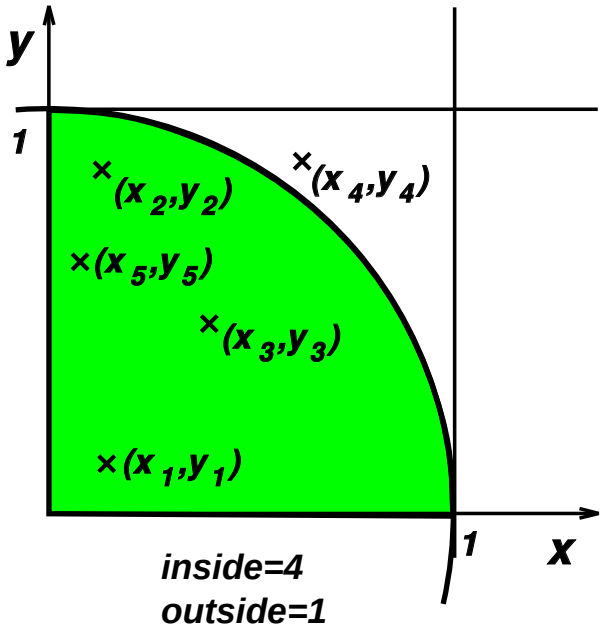
Computing PI by Montecarlo

Generate *random points* (x_j, y_j). The probability to fall in the unit circle is $\pi/4$.

$$\pi = 4\frac{\text{(\#inside)}}{\text{(\#total)}}$$



Montecarlo methods are *very easily parallelizable*, but *not deterministic* and exhibit a *slow rate of convergence* ($O(\sqrt{N})$).

(docver "texstuff-1.0.36-227-gdb205c4") (docdate "Sat Apr 14 21:47:34 2012 +0200") (procdate "Sat Apr 14 21:48:12 2012 +0200")

Assignement Nbr. 8

Computing PI by Montecarlo with OpenMP

- The sequential program below computes the number π with a Montecarlo strategy (see 163).
- In order to generate the random numbers we use a the *drand48_r()* function from the GNU Libc library. If you can not use this library try another random generator, but check that it must be a *reentrant* library, i.e. that it can be used in parallel.
- The typical call sequence for *drand48_r()* is

```
1  // This buffer stores the data for the
2  // random number generator
3  drand48_data buffer;
4  // This buffer must be initialized first
5  memset(&buffer, '\0',sizeof(struct drand48_data));
6  // Randomize the generator
7  srand48_r(time(0),&buffer);
8  ...
9  double x;
10 // Generate a random numbers x
11 drand48_r(&buffer,&x);
```

- Note: The variable *buffer* (i.e. the internal state of the random number generator) must be *private* to each thread, and also the initialization must

be performed in each thread.

- Your task is to parallelize this program with OpenMP.
- Compute speedup numbers for different number of processors. Try to run it on a computer with at least 4 cores.
- Try different types of scheduling (static, dynamic, guided...). Try several chunk sizes also, and determine a good combination. Discuss.

```
1  #include <cassert>
2  #include <cstdio>
3  #include <cstdlib>
4  #include <ctime>
5  #include <cstring>
6  #include <cmath>
7
8  #include <stdint.h>
9  #include <inttypes.h>
10 #include <unistd.h>
11
12 #include <omp.h>
13
14 using namespace std;
15
16 int main(int argc, char **argv) {
17   uint64_t chunk=100000000, inside=0;
18   double start = omp_get_wtime();
19   drand48_data buffer; // This buffer stores the data for the
20                        // random number generator
21   // This buffer must be initialized first
22   memset(&buffer, '\0',sizeof(struct drand48_data));
23   // Then we randomize the generator
```


(docver "texstuff-1.0.36-227-gdb205c4") (docdate "Sat Apr 14 21:47:34 2012 +0200") (procdate "Sat Apr 14 21:48:12 2012 +0200")

```
24    srand48_r(time(0),&buffer);
25    int nchunk=0;
26    while (1) {
27      for (uint64_t j=0; j<chunk; j++) {
28        double x,y;
29        // Generate a pair of random numbers x,y
30        drand48_r(&buffer,&x);
31        drand48_r(&buffer,&y);
32        inside += (x*x+y*y<1-0);
33      }
34      double
35        now = omp_get_wtime(),
36        elapsed = now-start;
37      double npoints = double(chunk);
38      double rate = double(chunk)/elapsed/1e6;
39      nchunk++;
40      double mypi = 4.0*double(inside)/double(chunk)/nchunk;
41      printf("PI %f, error %g, comp %g points, elapsed %fs, rate %f[Mpoints/s]
42            mypi,fabs(mypi-M_PI),npoints,elapsed,rate);
43      start = now;
44    }
45    return 0;
46  }
```


(docver "texstuff-1.0.36-227-gdb205c4") (docdate "Sat Apr 14 21:47:34 2012 +0200") (procdate "Sat Apr 14 21:48:12 2012 +0200")