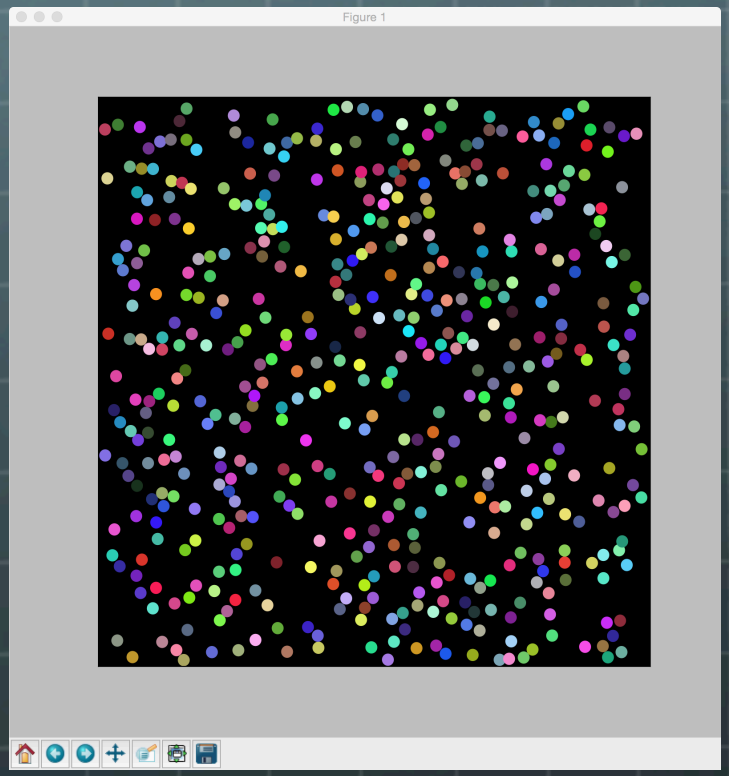
Improving the performance of a physics simulator

Kendrick Lo
AC298R

Background

- physics simulator and animation system
  - 10,000 “balls” in a $[0, 1]^2$ space
  - written in Python
- objective is to improve the performance of this code
  - parallel techniques
  - algorithmic improvements



Methodology

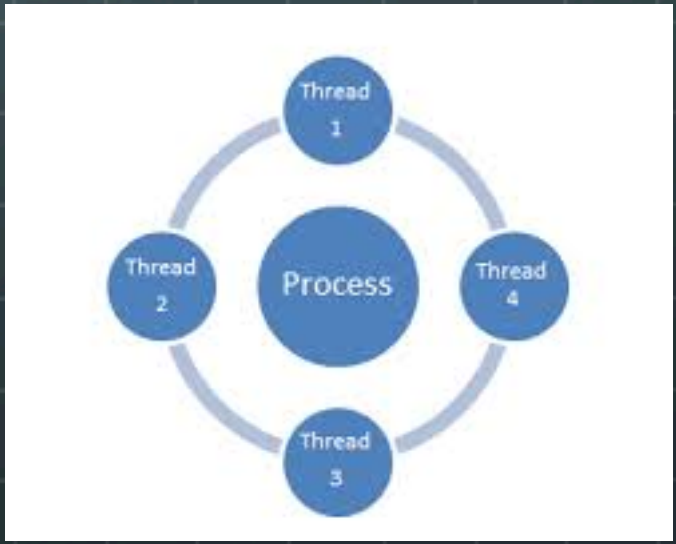
- Four techniques employed to improve on the code
  - multi-threading
  - spatial decomposition ("gridding")
  - spatially coherent sorting
  - locking
- Baseline for comparison:
  - 14 simulation *Frames per second (FPS)*

Existing code structure

- *driver.py*
  - calls function to update the animation display
  - keeps track of simulated "time"
- *physics.pyx*
  - written in Cython
  - checks for boundary conditions (walls) and collisions
  - returns updated positions

Multi-threading

- execute multiple threads concurrently
- similar to 3D projection where pixels can be rendered independently – embarrassingly parallel
- split work (boundaries, collisions, updates)
  - *Cython: prange()*
- 4 threads:
  - 20 FPS (43% increase)



ball index	x	y
0	0.21	0.15
1	0.92	0.13
2	0.36	0.31
3	0.92	0.41
4	0.95	0.51
5	0.12	0.14
6	0.31	0.14
7	0.65	0.49
…	…	…

Collision detection

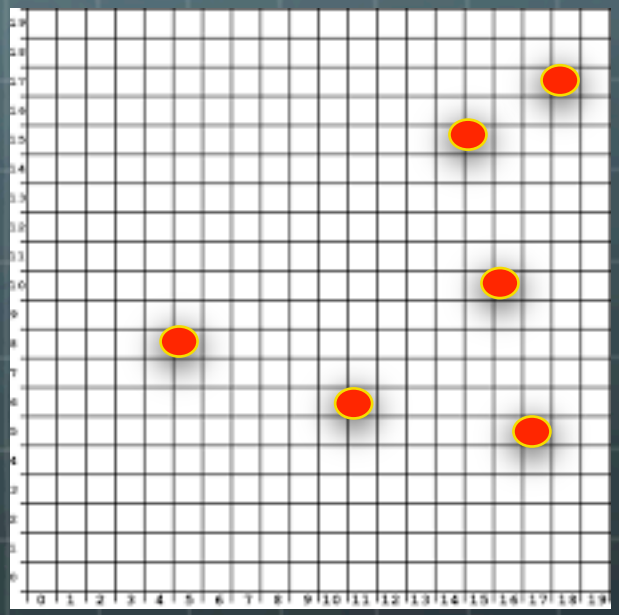
- array stores positions of each ball (x, y) by ball index
- collision detecting algorithm involves two loops:
  - outer loop iterates through all elements
  - inner loop compares each element to higher-indexed elements
- $O(N^2)$ algorithm
  - N = 10 -> ~100k comparisons
  - N = 100 -> ~10,000k comparisons
  - N = 1000 -> ~ 1M comparisons
  - N = 10000 -> …

Spatial Decomposition

- gridding
  - for each position, store the ball index
  - most grid elements are empty
  - check for neighbours (24)
  - $O(N^2) \rightarrow O(N)$!
- 1 thread…? (baseline: 14 FPS)

  734 FPS… 52x faster
- 4 threads

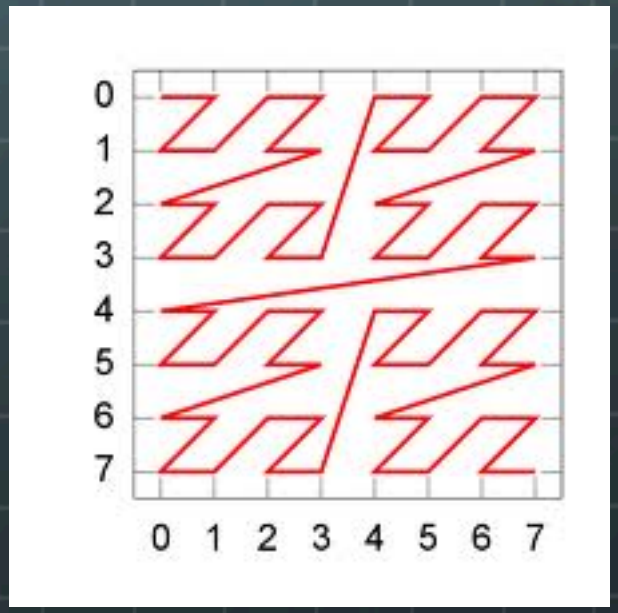
  1340 FPS… ~100x faster!



ball index	x	y
0	0.21	0.15
1	0.92	0.13
2	0.36	0.31
3	0.92	0.41
4	0.95	0.51
5	0.12	0.14
6	0.31	0.14
7	0.65	0.49
…	…	…

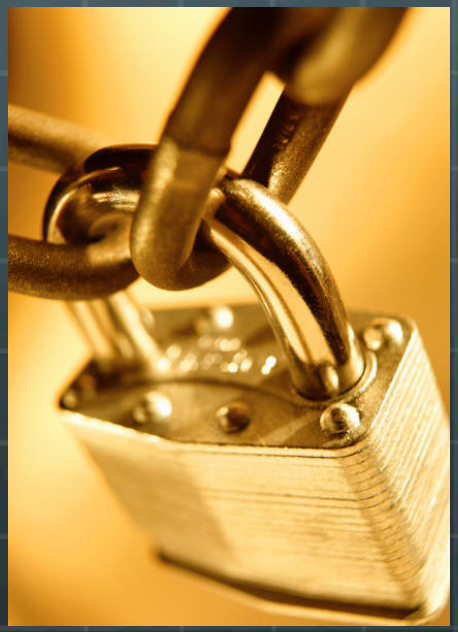
Spatially Coherent Sorting

- spatial coherence:
  objects nearby in space should be nearby in memory
- sort positions within array so that close balls have array indices close in value
- naive algorithm…?
- Morton ordering
  - 1 thread: 894 FPS (vs. 734)
  - 4 threads: 1667 FPS (vs. 1340)

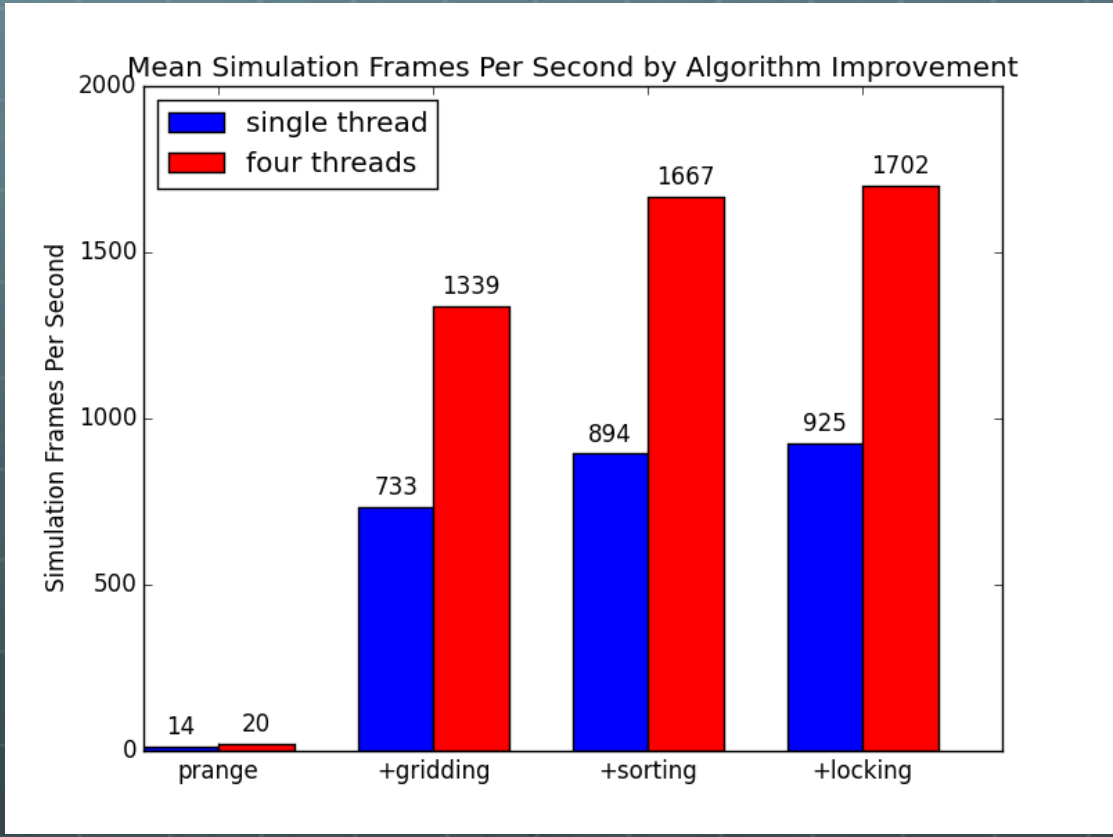


Locking

- *lock*: programming language construct that allows one thread to prevent other threads from accessing something shared
- in our simulation, we want to prevent simultaneous updates to object velocities during collision checking
- usually there is overhead associated with locking, but …
  - 1 thread: 925 FPS (vs. 894)
  - 4 threads: 1702 FPS (vs. 1667)
- performance increased… why?



Summary



- We achieved a 122x improvement over the baseline using a variety of techniques
- Parallel vs. algorithmic improvements
- Takeaway:
  - THINK!