



UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA



π computation with Montecarlo

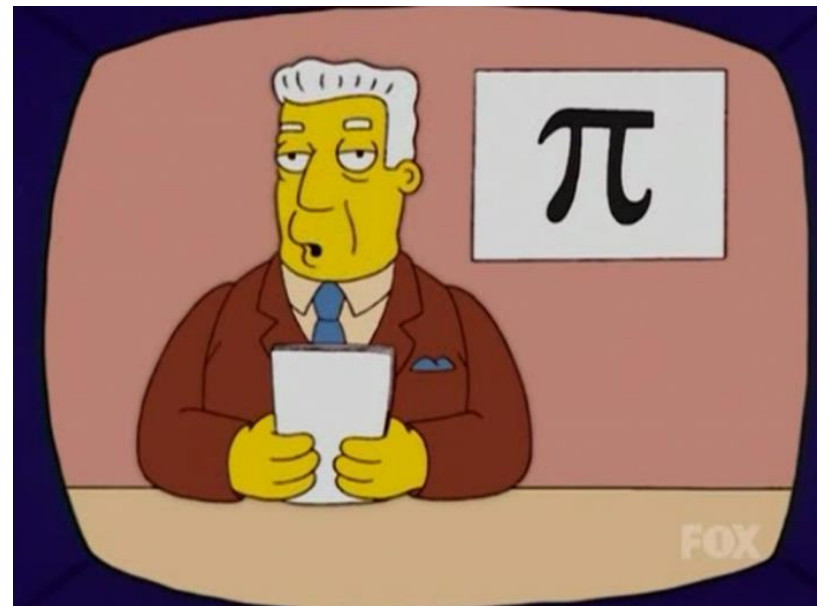
```
3.141592653589793238462643383
279502884197169399375105820974944
59230781640628620899862803482534211
70679821480865132823066470938446095
50982231 725359408 128481117
45028410 270193852 110555944
622948 954930381 964288109
75 665933446 128475 6482
3378678316 5271201909
145648586 9234603486
1045432664 8213393607
2602491412 7372458700
66063155881 74881520920 962829
25409171536 43678925903600113305
3054882046652 1384146931941611609
43305727036375 959195309218611738
19326117931051 18548074462379962
7495673518657 527248912279381
8301194912 9833673362
44065 66430
```

Paolo Burgio
paolo.burgio@unimore.it



PI

- ✓ No need to explain...
 - 3.1415926535897932384626433832795028841971693993751058209...
- ✓ Never end
 - No algorithm can compute its value in finite time





Save the date!!





Monte-Carlo methods

- ✓ Random-based experiments

Used in

- ✓ Solving deterministic problems (e.g., π computation)
- ✓ Studying random systems





Monte-Carlo PI computation

Rationale behind

- ✓ Correlation between π and the area of circle

$$A_{circle} = \pi * R_{circle}^2$$

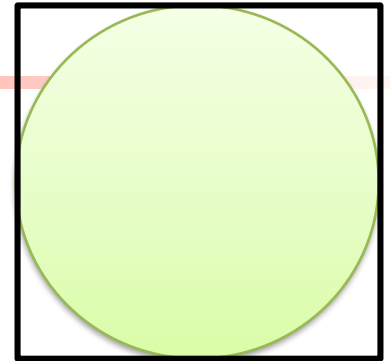
- ✓ Mmmm...



Of squares and circles

- ✓ Consider a square enclosing the circle. Its area is...

$$Side_{square} = R_{circle} * 2$$



- ✓ So, its area is...

$$A_{square} = Side_{square}^2 = (R_{circle} * 2)^2$$

- ✓ Given that..

$$A_{circle} = \pi * R_{circle}^2$$



Hmmmm...

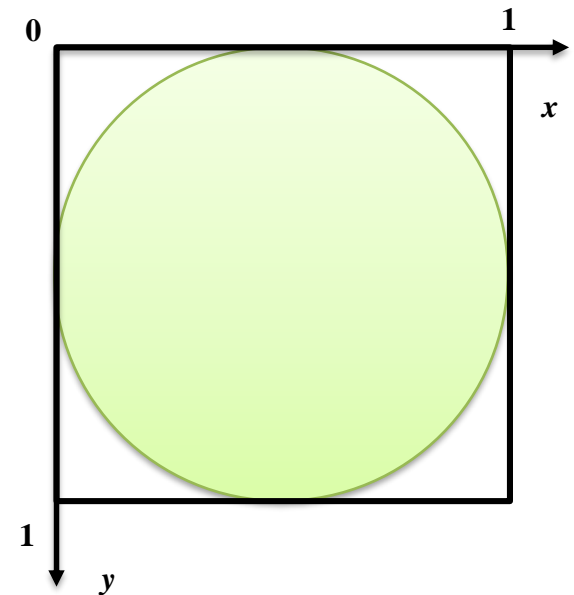
✓ Hmmmm....



$$\frac{A_{circle}}{A_{square}} = \frac{\pi * R_{circle}^2}{R_{circle}^2 * 4} = \frac{\pi}{4}$$

✓ ...SO....

$$\pi = 4 * \frac{A_{circle}}{A_{square}}$$





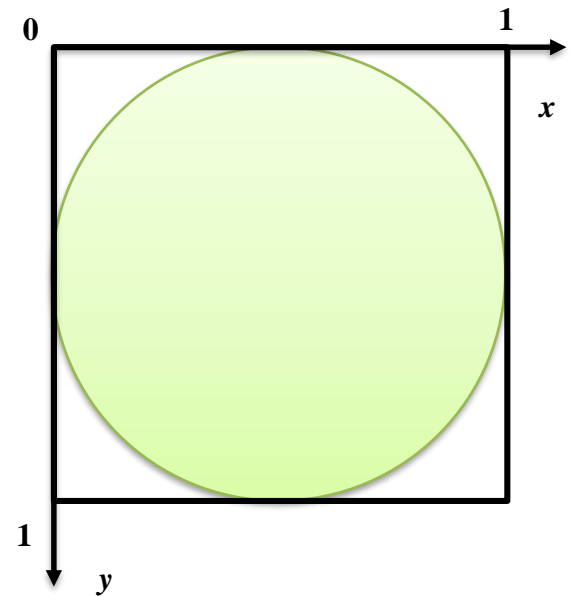
Now, the problem is..

- ✓ How to find the area of a circle
 - ..without knowing how much is π ??

Go random!

1. Throw randomly (many) point inside the square
 - Point = (x, y) with $x, y \in [0, R_{circle}]$
2. Count how many are within circle
3. Count how many are outside the circle
4. Compute the ratio
5. Multiply by 4

$$\pi = 4 * \frac{A_{circle}}{A_{square}}$$





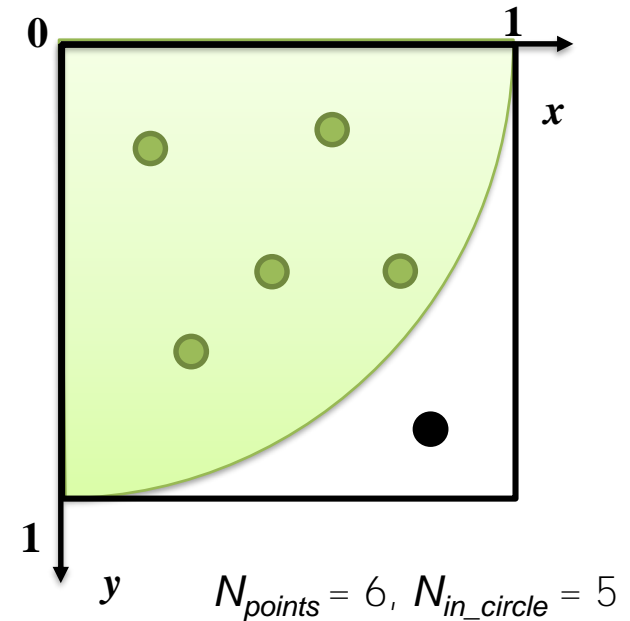
Optimizing the algorithm

- ✓ Consider only $\frac{1}{4}$ of the circle
 - R_{circle} from 0 to 1
 - $Side_{square}$ from 0 to 1
- ✓ Here, 5/6 are inside the circle
 - $N_{points} = 6 \Rightarrow \pi = 4 * 5 / 6 = 3.33$

How to compute it?

1. All points are always inside the square
2. A given point is inside the circle iff?
3. The more N_{points} , the more precision!

- ✓ It's sooo parallel!
 - Each thread generates a point and performs 2.





Exercise

Let's
code!

- ✓ Compute PI using Montecarlo Method
 - 10k iterations
 - Parametrizable



Random number generation

- ✓ Generate random (float) number between 0 and 1
 - Code/utills.c
 - Credits: Francesco Bellei

```
#include <stdlib.h>
float randNumGen()
{
    //Generate a random number
    int random_value = rand_r( /* Add thread-unique seed here */ );

    //make it between 0 and 1
    float unit_random = random_value / (float) RAND_MAX;

    return unit_random;
}
```





Exercise

Let's
code!

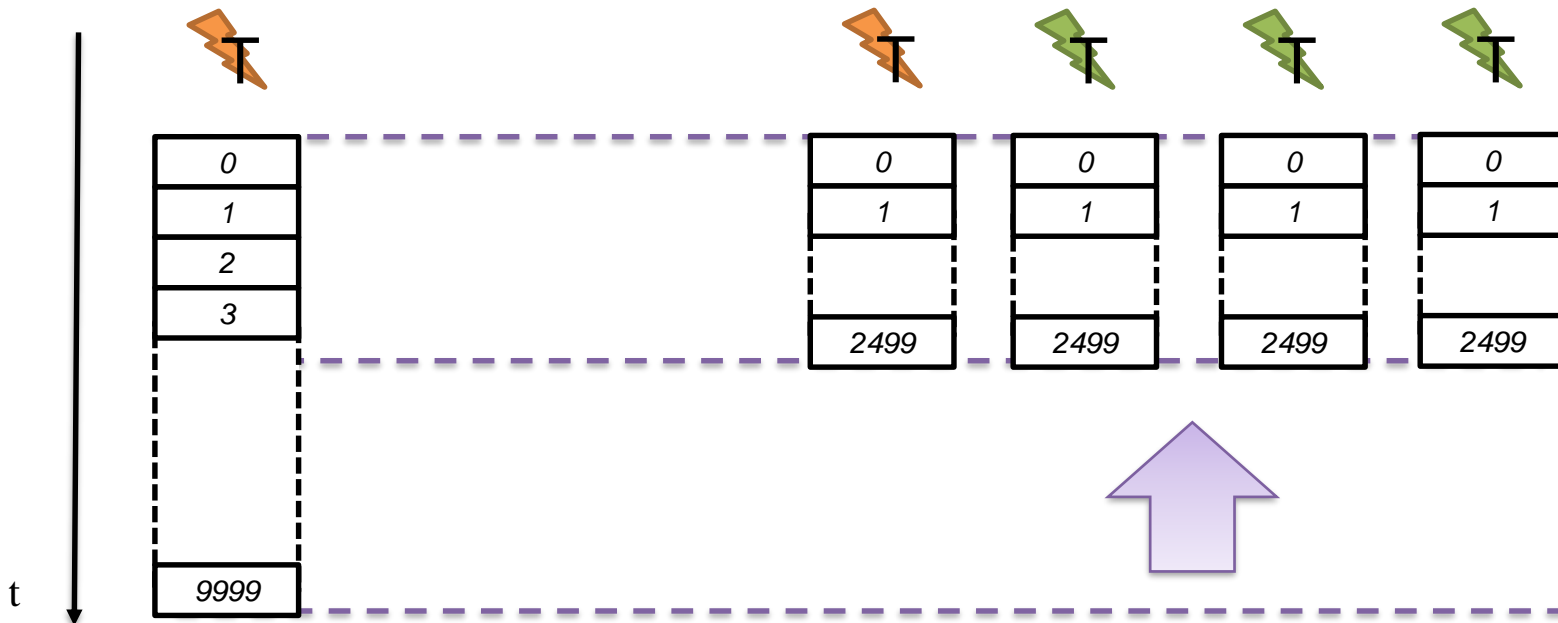
- ✓ Now, parallelize Montecarlo over N threads
- ✓ Need only $10k/N$ iterations!
 - In my laptop, w/Cygwin, $N = 4$
- ✓ Potentially, N times faster!
 - ...it won't be...

- ✓ How to know the number of (virtual) cores in Unix systems
 - `$ cat /proc/cpuinfo`



Sequential vs parallel

- ✓ 10k iterations
 - 1 rectangle = 1 iteration
 - You save N time!





Timing measurements

- ✓ Enable timing analysis of sequential/parallel code
 - http://algogroup.unimore.it/people/marko/courses/programmazione_parallel/PP1617/esercitazioni/pi_montecarlo/utils.c

```
#include <time.h>
#include <sys/time.h>

#define SECONDS 1000000000

unsigned long long gettime(void)
{
    struct timespec t;
    int r;

    r = clock_gettime(CLOCK_MONOTONIC, &t);
    if (r < 0)
    {
        printf("Error to get time! (%i)\n", r);
        return -1L;
    }

    return (unsigned long long) t.tv_sec * SECONDS + t.tv_nsec;
}
```



Some hints...

- ✓ Create `_printf` macro
 - Enables you adding/removing debug prints
 - "The problem of parallel debugging"
 - Printf "THREAD_ID/NTHREADS"
- ✓ Enable/disable OMP code with macro
 - First, test with Sequential!
 - If so, you can leave `-fopenmp` even for sequential code

```
#define _printf(...) printf(__VA_ARGS__)  
// #define _printf(...)  
  
#define USE_OMP  
  
void foo(void)  
{  
  
#ifdef USE_OMP  
    #pragma omp parallel  
#endif  
    {  
#ifdef USE_OMP  
        // Parallel code  
        _printf("[Thread %d/%d]\n",  
            omp_get_thread_num(),  
            omp_get_num_threads());  
#endif  
  
    }  
}
```



Standard-defined macro

OpenMP specifications

In implementations that support a preprocessor, the `_OPENMP` macro name is defined to have the decimal value `yyyymm` where `yyyy` and `mm` are the year and month designations of the version of the OpenMP API that the implementation supports.

- ✓ But...
 - *"In implementations that support a preprocessor"*



Exercise

Let's
code!

- ✓ Do this at home, varying N
 - $N = 1, 2, 4, 8, 16$
 - Run each experiment 3-5 times
 - Put avg values in an excel, and let's discuss



How to run the examples

Let's
code!

✓ Download the code/ folder from the course website

✓ Compile

✓ `$ gcc -fopenmp code.c -o code`

✓ Run (Unix/Linux)

`$./code`

✓ Run (Win/Cygwin)

`$./code.exe`



References



- ✓ "Calcolo parallelo" website
 - http://allogroup.unimore.it/people/marko/courses/programmazione_parallela/

- ✓ My contacts
 - paolo.burgio@unimore.it
 - <http://hipert.mat.unimore.it/people/paolob/>

- ✓ PThreads
 - <https://computing.llnl.gov/tutorials/pthreads/>
 - <http://php.net/manual/en/book.pthreads.php>
 - <http://www.google.com>

- ✓ OpenMP specifications
 - <http://www.openmp.org>
 - <http://www.openmp.org/mp-documents/openmp-4.5.pdf>
 - <http://www.openmp.org/mp-documents/OpenMP-4.5-1115-CPP-web.pdf>