• HW1 Problem 5
  • Connected components will be extra credit.
• Get Cython installed & working for next HW.
Multiprocessing, Multithreading, & Communication costs
https://github.com/thouis/mandelbrot-parallel
Serial

Just call the mandelbrot on the array of coordinates:

```python
mandelbrot.mandelbrot(in_coords, out_counts,
max_iterations=1024)
```

mandelbrot_single.py
Serial
(1 out of 4 processors)

Single - no serialization / communication
200 M complex multiply-adds / second
Serial
(1 out of 4 processors)

Single - no serialization / communication
200 M complex multiply-adds / second
Multiprocessing

Dice up into blocks, communicate them to worker processes, read back results.

```python
in_blocks = by_blocks(in.coords, 10)  # 10 blocks / side
tmp_blocks = pool.map(wrap_mandelbrot, in_blocks)
for src, dest in zip(tmp_blocks,
                     by_blocks(out_counts, 10)):
    dest[...] = src
```

mandelbrot_multi.py
4 processes

Multiproc (v1.0) - serialize blocks
490 M cmas / second

Single - no serialization / communication
200 M complex multiply-adds / second
4 processes

Goal on 4-core machine is 800M cmas / second

Multiproc (v1.0) - serialize blocks
490 M cmas / second

Single - no serialization / communication
200 M complex multiply-adds / second
Multiprocessing - Shared memory & slices

Share memory, serialize slice coordinates.
Slices?

- $A[0:10, 0:10]$ - slice of $A$
  - Uses **same memory** as $A$.
- We can create slices outside of arrays with
  
  $\text{slice}(\text{start, end, step})$
  
  $[:10, :10] \rightarrow (\text{slice}(0, 10, 1), \text{slice}(0, 10, 1))$
in_coords = make_shared(in_coords)
out_counts = make_shared(out_counts)

def wrap_mandelbrot(ij_slice, iterations=1024):
    mandelbrot.mandelbrot(in_coords[ij_slice],
                         out_counts[ij_slice],
                         iterations)

slices = by_block_slices(in_coords, 10)
pool.map(wrap_mandelbrot, slices)
Multiprocessing - Shared memory & slices

```python
in_coords = make_shared(in_coords)
out_counts = make_shared(out_counts)

def wrap_mandelbrot(ij_slice, iterations=1024):
    mandelbrot.mandelbrot(in_coords[ij_slice],
                          out_counts[ij_slice],
                          iterations)

slices = by_block_slices(in_coords, 10)
pool.map(wrap_mandelbrot, slices)
```

mandelbrot_shared_blocks.py
4 processes, shared memory

Shared Blocks (v2.0) - serialize slices
550M cmas / second

Multiproc (v1.0) - serialize blocks
490 M cmas / second

Single - no serialization / communication
200 M complex multiply-adds / second
Memory access patterns

• How is memory accessed when working by blocks?

• What happens at the edge of a block?

• How does this interact with per-core caches?
Multiprocessing v3

- Use shared memory for input and output arrays.
- Instead of slicing blocks, slice rows.
- Why rows?
in_coords, out_counts = make_coords()
in_coords = make_shared(in_coords)
out_counts = make_shared(out_counts)

...

slices = by_rows(in_coords, 10)
pool.map(wrap_mandelbrot, slices)
in_coords, out_counts = make_coords()
in_coords = make_shared(in_coords)
out_counts = make_shared(out_counts)

... 

slices = by_rows(in_coords, 10)
pool.map(wrap_mandelbrot, slices)
Shared memory, by rows

mandelbrot_multi_rows.py
Shared memory, by rows

Shared Blocks (v3.0) - serialized rows
620 cmas / second

Shared Blocks (v2.0) - serialize slices
550M cmas / second

Multiproc (v1.0) - serialize blocks
490 M cmas / second

Single - no serialization / communication
200 M complex multiply-adds / second
Multithreading

- Let’s not get into the details, right now.
- replace Pool(4) with ThreadPool(4)
- Same slicing strategy.
Multiprocess vs. Threading

Threading - serialized rows
620 cmas / second

Shared Blocks (v3.0) - serialized rows
620 cmas / second
Remove serialization

Give Nth thread every Nth line:

`slices = [slice(threadidx, in_coords.shape[0], 4)
          for threadidx in range(4)]`

Only 4 slices —> serialization time goes away.
Without Serialization
Without Serialization

Threading - no serialization
720 cmas / second

Threading - serialized rows
620 cmas / second

Shared Blocks (v3.0) - serialized rows
620 cmas / second
Questions for you

How much difference does Multiprocessing vs. Threading make?

Is there a better way to slice the array into 4 parts?
Four horizontal bands?
Four vertical bands?

Post on Piazza, discuss Wednesday.