Universidade de São Paulo – USP
Instituto de Ciências Matemáticas e Computação – ICMC

Trabalho de sistemas operacionais

Código da Disciplina: SSC0640

Prof. Julio Cezar Estrella

Aluno: Lucas Nascimento Camolezi
Nº USP: 10276932
Aluno : Afonso Henrique Piacentini Garcia
Nº USP: 9795272

São Carlos
2019

Sumário

1 – Introdução:

O número PI é a proporção matemática entre o perímetro e o diâmetro de uma circunferência.

Essa proporção, apesar de conhecida a milhares de anos, continua sendo fonte de pesquisa em diversas áreas da ciência. Continuam-se os estudos sobre suas peculiaridades, e sempre são buscadas formas de tornar o cálculo desse valor mais preciso e rápido.

Neste trabalho, utilizaremos três métodos diferentes para calcular o valor de PI, primeiramente de forma sequencial, e em seguida de forma paralelizada, sendo eles o algoritmo de Borwein, o algoritmo de Gauss-Legendre e o algoritmo de Monte Carlo.

1.1 – Borwein:

Peter Borwein desenvolveu diversos algoritmos para realizar o cálculo do valor de PI. No caso, foi utilizada a fórmula BBP, criada em 1995 por Peter Borwein junto com Simon Plouffe e David Harold Bailey.

1.2 – Gauss-Legendre:

O Algoritmo de Gauss-Legendre foi baseado nos trabalhos individuais de Carl Friedrich Gauss (1779 – 1815) e Adrien-Marie Legendre (1799-1855) unidos com algoritmos modernos. É caracterizado por convergir muito rapidamente, porém tem como principal desvantagem o uso excessivo de memória de processamento, podendo ser desvantajoso em alguns casos.

1.3 – Monte Carlo:

O método de Monte Carlo é uma maneira para realizar aproximações utilizando simulações estocásticas.

O Método citado acima é utilizado em diversas áreas e aplicações científicas, e nesse trabalho utilizaremos o algoritmo homônimo, que utiliza desse método para realizar o cálculo do valor e PI.

2 – Demonstrações:

2.1 – Borwein:

2.1.1 – Sequencial:

Sendo k o número de interações desejadas, realizamos:

$$\sum_{0}^{k} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \simeq \pi$$

2.1.2 – Paralelo:

Foi utilizado a mesma fórmula que o algoritmo sequencial. Porém o valor de K foi dividido para cada thread. Como foi utilizado um processador com 4 núcleos foram criadas quatro Threads, cada uma executa 25% do valor do Kmax escolhido. Após a finalização de todas as threads o valor obtido em cada thread é somado e o resultado final é obtido.

2.2 – Gauss-Legendre:

2.2.1 – Sequencial:

Iniciando as variáveis:

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

Sendo n um valor entre 0 e k, com k sendo o número de interações:

$$a_{n+1} = \frac{a_n + b_n}{2} \qquad b_{n+1} = \sqrt{(a_n b_n)}$$

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

Dessa forma, temos:

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

2.2.2 – Paralelo:

No código do Gauss paralelo foram utilizadas quatro threads, uma para cada possível foŕmula do gauss. Como a fórmula do Tn+1 depende do An+1 ela não pode ficar em uma thread separada (precisa ser calculado sequencialmente a An+1). Logo Tn+1 foi calculado em conjunto (na mesma thread) que An+1. Bn+1 e Tn+1 foram colocas em threads próprias. Por último foi criado uma thread para manejar a sincronização e atualização para a próxima iteração da fórmula de gauss.

2.3 – Monte Carlo:

2.3.1 – Sequencial:

Inicialização de variáveis:

$$T = N°iteraçõe s \qquad S = 0$$

Para cada iteração do código, tem-se:

$$x = random\,(0,1) \qquad y = random\,(0,1)$$

$$se\ \sqrt{x^2 + y^2} \leq 1,\ temos\ que\ S = S + 1$$

A aproximação de π será:

$$\pi \simeq 4\frac{S}{T}$$

2.3.2 – Paralelo:

Para realizar a paralelização do algoritmo, a lógica utilizada foi exatamente a mesma do código sequencial. Porém, para tornar o processo mais eficiente, as interações foram divididas em blocos iguais e realizadas em paralelo

nos 4 núcleos do hardware utilizado. Por fim, os resultados obtidos foram somados.

3 – Resultados:

3.1 – Borwein: 10^5 *Iterações*

Tempo	Sequencial	Paralelo
Teste 1	189.97	89.33
Teste 2	191.84	87.75
Teste 3	192.35	87.96
Tempo Médio	191.39	88.35

3.2 – Gauss-Legendre: 10^5 *Iterações*

Tempo (s)	Sequencial	Paralelo
Teste 1	58.13	98.57
Teste 2	58.10	104.98
Teste 3	56.73	102.67
Tempo Médio	57.65	102.07

3.3 – Monte Carlo: 10^9 *Iterações*

Tempo	Sequencial	Paralelo
Teste 1	24.25	7.37
Teste 2	24.81	6.93
Teste 3	24.75	7.10
Tempo Médio	24.60	7.13

4 – Sistemas utilizados:

4.1 – Software:

Considerando que realizamos cálculos com grandes precisões e muitas casas decimais, os tipos básicos de dados já presentes em C não foram suficientes para as operações realizadas. Para isso, contamos com o uso do GMP (GNU Multiple Precision Arithmetic Library), que basicamente tornou o limite de memória do computador utilizado e o tempo de execução das operações o único limitador de precisão nessas.

Além disso, quando realizamos os cálculos em paralelo, foi utilizado API Posix PThreads para tornar mais simples a criação, manipulação e sincronização de Threads na linguagem C.

Para compilação do código foi utilizado o comando:

```
gcc arquivo.c -o nomeSaida -lpthread -lgmp -lm
```

4.2 – Hardware:

O hardware utilizado para a realização dos experimentos foi o próprio hardware do PC.

Especificações do PC utilizado:

- Intel i5-7200u
- 8GB RAM

5 – Considerações finais:

5.1 – Problemas encontrados:

Um grande problema que tivemos foi o fato de não conseguir realizar 10^9 interações no algoritmos de Borwein e no de Gauss-Legendre(o máximo que conseguimos realizar foram 10^5 interações), tanto nas suas versões sequenciais quanto nas paralelizadas. Em ambos os algoritmos, isso ocorreu por limitações de hardware e pelo tempo de execução ser muito grande com quantidades maiores de interações.

Outro problema enfrentado foi o fato de o algoritmo de Gauss-Legendre não apresentar boas condições de paralelização, e em

consequência, quando paralelizamos este código o tempo de execução dos testes foi maior do que o tempo de execução dos testes em sequencial.

5.2 – Conclusões:

Primeiramente, este trabalho demonstrou como a paralelização de códigos utilizando Threads pode ser benéfica em muitas situações. Por um lado, ela é muito efetiva por diminuir o tempo de execução de programas com muitas interações nos quais cada interação é independente, mas por outro lado utilizar esse mecanismo pode gerar Racing Conditions, além de necessitar de recursos específicos de Hardware e Software para funcionar.

Ambos os algoritmos de Borwein e o de Gauss-Legendre foram desenvolvidos especificamente para realizar o cálculo das casas decimais de π da forma mais precisa possível, assim obtendo resultados muito melhores neste aspecto. O código que utilizou o método de Monte Carlo (um método estatístico com diversos usos diferentes), por outro lado, conseguiu realizar interações muito mais rápidas, porém menos precisas.

6 – Bibliografia:

http://recologia.com.br/2013/07/estimando-o-valor-de-pi-usando-o-metodo-de-monte-carlo/

https://computing.llnl.gov/tutorials/pthreads/

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

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

https://pt.wikipedia.org/wiki/F%C3%B3rmula_BBP