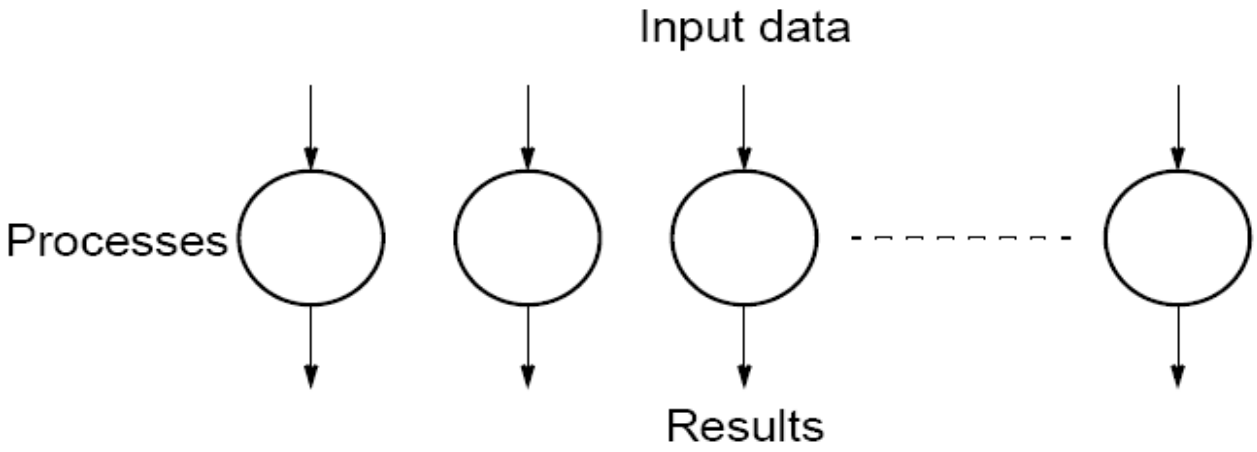
Parallel Techniques

- Embarrassingly Parallel Computations
- Partitioning and Divide-and-Conquer Strategies
- Pipelined Computations
- Synchronous Computations
- Asynchronous Computations
- Strategies that achieve load balancing

Embarrassingly Parallel Computations

Embarrassingly Parallel Computations

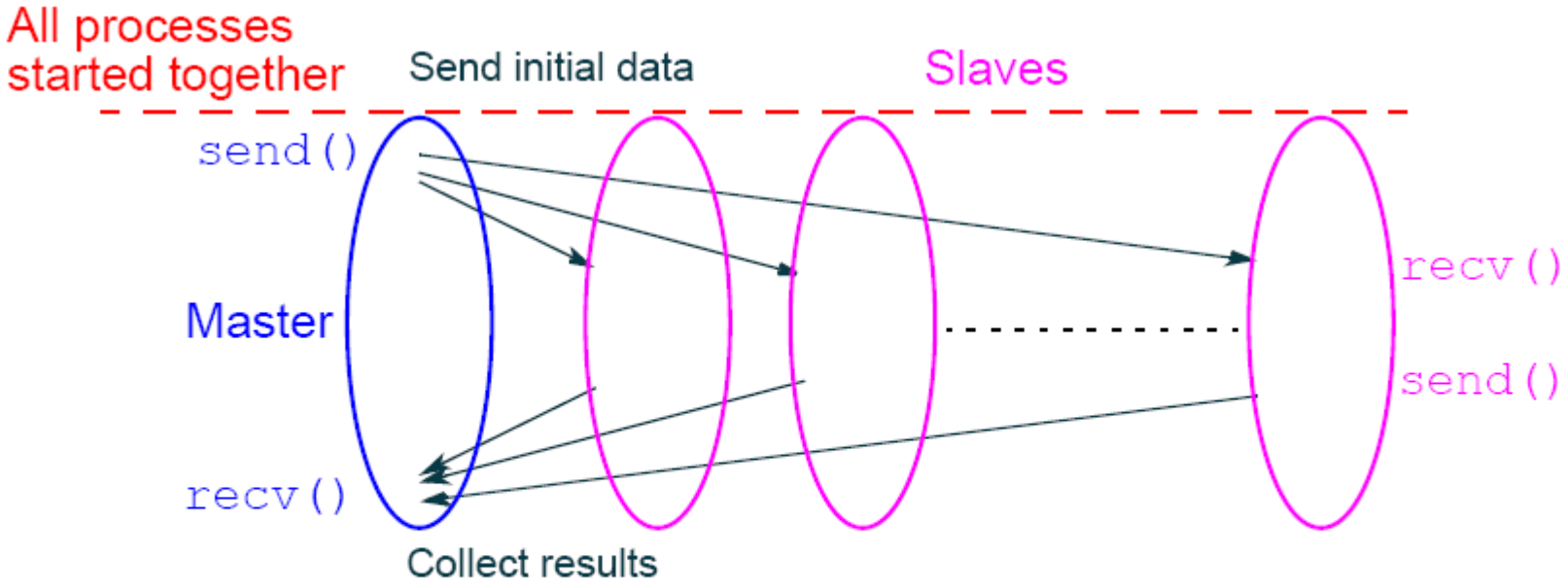
A computation that can obviously be divided into a number of completely independent parts, each of which can be executed by a separate process(or).



No communication or **very little communication** between processes

Each process can do its tasks without any interaction with other processes

Practical embarrassingly parallel computation with static process creation and master-slave approach



Usual MPI approach

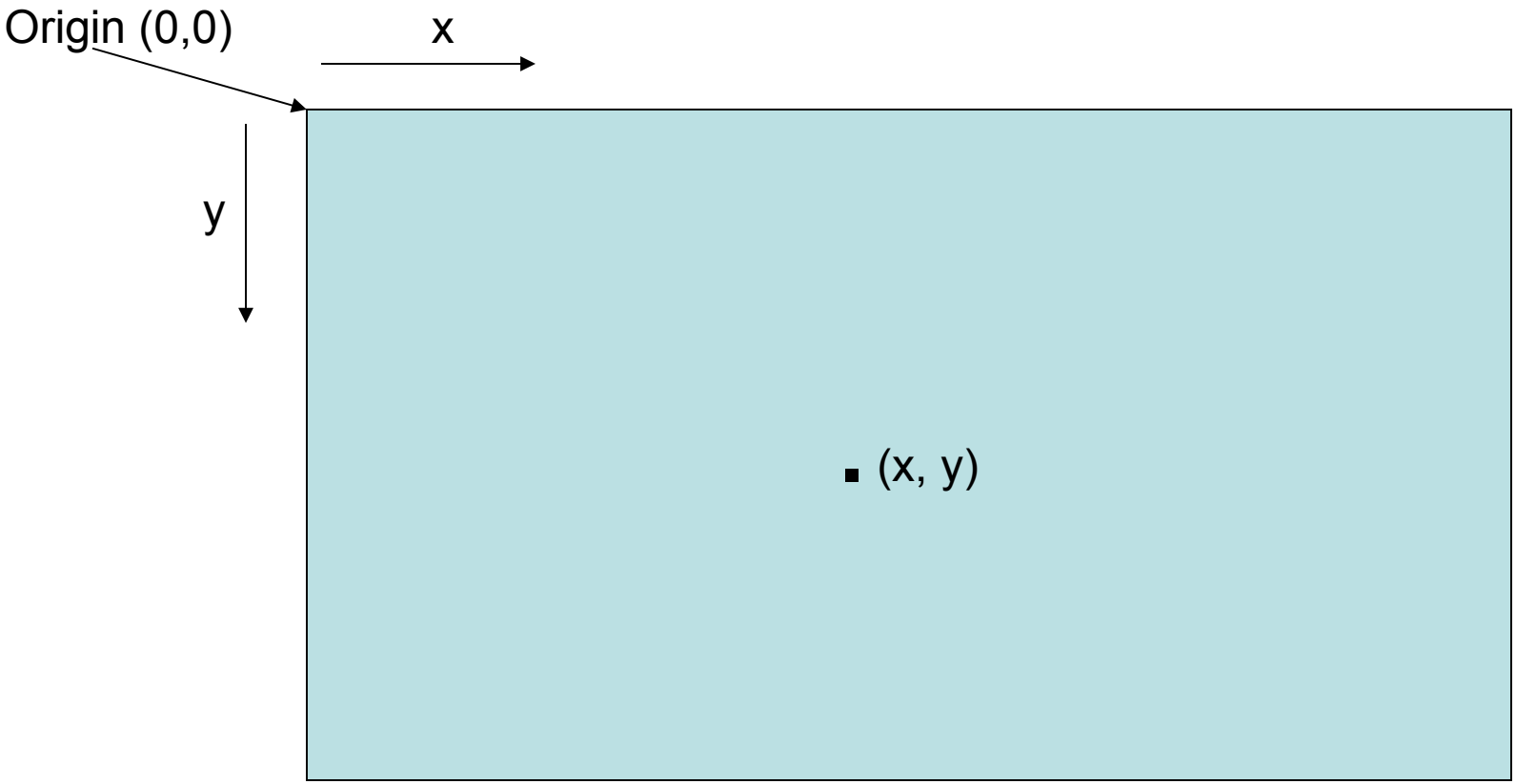
Embarrassingly Parallel Computation Examples

- Low level image processing
- Mandelbrot set
- Monte Carlo Calculations

Low level image processing

Many low level image processing operations only involve local data with very limited if any communication between areas of interest.

Image coordinate system



Some geometrical operations

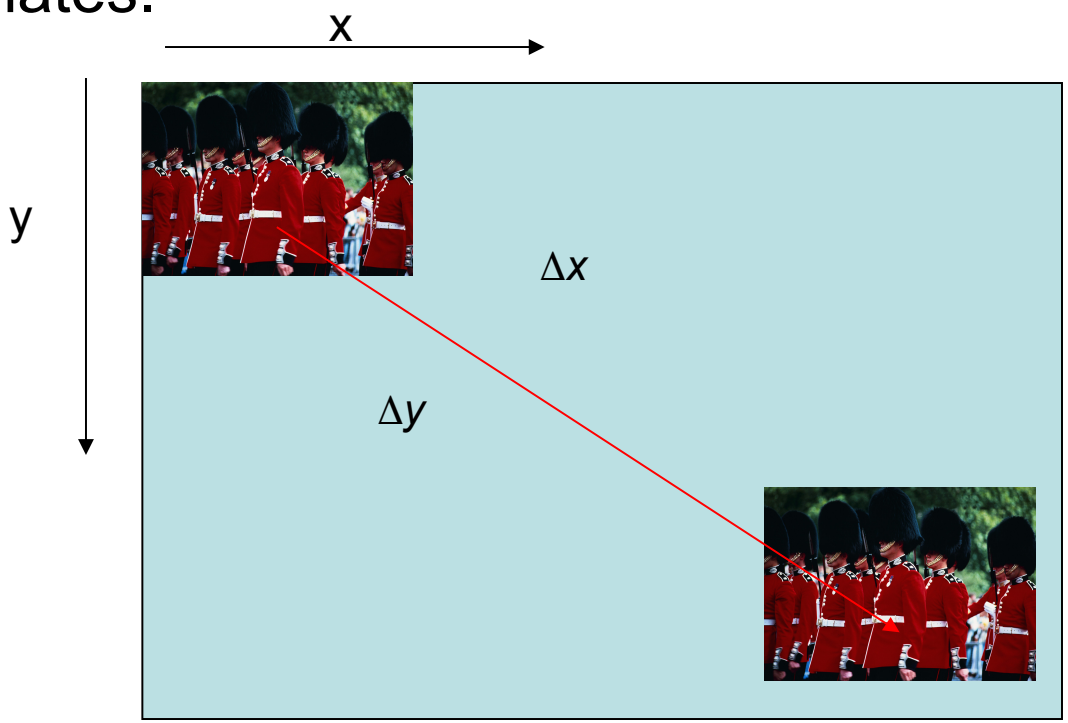
Shifting

Object shifted by Δx in x-dimension and Δy in y-dimension:

$$x' = x + \Delta x$$

$$y' = y + \Delta y$$

where x and y are original coordinates and x' and y' are the new coordinates.



Some geometrical operations

Scaling

Object scaled by factor S_x in x-direction and S_y in y-direction:

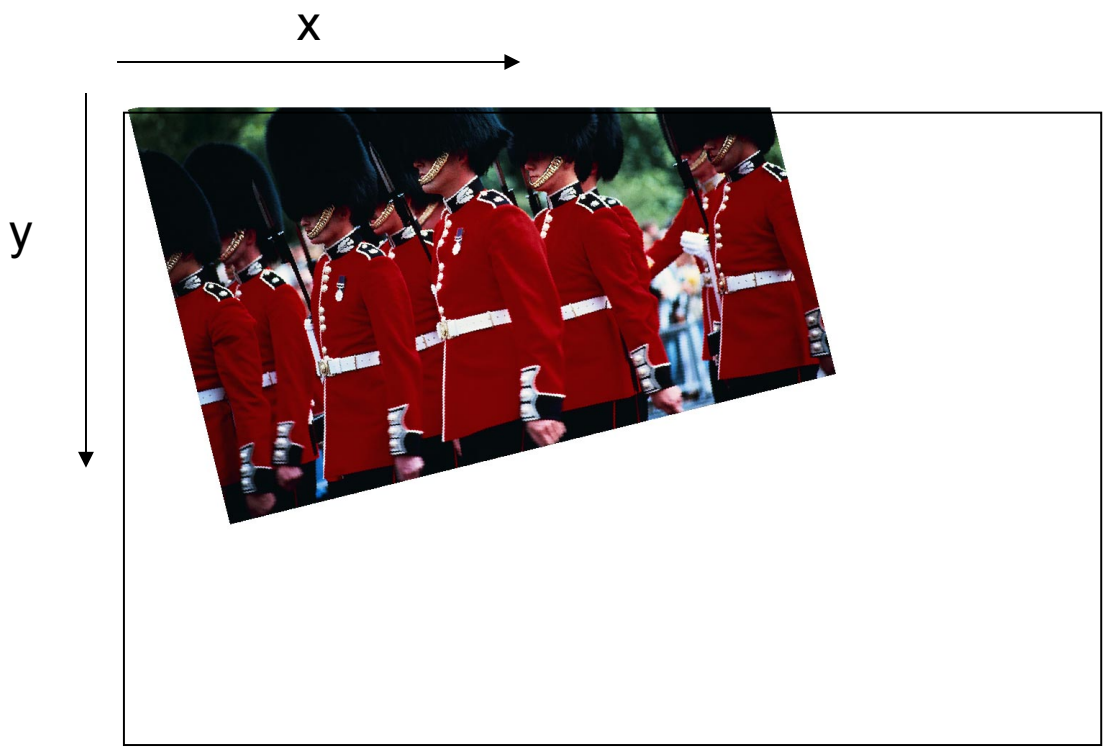
$$x' = xS_x$$

$$y' = yS_y$$

Rotation

Object rotated through an angle q about the origin of the coordinate system:

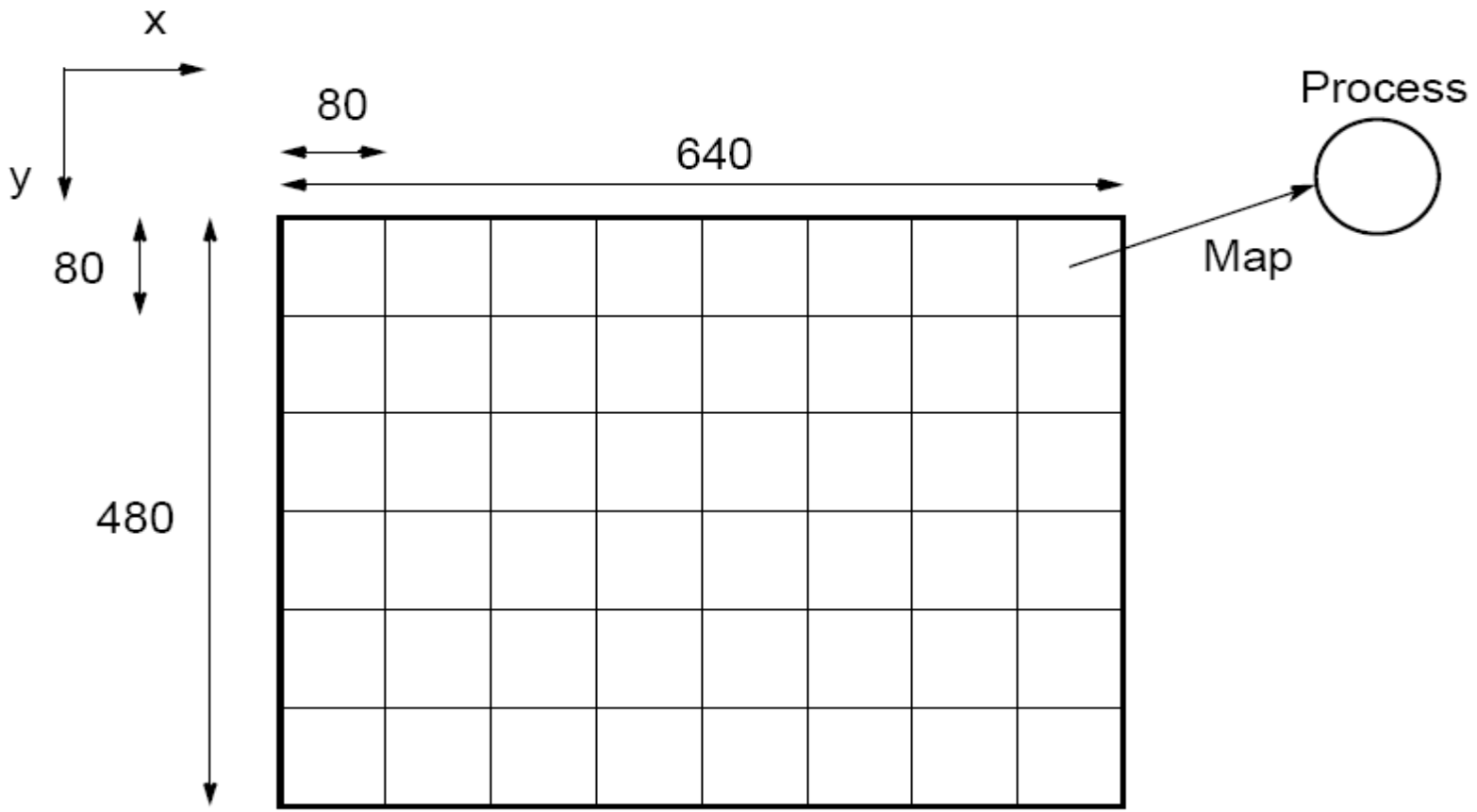
$$x' = x\cos\theta + v\sin\theta$$



Parallelizing Image Operations

Partitioning into regions for individual processes

Example



Square region for each process (can also use strips)

Question

Is there any inter–process communication?

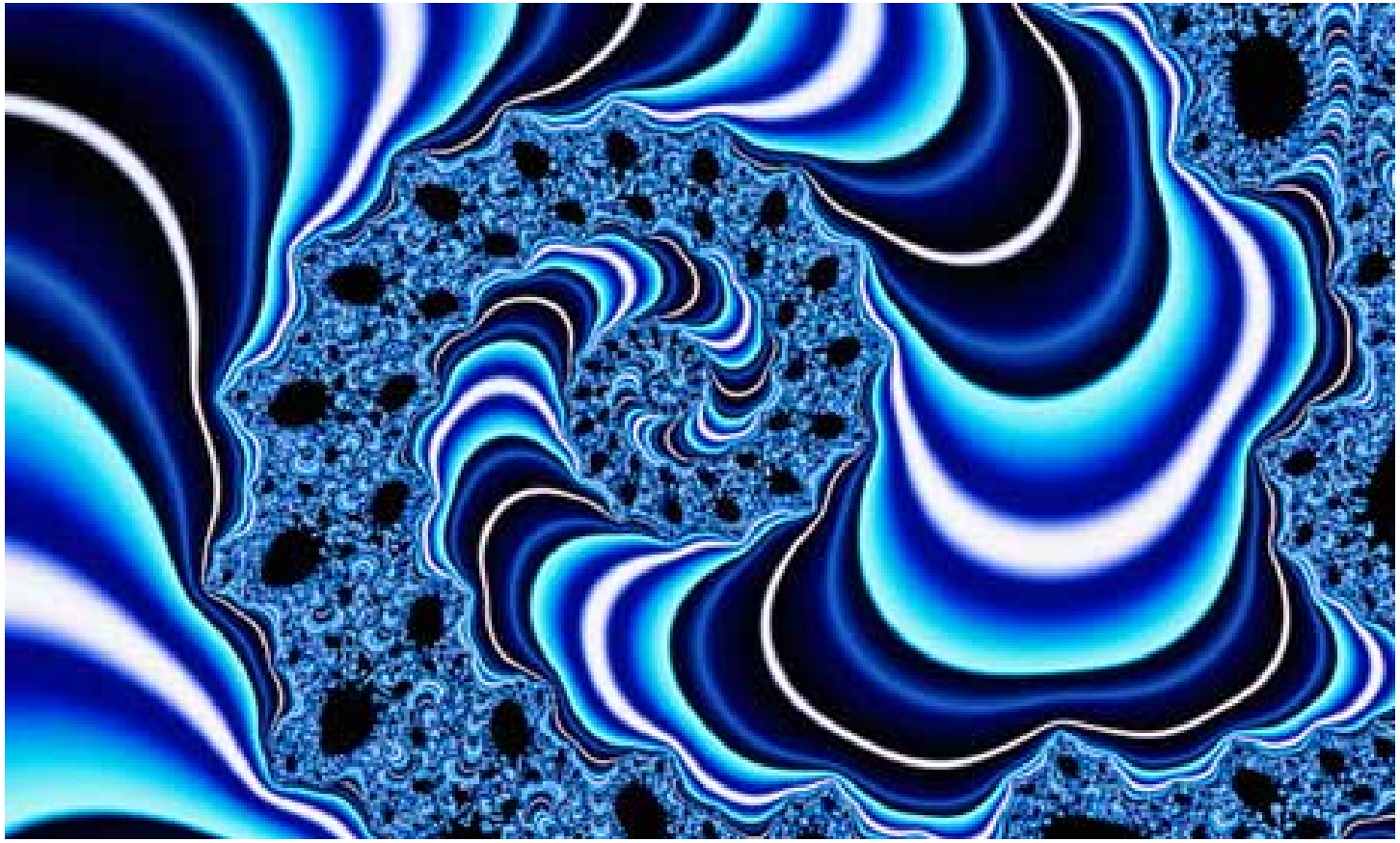
Answer

Benoît B. Mandelbrot (1924-2010), Father of Fractal Geometry

- http://laughingsquid.com/benoit-b-mandelbrot-1924-2010-father-of-fractal-geometry/
- http://vimeo.com/1908224

Died on Oct 15, 2010

Fractals



Photograph: © Stocktrek/Corbis

Mandelbrot Set

Set of points in a complex plane that are quasi-stable (will increase and decrease, but not exceed some limit) when computed by iterating the function:

$$z_{k+1} = z_k^2 + c$$

where z_{k+1} is the $(k + 1)th$ iteration of complex number:

$$z = a + bi$$

and c is a complex number giving position of point in the complex plane. The initial value for z is zero.

Mandelbrot Set continued

Iterations continued until magnitude of *z* is:

- Greater than 2

or

- Number of iterations reaches arbitrary limit.

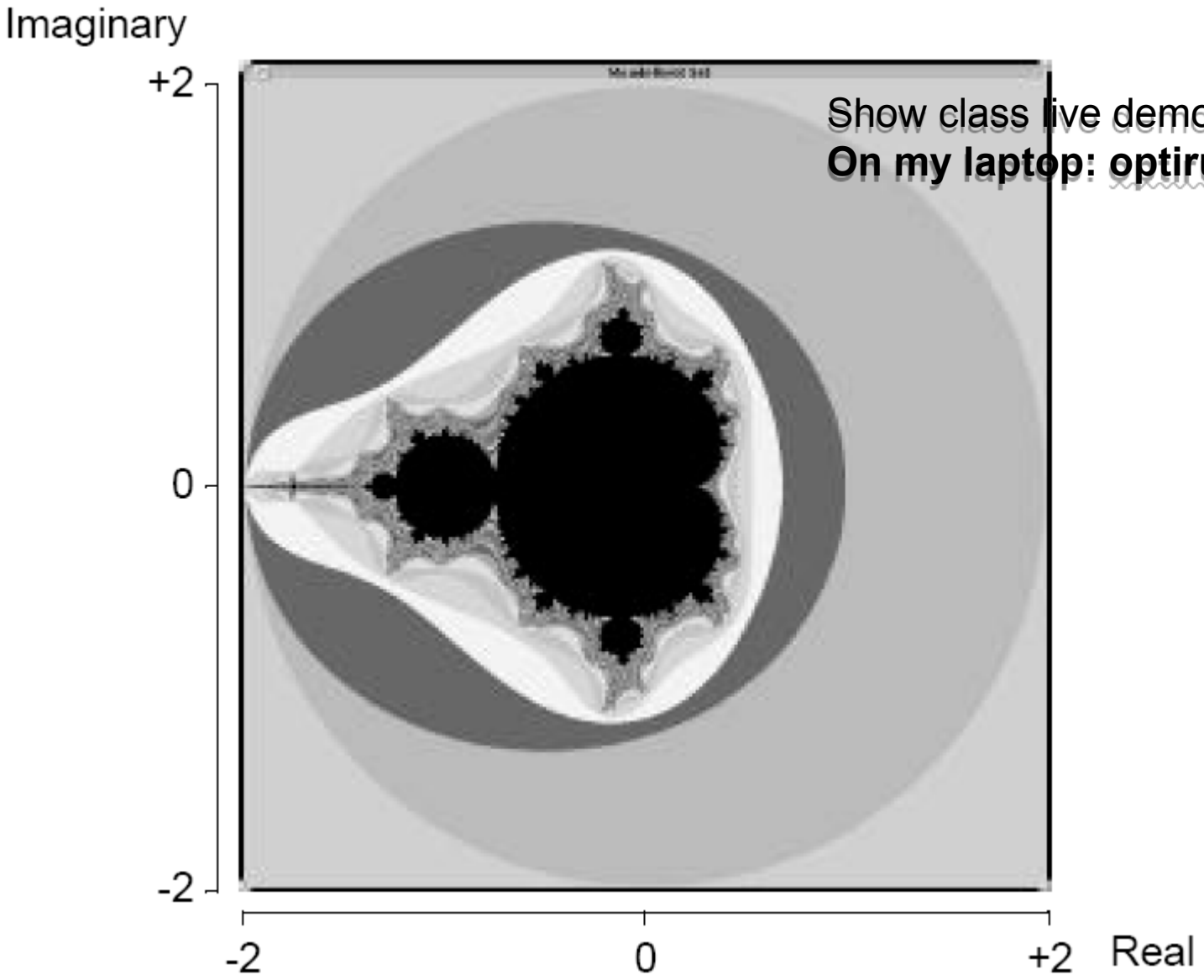
Magnitude of *z* is the length of the vector given by:

$$z_{length} = \sqrt{a^2 + b^2}$$

Sequential routine computing value of one point returning number of iterations

```
structure complex {
        float real;
        float imag;
};
int cal_pixel(complex c)
{
        int count, max;
        complex z;
        float temp, lengthsq;
        max = 256;
        z.real = 0; z.imag = 0;
        count = 0;                              /* number of iterations */
        do {
                temp = z.real * z.real - z.imag * z.imag + c.real;
                z.imag = 2 * z.real * z.imag + c.imag;
                z.real = temp;
                lengthsq = z.real * z.real + z.imag * z.imag;
                count++;
        } while ((lengthsq < 4.0) && (count < max));
        return count;
}
```

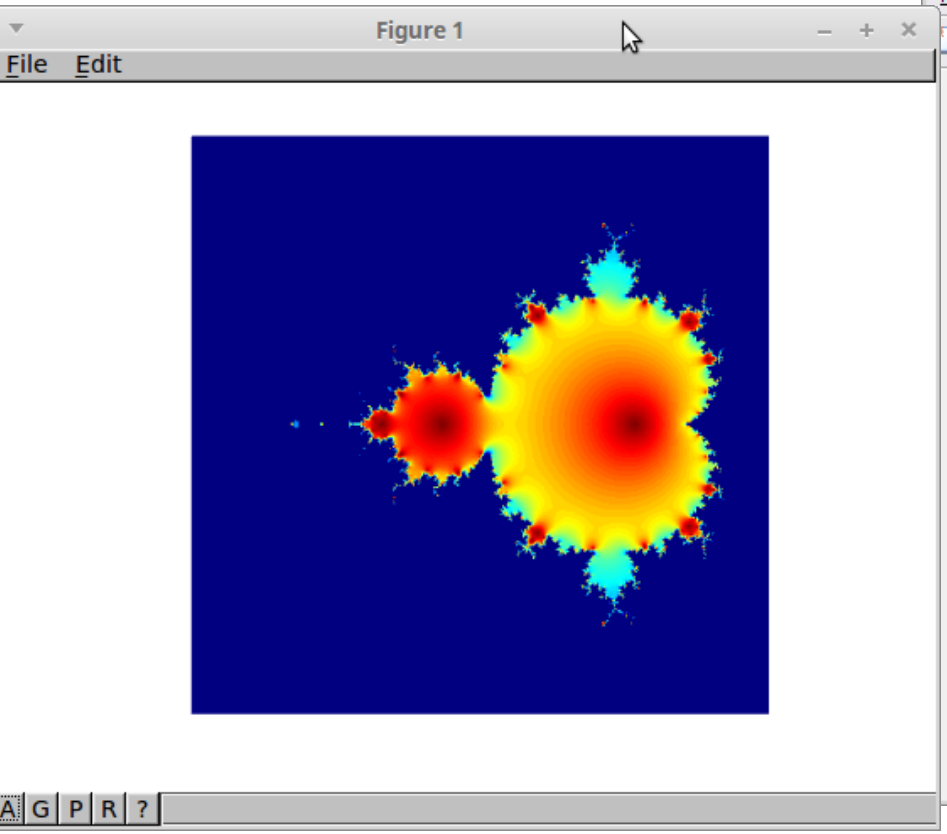
Mandelbrot set

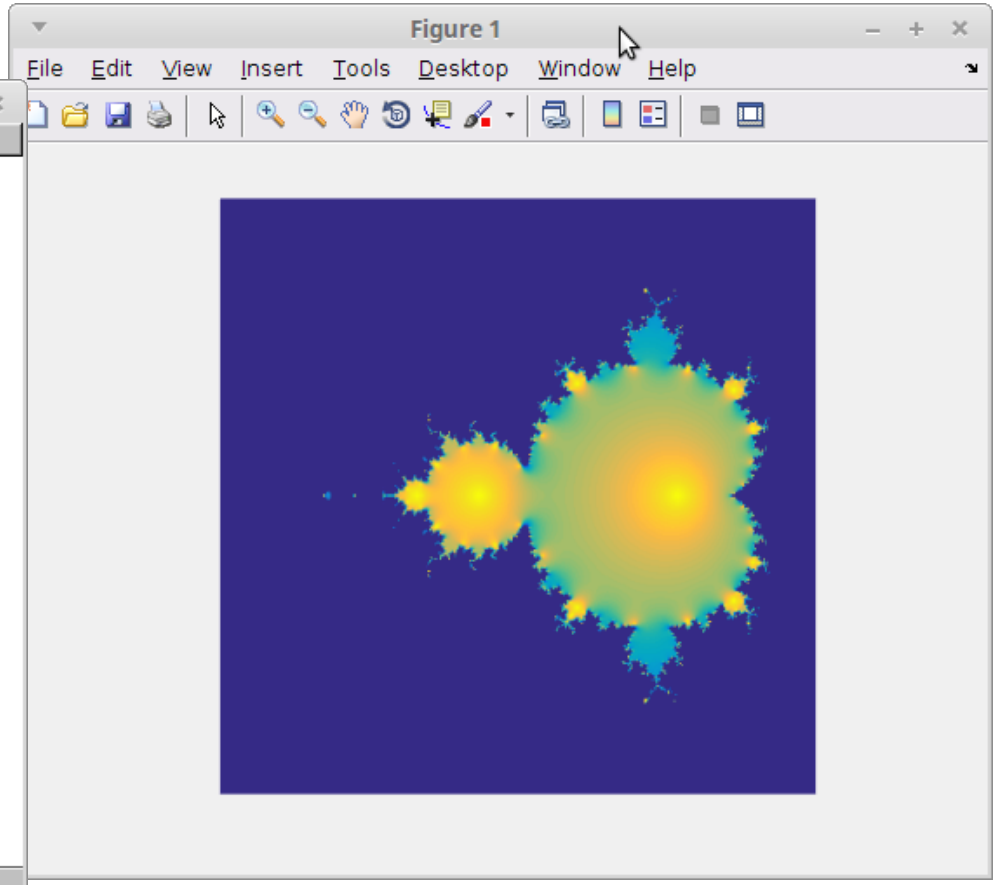


Show class live demo using Octave

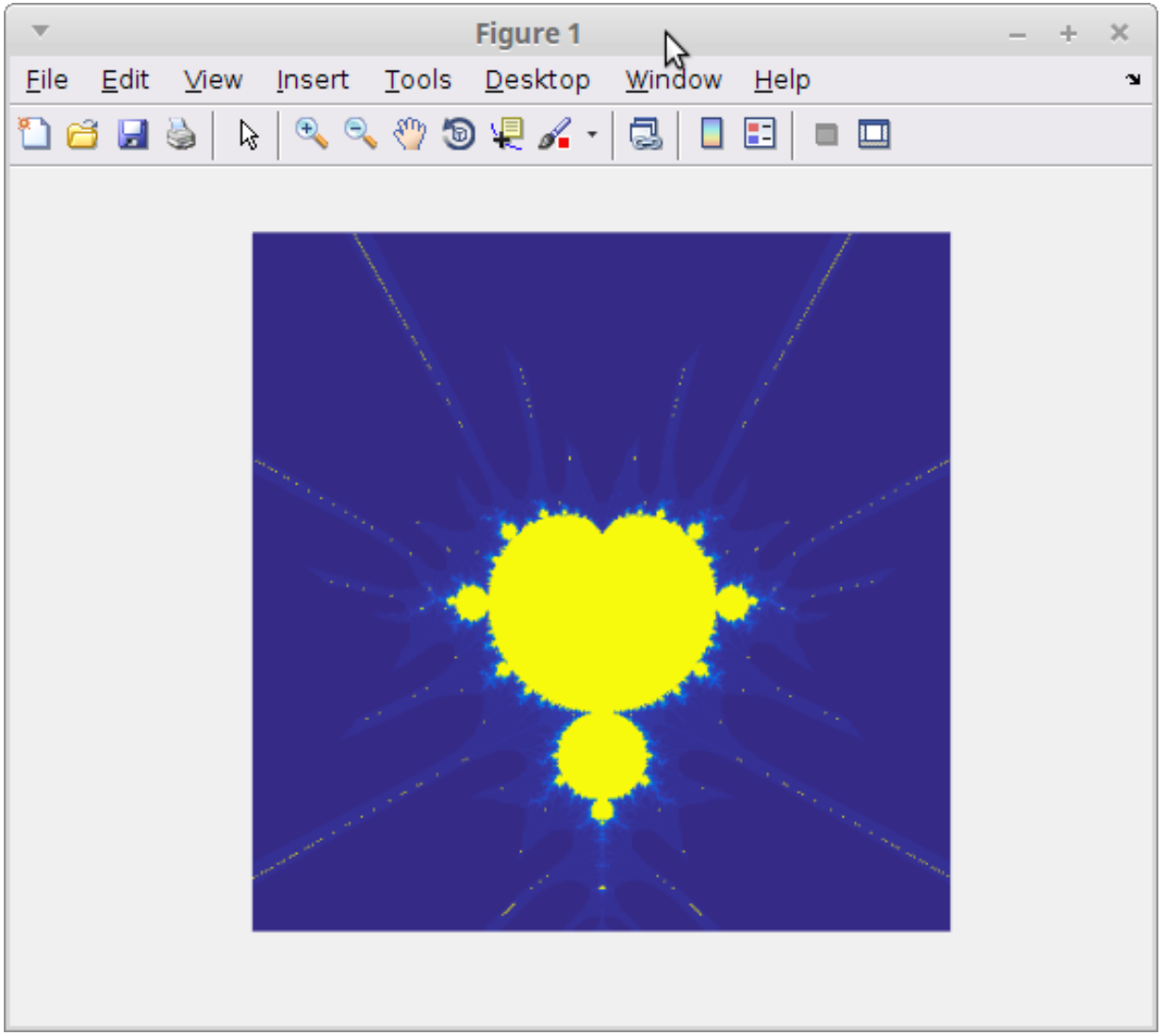
On my laptop: optirun qtoctave

Output of `mandel.m` executed with Matlab (right) and Octave (left)





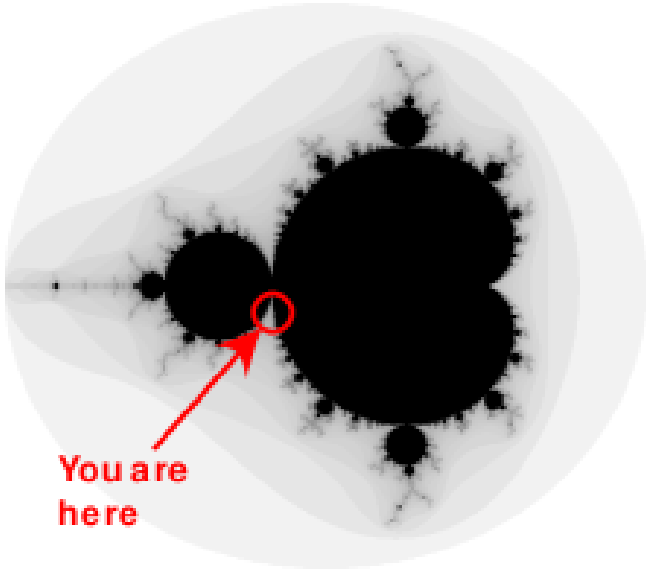
Output of `mandelg.m` executed with Matlab



http://blogs.mathworks.com/loren/2011/07/18/a-mandelbrot-set-on-the-gpu/

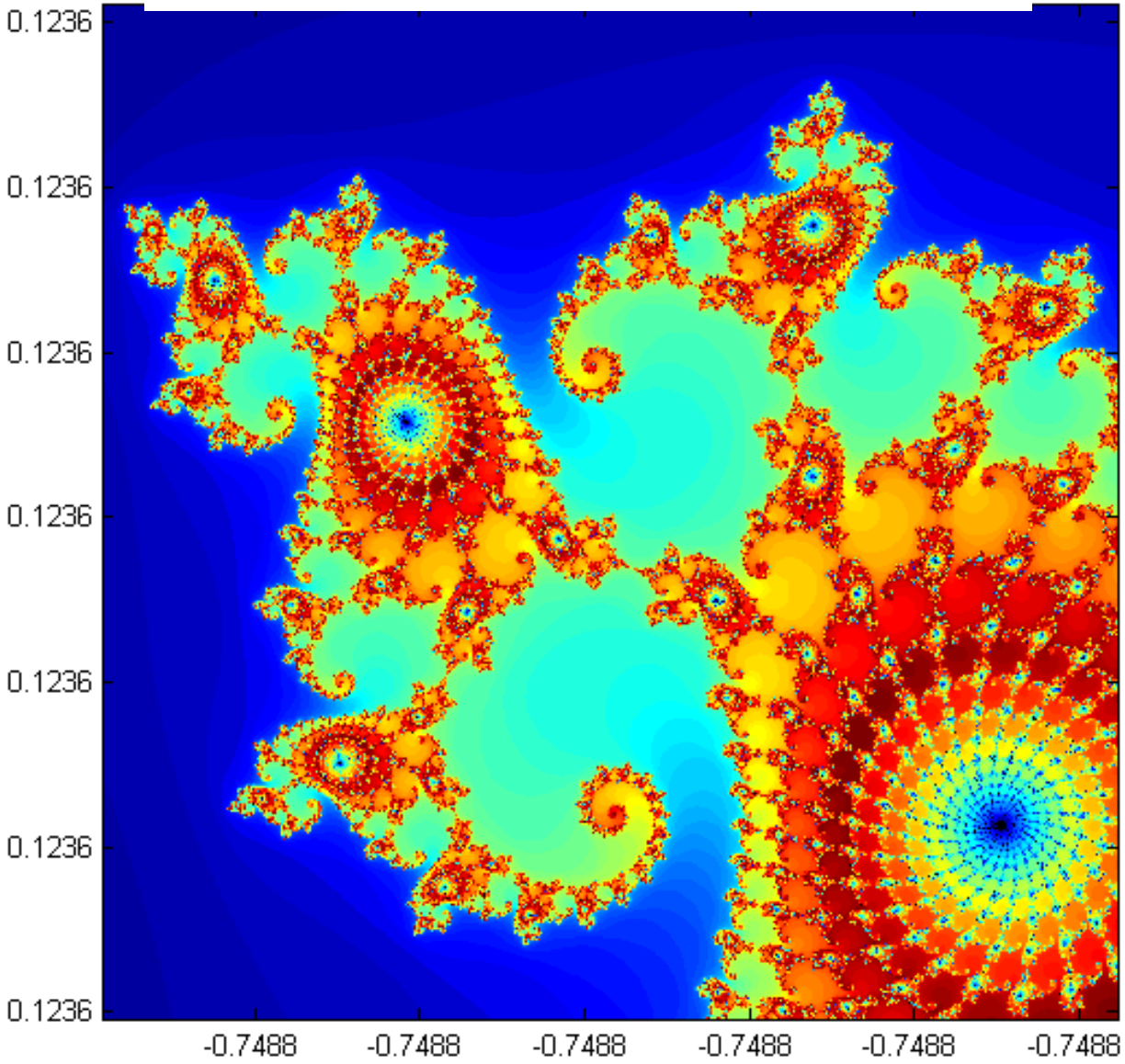
Setup

The values below specify a highly zoomed part of the Mandelbrot Set in the valley between the main cardioid and the p/q bulb to its left.



The following code forms the set of starting points for each of the calculations by creating a 1000x1000 grid of real parts (X) and imaginary parts (Y) between these limits. For this particular location I happen to know that 500 iterations will be enough to draw a nice picture.

Output of `mandel_matlab_new.m` executed with Matlab



```
maxIterations = 500;
gridSize = 1000;
xlim = [-0.748766713922161, -0.748766707771757];
ylim = [ 0.123640844894862,  0.123640851045266];
```

The Mandelbrot Set in MATLAB

Below is an implementation of the Mandelbrot Set using standard MATLAB commands running on the CPU. This calculation is vectorized such that every location is updated at once.

```
% Setup
t = tic();
x = linspace( xlim(1), xlim(2), gridSize );
y = linspace( ylim(1), ylim(2), gridSize );
[xGrid,yGrid] = meshgrid( x, y );
z0 = xGrid + 1i*yGrid;
count = zeros( size(z0) );

% Calculate
z = z0;
for n = 0:maxIterations
    z = z.*z + z0;
    inside = abs( z )<=2;
    count = count + inside;
end
count = log( count+1 );

% Show
cpuTime = toc( t );
set( gcf, 'Position', [200 200 600 600] );
imagesc( x, y, count );
axis image
colormap( [jet();flipud( jet() );0 0 0] );
title( sprintf( '%1.2fsecs (without GPU)', cpuTime ) );
```

Mandel_matlab_new.m

Parallelizing Mandelbrot Set Computation

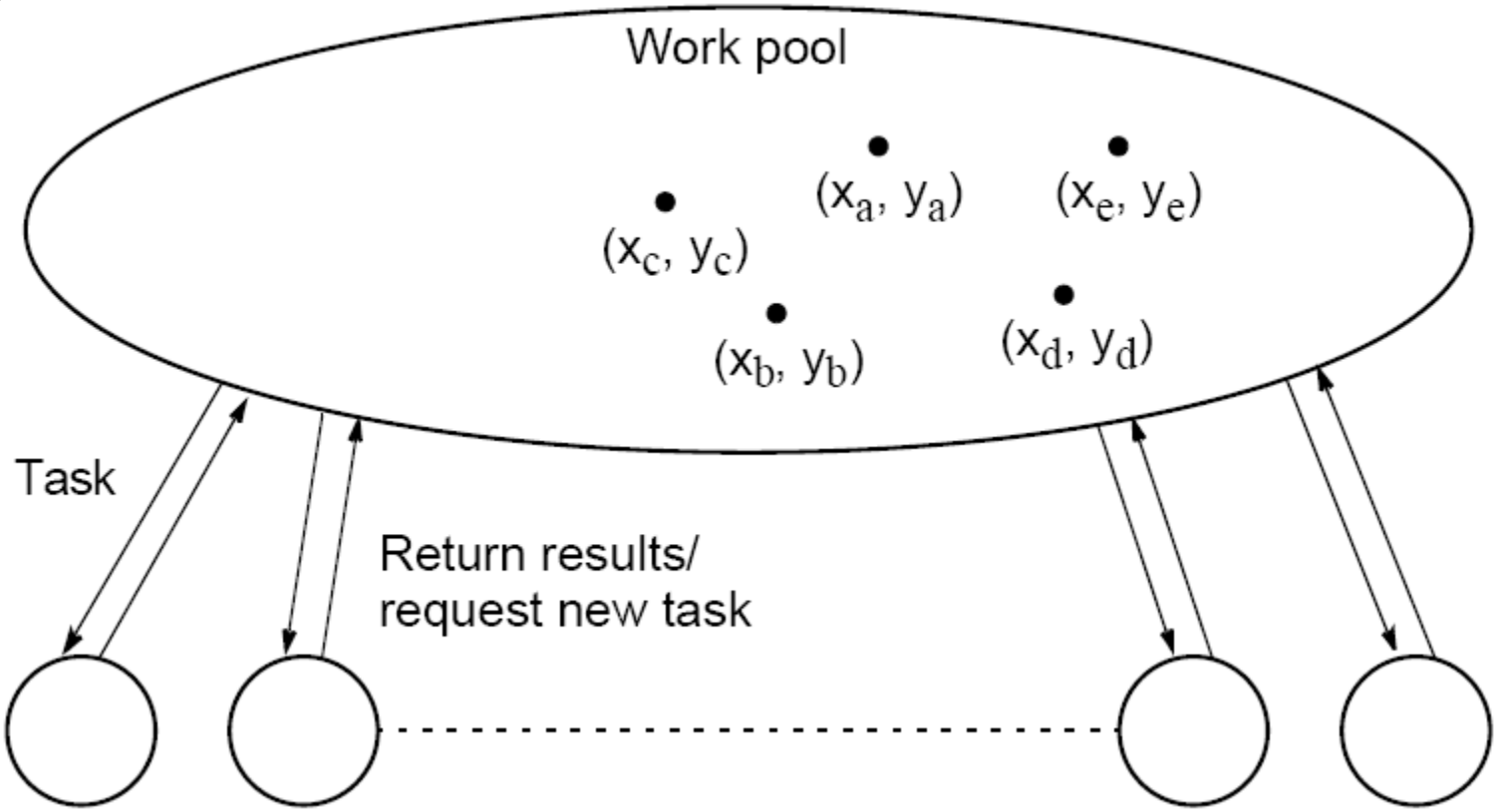
Static Task Assignment

Simply divide the region in to fixed number of parts, each computed by a separate processor.

Not very successful because different regions require different numbers of iterations and time.

Dynamic Task Assignment

Have processor request regions after computing previous regions



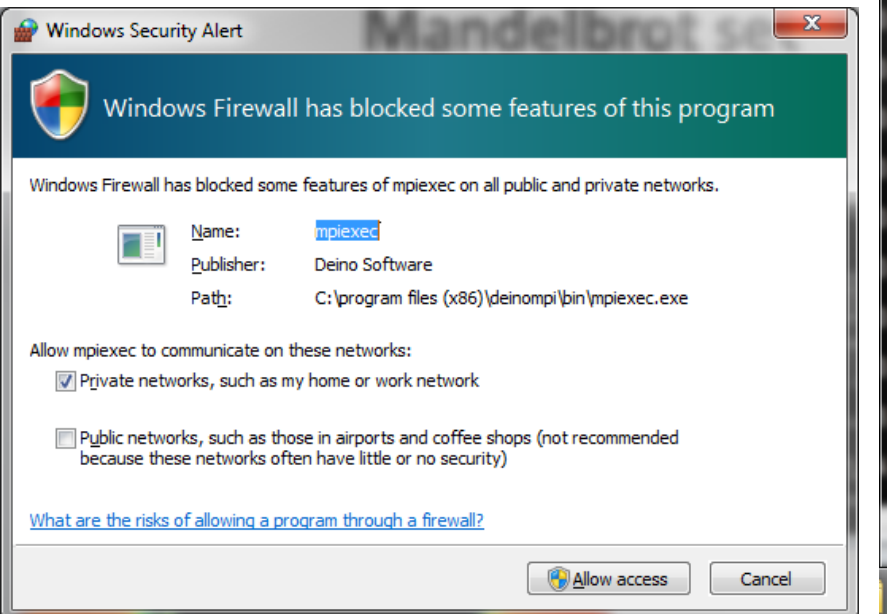
Parallel Mandelbrot demo

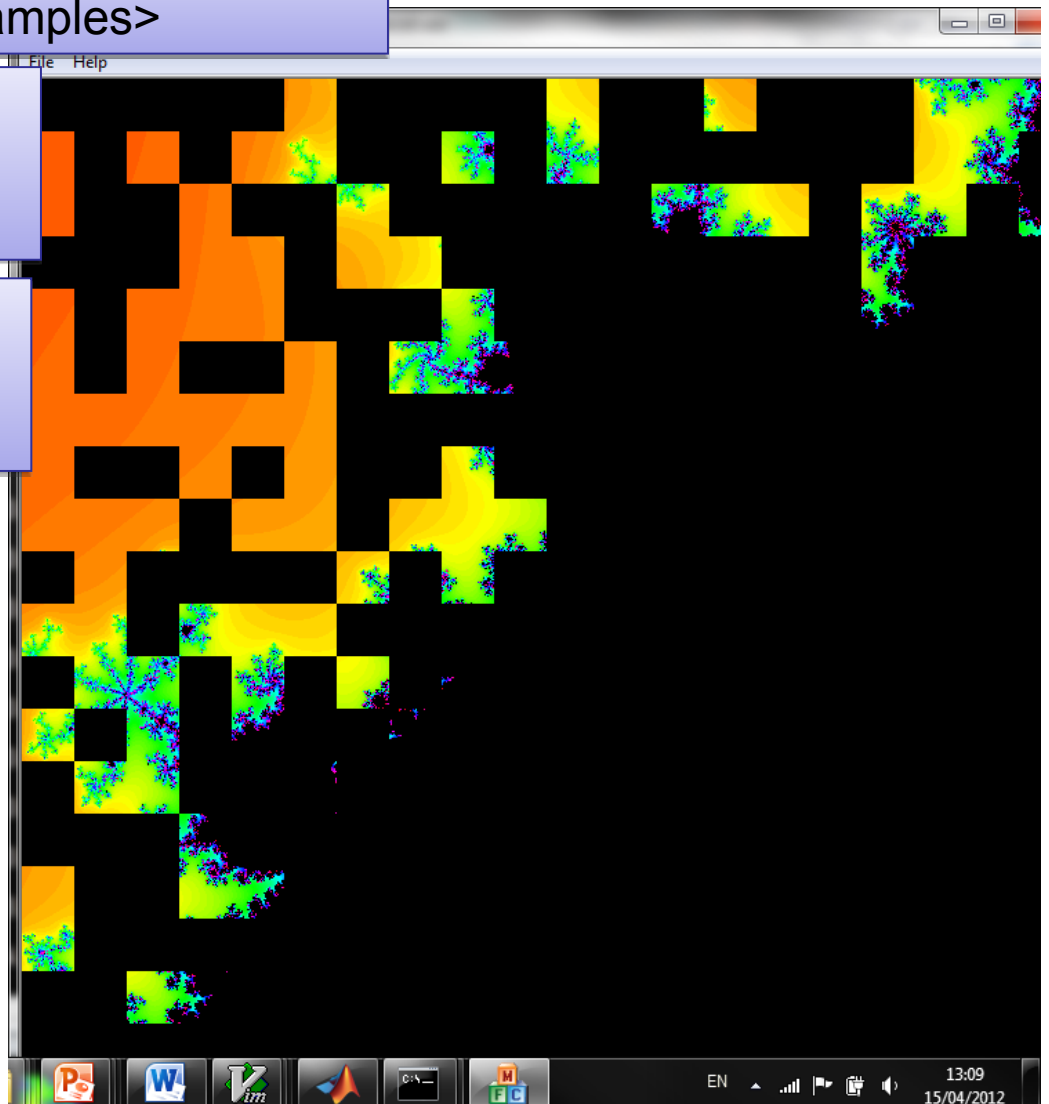
Open two command shell windows and change directory to: C:\Program Files (x86)\DeinoMPI\examples>

On window #1: Start the server:

..\bin\mpiexec –n 4 pmandel.exe

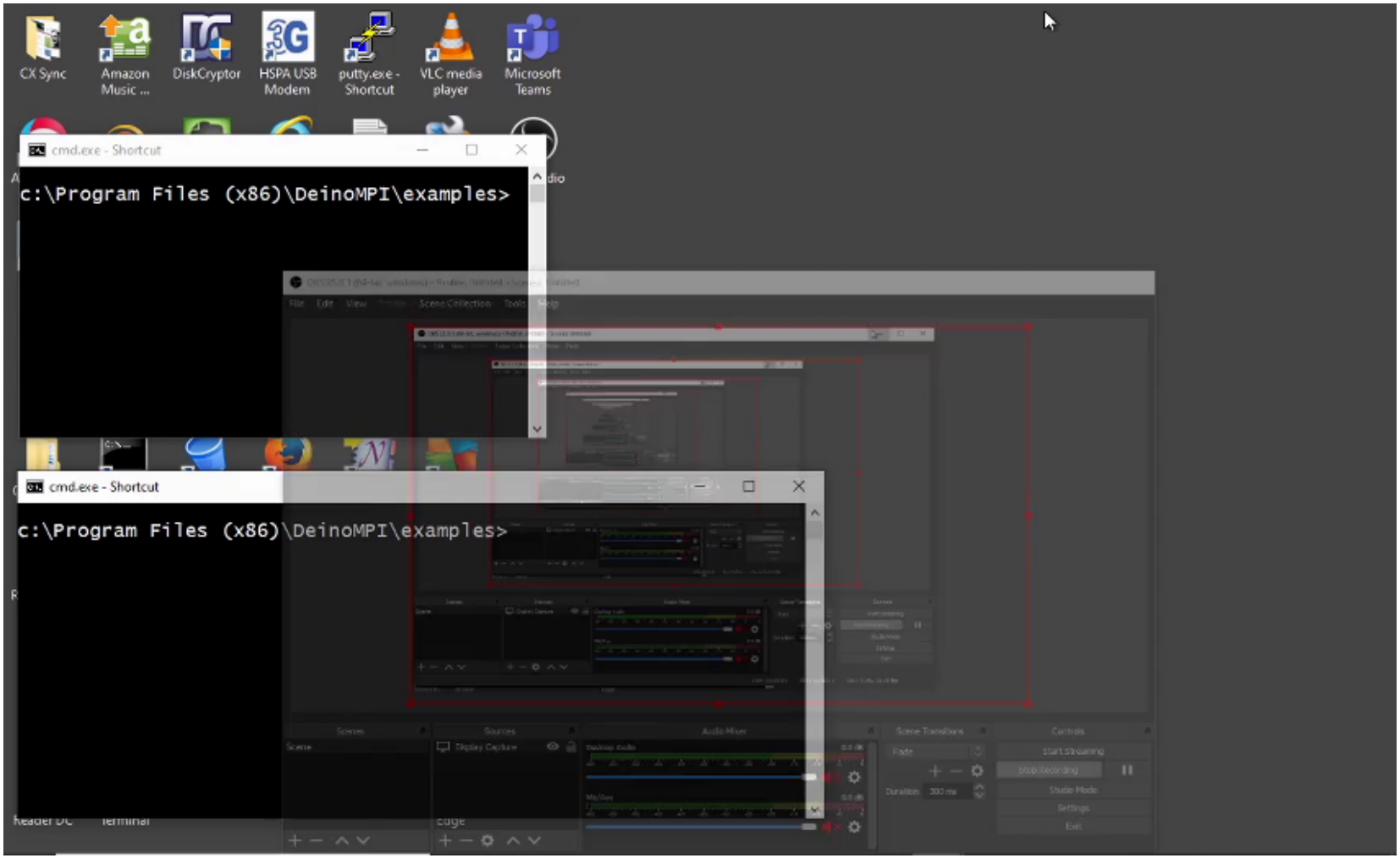
On window #2: Start the visualization client pman_vis.exe
Select the TCP ports



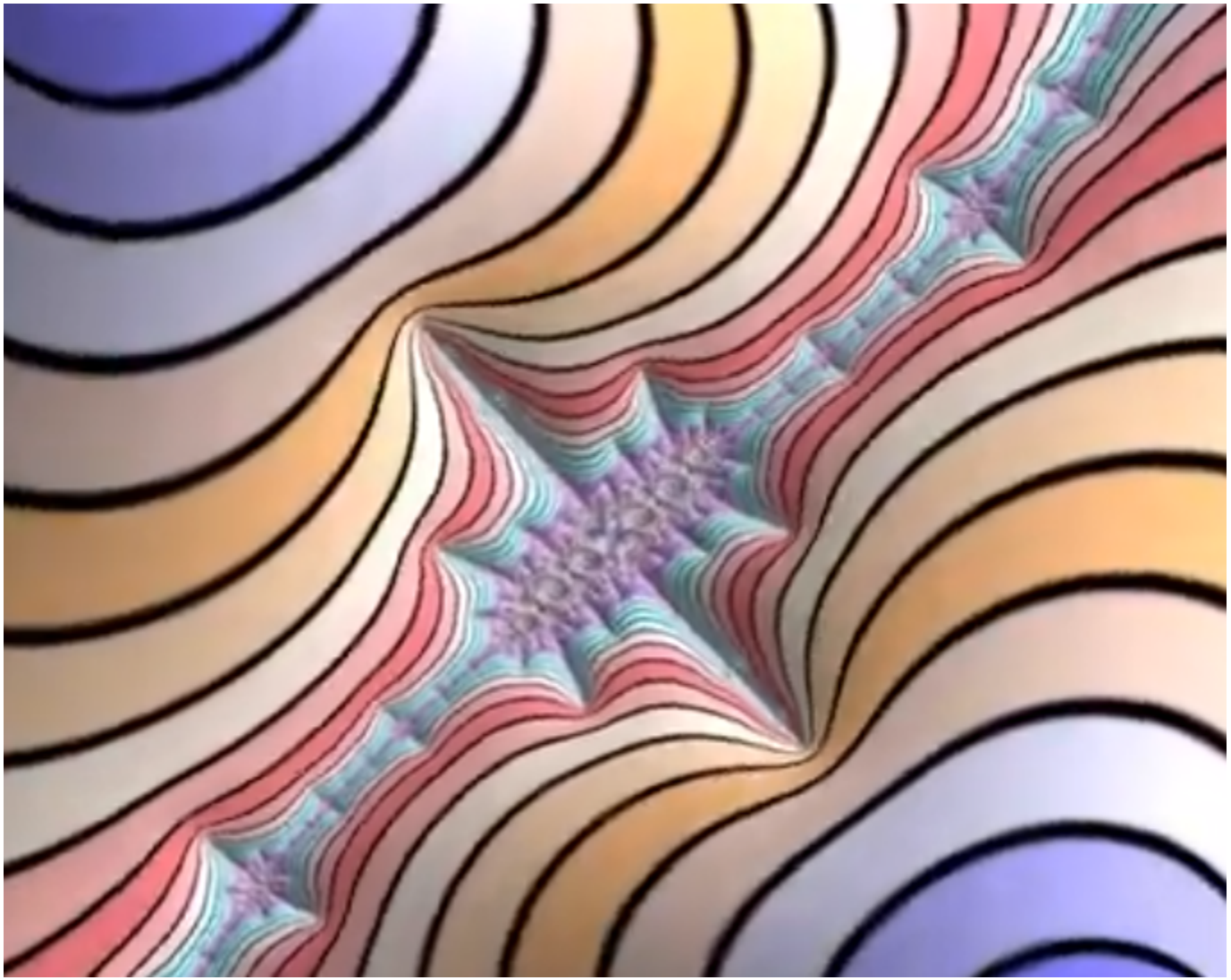


Parallel Mandelbrot demo

Screen recording demo from my other laptop



Link: Youtube video



Monte Carlo Methods

Another embarrassingly parallel computation.

Monte Carlo methods use of random selections.

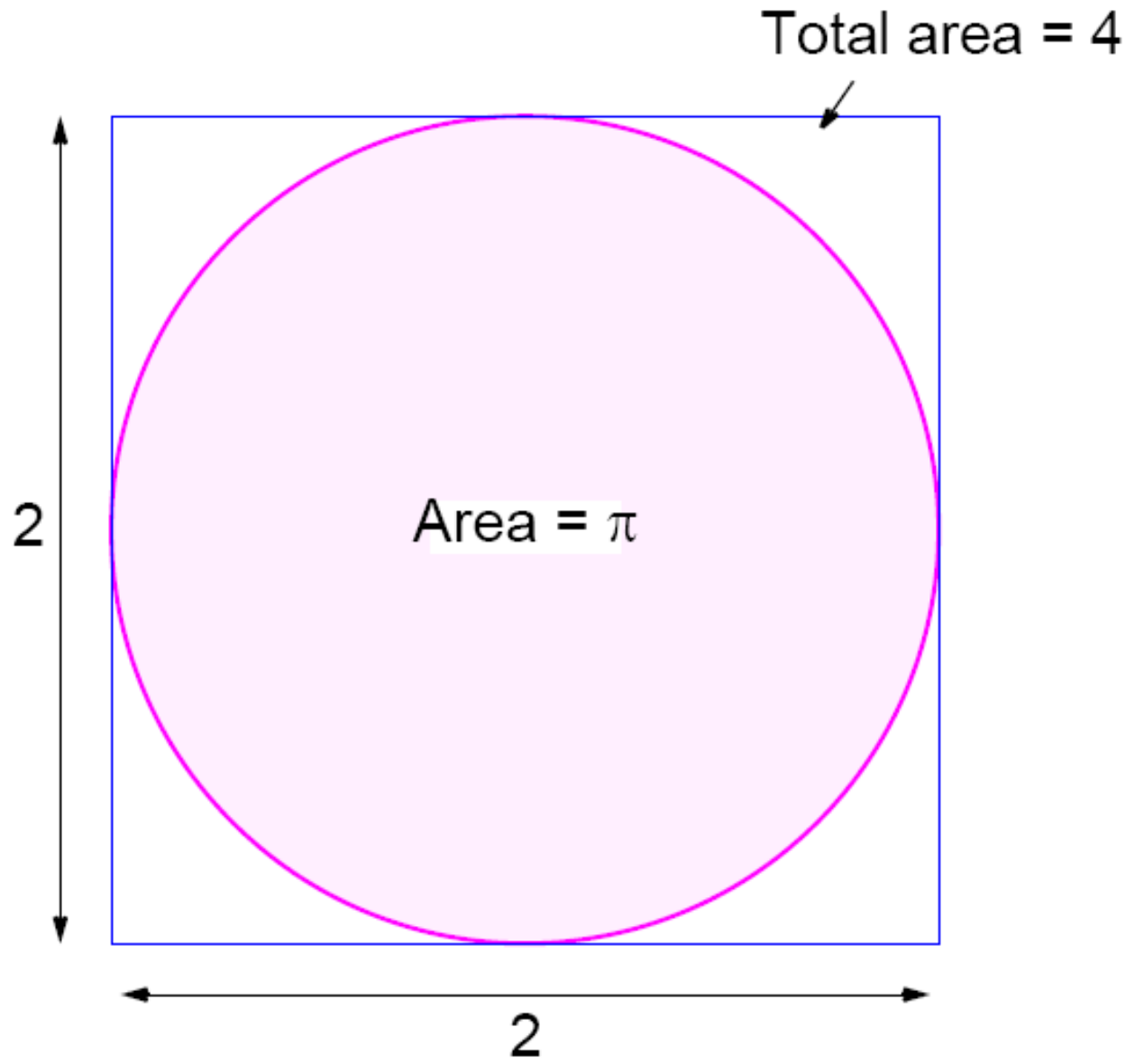
Example - To calculate π

Circle formed within a 2 x 2 square. Ratio of area of circle to square given by:

$$\frac{\text{Area of circle}}{\text{Area of square}} = \frac{\pi(1)^2}{2 \times 2} = \frac{\pi}{4}$$

Points within square chosen randomly. Score kept of how many points happen to lie within circle.

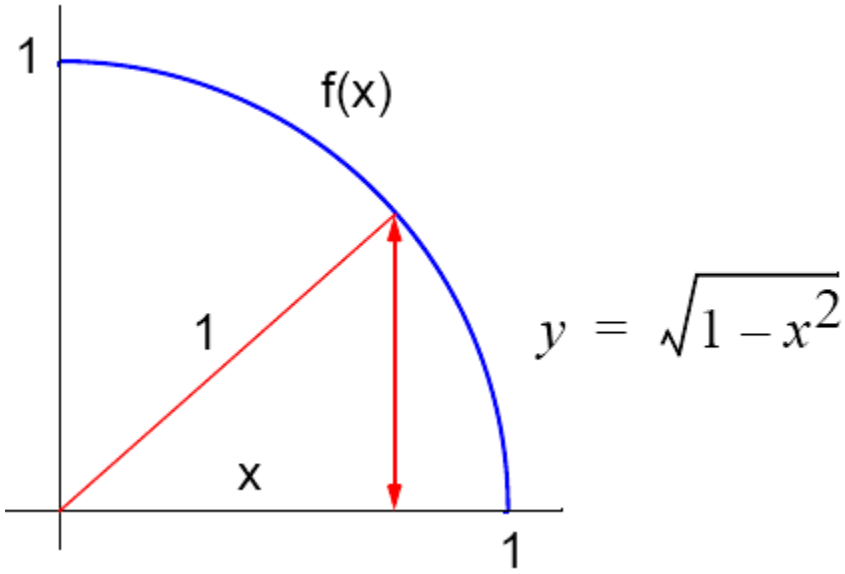
Fraction of points within circle will be $\pi/4$, given sufficient number of randomly selected samples.


Total area = 4
2
Area = π
2

Computing an Integral

One quadrant can be described by integral

$$\int_0^1 \sqrt{1-x^2}\,dx = \frac{\pi}{4}$$

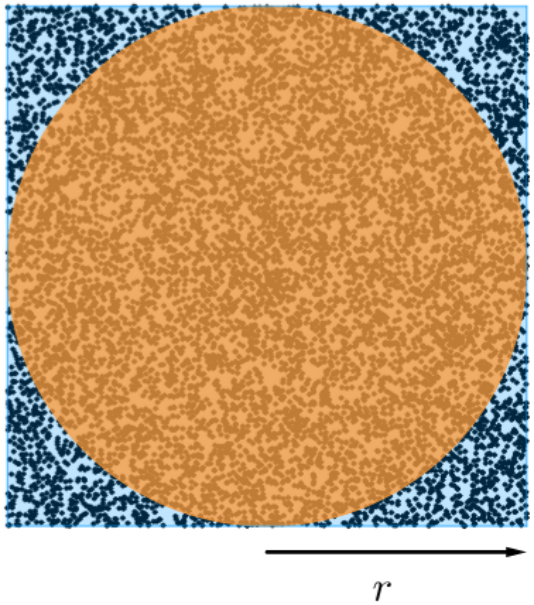


Random pairs of numbers, (x_r,y_r) generated, each between 0 and 1.

Counted as in circle if $y_r \le \sqrt{1-x_r^2}$; that is, $y_r^2 + x_r^2 \le 1$.

Demo

- https://www.geogebra.org/m/cGvXEE36



Start

Stop

Reset

$$\frac{Area\ of\ circle}{Area\ of\ square} = \frac{\pi r^2}{(2r)^2}$$
$$= \frac{\pi r^2}{4r^2}$$
$$= \frac{\pi}{4}$$

$$\frac{\pi}{4} = \frac{7868}{10000} = 0.7868$$

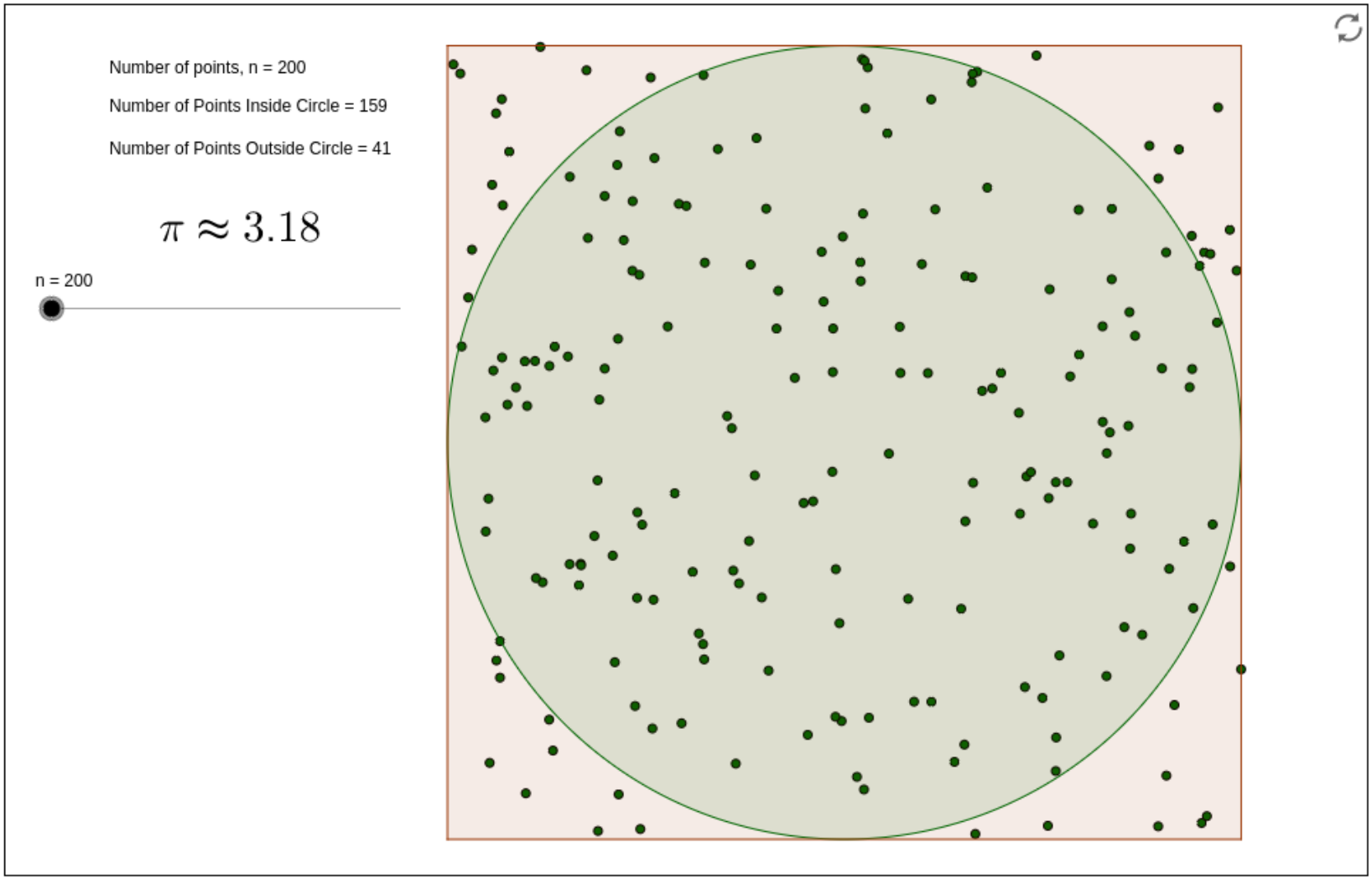
$$\pi = 3.1472$$

Error = 0.18%

https://www.geogebra.org/m/
xtCqCxKM

Monte Carlo Method to Approximate Pi

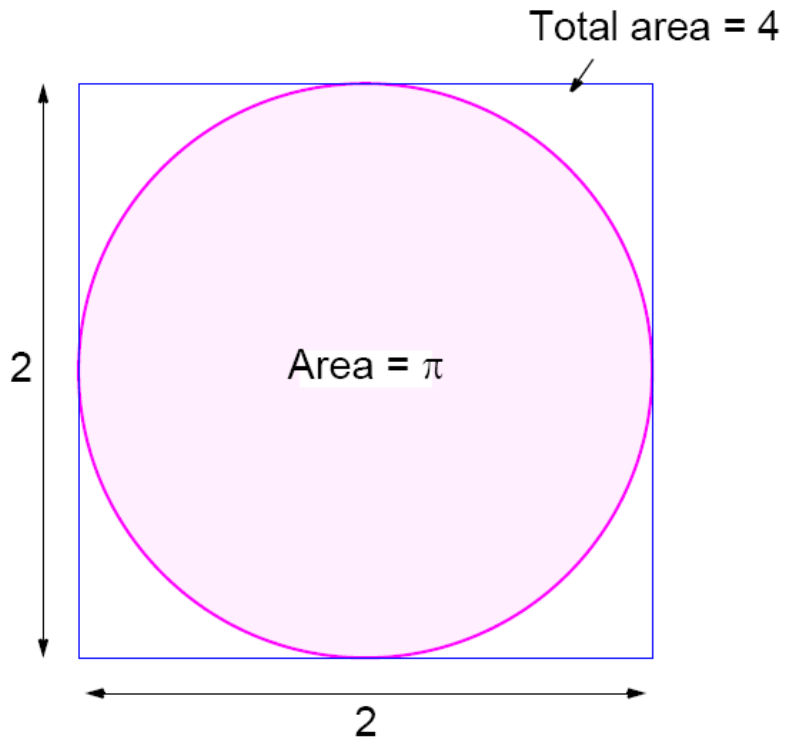
Move the slider to approximate the value of pi. (When the value of n gets large, the applet will slow down)

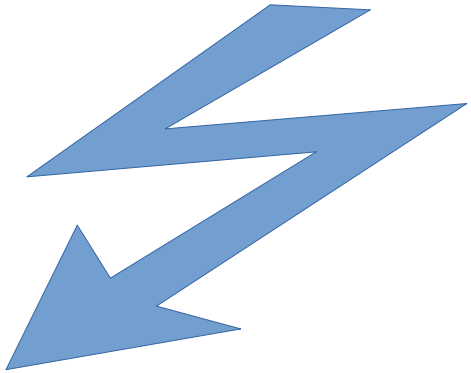


Master-Worker (M-W) Demo

- Embarrassingly Parallel Computing paradigm
- Calculation of π

$$\frac{\text{Area of circle}}{\text{Area of square}} = \frac{\pi(1)^2}{2 \times 2} = \frac{\pi}{4}$$





. A much better method is to use Leibniz's expansion of arctan(1):

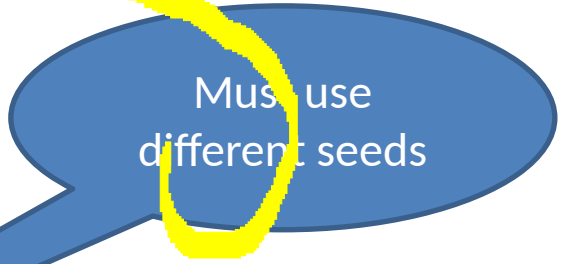
$$\frac{\pi}{4} = \sum_{k} \frac{(-1)^k}{2 * k + 1}$$

Pseudo Code - Serial Version

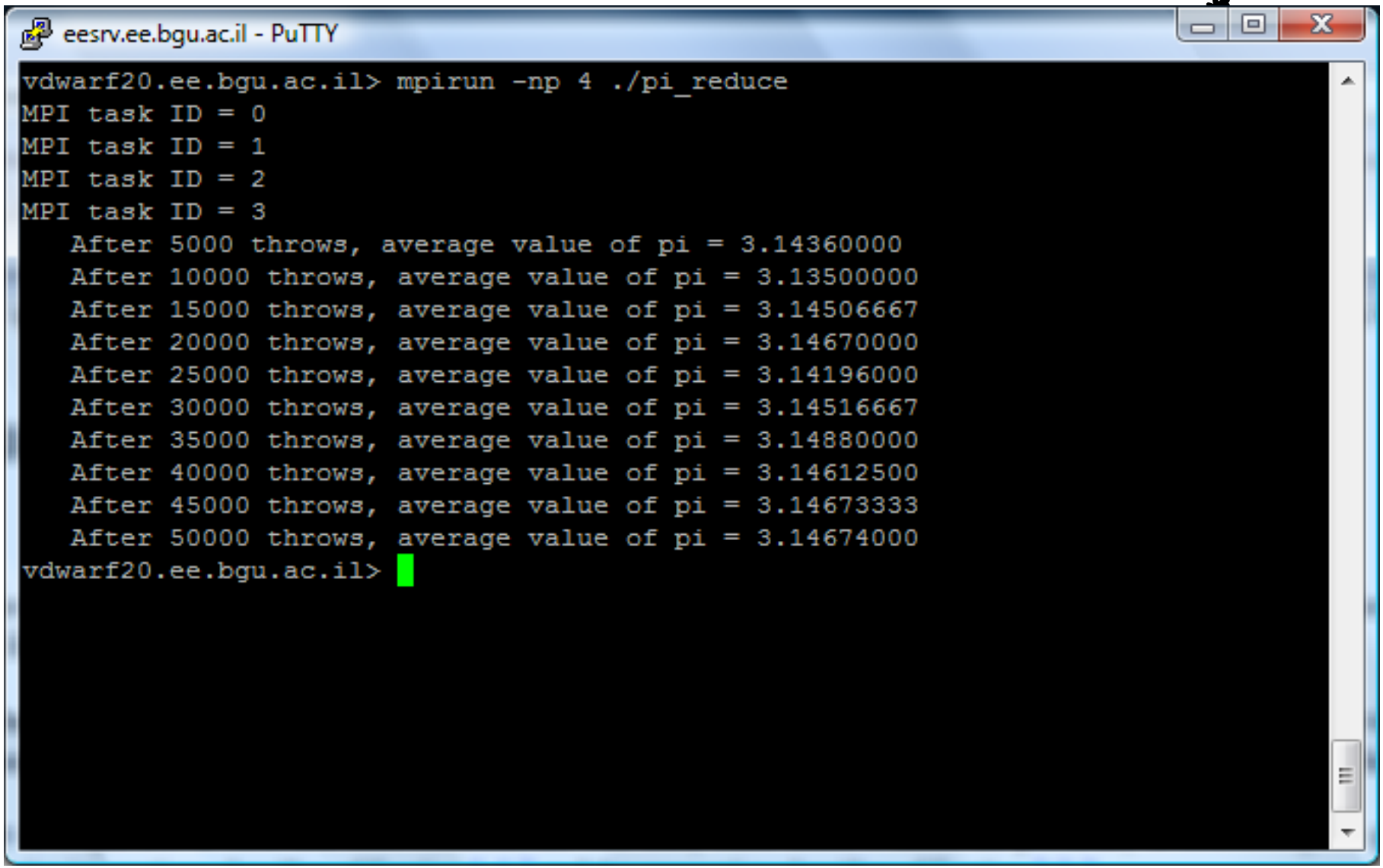
```
npoints = 10000
circle_count = 0
do j = 1,npoints
 generate 2 random numbers between 0 and 1
 xcoordinate = random1
 ycoordinate = random2
 if (xcoordinate, ycoordinate) inside circle
then
    circle_count = circle_count + 1
end do
PI = 4.0*circle_count/npoints
```

Pseudo Code - Parallel Version

```
npoints = 10000
circle_count = 0
p = number of tasks
num = npoints/p
find out if I am MASTER or WORKER
do j = 1,num
   generate 2 random numbers between 0 and 1
   xcoordinate = random1
   ycoordinate = random2
   if (xcoordinate, ycoordinate) inside circle then
      circle_count = circle_count + 1
end do
if I am MASTER
   receive from WORKERS their circle_counts compute PI
(use MASTER and WORKER calculations)
else if I am WORKER
   send to MASTER circle_count
endif
```



M – W Monte-Carlo calculation of π



- Reference to the source code: http://www.pdc.kth.se/training/Tutor/MPI/Templates/pi/index.html#top
- pi_send.c
- pi_reduce.c
- dboard.c
- make.pi.c

- However, I had to modify the scaling of random numbers – see next slide

0<r<1

- Instead of:

```
cconst = 2 << (31 - 1);
r = (double)random()/cconst;
```

- I had to change the code to:

```
r = ((double)rand() / ((double)
(RAND_MAX)+(double)(1)));
```

Alternative (better) Method

Use random values of x to compute $f(x)$ and sum values of $f(x)$:

$$\text{Area} = \int_{x_1}^{x_2} f(x)\,dx = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} f(x_r)(x_2 - x_1)$$

where x_r are randomly generated values of x between x_1 and x_2.

Monte Carlo method very useful if the function cannot be integrated numerically (maybe having a large number of variables)

Example

Computing the integral

$$I = \int_{x_1}^{x_2} (x^2 - 3x)\, dx$$

Sequential Code

```
sum = 0;
for (i = 0; i < N; i++) {                /* N random samples */
    xr = rand_v(x1, x2);                  /* generate next random value */
    sum = sum + xr * xr - 3 * xr;     /* compute f(xr) */
}
area = (sum / N) * (x2 - x1);
```

Routine randv(x1, x2) returns a pseudorandom number between x1 and x2.

For parallelizing Monte Carlo code, must address best way to generate random numbers in parallel - see textbook

Acceptance-rejection method (Von Neumann)

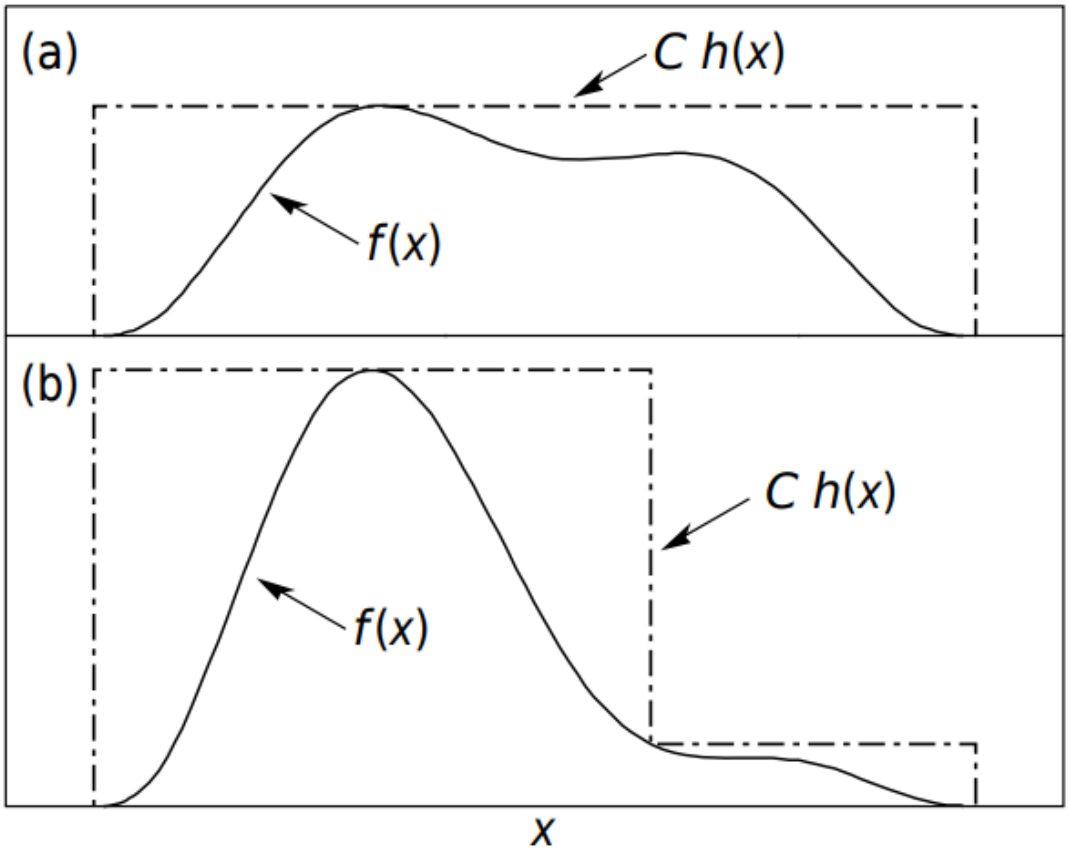


Figure 34.2: Illustration of the acceptance-rejection method. Random points are chosen inside the upper bounding figure, and rejected if the ordinate exceeds $f(x)$. The lower figure illustrates a method to increase the efficiency (see text).