Universidade de São Paulo

Prof. Dr. Julio Cezar Estrella

Sistemas Operacionais I

Gabriel de Andrade Dezan - 10525706

Ivan Mateus de Lima Azevedo - 10525602

Cálculo do número π

São Carlos - SP

2019

Introdução

Gauss-Legendre

O algoritmo de Gauss-Legendre é um algoritmo conhecido por ser rapidamente convergente: nas primeiras 25 iterações já calcula aproximadamente 45 milhões de dígitos corretos de π. Ele é baseado nos trabalhos dos matemáticos Johann Carl Friedrich Gauss e Adrien-Marie Legendre.

Ele funciona substituindo sucessivamente dois números por sua média aritmética e sua média geométrica. Fazendo isso, é possível aproximar a média aritmética-geométrica dos números.

A desvantagem deste algoritmo é que para que se consiga essa alta precisão, é necessário uma quantidade absurda de memória. Ou seja, é um algoritmo muito guloso. Por conta disso, não foram feitas 1 bilhão de iterações, por conta do longo tempo de execução, mas sim 100 mil.

Borwein

O algoritmo de Borwein foi desenvolvido pelos irmãos Jonathan e Peter Borwein. Ele funciona de forma que o valor que é aproximado é 1/π. A ordem de convergência deste algoritmo é quadrática.

Monte Carlo

O método Monte Carlo é um método que se baseia em repetições de amostras randômicas a ponto de de extrair um valor desconhecido. Existem inúmeras implementações desse método, em muitas áreas diferentes, como cassinos e finanças.

Black-Scholes

Um modelo que utiliza uma adaptação de fórmula de física para descrever a precificação de derivativos. No nosso caso é utilizada a simulação de Monte Carlo para se calcular o preço de opções européias utilizando o modelo de Black Scholes.

Implementação

Gauss-Legendre

- Versão sequencial:

  As variáveis são inicializadas com:

  $a_0 = 1$

  $b_0 = \frac{1}{\sqrt{2}}$

  $t_0 = 0.25$

  $p_0 = 1$

A cada iteração, a variável a é aproximada para sua média aritmética e a b, por sua média geométrica. As variáveis t e p são calculadas como se segue:

$$a_{n+1} = \frac{a_n+b_n}{2}$$

$$b_{n+1} = \sqrt{a_n b_n}$$

$$t_{n+1} = t_n - p_n(a_n - a_{n+1})^2$$

$$p_{n+1} = 2p_n$$

O valor de π é aproximado por:

$$\pi \approx \frac{(a_n+b_n)^2}{4t_n}$$

- Versão paralela:

Da mesma forma que na versão sequencial, as fórmulas e os valores iniciais são os mesmos. A diferença é que há duas threads: uma calcula os valores de a e b (thread_AB) e outra, os de t e p (thread_T).

A estratégia utilizada foi de um buffer circular de tamanho igual a $0.3N$ (esse valor foi definido através de vários testes), dessa forma, o problema se resumiu a um problema de produtor/consumidor em que o produtor é a thread_AB e o consumidor

é a thread_T. Os valores que são guardados no buffer são os de a_{n+1} e a_n. Assim, a cada operação de inserção no buffer, são inseridas duas variáveis por vez e a cada operação de remoção, duas variáveis são retiradas por vez.

Tempo de execução do algoritmo com 100.000 iterações:

Sequencial: 53,28s

Paralelo: 54,34s

Borwein

- Versão sequencial:

  As variáveis iniciais são:

  $$a_0 = 2(\sqrt{2} - 1)^2$$

  $$y_0 = \sqrt{2} - 1$$

  A cada iteração, a e y são aproximados por:

  $$a_{n+1} = a_n(1 + y_{n+1})^4 - 2^{2n+3} y_{n+1}(1 + y_{n+1} + {y_{n+1}}^2)$$

  $$y_{n+1} = \frac{1-(1-{y_n}^4)^{\frac{1}{4}}}{1-(1+{y_n}^4)^{\frac{1}{4}}}$$

  Por fim, π é aproximado por:

  $$\pi \approx \frac{1}{a_n}$$

- Versão paralela:

Na versão paralela, os valores iniciais e as equações utilizadas também são os mesmos, assim como no algoritmo anterior. E da mesma forma também foi utilizada a abordagem de produtor/consumidor.

Nesse caso a thread produtora é a thread_Y (que calcula os valores de y) e a consumidora, a thread_A (calcula os valores de a). O tamanho do buffer é o mesmo utilizado anteriormente, visto que empiricamente foi o melhor valor de tamanho encontrado.

Tempo de execução do algoritmo com 100.000 iterações:

Sequencial: 12,18s

Paralelo: 12,87s

Monte Carlo

- Versão sequencial:

  O método de Monte Carlo foi usado para calcular o valor de π seguinte forma:

Seja uma circunferência de raio 0,5 que se está inscrita em um quadrado de lado 1. Ela possui uma área com valor $\pi \cdot r^2 = \pi/4$, enquanto o quadrado tem uma área de valor 1.

Com essa informação, geramos vários pontos aleatórios na área do quadrado e conferimos se os pontos se encontram dentro da circunferência ou não.

Se dividirmos o total de pontos que se encontram no círculo pelo total de pontos gerados, encontraremos uma razão que se aproxima da razão entre as áreas da circunferência e do quadrado.

Logo:

$$\frac{\pi}{4} \approx \frac{n_{circulo}}{n_{total}}$$

No código, em cada iteração eram gerados dois números aleatórios entre 0 e 1, que seriam as coordenadas x e y do ponto.

Em seguida, era calculada a distância entre o ponto e o centro da circunferência. Se a distância for menor ou igual a 1, significa que o ponto se encontra dentro dela e, então, a variável `circle` (contagem de pontos dentro da circunferência) era incrementada em 1.

No fim de cada iteração, a variável `square` (contagem de pontos dentro do quadrado) também era incrementada em 1 pois todo ponto está dentro do quadrado.

Por fim π era calculado utilizando a fórmula apresentada anteriormente.

- Versão paralela:

A versão paralela fazia as mesmas operações, porém, como as variáveis square e circle poderiam ser acessadas por threads diferentes, gerando conflito, cada thread teve sua própria variável `square` e `circle`, que foram somadas ao fim, quando as threads foram reunidas.

O cálculo final foi realizado em apenas uma thread, com os valores finais de `square` e `circle` já estabelecidos.

Tempo de execução do algoritmo com 300.000.000 iterações:

Sequencial: 38,44s

Paralelo: 17,67s

Monte Carlo

- Versão sequencial:

O algoritmo recebe essas variáveis de entrada:

S: valor da ação

E: preço de exercício da opção

r: taxa de juros livre de risco (SELIC)

σ: volatilidade da ação

T : tempo de validade da opção

M : número de iterações

Como exibido na Especificação do Trabalho, esse é o pseudo-código do modelo de Black-Scholes utilizando o método de Monte Carlo

Pseudo-código do algoritmo de Black Scholes com Monte Carlo

1: **for** $i = 0$ to $M - 1$ **do**
2: $t := S \cdot \exp\left((r - \frac{1}{2}\sigma^2) \cdot T + \sigma\sqrt{T} \cdot \text{randomNumber}()\right)$ ▷ t é uma variável temporária
3: trials [i] $:= \exp(-r \cdot T) \cdot \max\{t - E, 0\}$
4: **end for**
5: mean $:= mean($ trials $)$
6: stddev $:= stddev($ trials , mean$)$
7: confwidth $:= 1.96 \cdot \text{stddev}/\sqrt{M}$ ▷ Cálculo do intervalo de confiança
8: confmin := mean − confwidth
9: confmax := mean + confwidth

Adaptado de http://www.cs.berkeley.edu/~ydlick/cs194f07

E ele foi implementado dessa forma no método sequencial, com adaptações para a linguagem C.

- Versão paralela:

A diferença da versão paralela é que o loop é dividido pelo número de threads, então as operações ocorrem simultaneamente, o que diminui o número de iterações aparentes.

Tempo de execução do algoritmo com 5.000.000 iterações:

Sequencial: 19,92s

Paralelo: 08,05s

Compilação

Os códigos devem, naturalmente, serem compilados uma primeira vez antes de serem executados. Algumas bibliotecas utilizadas, como PThreads e GMP já fazem parte das bibliotecas do compilador gcc. Por esse motivo e pelo fato dele ser o compilador de C mais amplamente usado, optamos por usá-lo na compilação dos nossos códigos.

Foi criado um Makefile para o auxílio da compilação, portanto, para compilar todos os códigos, basta executar o comando abaixo

```
make all
```

Caso o usuário queira, por exemplo, compilar apenas o arquivo borwein_paralelo, ele precisa rodar:

```
make borwein_paralelo
```

Para compilar apenas usando o gcc, sem usar o Makefile, o usuário pode seguir a seguinte referência:

Borwein: `gcc -o borwein borwein.c -lgmp`

Borwein Paralelo: `gcc -o borwein_paralelo borwein_paralelo.c -lgmp -lpthread`

Gauss-Legendre: `gcc -o gauss-legendre gauss-legendre.c -lgmp`

Gauss-Legendre Paralelo: `gcc -o gauss-legendre_paralelo gauss-legendre_paralelo.c -lgmp -lpthread`

Black-Scholes: `gcc -o black-scholes black-scholes.c rand_bm.c -lm -lgmp -lmpfr`

Black-Scholes Paralelo: `gcc -o black-scholes_paralelo black-scholes_paralelo.c rand_bm.c -lm -lgmp -lpthread -lmpfr`

Monte-Carlo: `gcc -o monte-carlo monte-carlo.c rand_bm.c -lm -lgmp`

Monte-Carlo Paralelo: `gcc -o monte-carlo_paralelo monte-carlo_paralelo.c rand_bm.c -lm -lgmp -lpthread`

Execução

Após os arquivos serem compilados, eles podem ser executados. Para executar, por exemplo, o algoritmo de Gauss-Legendre paralelo, basta executar:

```
./gauss-legendre_paralelo
```

Porém, como estaremos também fazendo a contagem do tempo, é importante utilizar o comando:

```
/usr/bin/time -f “%e” ./gauss-legendre_paralelo
```

Observação: O algoritmo de Black-Scholes, tanto sequencial como paralelo, precisa de um arquivo de entrada, com os valores necessários. Para executá-lo:

```
/usr/bin/time -f “%e” ./gauss-legendre_paralelo <
            entrada_blackscholes.txt
```

Hardware

Processor: Intel Core i7-3770 @ 3.40GHz (8 cores)

Memory: 16384MB

Disk: 447GB

Software

OS: Ubuntu 18.04
Kernel: 4.4.0-18362-Microsoft (x86_64)
File-System: wslfs
System Layer: wsl