Trabalho Final
Sistemas Operacionais

Bruno Feitosa[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo

feitosa.bruno@usp.br

Abstract. *This paper discuss and compares three different methods to calculate π, one method to calculate Call prices on Stock Market options, and how these methods can use POSIX PThreads to optimize these methods through parallelization.*

Resumo. *Este artigo discute e compara três métodos diferentes para calcular π, um método para calcular o valor de opções Call no Mercado de Ações, e como estes métodos podem utilizar POSIX PThreads para otimizar estes métodos através de paralelização.*

1. Resumo dos Algoritmos

A seguir serão brevemente descritos os algoritmos usados neste trabalho. Vale notar, que devido a estes algoritmo utilizarem números com tamanhos e precisões maiores do que as padrões da linguagem de implementação, foram utilizadas duas bibliotecas, a GNU Multiple Precision Library e a GNU Multiple Precision Floating point with correct Rounding Library.

1.1. Cálculo do π - Método Gauss-Legendre

Este algoritmo é uma aplicação da regra de quadratura Gauss-Legendre, para aproximação numérica de uma integral definida. O algoritmo se baseia em substituir dois números por suas médias aritméticas e geométricas, para aproximá-las a média geométrica-aritméticas. Para o cálculo do valor de π, é usada a função trigonométrica do sen e o ângulo de $\pi/4$.

1.2. Cálculo do π - Método Bailey-Borwein-Plouffe

O método Bailey-Borwein-Plouffe é baseado na fórmula BBP para o cálculo do π [David H. Bailey and Plouffe 1997]. Sua vantagem está no fato de que todos os fatores da fórmula são múltiplos de 8, o que acelera a computação em sistemas hexadecimais.

1.3. Cálculo do π - Método Monte Carlo

O método de Monte Carlo é um método estocástico, que relaciona a área de um quadrado com lado $2.R$ e a área de um círculo inscrito nele com raio R. Como a relação entre estas áreas é π, a seleção aleatória de pontos dentro do quadrado tem $\pi/4$ vezes mais chance de caírem dentro do círculo do que fora dele.

1.4. Cálculo de Preços de Opções - Método Black-Scholes

O método Black-Scholes é utilizado para se calcular Opções (*Call* e *Put*) de Ações. Opções são derivativos de Ações, ou seja, são Ativos que dependem de um outro Ativo, que nesse caso, é a Ação de referência da Opção. Como opções agem como contratos de Compra/Venda dos Ativos de referência, seus preços estão ligados ao preço dos Ativos de referência e ao estado do mercado em si. Como sua precificação não é tão direta como a negociação de Ações no Mercado, método como o de Black-Scholes permitem uma melhor estimativa de seus preços.

2. Metodologia dos Algoritmos e Discussões

A seguir são explicitadas as fórmulas de cada algoritmo, assim como comentários a respeito sobre condições intrínsecas de execução, particularidades, ou oportunidades de paralelização.

2.1. Algoritmo Gauss-Legendre

$$\begin{aligned} a_{n+1} &= \frac{a_n + b_n}{2} & a_0 &= 1 \\ b_{n+1} &= \sqrt{a_n b_n} & b_0 &= \frac{1}{\sqrt{2}} \\ t_{n+1} &= t_n - p_n(a_n - a_{n+1})^2 & t_0 &= \frac{1}{4} \\ p_{n+1} &= 2p_n & p_0 &= 1 \end{aligned} \tag{1}$$

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

Devido a natureza sequencial do algoritmo (o passo seguinte usa o resultado do passo anterior), sua paralelização não pode ser feita em relação a iteração, mas sim somente em relação aos quatro fatores (a, b, t, e p). Dentre eles, b_{n+1} possui a maior complexidade de cálculo, por possuir o cálculo de uma raiz quadrada. Nenhuma das variáveis utilizadas no seu cálculo será alterada durante a execução da iteração, então seu cálculo pode ser feito de forma independente das restantes. O fator a_{n+1} possui a segunda maior complexidade, e seria um outro candidato a paralelização. Porém, o valor de a_{n+1} é utilizado na iteração corrente para o cálculo de t_{n+1}. Dessa forma, a mais interessante forma de paralelização deste algoritmo está em paralelizar o cálculo de b_{n+1} em relação ao cálculo de a_{n+1} e t_{n+1}. A escolha de calcular p_{n+1} junto com b_{n+1} ou a_{n+1} e t_{n+1} é irrelevante devido a baixa complexidade da operação.

2.2. Algoritmo Bailey-Borwein-Plouffe

$$\pi = \sum_{k=0}^{\infty} \left(\frac{1}{16^k}\right) \cdot \left(\frac{4}{8.k+1} - \frac{2}{8.k+4} - \frac{1}{8.k+5} - \frac{1}{8.k+6}\right) \tag{3}$$

$$\pi = \sum_{k=0}^{\infty} a_k . \left(p_k^1 + p_k^2 + p_k^3 + p_k^4\right) \tag{4}$$

Em contrapartida ao método Gauss-Legendre, o método BBP pode ser paralelizado tanto em iteração (todas as iterações são independentes entre si) quanto em parcelas (os termos a_k e p_k^n são independentes entre si). Estas características, combinadas ao fato de que os fatores de multiplicação são múltiplos de 8, demonstram a capacidade de execução rápida deste algoritmo. Para este trabalho, foi feita a paralelização em parcelas (cada iteração calculou as parcelas em paralelo).

2.3. Algoritmo Monte Carlo

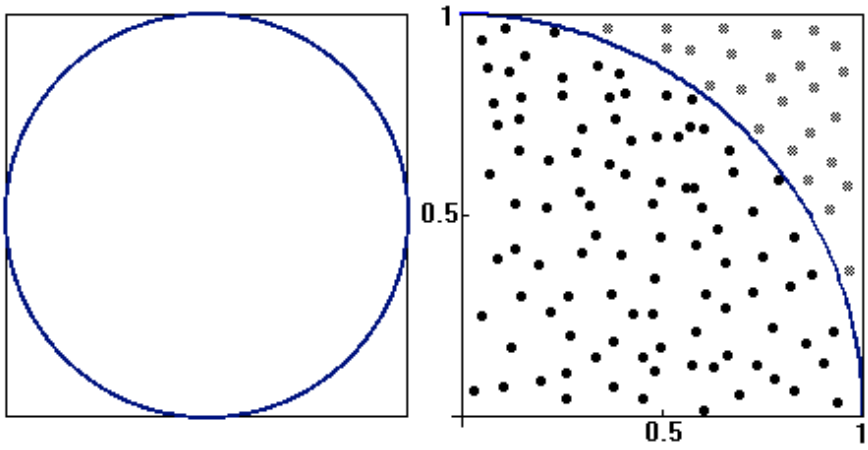

Figura 1. Visualização do Método de Monte Carlo

Para o método de Monte Carlo, cada ponto pode ser gerado e testado (se a soma de seus quadrados é maior que 1) de forma completamente independente. Dessa forma, o algoritmo é paralelizável a nível de iteração. A dificuldade da paralelização deste algoritmo está em relação a geração de números aleatórios.

2.4. Algoritmo Black-Scholes

$$\begin{aligned} \texttt{t} &= S.e^{(r-\frac{v^2}{2}).T}.e^{(v.\sqrt{T}).randomNumber()} \\ \texttt{aux1} &= S.e^{(r-\frac{v^2}{2}).T} \\ \texttt{aux2} &= (v.\sqrt{T}) \\ \texttt{trial[i]} &= e^{-r.T}.\texttt{max}(\texttt{t} - E; 0) \\ \texttt{aux3} &= e^{-r.T} \end{aligned} \tag{5}$$

$$\texttt{t} > E \quad ... \quad randomNumber() < \frac{ln(E/\texttt{aux1})}{\texttt{aux2}} \tag{6}$$

Similar ao método de Monte Carlo, o método de Black-Scholes baseado em Monte Carlo gera N testes com números aleatórios entre 0 e 1, e infere o resultado a partir de dados estocásticos. Dessa forma, o algoritmo é paralelizável a nível de iteração. Uma ressalva deste algoritmo é que ele não é válido para condições onde todos os números aleatórios gerados não são capazes de satisfazer a condição de validade demonstrada na Equação 6, visto que todo teste `trial[i]` resultará em 0.

3. Resultados e Discussões

3.1. Algoritmo Gauss-Legendre

A rápida convergência do algoritmo de método de Gauss-Legendre para o cálculo de π foi observada pela forma como o resultado se tornava errôneo caso muitas iterações fossem feitas. Estes erros surgiam do fato da parcela t_n assumir valores além da precisão escolhida. Os limites observados relacionados com a precisão escolhida podem ser observados na Tabela 1.

Tabela 1. Limite de Iterações e Precisão (em bits)

Número de Iterações	Precisão (bits)
600	256
1000	512
2000	1024
15000	8192

Uma precisão com tamanho de 1MB causou um travamento na execução do programa. Com o objetivo de estressar o algoritmo e a biblioteca, foi escolhida a precisão de 1KB. Os tempos de execução do algoritmo não-paralelizado e paralelizado podem ser vistos na Tabela 2. Os valores refletem os tempos mínimo e máximo entre 5 execuções.

Tabela 2. Tempo de Execução: Método Gauss-Legendre

	Não Paralelizado (s)	Paralelizado (s)
real	0,229 - 0,235	0,682 - 0,720
user	0,229 - 0,235	0,407 - 0,486
system	0,000	0,300 - 0,371

A aplicação paralelizada teve um desempenho pior para os mesmos parâmetros de execução. Seria necessária uma futura investigação para verificar se a piora de desempenho foi causada pela criação e junção de threads ou por um mal isolamento das variáveis da thread de cálculo da parcela b_n em relação as variáveis restantes.

Funcionalmente, o algoritmo calculou corretamente o valor do π.

3.2. Algoritmo Bailey-Borwein-Plouffe

Para melhor comparação com os algoritmos restantes, a precisão e número de iterações escolhidas no algoritmo de Gauss-Legendre foi mantida para todos os programas. A única exceção a esta escolha foi o algoritmo de Monte Carlo, discutido futuramente.
Os tempos de execução do algoritmo não-paralelizado e paralelizado podem ser vistos na Tabela3. Os valores refletem os tempos mínimo e máximo entre 5 execuções.

Novamente, a aplicação paralelizada teve um desempenho pior do que a não paralelizada. As observações sobre os possíveis motivos são as mesmas a respeito do

Tabela 3. Tempo de Execução: Método Bailey-Borwein-Plouffe

	Não Paralelizado (s)	Paralelizado (s)
real	0,473 - 0,483	1,771 - 1,871
user	0,473 - 0,479	1,467 - 1,648
system	0,004 - 0,005	2,142 - 2,388

método de Gauss-Legendre.

Funcionalmente, o algoritmo calculou corretamente o valor do π. Adicionalmente, em um contexto não-otimizado para operações hexadecimais, o método de Gauss-Legendre demonstrou melhor desempenho do que o método Bailey-Borwein-Plouffe. Em um contexto de otimização especializado para operações hexadecimais, é possível prever como o método Bailey-Borwein-Plouffe resulta e menos consumo de memória e execução, dados seus desempenhos próximos.

3.3. Algoritmo Monte Carlo

Devido a natureza estocástica do algoritmo, o uso de poucas iterações leva a um erro muito grande no cálculo do valor do π. Dessa forma, para este algoritmo em particular foram utilizadas 10^9 iterações para o cálculo de π.

Como a execução do aplicativo teve um desempenho muito pior em comparação aos anteriores, e a escolha da estratégia para cálculo de valores randômicos foi ruim para paralelização, não foram feitos testes para o algoritmo paralelizado (que possuiu desempenho extremamente pior em comparação ao não paralelizado devido ao uso da biblioteca errada de gerador de números randômicos).

Os tempos de execução do algoritmo não-paralelizado podem ser vistos na Tabela 4 para uma única execução do algoritmo.

Tabela 4. Tempo de Execução: Método Monte Carlo

	Não Paralelizado
real	7m7,959s
user	7m7,940s
system	0m0,012s

Funcionalmente, o algoritmo calculou corretamente o valor do π, porém com um desempenho e corretude muito inferior aos algoritmos anteriores.

3.4. Algoritmo de Black-Scholes

As entradas de referência na especificação do trabalho não são confiáveis. Tanto a calculadora de referência, quanto o algoritmo, usam os valores de volatilidade, juros, e

tempo até a execução do contrato como valores percentuais entre 0 e 1. Dessa forma, uma volatilidade de 10% seria escrita como 0.10, um juros de 1% seriam representados como 0.01, e um período de 1 mês seria representado como $1/12$, ou seja, $0.08333\ldots$.

Os dados de referência na especificação do trabalho estão indicando volatilidade de 1000%, juros de 100%, e período de execução de 1 ano. Isso é refletido na calculadora, que prevê a Opção tendo um valor da mesma ordem do preço da Ação de referência. Normalmente, o preço de uma opção é somente uma fração do preço da ação, visto que são contratos com prazos típicos de um mês.

Apesar de aplicar corretamente o algoritmo conforme apresentado na especificação de referência, nenhum dos resultados esteve em acordo com a calculadora de referência. Por este motivo, e por não ter sido utilizada uma aplicação correta de geração de números randômicos que fosse *thread-safe* E não bloqueasse execução de threads acessando a mesma função de geração de número aleatório, este algoritmo não foi explorado.

4. Conclusões

A paralelização de programas tem uma alta capacidade de reduzir drasticamente o tempo de execução de algoritmos, contanto que os mesmos exibam características que permitam a paralelização.

Porém, isso não implica que a paralelização é uma tarefa simples. Bibliotecas *thread-safe* bloqueiam a execução de threads que tentam acessá-las quando há concorrência. A passagem de parâmetros para cada thread também pode causar problemas de concorrência caso não sejam explicitamente separados para cada thread.

Este trabalho conseguiu realizar uma boa análise e aplicação dos algoritmos, apontar corretamente as oportunidades de paralelização, porém não conseguiu executar a paralelização usando as POSIX PThreads.

Referências

The GNU MP Bignum Library. In *https://gmplib.org/*.

The GNU MPFR Library. In *https://www.mpfr.org/*.

David H. Bailey, P. B. B. and Plouffe, S. (1997). On the rapid computation of various polylogarithmic constants. In *Mathematics of Computation*, volume 66, pages 903—-913.