Métodos sequeciais e paralelos para cálculo do número Pi

Leonardo Alves Paiva[1], Rafael Pastre[1]

NUSP: *10276911*, *9783675*

[1] Escola de Engenharia de São Carlos / Instituto de Ciências Matemáticas e Computação – Universidade de São Paulo (USP)
São Carlos – SP – Brazil

{leonardoalvespaiva,rafael.pastre}@usp.br

Resumo. *Este relatório descreve o trabalho implementado na disciplina de Sistemas Operacionais I do curso de Engenharia de Computação da USP - São Carlos. É tratada a implementação de três métodos diferentes para cálculo do número pi: Gauss-Legendre, Borwein e Monte Carlo. Além desses, há o algoritmo de Black Sholes para cálculo de opções europeias de ações, o qual também faz uso do método de Monte Carlo. Todos os métodos foram feitos de forma sequencial e paralela para posterior comparação.*

Sumário

1. Algoritmos

1.1. Gauss-Legendre

O algoritmo de Gauss-Legendre foi desenvolvido pelos trabalhos individuais de Carl Friedrich Gauss (1779-1815) e Adrien-Marie Legendre (1799-1855). Seu intuito é o cálculo do número π (pi), o que faz com grande notabilidade por ser rapidamente convergente, produzindo 45 milhões de dígitos corretos em 25 iterações.

Parte-se dos valores iniciais:

$$a_0 = 1 \qquad b_0 = \frac{1}{\sqrt{2}} \qquad t_0 = \frac{1}{4} \qquad p_0 = 1.$$

E a cada iteração:

$$\begin{aligned} a_{n+1} &= \frac{a_n + b_n}{2}, \\ b_{n+1} &= \sqrt{a_n b_n}, \\ t_{n+1} &= t_n - p_n(a_n - a_{n+1})^2, \\ p_{n+1} &= 2p_n. \end{aligned}$$

Por fim, o valor de π é aproximado por:

$$\pi \approx \frac{(a_{n+1} + b_{n+1})^2}{4t_{n+1}}.$$

1.2. Borwein

O algoritmo foi desenvolvido por Jonathan e Peter Borwein e calcula o valor de π. O algoritmo implementado neste trabalho foi o iterativo de convergência quadrática.

Seus valores iniciais são:

$$\begin{aligned} a_0 &= \sqrt{2} \\ b_0 &= 0 \\ p_0 &= 2 + \sqrt{2} \end{aligned}$$

E a cada iteração:

$$\begin{aligned} a_{n+1} &= \frac{\sqrt{a_n} + 1/\sqrt{a_n}}{2} \\ b_{n+1} &= \frac{(1 + b_n)\sqrt{a_n}}{a_n + b_n} \\ p_{n+1} &= \frac{(1 + a_{n+1})\, p_n b_{n+1}}{1 + b_{n+1}} \end{aligned}$$

Ao final, *p* converge quadraticamente para π, ou seja, a cada iteração os dígitos corretos dobram.

1.3. Monte Carlo

O termo Monte Carlo é usado para definir qualquer método de uma classe de métodos estatśticos que fazem amostragens aleatórias para obterem resultados numéricos. São feitas diversas simulações a fim de se obter resultados baseados em probabilidades. O método de Monte Carlo é amplamente utilizado na matemática, física e biologia.

Para o cálculo do número π, o método de Monte Carlo implementado funciona sorteando as coordenadas x e y com valores entre 0 e 1, o que limita os pontos a estarem em um quadrado de lado unitário, no primeiro quadrante e com vértice na origem. Então veirfica-se se o ponto sorteado estão dentro do primeiro quadrante do círculo unitário $x^2 + y^2 < 1$. Se sim, soma-se um ao total de acertos. Para amostragens massivas, a probabilidade de os pontos sorteados caírem no círculo tende a áres do quarto de círculo sobre o a área do quadrado: $(\pi / 4)$.

1.3.1 Black Scholes

Apesar de não ter sido explicitamente pedido a implementação deste algoritmo, resolveu-se implementá-lo por ser uma aplicação prática do método de Monte Carlo e por contribuir para a aprendizagem.

O modelo de Black-Scholes é muito usado para determinar preços de derivativos de ações, no caso opções europeias de ações. Foi desenvolvido na década de setenta por Fisher Black, Robert Merton e Myron Scholes. A fórmula para o cálculo baseia-se em algumas suposições, entre elas:

- Sem custos de transação;
- A ação não paga dividendos;
- O preço segue um movimento *Browniano;*
- Negociação contínua;
- Taxa juros livre de risco constante e igaual a todos.

Seus parâmetros de entrada são:

- S: valor da ação
- E: preço de exercício da opção
- r: taxa de juros livre de risco (SELIC)
- σ (ou v ou vol): volatilidade da ação
- T : tempo de validade da opção
- M : número de iterações

E sua saída é o intervalo de confiança que o preço do derivativo pode estar.

2. Implementação e Soluções

2.1. Gauss-Legendre

Analisando o algoritmo de Gauss-Legendre, percebe-se que em cada iteração do método as atualizações das variáveis “a”, “b”, e “p” são independentes, enquanto a atualização da variável “t” depende de uma atualização prévia da variável “a”. Dessa forma, para paralelizar este método foram utilizadas 3 threads. A primeira atualiza as variáveis “a” e “t” em sequência, a segunda atualiza a variável “b” , por fim, a terceira atualiza a variável “p”. Vale ressaltar ainda que, utilizar uma quarta thread pra atualizar unicamente “t” não faz sentido nesse cenário, devido a natureza sequencial da atualização de “a” e “t”, e, além disso, tal paralelização apenas causaria dificuldades de implementação e perda de desempenho, devido ao fato que estas variáveis nunca serão atualizadas ao mesmo tempo, e pela necessidade de adicionar mecanismos para sincronização destas variáveis.

Outro ponto importante sobre a paralelização deste algoritmo é o sincronismo necessário para o seu funcionamento. Em uma iteração do método necessita-se de que todas as variáveis estejam atualizadas previamente, ou seja, não se pode atualizar uma variável mais de uma vez sem que todas as outras estejam atualizadas. Dessa forma, na implementação deste método utilizou-se uma “barreira”, implementada pela biblioteca “pthread”, como mecanismo de sincronização das threads. Seu funcionamento consiste em definir um ponto no qual nenhuma thread pode avançar até que a execução de todas as theads alcance este ponto. Assim, cada thread foi divida com barreiras em duas partes, uma para calcular o novo valor das variáveis correspondentes, e outra para atualizar o valor das variáveis, garantindo dessa forma, que nenhuma variável seja sobrescrita até que todos os cálculos de uma iteração realizados.

Na implementação do método de Gauss-Legendre vale ressaltar também que conforme as iterações aumentam, os valores das variáveis “a”, “b”, “p”, e “t” podem ultrapassar os valores que podem ser representados por uma variável do tipo double padrão, por isso, foi utilizada a biblioteca GMP para realizar o cálculo destas variáveis, possibilitando assim que estas atingissem valores maiores, e que o algorítimo alcançasse um maior número de iterações. Entretanto, foi observado que um aumento de precisão de 10 vezes nas variáveis do tipo “mpf” causava um aumento de aproximadamente 10 vezes no tempo de execução, e por este motivo, o número de iterações ficou limitado em 100000, visto que um aumento das iterações para 1000000 aumentaria o tempo de execução em 100 vezes, considerando o aumento de iterações e o aumento da precisão dos números.

2.2. Borwein

A paralelização do método de Borwein é semelhante a do método de Gauss-Legendre, uma vez que, em uma iteração deste método as atualizações das variáveis “a” e “b” são independentes, enquanto a atualização da variável “p” depende das atualizações das variáveis “a” e “b”. Dessa forma, para paralelizar este método, foram utilizadas 2 threads. A primeira atualiza as variáveis “a” e “p”, e a segunda atualiza a variável “b”. Vale ressaltar ainda que, utilizar uma terceira thread pra atualizar

unicamente “p” não faz sentido nesse cenário, pois, semelhante ao método de Gauss-Legendre, a natureza atualização de “t” é sequencial, além disso, tal paralelização apenas causaria dificuldades de implementação e perda de desempenho, devido ao fato destas variáveis nunca serem atualizadas ao mesmo tempo, e da necessidade de adicionar mecanismos para sincronização das variáveis.

A sincronização deste algoritmo também foi feita de forma semelhante a do método de Gauss-Legendre, utilizando barreiras para dividir cada thread em duas regiões. Entretanto, como a atualização de “p” depende de ambas “a” e “b”, o cálculo de “p” foi deixado para a segunda região do código, garantindo assim que “p” seja calculado após a atualização de “a” e “b”. Ou seja, as threads foram organizadas da seguinte maneira: Na primeira região, a primeira thread calcula o novo valor de “a” e a segunda thread calcula o segundo valor de “b”. Na segunda região, a primeira thread calcula o novo valor de “p” e atualiza “a” e “p” em sequência, e a segunda thread atualiza “b”.

Na implementação do método de Borwein, assim como no de Gauss-Legendre, os valores das variáveis “a”, “b”, e “p” também podem ultrapassar os valores que podem ser representados por uma variável do tipo double padrão, por isso, foi utilizada a biblioteca GMP para realizar o cálculo destas variáveis, possibilitando assim que estas atingissem valores maiores, e que o algorítimo alcançasse um maior número de iterações. Entretanto, foi observado que um aumento de precisão de 10 vezes nas variáveis do tipo “mpf” causava um aumento de aproximadamente 10 vezes no tempo de execução, e por este motivo, o número de iterações ficou limitado em 100000, visto que um aumento das iterações para 1000000 aumentaria o tempo de execução em 100 vezes, considerando o aumento de iterações e o aumento da precisão dos números.

2.3. Monte Carlo

A paralelização do método de Monte Carlo é a mais simples quando comparada com os outros métodos. Como este algorítimo utiliza pontos aleatórios e independentes, as iterações totais do método podem ser divididas igualmente para cada thread, e o valor final somado ao fim da execução das threads. Vale ressaltar também que a implementação desta abordagem é vantajosa pois não necessita da sincronização entre threads e também torna genérico o número de threads utilizadas. O único cuidado que deve ser tomado é que a função que gera os números aleatórios não pode utilizar um “recurso limitado”, pois isto impede a execução de múltiplas threads ao mesmo tempo. Durante os testes com a função rand() padrão do C, foi verificado que este era o caso para esta função. Visando contornar este problema, foi implementada uma função pseudo-aleatória customizada, de forma que o próximo valor dependesse apenas do valor anterior, e que este valor fosse armazenado em cada thread.

2.3.1 Black Scholes

A paralelização do método de Black Scholes é feita da mesma forma que a de Monte Carlo, as iterações são divididas igualmente entre as threads, garantindo que cada thread execute um número menor de iterações. Está implementação também foi feita de forma que o número de threads utilizadas fosse generalizado, e também utiliza a mesma função randômica implementa anteriormente.

A implementação do algoritmo de Black Scholes utiliza ainda a média e o desvio padrão dos pontos aleatórios gerados, o que causa uma dificuldade na implementação, pois para calcular o desvio padrão de todos os pontos de um conjunto, seria necessário a utilização de um vetor que armazenasse todos estes pontos, e que portanto causaria problemas de alocação de memória conforme o número de pontos aumentasse. Para solucionar este problema, os cálculos da média e variância foram divididos em grupos de pontos, reduzindo dessa forma a memória necessária. Vale também justificar que esta abordagem é válida, pois, considerando subconjuntos de mesmo tamanho, a média de todo o conjunto é a média das médias de cada subconjunto, e, como pode-se aproximar as médias de cada subconjunto como sendo igual a média total, a variância de todo o conjunto pode ser calculada através da média das variância de cada subconjunto, e, finalmente, obtém-se o desvio padrão pela raiz quadrada da variância.

3. Experimento

3.1. Especificação do hardware

A especificação do hardware e sistema operacional foi obtida através do software *Phoronix Test Suite v5.2.1* com o seguinte comando: *phoronix-test-suite system-info*. A especificação obtida foi a seguinte:

Hardware:

Processor: Intel Core i3-2330M @ 2.20GHz (4 Cores), Motherboard: Acer JE50_HR, Chipset: Intel 2nd Generation Core Family DRAM, Memory: 6144MB, Disk: 320GB SAMSUNG HM321HI, Graphics: Intel HD 3000 (1100MHz), Audio: Realtek ALC269VB, Network: Broadcom and subsidiaries NetLink BCM57785 Gigabit PCIe + Qualcomm Atheros AR9287 Wireless

Software:

OS: Ubuntu 16.04, Kernel: 4.4.0-151-generic (x86_64), Desktop: GNOME Shell 3.20.4, Display Server: X Server 1.18.4, Display Driver: intel 2.99.917, OpenGL: 3.3 Mesa 18.0.5, Compiler: GCC 5.4.0 20160609, File-System: ext4, Screen Resolution: 1366x768

3.2. Metodologia

Para executar os programas na máquina especificada anteriormente, procurou-se evitar todos os processos em segundo plano, deixando somente os programas executando. Procurou-se usar executar somente o código avaliado e recursos essenciais do sistema operacional. Usou-se o maior número de iterações possíveis (no caso 10^5) para se ter uma avaliação e uma comparação de resultados satisfatórias. O grande fator limitante foi o tempo, pois, com o aumento iterações, o tempo de execução dos programas ficaria inviável podendo atravessar dias.

Os programas foram compilados e executados conforme especificado na próxima seção.

3.3. Execução dos códigos

Instruções para compilação dos programas:

Códigos sequenciais:

- Gauss-Legendre:

```
gcc gauss_legendre.c -o gauss_legendre -lm -lgmp
```

- Borwein:

```
gcc borwein -o borwein -lm -lgmp
```

- Monte Carlo:

```
gcc monte_carlo.c -o monte_carlo -lm
```

- Black-Scholes:

black_scholes.c -o black_scholes -lm

Códigos paralelos:

- Gauss-Legendre:

  gcc gauss_legendre.c -o gauss_legendre -lm -lgmp -lpthread

- Borwein:

  gcc borwein -o borwein -lm -lgmp -lpthread

- Monte Carlo:

  gcc monte_carlo.c -o monte_carlo -lm -pthread

- Black-Scholes:

  black_scholes.c -o black_scholes -lm -pthread

Instruções para execução dos programas:

- Gauss-Legendre:

  time ./gauss_legendre

- Borwein:

  time ./borwein

- Monte Carlo:

  time ./monte_carlo

- Black-Scholes:

  time ./black_scholes

Para a excecução do metodo de Black-Scholes, é necessária a presença do arquivo entrada_blackscholes.txt no seguinte formato:

```
S  <valor_de_S>
E  <valor_de_E>
r  <valor_de_R>
v  <valor_de_v>
T  <valor_de_T>
M  <valor_de_M>
```

4. Resultados

Os algoritmos de Gauss-Legendre e Borwein por necessitarem de muita memória e tempo, tiveram suas iterações limitadas a 10^5, enquanto o cálculo por Monte Carlo e de Black Sholes tiveram suas iterações definidas em 10^9.

Sequencial:

Gauss-Legendre	Borwein	Monte Carlo	Black Scholes
116,61	674,63	34,55	109,64

Tabela 1. Tempo da execução dos programas sequenciais (em segundos)

Gauss-Legendre	Borwein	Monte Carlo
3,141593	3,141593	3,141590

Tabela 2. Resultado da execução dos programas sequenciais que calculam π

Paralelo:

Gauss-Legendre	Borwein	Monte Carlo	Black Scholes
129,60	560,84	19,24	40,49

Tabela 3. Tempo da execução dos programas paralelos (em segundos)

Gauss-Legendre	Borwein	Monte Carlo
3,141593	3,141593	3,158397

Tabela 4. Resultado da execução dos programas paralelos que calculam π

Speedup Sequencial/Paralelo:

Gauss-Legendre	Borwein	Monte Carlo	Black Scholes
0,9	1,20	1,80	2,71

5. Conclusão

Como pode-se ver nos resultados, nem sempre a paralelização leva a execuções mais otimizadas, como no caso de Gauss-Legendre, já que o algoritmo possui natureza sequencial e uma versão paralela pode gastar muito tempo com sincronização entre as threads, o que invibializaria um possível ganho. Em outros algoritmos, a paralelização aumenta o desempenho, como no algoritmo de Borwein, que teve um ganho em torno de 20%.

Algoritmos, altamente paralelizáveis, como os dos métodos de Monte Carlo para cálcuo de pi e de Black Sholes, apresentam ganho significativamente maior, já que cada iteração é independente das demais. Sendo que cada uma poderia ocupar uma thread. Nestes casos, o ganho por paralelismo é fortemente limitada pelo número de núcleos de processamento que podem dividir o trabalho no sistema computacional. Embora se ressalte que devido às características destes métodos, o resultado pode não ser tão preciso quanto os demais, ainda que sejam mais fáceis de calcular.

Por fim, é importante notar que o paralelismo, quando bem aplicado, pode resultar em grandes vanteagens na questão de desempenho para o usuário, ainda mais se for levado em consideração a forte presença de processadores *multi-cores* nos computadores pessoais atualmente, refletindo diretamente no usuário final.

Referências

https://pt.wikipedia.org/wiki/Algoritmo_de_Gauss-Legendre

https://pt.wikipedia.org/wiki/Algoritmo_de_Borwein

http://olaria.ucpel.tche.br/pdist/lib/exe/fetch.php?media=pi-monte-carlo.pdf

https://pt.wikipedia.org/wiki/M%C3%A9todo_de_Monte_Carlo

https://www.sunoresearch.com.br/artigos/black-scholes/

https://pt.wikipedia.org/wiki/Black-Scholes

https://gmplib.org/