UNIVERSIDADE DE SÃO PAULO

UNIVERSIDADE DE SÃO PAUO

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E COMPUTAÇÃO

Cálculo do π através de métodos, de forma sequencial e paralela.

RVC. Beber *G. Gibertoni* *C. Casagrande*

Cálculo do π através de métodos, de forma sequencial e paralela

Rodrigo Vicente Casagrande Beber[1], Guilherme Gibertoni[2], Cassiano Casagrande[3]

[1] Instituto de Ciências Matemáticas e Computação Universidade de São Paulo(USP)

rvcbeber@gmail.com - 7152490

[2] Instituto de Ciências Matemáticas e Computação Universidade de São Paulo(USP)

kuramagui@gmail.com - 7152802

[3] Instituto de Ciências Matemáticas e Computação Universidade de São Paulo(USP)

cassianokleinert@gmail.com - 7152819

Resumo. Este trabalho é um relatório sobre 3 métodos para se calcular o número π, tanto de forma sequencial como paralela, a fim de se obter uma grande precisão (10 milhões de casas).

Palavras-Chave: Paralelismo, Programação Concorrente

Sumário

1 – Introdução

Nesse relatório iremos abordar 3 métodos, Gauss Legendre, Borwein e Monte Carlo para fazer o cálculo do π. Para todos utilizamos a biblioteca GMP, de livre distribuição e já equipada com estruturas de dados para lidar com cálculos de 10 milhões de digitos de precisão. Cada método será abordado pelo algoritmo proposto no meio científico, bem como a implementação feita. Será tratado também a paralelização de cada método, e como isso afetou o desempenho.

No capítulo 2 iremos falar sobre o método de Gauss Legendre, no capítulo 3 sobre Borwein e no capítulo 4 sobre o método de Monte Carlo. No capítulo 5 apresentamos as referências utilizadas.

2 – Gauss Legendre

2.1 – Algoritmo

O algoritmo de Gauss-Legendre foi desenvolvido na década de 70 através da associação da técnica de média aritimética-geométrica de Gauss com a relação de Legendre para integrais elípticas. Essa associação também é conhecida por algoritmo de Brent-Salamin, sendo que ela foi proposta independentemente por Richard Brent e Eugene Salamin na mesma época[1].

A técnica de Gauss é conhecida por ter uma rápida convergência, de ordem quadrática. Com a relação de Legendre é possível expressar o número π através de uma sequência de médias aritimética-geométricas.

Essa sequência caracteriza um processo iterativo compacto, após as devidas simplificações matemáticas e uso de teoremas que não vem ao caso aqui demonstrar. As fórmulas a seguir mostram os cálculos a serem realizados a cada passo da iteração.

$$
\begin{aligned}
a_{n+1} &= \frac{a_n + b_n}{2}, \\
b_{n+1} &= \sqrt{a_n b_n}, \\
t_{n+1} &= t_n - p_n(a_n - a_{n+1})^2, \\
p_{n+1} &= 2p_n. \\
\pi &\approx \frac{(a_n + b_n)^2}{4t_n}.
\end{aligned}
$$

Figura 1 – Cada passo da iteração para cálculo do Pi

Nesta nomenclatura, a_n representa a sequência de médias aritiméticas, b_n a sequência de médias geométricas, p_n o número de dígitos corretos a cada passo, e t_n o intervalo da sequência de médias a ser considerado no cálculo do π. O objetivo deste processo é aproximar a_n e b_n até que a diferença entre os dois seja menor que a precisão desejada:

$$(a_n - b_n)^2 < 10^{-6}$$

É importante notar que através de cálculos anteriores foi determinado um conjunto de valores para iniciar a sequência[2]

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

Figura 2 – Valores iniciais para as sequências

2.2 – Precisão e Paralelização

Foi implementado duas versões do algoritmo da Figura 1. Na primeira traduzimos as equações diretamente para o código e o executamos em um laço

até que as 10 milhões de casas de precisão fossem atingidas. Nesse processo concluímos que 24 iterações foram necessárias para tal resultado, condizendo com o proposto pelo algoritmo já que

$$2^{24} > 16x10^{6} \quad e \quad 2^{23} < 10^{6}$$

Para paralelizar o algoritmo e obter maior desempenho nos cálculos, identificamos dentre as operações realizadas quais apresentam variáveis independentes entre si. Das cinco operações realizadas – calculo do π, calculo do A_{n+1}, B_{n+1}, T_{n+1}, P_{n+1} – apenas o cálculo de T_{n+1} apresenta dependências. Sendo assim, criamos uma thread para cada cálculo e executamos de forma concorrente, com a thread responsável pelo T_{n+1} aguardando o término das outras para iniciar sua execução.

Aplicando o processo acima de paralelização mantivemos as fórmulas intactas, mas permitimos que em computadores com mais de um núcleo mais de uma thread possa ser executada simultaneamente. Os resultados foram satisfatórios, ainda obtendo as 10 milhões de casas corretas do π obtivemos um *speed-up* de pouco mais de 50%.

O cálculo do *speed-up* obtém a razão entre os tempos de processamento tradicional e concorrente, e ressalta a porcentagem de acréscimo ou decréscimo de desempenho[3]:

$$S_p = \frac{T_1}{T_p}$$

Figura 3 – Fórmula do Speed-up para p processadores

Em nosso caso executamos os algoritmos em um processador Intel i5-2410M de 2 núcleos físicos e com capacidade para 4 threads simultâneas. Na tabela abaixo consta os tempos obtidos nesta máquina.

Algoritmo	Tempo Exec 1 (s)	Tempo Exec 2 (s)	Tempo Médio (s)	Speed-Up
Tradicional	80.357	76.479	78.418	$S_2 \sim 1.52$
Paralelo	51.615	51.851	51.733	

Tabela 1 – Tempos de execução e Speed-Up

3 – Borwein

3.1 – Algoritmo

O algoritmo de Borwein é um algoritmo para o cálculo do valor aproximado de π, ele foi proposto por Jonatham e Peter Borwein.
Com o passar dos anos eles publicaram diversas revisões do algoritmo, aumentando a razão de convergência dos algoritmos.
Esse relatório refere-se ao algoritmo de 1987, que tem convergência quadrática:

$$X0 = 2^{\frac{1}{2}}$$

$$Y1 = 2^{\frac{1}{4}}$$

$$P0 = 2^{\frac{1}{2}} + 2$$

$$X(n+1) = \frac{Xn^{\frac{1}{2}} + Xn^{-\frac{1}{2}}}{2}$$

$$Y(n+1) = \frac{Yn * Xn^{\frac{1}{2}} + Xn^{-\frac{1}{2}}}{Yn + 1}$$

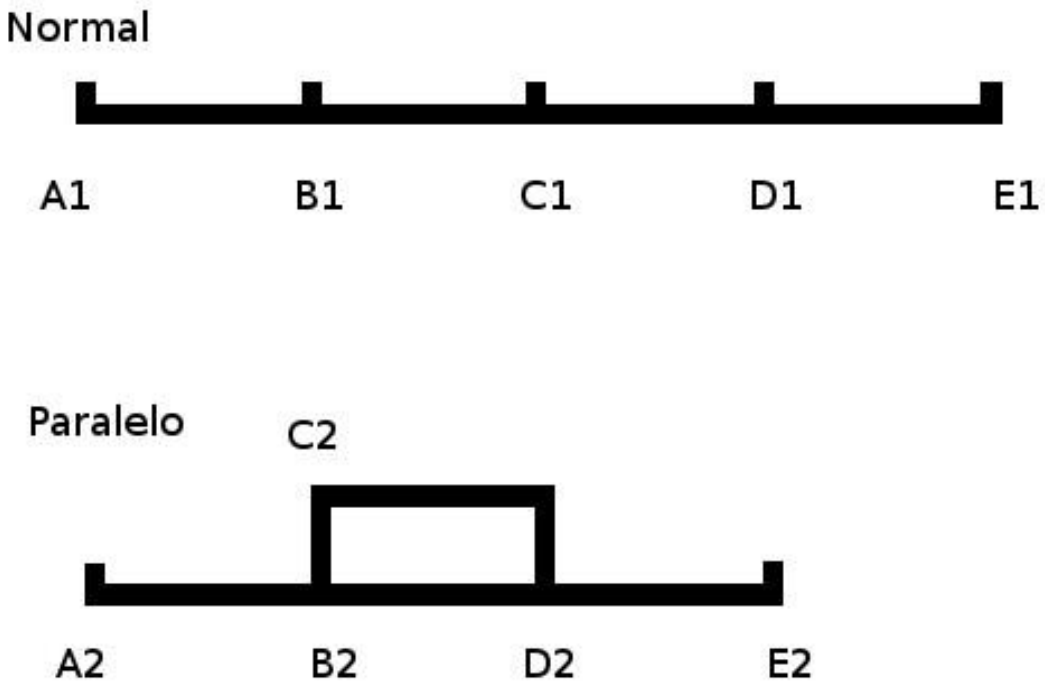
$$Pn = Pn - 1 * \frac{Xn + 1}{Yn + 1}$$

A sequênca Pn converge para π de forma que:

$$Pn - \pi \leq 10^{2^{n+1}}$$

3.2 – Implementação

A implementação normal do algoritmo foi feita de forma otimizada. Valores como $Xn^{\frac{1}{2}}$ e $Xn^{-\frac{1}{2}}$ que sâo utilizados diversas vezes na mesma iteração são calculados apenas uma vez.



A figura mostra como são executadas cada iteração das duas versões do algoritmo. Em A, calcula-se $Xn^{\frac{1}{2}}$ e $Xn^{-\frac{1}{2}}$. Em B calcula-se Yn e $Y(n+1)$. Em C calcula-se Xn e $\left(P(n-1)*(X(n+1))\right)$ e em D divide-se o resultado obtido por $Y(n+1)$.

3.2.1 – Implementação paralela do algoritmo

O algoritmo de Borwein não é facilmente paralelizável devido ao grau de dependência entre as variáveis calculadas. Para pareleliza-lo inicialmente aferiu-se os tempos aproximados de execução de cada uma das operações aritméticas básicas:
add_ui ~ 0s
add ~ 0.01s
mul ~ 1.07s
div ~ 2.43s
sqrt ~3.37s

Portanto, para melhorar a performance por meio da paralelização, deve-se tentar paralelizar operações como mul, div e sqrt.

A execução do programa foi divida então em duas threads, que executam sem dependência de dados, e cujo tempo de execução é o seguinte;
Thread: 1 3.64s
Thread: 2 1.10s

A divisão de tempo entre as threads está longe do ideal, no entanto, para 25 iterações(que sâo suficientes para calcular π com 10 milhões de casas decimais) o speed up teórico seria de cerca de 26 segundos, desconsiderando o overhead da criação das threads.

Os valores aferidos para a execução dos programas, compilados no gcc 4.2 com parametro -Os é de:

- Tempo gasto no sequencial: 2m8.583s
- Tempo gasto no paralelo: 1m41.643s

A diferença entre os tempos é de aproximadamente 27s, sendo que a versão paralela é 57% mais rápida que a sequencial.

Essa paralelização do algoritmo não é escalável para mais de dois processadores, ou seja, não haveria melhora na performance caso seja adicionado mais que dois processadores.

4 – Monte Carlo

4.1 – Algoritmo

O algoritmo de Monte Carlo para cálculo do π consiste em calcular a área de ¼ de uma circunferência e a partir desse valor, estimar o valor do π.

Temos por Monte Carlo, que a área de ¼ da circunferência pode ser dada pelo raio ao quadrado, multiplicado pelo número de pontos encontrados dentro da circunferência divididos pelo número total de pontos aleatórios gerados. Segue a fórmula: $A = r^2 * \frac{Pin}{Pout}$.

Temos também que a área de ¼ da circunferência é: $A = \frac{1}{4} * \pi r^2$. Substituindo uma fórmula na outra obtemos o valor aproximado do π através da fórmula: $\pi = 4 * \frac{Pin}{Pout}$ [4], [5].

A implementação pode ser feita através do seguinte algoritmo:

Entrada: número de iterações da execução em n
Resultado: valor estimado do π 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 $\leftarrow 4 * \frac{P_{in}}{P_{total}}$

4.2 – Implementação

A implementação do algoritmo foi feito em linguagem C, em conjunto com a biblioteca GMP(para cálculos de BIG NUMBERS).

Tanto para o caso sequencial como para o paralelo foram feitos 1 bilhão de iterações, porém devido ao método, não se obteve muita precisão. Isso se deve ao fato que se divide o número de pontos encontrados dentro da circunferência pelo número de pontos totais aleatórios gerados, portanto a precisão sempre será pequena.

4.3 – Paralelismo

Para fazer o paralelismo, criamos diversas threads, e cada thread calcula uma parcela dos números de pontos sorteados. Feito isso, somamos os totais de pontos encontrados dentro da circunferência de cada thread, e dividimos pelo total de pontos

sorteados. Como esperado o tempo diminuiu. Apresentamos a seguir o tempo gasto tanto no caso sequencial como no caso paralelo:

- Tempo gasto no sequencial: 1m28.807s
- Tempo gasto no paralelo: 0m43.987s

Como podemos ver, houve uma melhora, diminuindo o tempo de processamento. Podemos calcular o speed-up com a fórmula apresentada abaixo:

$$S_p = \frac{T_1}{T_p}$$

Obtendo um $Sp = 2{,}01$.

5 – Discussão dos Resultados

Todos os algoritmos foram executados na mesma máquina, com processador Intel i5-2410M de dois núcleos de processamento e capacidade máxima de até 4 threads de execução em paralelo. Como esperado pela análise matemática dos algoritmos, temos a seguinte ordem de eficiência:

Gauss-Legendre > Borwein > MonteCarlo

Ao paralelizar estes algoritmos nesta máquina, a ordem se manteve. O algoritmo Gauss-Legendre pode ser resolvido por completo em apenas 4 operações que podem ser paralelizadas -- uma em cada thread. Este algoritmo aproveitou o total potencial da máquina.

Já o algoritmo de Borwein só utilizou 2 threads, apresentando um ganho de eficiência considerável, mas sem escalabilidade para uso do poder total do processador.

O algoritmo de MonteCarlo necessita de muitas iterações devido ao seu caráter aleatório. A paralelização destas iterações pode ser realizada, sendo cada uma colocada em threads separada. Dessa forma a mesma quantidade de iterações pode ser executada em um tempo muito menor já que várias delas vão ser executadas em paralelo. Para esta arquitetura de dois núcleos o algoritmo apresentou um ganho de desmpenho de 100%.

A paralelização apresentou um ganho considerável nos 3 algoritmos, no qual pudemos presenciar o potencial das threads e da arquitetura multi-núcleo de nosso sistema. Entretanto, esse ganho de desempenho não altera o caráter e ordem de complexidade dos algoritmos. Apesar do algoritmo de MonteCarlo executar um pouco mais rápido que o de Gauss-Legendre no modo concorrente, Gauss-Legendre conseguiu atingir as 10 milhões de casa de corretude e MonteCarlo apenas 2.

6 – Referências

[1] SALAMIN, E. **Computation of pi Using Arithmetic-Geometric Mean**, Mathematics of Computation. 1976

[2] BRENT, R. **Multiple-precision zero-finding methods and the complexity of elementary function evaluation.** Academic Press, New York. p18. 1975

[3] ACZEL, J. **A new formula for speedup and its characterization.** Springer. 1997.

[4] REAL, R. A. **Aproximação do π pelo Método de Monte Carlo.** Disponível em http://www.inf.ufrgs.br/gppd/disc/cmp134/trabs/T2/021/html/index.html. Acessado em: 21 de Março de 2013.

[5] RONALD, M. **The Java Programmers Guide to Numerical Computation**. Pearson Educational. p. 353. 2003