# Objetivos

Desenvolver um pequeno aplicativo que possa demonstrar a utilização de sockets como meio de comunicação em sistemas distribuídos.

# Método de Monte Carlo

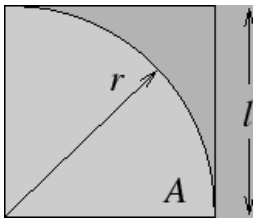
O método de Monte Carlo é uma forma de resolver problemas usando números aleatórios. O método explora as propriedades estatísticas dos números aleatórios para assegurar que o resultado correto é computado da mesma maneira que num jogo de casino para se certificar de que a "casa" sempre terá lucro. Por esta razão, a técnica de resolução de problemas é chamada de método de Monte Carlo.

Para resolver um problema através do método de Monte Carlo é usado uma série de tentativas aleatórias. A precisão do resultado final depende em geral do número de tentativas. Esse equilíbrio entre a precisão do resultado e o tempo de computação é uma característica extremamente útil dos métodos de Monte Carlo. Se queremos somente uma solução aproximada, então um método de Monte Carlo pode ser bastante rápido.

**Exemplo: computação de *pi***

Neste exemplo um algoritmo de Monte Carlo simples é usado para calcular o valor de π usando uma sequência de números aleatórios. Considere o quadrado colocado no plano x-y com a extremidade inferior esquerda na origem como mostrado na figura abaixo. A área do quadrado é $r^2$,

onde *r* é o comprimento do lado. Um quarto de um círculo é inscrito no quadrado. Seu raio é *r* e seu centro está na origem do plano x-y. A área da quarta parte do círculo é $\pi r^2/4$




Considerando A a área de $\frac{1}{4}$ de círculo parcialmente representado na figura acima, tem-se que:

$$= \frac{1}{4} * \pi * r^2$$

Pelo Método de Monte Carlo,

$$= l^2 * \frac{P_{in}}{P_{total}} \qquad (2)$$

sendo $P_{in}$ os pontos lançados aleatoriamente que ficaram dentro da área em cinza, e $P_{total}$ o número total de pontos gerados.

Unindo as equações 1, e 2 obtém-se:

$$\frac{1}{4} * \pi * r^2 = l^2 * \frac{P_in}{P_total}$$

$$\pi = \frac{l^2 * \frac{P_{in}}{P_{total}}}{\frac{1}{4} * r^2}$$

Considerando-se $l = r = 1$ obtém-se:

$$\pi = 4 * \frac{P_{in}}{P_{total}} \quad (3)$$

O algoritmo de cálculo consiste em gerar pontos $P(x, y)$ sendo $0 \leq x \leq 1$ e $0 \leq y \leq 1$ (vide figura 2). Identificar e contar os pontos $P_{in}$ e $P_{total}$, e aplicar a equação 3.

**Entrada:** número de iterações da execução em $n$
**Resultado:** valor estimado do $\pi$ em PI

1: **para** $P_{total} = 1$ to $n$ **faça**
2: $x \leftarrow$ número aleatório entre 0 e 1
3: $y \leftarrow$ número aleatório entre 0 e 1
4: **se** $(x^2 + y^2 \leq 1)$ **então**
5: $P_{in} \leftarrow P_{in} + 1$
6: **fim se**
7: **fim para**
8: PI $\leftarrow 4 * \frac{P_{in}}{P_{total}}$

**Figura:** Algoritmo de Cálculo do $\pi$

## *Implementação monoprocessada*

```
import random
import time
import sys

def pi(trials):
        hits = 0
        for i in xrange(trials):
                x = random.random()
                y = random.random()
                if (((x*x) + (y*y)) < 1):
                        hits = hits + 1
        return 4.0 * hits / trials
t0=time.time()
print pi(int(sys.argv[1]))
tf=time.time()
print 'O tempo gasto na execucao eh: ',tf-t0,'[s]'
```

# Versão distribuída

Considerando que o processamento de cada uma das iterações pode ocorrer de forma independente, a forma mais simples de se pensar em particionar o problema é dividindo igualmente o numero de iterações pelo número de processadores disponíveis. A medição do poder computacional disponibilizado é realizada com base no tempo de execução de cada tarefa.

## *Servidor*

```
import random #gerador de numeros aleatorios
from socket import * #sockets

#funcao que faz o calculo do numero de pontos
def pi(trials):
        hits = 0
        for i in xrange(trials):
                x = random.random()
                y = random.random()
                if (((x*x) + (y*y)) < 1):
                        hits = hits + 1
        return hits

HOST = ''
PORT = 2343

s = socket(AF_INET, SOCK_STREAM) #cria um socket
s.bind((HOST,PORT)) #conecta o socket na porta
s.listen(1) # fica esperando conexoes
print "Escutando conexoes"
while 1:
        conn, addr = s.accept() #recebe uma conexao
        data = conn.recv(1024) #recebe o dado enviado pelo cliente
        if data == "PING": #se o comando recebido foi um ping
                print "ping recebido"
        else:#senao a informacao recebida eh o numero de tentativas
                print "Calculando para o cliente "+str(addr[0])
                hits = pi(int(data)) #calcula o numero de pontos usando o numero
de tentativas recebidas do cliente
                print hits
                conn.send(str(hits)) #retorna ao cliente o numero de pontos
                conn.close() # fecha a conexao
```

## *Cliente*

```
from socket import *
import sys
import time
from threading import Thread

PORT = 2343

#subclasse da classe Thread. essa classe conecta com o servidor e envia os dados
para serem calculados
class cliente(Thread):
        def __init__ (self,ip,trials): #construtor da classe
                Thread.__init__(self) #chama o construtor da classe pai
                self.ip = ip #o ip que deve conectar
```

```
                self.hits = -1 #armazena o resultado
                self.trials = str(trials) #numero de tentativas
        def run(self): #essa eh a parte que serah executada
                print "thread do ip "+str(self.ip)+" iniciou"
                #abre a conexao com o servidor
                s = socket(AF_INET,SOCK_STREAM)
                s.connect((self.ip,PORT))
                #envia para o servidor o numero de tentativas
                s.send(self.trials)
                #fica esperando o resultado do calculo do servidor
                self.hits = s.recv(1024)
                s.close()
                print "trhread do ip "+str(self.ip)+" finalizou com o
resultado="+str(self.hits)

#funcao que verifica se os servidores estao ativos
def ativos(servidores):
        ativos = []#lista com os servidores ativos
        for ip in servidores:
                try:
                        print "testando "+str(ip)
                        s = socket(AF_INET,SOCK_STREAM)
                        s.connect((ip,PORT))
                        s.send("PING")#manda um comando ping para ver se o
servidor ainda esta ativo
                        s.close()
                        ativos.append(ip)
                except:
                        #print "Unexpected error:", sys.exc_info()[0]
                        pass
        return ativos

trials = sys.argv[1] #recebe do parametro do usuario o numero maximo de
tentativas
print "Verificando servidores ativos"
#verifica o numero de servidores ativos
servidores = ativos(['192.168.200.14', '200.135.240.1', '192.168.200.11',
'192.168.200.5'])
if len(servidores) == 0:
        print "Nao existem servidores ativos"
        sys.exit()
print "Servidores ativos:"+str(servidores)
#calcula o numero de tentativas que cada servidor deve executar
trials_por_servidor = int(trials) / len(servidores)
resultado = [] #lista com as threads em execucao
t0=time.time() #usado para o calculo do tempo de execucao

for ip in servidores:
        atual = cliente(ip, trials_por_servidor) #cria uma nova thread
        resultado.append(atual) #adiciona a thread na lista de threads
        atual.start() #inicia a thread

total_hits = 0 #total dos resultados
for r in resultado:
        r.join()#espera ate a thread terminar
        total_hits += int(r.hits)

#faz o calculo do pi usando os resultados enviados por cada servidor
pi = 4.0 * int(total_hits) / int(trials)

print "Pi :" + str(pi)
tf=time.time()
print 'O tempo gasto na execucao eh: ',tf-t0,'[s]'
```

# Resultados

## *Versão monoprocessada*

```
python monoprocessado.py 10000000
3.141672
O tempo gasto na execucao eh:  15.409099102 [s]
```

## *Versão distribuída*

### Com um servidor local

```
python server.py &
python cliente.py 10000000
Verificando servidores ativos
testando 192.168.200.14
testando 200.135.240.1
testando 192.168.200.11
testando 192.168.200.5
Servidores ativos:['192.168.200.14']
trhread do ip 192.168.200.14 iniciou
trhread do ip 192.168.200.14 finalizou com o resultado=7855196
Pi :3.1420784
O tempo gasto na execucao eh:  16.339785099 [s]
```

### Com um servidor remoto

```
python cliente.py 10000000
Verificando servidores ativos
testando 192.168.200.14
testando 200.135.240.1
testando 192.168.200.11
testando 192.168.200.5
Servidores ativos:['192.168.200.11']
thread do ip 192.168.200.11 iniciou
thread do ip 192.168.200.11 finalizou com o resultado=7854467
Pi :3.1417868
O tempo gasto na execucao eh:  24.9854791164 [s]
```

### Com dois servidores

```
python cliente.py 10000000
Verificando servidores ativos
testando 192.168.200.14
testando 200.135.240.1
testando 192.168.200.11
```

```
testando 192.168.200.5
Servidores ativos:['192.168.200.14', '192.168.200.11']
thread do ip 192.168.200.14 iniciou
thread do ip 192.168.200.11 iniciou
thread do ip 192.168.200.14 finalizou com o resultado=3928204
thread do ip 192.168.200.11 finalizou com o resultado=3925699
Pi :3.1415612
O tempo gasto na execucao eh:  12.5829679966 [s]
```

**Com três servidores**

```
python cliente.py 10000000
Verificando servidores ativos
testando 192.168.200.14
testando 200.135.240.1
testando 192.168.200.11
testando 192.168.200.5
Servidores ativos:['192.168.200.14', '192.168.200.11', '192.168.200.5']
thread do ip 192.168.200.14 iniciou
thread do ip 192.168.200.11 iniciou
thread do ip 192.168.200.5 iniciou
thread do ip 192.168.200.5 finalizou com o resultado=2617234
thread do ip 192.168.200.14 finalizou com o resultado=2618522
thread do ip 192.168.200.11 finalizou com o resultado=2617572
Pi :3.1413312
O tempo gasto na execucao eh:  8.77324795723 [s]
```

Observações: Como o método de monte carlo se baseia em geração de números aleatórios o tempo de execução pode variar, assim como a precisão do resultado. Mesmo assim foi possível visualizar um ganho de performance na versão distribuída quando foi adicionado mais máquinas. No teste da versão distribuída com somente um servidor remoto o resultado foi piór do que a versão monoprocessada pois agora temos o tempo de comunicação entre o cliente e o servidor que foi acrescentado.

# Trabalho

Desenvolver a versão cliente/servidor usando sockets em C.

# Referências


http://www.brpreiss.com/books/opus7/

http://www.inf.ufrgs.br/procpar/disc/cmp134/trabs/T2/021/html/index.html