ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Spring 2012

Encapsulating the built-in python RNG

an example implementation, in `Mersenne.py`

```
1  import random
2  from PointCloud import PointCloud
3
4  class Mersenne(PointCloud):
5      """
6      A point generator implemented using the Mersenne Twister random number
7      generator that is available as part of the python standard library
8      """
9
10     # interface
11     def point(self, box):
12         """
13         Generate a random point in the interior of {box}
14         """
15         # unpack the bounding box
16         tail, head = box
17         intervals = tuple(zip(tail, head))
18         # build the point p by caling random the right number of times
19         p = [ random.uniform(left, right) for left, right in intervals ]
20         # and return it
21         return p
```

Representing the region of integration

- the next step is building representations of the region of integration
- in `Shape.py`

```
1  class Shape(object):
2      """
3      The abstract base class for representations of geometrical regions
4      """
5
6      # interface
7      def interior(self, point):
8          """
9          Predicate that checks whether {point} falls on my interior
10         """
11         raise NotImplementedError(
12             "class {.__name__!r} should implement 'interior'".format(type(self)))
```

The interior of a circle, in `Disk.py`

```
1  from Shape import Shape
2
3  class Disk(Shape):
4      """
5      A representation of a circular disk
6      """
7
8      # interface
9      def interior(self, point):
10         """
11         Predicate that checks whether {point} falls on my interior
12         """
13         r2 = self.radius**2
14         x0, y0 = self.center
15         x, y = point
16         dx = x - x0
17         dy = y - y0
18
19         if dx*dx + dy*dy > r2:
20             return False
21
22         return True
23
24     # meta methods
25     def __init__(self, radius=1.0, center=(0.0, 0.0)):
26         self.radius = radius
27         self.center = center
28         return
```

A first pass at the driver script

```
def gauss():
    """
    The driver for the object oriented solution
    """
    from Disk import Disk
    from Mersenne import Mersenne

    # inputs
    N = 10**5
    box = [(0,0), (1,1)]
    # the point cloud generator
    cloud = Mersenne()
    # the region of integration
    disk = Disk(center=(0,0), radius=1)

    # the integration algorithm
    total = 0
    interior = 0
    while total < N:
        point = cloud.point(box)
        if disk.interior(point):
            interior += 1
        total += 1

    # print out the estimate of π
    print("pi: {0:.8f}".format(4*interior/N))
    return
```

Comparing the performance of our three implementations

N	C++ t(sec)	python t(sec)	naïve OO t(sec)
10^0	.002	.014	.014
10^1	.002	.014	.014
10^2	.002	.014	.014
10^3	.002	.015	.020
10^4	.004	.027	.078
10^5	.026	.144	.625
10^6	.230	1.265	6.242
10^7	2.277	12.624	61.583
10^8	22.749	130.430	
10^9	227.735		