  # (10, 1, 1, 1)
np.save('dummydata.npy', arr)
```

Then, comment out the lines saving the image and instead just print the array to manually inspect the results.

2. **Autolab** Further modify the example to compute the full reduction. Remember to divide by the number images at the end to compute the mean. Hint: each step of the reduction should be a separate call to \texttt{pool.map}.
3. Play with the chunk size (the `chunk` variable in the code). What value gives the best performance for you? Note, this will depend on the size of the subset you use, so make sure to try the program for at least the 100,000 subset.
4. Run your program as a batch job for varying number of processes and create a speedup plot. Run the program for at least the 100,000 subset. For consistent results, you should run the program 3 times and use the average time. What do you see? Is anything surprising? Hint: **this will take time** for the 100K subset or above. While you wait, start the next exercise.
5. **Autolab** Repeat the previous exercise, but run Python under `numactl --interleave=all`. What changed? Hint: see lecture slides for examples with `numactl`.
6. Compare your fastest runtime with `np.sum`. Do you manage to compute the sum faster? If so, by how much?