Universidade Federal do Rio Grande do Norte

Instituto Metrópole Digital

Análise e Comparação de Speedup e Eficiência do código serial e paralelo nos problemas de estimativa do valor de PI com Monte Carlo e da integração numérica usando a regra dos trapézios

João Vitor Venceslau Coelho

Natal/RN

2020

Introdução

Este relatório consiste na explicação dos problemas abordados, sendo eles: a ***Estimativa do valor de PI*** e a ***Integração numérica***, essa explicação busca deixar claros todos os detalhes dos problemas e das soluções utilizadas, sendo elas: **o *método de Monte Carlo*** para o problema da estimativa de PI e a ***regra dos trapézios*** para a integração numérica. As soluções implementadas também serão explicadas, isto é, a versão serial e a paralela dos códigos implementados para cada uma das soluções dos problemas abordados será explanada, em seguida os resultados obtidos serão mostrados e comentados. Será também brevemente comentada a corretude dos algoritmos implementados e serão expostas as análises do speedup, da eficiência e da escalabilidade dos algoritmos paralelos para os seriais. Ao final, um resumo das atividades e análises realizadas será relatado, sintetizando os principais pontos do relatório.

Estimando o valor de PI via método de Monte Carlo

Os métodos de Monte Carlo são uma ampla classe de algoritmos computacionais que dependem de amostragem aleatória repetida para obter resultados numéricos.
A estimativa do valor de PI, segundo o método de Monte Carlo, consiste em sortear n pares de números em uma circunferência de raio A inscrita num quadrado de lado $2 \cdot A$. Contamos quantos desses n pares estão dentro da área delimitada pela circunferência (acertos) e quantos estão fora. Por fim, é calculada a proporção entre os acertos e o número total de pontos gerados, sendo esse valor uma aproximação para PI.

Integração numérica usando a regra dos trapézios

A integração numérica permite calcular o valor aproximado de uma integral definida sem conhecer uma expressão analítica para a sua primitiva. Assim, integrar numericamente uma função $y = f(x)$ num intervalo $[a, b]$ pode ser o mesmo que integrar um polinômio $P_n(x)$ que aproxima $f(x)$ em um determinado intervalo.
A aproximação mais simples possível, que pode ser calculada sem dificuldade, seria a área do trapézio definido pelos pontos $(a, f(a))$ e $(b, f(b))$, onde $f(a)$ e $f(b)$ são as bases e $(b - a)$ é a altura. Porém, o erro desta aproximação ainda é muito alto, dessa forma, uma estratégia para melhorar a qualidade da aproximação é dividir o intervalo de integração em diversos subintervalos menores, aproximando a integral em cada um desses subintervalos pela área dos respectivos trapézios.

Desenvolvimento

Abaixo são explicados os algoritmos implementados, tanto as versões seriais quanto as paralelas de cada um dos dois problemas abordados, isto é, o ***Algoritmo para estimar o valor de PI via método de Monte Carlo*** e o ***Algoritmo para integração numérica usando a regra dos trapézios***.

Soluções implementadas

Algoritmo Serial para estimar PI via método de Monte Carlo

O algoritmo recebe como entrada dois argumentos por linha de comando, são eles: a **seed**, usada para gerar os números pseudo-aleatórios e o **número de sorteios** a serem realizados no algoritmo. Após inicializar a **seed** dos sorteios, é criado um laço, em que, enquanto o número de sorteios restantes for maior que 0, sorteia-se dois valores entre 0 e 1, sendo um para a coordenada 'x' e outro para a 'y', verifica-se, dessa forma, se a soma delas, elevadas ao quadrado $(x^2 + y^2)$, é menor ou igual a 1, se for, aumenta-se o contador de "acertos", isto é, sorteios onde o par sorteado está dentro da área da circunferência, em seguida diminui-se uma unidade da contagem de sorteios e, ao término do laço, tem-se o número de "acertos". Então para estimar o valor de PI, multiplica-se por quatro a razão entre o número de "acertos" obtidos e o número de sorteios realizados, dessa forma imprimimos na tela o valor estimado para PI.

Algoritmo Paralelo para estimar PI via método de Monte Carlo

O funcionamento do algoritmo paralelo para estimar PI é praticamente o mesmo do serial, sendo que a tarefa paralelizada é a contagem de "acertos" entre os sorteios realizados. Após ser chamado o ***MPI_Init*** e obter o número de processos que serão usados, é calculado quantos sorteios o processo atual irá realizar, dividindo o número de sorteios solicitados pelo número de processos criados, então realiza-se a mesma contagem descrita no algoritmo serial com esse número resultante da divisão. Chama-se o ***MPI_Reduce*** para somar, os "acertos" que cada processo obteve, e enviar o resultado para o processo 0, onde PI é estimado usando a operação, descrita anteriormente de multiplicar por quatro a razão entre o número de "acertos" e o total de sorteios realizados. Sendo assim, apenas o processo 0 possui a soma correta, então somente ele irá imprimir na tela o valor estimado para PI, os outros processos irão finalizar normalmente a sua execução.

Algoritmo Serial para a integração numérica usando a regra dos trapézios

O algoritmo recebe como entrada três argumentos por linha de comando, são eles: o **início e fim do intervalo** (a, b), respectivamente, a ser integrado, e por último a **quantidade de trapézios** (n) a serem usados no algoritmo. Calcula-se o h que será usado na operação, isto é a altura que cada trapézio terá, dividindo a diferença entre o fim do intervalo e início, pelo número de trapézios que serão utilizados: $h = \frac{b-a}{n}$. Dessa forma, inicia-se o cálculo da integral da função escolhida, no caso, a função é a ***Rastrigin*** em 2 dimensões, com a aproximação, somando as imagens da função nos pontos a e b, dividindo o resultado por dois e o armazenando-o para usá-lo posteriormente. Em seguida o laço que irá somar o restante da integral, é iniciado, calculando o valor do x_i, que será usado na função, partindo do ponto a e somando a altura de i trapézios: ($x_i = a + i \cdot h$), o resultado da função, no ponto x_i**,** é somado à integral calculada até o momento e a operação é repetida até todos os $n-1$ trapézios restantes serem considerados, por fim, o valor acumulado da integral é multiplicado pelo h e esse é o valor da integral numérica calculada usando a regra dos trapézios.

Algoritmo Paralelo para a integração numérica usando a regra dos trapézios

A execução do algoritmo paralelo para a regra dos trapézios é semelhante a do serial, sendo que a tarefa para ser paralelizada precisa antes ser subdividida em problemas menores, no caso, calcula-se a integral de subintervalos do intervalo informado por parâmetro e em seguida soma-se as integrais encontradas em cada subintervalo. Assim, é realizado o mesmo procedimento do código serial até o cálculo do h a ser utilizado, então chama-se o ***MPI_Init***, se obtém o número de processos que serão usados e o $rank\ do\ processo$ atual. Por conseguinte, o número de trapézios nesse processo ($local\_n$) é calculado, dividindo o total de trapézios, solicitados por parâmetro, pela quantidade de processos criados, dessa forma, o início do intervalo a ser considerado ($local\_a$) é encontrado somando a a, o resultado da multiplicação do $rank\ do\ processo$ com $local\_n$ e o h, sendo assim, calcula-se o fim do intervalo a ser considerado ($local\_b$) somando a $local\_a$ o $local\_n$ multiplicado por h. Depois de ter inicializado as devidas variáveis, inicia-se o cálculo da integral, repete-se o mesmo algoritmo descrito na versão serial, usando como parâmetros as variáveis calculadas anteriormente, $local\_a$**,** $local\_b$ e $local\_n$, sendo apenas h o mesmo para todos os processos. Em seguida o **MPI_Reduce** é chamado para somar as integrais encontradas por cada processo no seus subintervalos calculados e enviar o resultado final para o processo 0, onde este será exibido na tela enquanto os outros processos serão finalizados normalmente.

Resultados encontrados

Os experimentos foram realizados numa máquina com a seguinte configuração:

- CPU: AMD® Ryzen 5 2600 six-core processor × 12
- Memória: 12 Gigabytes (11,7 GiB)
- Sistema Operacional (SO): Ubuntu 20.04.1 LTS - 64 bits

Corretude do Algoritmo Serial para estimar PI via método de Monte Carlo

A corretude do algoritmo vêm pelo uso do método de Monte Carlo, que se baseia verificando se os números sorteados satisfazem uma propriedade previamente estabelecida. No caso da estimativa do PI, essa propriedade seria que a probabilidade de sortear um par de números dentro da área da circunferência seria igual a área da circunferência dividida pela área do quadrado no qual ela está inscrita.

$$P = \frac{\text{Área da circunferência}}{\text{Área do quadrado}}$$

Como a área de uma circunferência de raio 1 é igual a PI e a probabilidade P pode ser aproximada pela razão entre os sorteios dentro da area da circunferencia, os “acertos”, e o total de sorteios realizados, temos que:

$$\frac{Acertos}{Total\ de\ Sorteios} \approx P = \frac{\pi}{\text{Área do quadrado}}$$

Porém não foi utilizada uma circunferência de raio 1 e sim apenas uma seção de uma circunferência com centro na origem, a que se encontra no quadrante dos eixos positivos, inscrita num quadrado de lado 1, assim a área da circunferência nesta seção é de um quarto da área original, sendo, portanto, de $\frac{\pi}{4}$ e a área do quadrado igual a 1, assim temos:

$$\frac{Acertos}{Total\ de\ Sorteios} \approx P = \frac{\frac{\pi}{4}}{1} = \frac{\pi}{4}$$

Dessa forma, pode-se aproximar PI por:

$$\pi \approx 4 \cdot \frac{Acertos}{Total\ de\ Sorteios}$$

Para verificar se ocorre um “acerto” no sorteio, está sendo assumido que a circunferência tem centro na origem, portanto para verificar se um ponto (x, y) está dentro da área é usada a seguinte desigualdade:

$$x^2 + y^2 \leq 1$$

Corretude do Algoritmo Serial para estimar PI via método de Monte Carlo

A explicação da corretude da versão paralela do algoritmo é a mesma da versão serial, acrescentando que o cálculo da quantidade de acertos entre os sorteios é paralelizado, se fazendo necessário somar o total de acertos que cada processo obteve durante sua execução, utilizando essa soma, entre todos os processos, para a estimativa do PI.

Corretude do Algoritmo Serial para a integração numérica usando a regra dos trapézios

A corretude da regra dos trapézios para a obtenção da integração numérica pode ser ilustrada com o seguinte exemplo de execução:

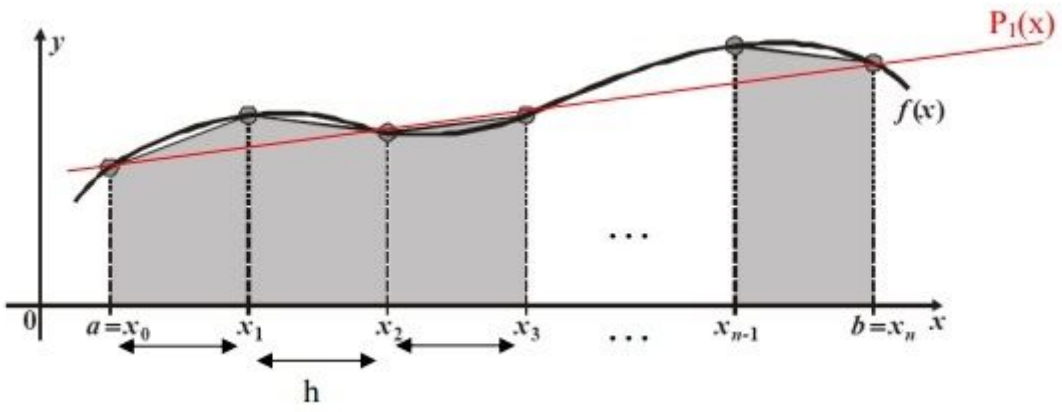


Imagem 1: Ilustração da regra dos trapézios. Fonte: Regra dos Trapézios - Cálculo Numérico

Onde a integral definida de a até b da função $f(x)$ é ilustrada pela área sob a curva formada pela função no gráfico. Essa área é aproximada pela área de n trapézios criados utilizando n pontos da função, cada um com altura h. Quanto mais trapézios existirem, menor o erro da aproximação e maior o custo de calcular a área total, isto é a integral da função. Assim, dado o intervalo $[a, b]$, a função $f(x)$ e a quantidade de trapézios a serem utilizados pode-se realizar uma aproximação da integral definida neste intervalo.

Corretude do Algoritmo Paralelo para a integração numérica usando a regra dos trapézios

A corretude da versão paralela do algoritmo se baseia na mesma ideia da utilizada para a versão serial, a ideia é que calculando a área de vários trapézios de uma altura h, criados usando pontos da função $f(x)$ num intervalo $[a, b]$, se obtenha uma aproximação da integral da função $f(x)$, a diferença é que agora a tarefa de calcular a área de todos os trapézios necessários foi dividida entre os processos envolvidos, assim a soma dos resultados que cada processo calcula deve ser somada no final e, realizando essas operações corretamente, temos a corretude do algoritmo. Salienta-se que o cálculo da área de um intervalo já é a tarefa que a versão serial realiza, apenas alteram-se os valores de a e de b para um subintervalo entre eles, enquanto a soma dos resultados obtidos em cada processo é facilmente calculada, justificando a integridade do algoritmo.

Análise de Speedup, Eficiência e Escalabilidade

Abaixo estão as tabelas com as médias dos tempos válidos obtidos, sendo que para cada instância analisada, foram coletados 10 tempos, em que os dois maiores e os dois menores foram desconsiderados como válidos para o cálculo dessa média.

Estimando o valor de PI via método de Monte Carlo

Neste problema, foram realizadas duas abordagens em relação a geração dos números pseudo-aleatórios: uma utilizando ***rand*** e a outra ***rand_r***. Abaixo temos os tempos obtidos em cada versão dos códigos seriais e paralelos. Salienta-se que a seed utilizada foi 0.

Problema	Serial	2	3	4	6	8	12
2200000000	34,32688	15,86407	10,58812	7,94423	5,48859	4,59764	3,08759
2400000000	37,68474	17,30513	11,54742	8,66671	5,98466	5,01152	3,37360
2600000000	40,77876	18,74646	12,50981	9,38771	6,47844	5,44221	3,65222
2800000000	43,85770	20,18858	13,46879	10,10812	6,98139	5,85788	3,93528
3000000000	46,92508	21,63005	14,43207	10,83094	7,48100	6,28207	4,21981
3200000000	50,08402	23,07168	15,39325	11,55152	7,97130	6,69724	4,50430
3400000000	53,01344	24,51286	16,35111	12,27250	8,47962	7,11137	4,78444
3600000000	56,11943	25,95441	17,31351	12,99432	8,96037	7,52518	5,06972
3800000000	59,22620	27,39352	18,27720	13,71708	9,46412	7,95561	5,35319
4000000000	62,39654	28,83912	19,23946	14,43968	9,95953	8,38070	5,63983
4200000000	65,31089	30,28041	20,19749	15,15846	10,44960	8,79070	5,91947
4400000000	68,31440	31,72256	21,15786	15,87793	10,95428	9,20802	6,20300

Tabela 1: Os tempos médios obtidos nos experimentos por tamanho do problema e algoritmo utilizado, assim como a quantidade de cores na versão paralela usando ***rand_r***.

Problema	Serial	2	3	4	6	8	12
1600000000	30,98110	26,51905	17,69005	13,27930	9,15818	7,57518	5,10946
1800000000	35,18845	29,83371	19,92306	14,93768	10,28430	8,52674	5,75264
2000000000	39,06540	33,14777	22,10799	16,58795	11,43713	9,47186	6,41377
2200000000	42,78120	36,46315	24,31701	18,25003	12,57679	10,40662	7,07303
2400000000	46,76031	39,77943	26,52734	19,92377	13,71261	11,35025	7,68161
2600000000	50,51987	43,11232	28,73470	21,57399	14,84818	12,30051	8,29850
2800000000	53,98400	46,40409	30,94669	23,21487	15,97642	13,18682	8,94055
3000000000	57,66845	49,72038	33,15408	24,87037	17,12978	14,18673	9,59626
3200000000	61,55654	53,03400	35,38159	26,53354	18,29212	15,11031	10,22528

Tabela 2: Os tempos médios obtidos nos experimentos por tamanho do problema e algoritmo utilizado, assim como a quantidade de cores na versão paralela usando ***rand***.

O uso do ***rand*** acabou sendo mais custoso, pois para um mesmo tamanho de problema o algoritmo com ***rand_r*** é mais ráPIdo tanto na versão serial quanto na paralela. Por exemplo, no tamanho do problema **2600000000** o tempo com o **rand_r** é de cerca de 40 segundos enquanto que com o **rand** é de 50, na versão serial, mas o principal a ser percebido é na questão paralela dos algoritmos, onde a versão com **rand_r** diminui o tempo gasto para cerca de 18.7 segundos, enquanto que com **rand** o tempo diminui para cerca de 43 segundos, ao utilizar 2 cores. A comparação do speedup e da eficiência, a seguir, irá ilustrar como é diferente o ganho em tempo da versão com **rand_r** para a de **rand**.

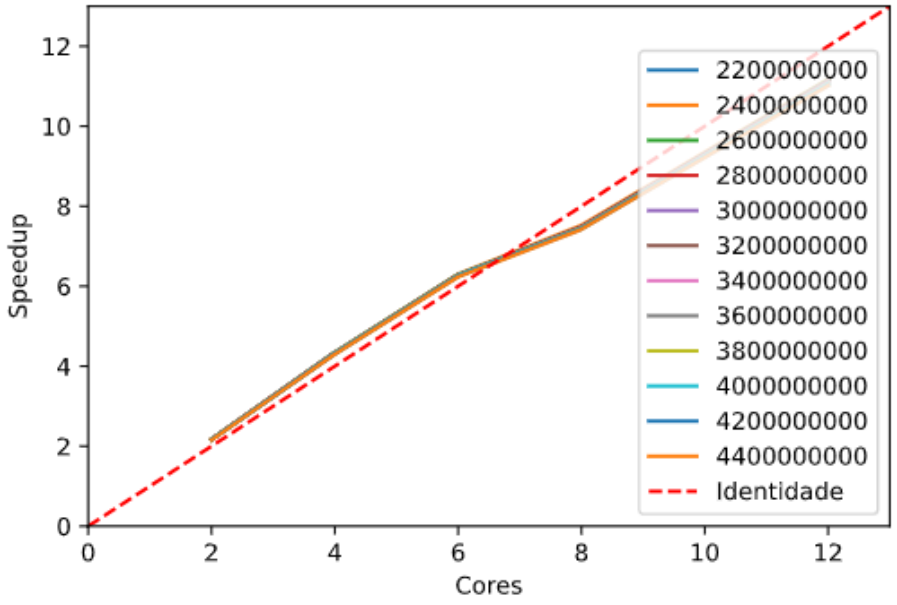


Imagem 2: Compara o speedup em cada um dos tamanhos do problema com o esperado. Na versão com o uso do **rand_r** o speedup fica mais próximo da reta identidade, chegando a obter speedup superlinear no início.

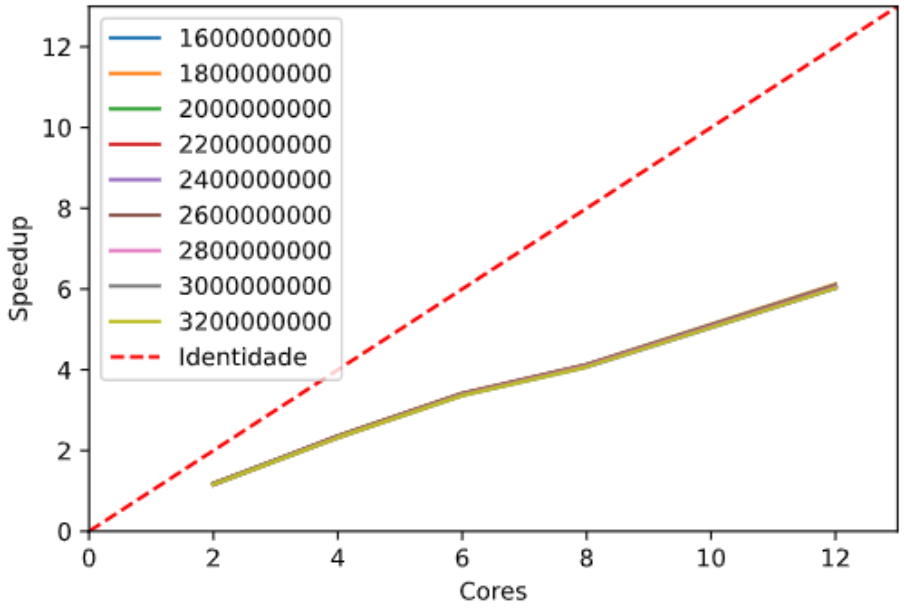
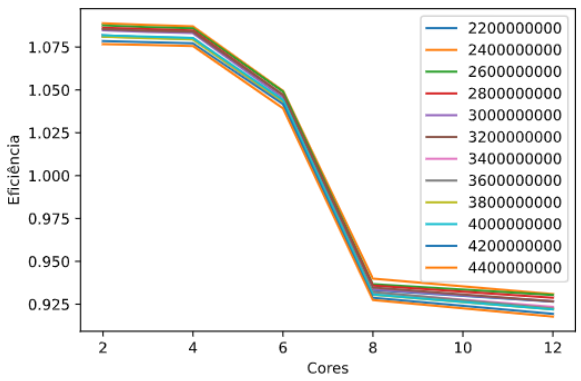
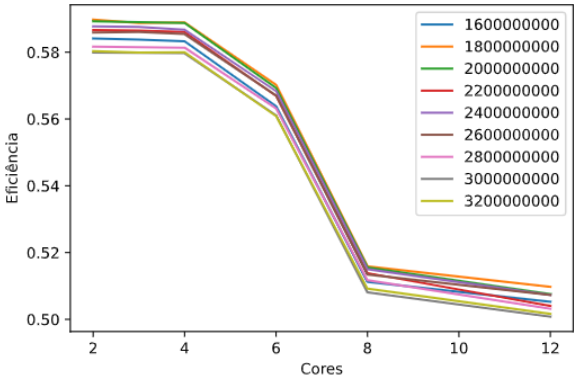


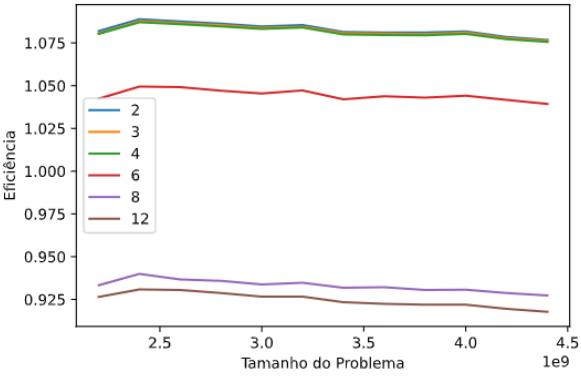
Imagem 3: Compara o speedup em cada um dos tamanhos do problema com o speedup esperado. Na versão com o uso do **rand** podemos perceber como o speedup é menor.

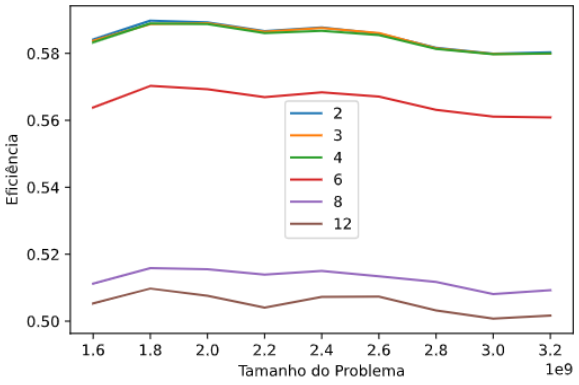
O speedup com o uso do **rand** acaba ficando comprometido, enquanto que com o uso do **rand_r** ele acaba ultrapassando o valor esperado em certos momentos. É perceptível em ambos os gráficos, porém mais facilmente no do **rand_r**, uma diminuição no speedup ao ser mudado de 6 para 8 cores, isso se deve pela configuração do hardware utilizado, que possui 6 cores físicos. Consequentemente temos um impacto na eficiência dos algoritmos, segue abaixo a comparação:





Imagens 4 e 5: Apresenta a eficiência por cores em cada tamanho de problema utilizado, possibilitando visualizar que as curvas são similares, porém os valores das mesmas diferem, sendo que a eficiência no primeiro varia em torno de 1.0, enquanto que a do segundo em torno de 0.54.





Imagens 6 e 7: Apresenta a eficiência por tamanho do problema com cada quantidade de cores utilizado, aqui podemos facilmente perceber a diferença na eficiência obtida em cada algoritmo, basta conferir o eixo y. Novamente percebe-se semelhanças nas curvas obtidas, porém, dessa vez, o ruído nos valores causa algumas diferenças.

Com base nesses dados, conclui-se que o algoritmo é **Fracamente Escalável** pois com o aumento do tamanho do problema e dos cores de forma proporcional, a eficiência se mantém praticamente constante, ocorrendo algumas oscilações. Essa eficiência acaba sendo menor com o uso dos cores lógicos (8 e 12), portanto para a análise realizada, foram considerados apenas os cores físicos, visto a diferença entre estes e os lógicos.
É importante mencionar que o ***rand_r*** se tornou obsoleto com o POSIX.1-2008, o uso do **random()** é recomendado no lugar de **rand** quando a aplicação necessita de uma "boa" aleatoriedade. Como é indicado aqui: https://linux.die.net/man/3/rand_r.

Integração numérica usando a regra dos trapézios

O intervalo utilizado para calcular as integrais foi de 0 até 100 e o tamanho do problema consiste no número de trapézios envolvidos para aproximar essas integrais. Abaixo estão as médias dos tempos que cada aproximação levou:

Problema	Serial	2	3	4	6	8	12
2000000000	33,98527	17,19834	11,46282	8,60459	6,04006	6,50407	4,43291
2200000000	37,49789	18,87410	12,61071	9,45574	6,61572	7,15948	4,87876
2400000000	40,81711	20,60625	13,75075	10,33194	7,21062	7,80316	5,33569
2600000000	44,34554	22,31847	14,88667	11,20475	7,80951	8,49092	5,80102
2800000000	47,85553	24,01481	16,06076	12,10140	8,45827	9,14397	6,26360
3000000000	51,36900	25,95964	17,18058	12,89366	9,06984	9,78318	6,70373
3200000000	54,86905	27,59841	18,31586	13,74558	9,62461	10,40799	7,18645
3400000000	58,27556	29,17839	19,45493	14,61706	10,23163	11,04525	7,63379
3600000000	61,71102	30,95249	20,61209	15,48367	10,88755	11,71953	8,08572
3800000000	65,14461	32,58794	21,75731	16,36523	11,47595	12,35414	8,53912
4000000000	68,54151	34,40410	22,91824	17,27539	12,09580	13,03109	9,01216

Tabela 3: Apresenta os tempos médios obtidos nos experimentos por tamanho do problema e algoritmo utilizado, assim como a quantidade de cores na versão paralela. O tempo utilizando 8 cores no algoritmo paralelo acabou sendo maior que usando 6 cores, isso deve ter sido consequência do uso de cores lógicos, pois os cores físicos da máquina são apenas 6.

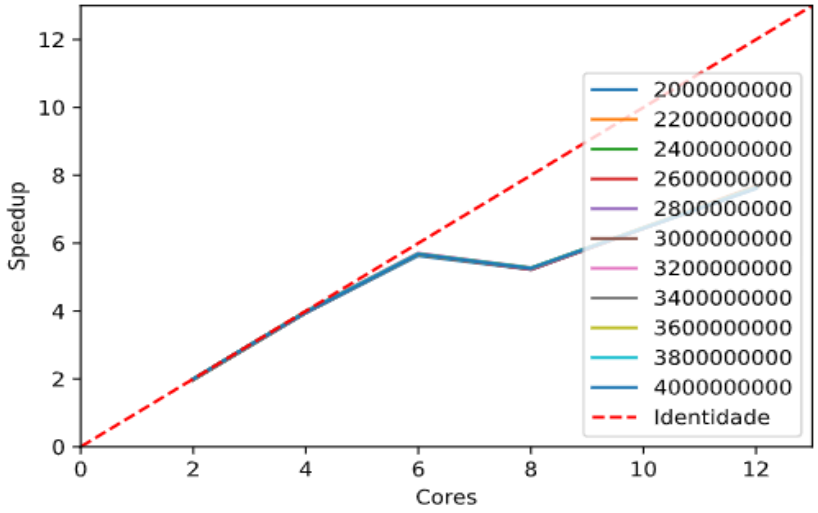


Imagem 8: Compara o speedup em cada um dos tamanhos do problema com speedup ideal, dessa forma, é possível ver uma nítida alteração na linha do speedup, que se dá na mudança de 6 para 8 cores, isso acontece, como foi dito anteriormente, pelo uso de cores lógicos da máquina. É interessante mencionar, também, que inicialmente o speedup acompanha a identidade, mas é perceptível que um distanciamento ainda ocorre ao passarmos de 4 para 6 cores.

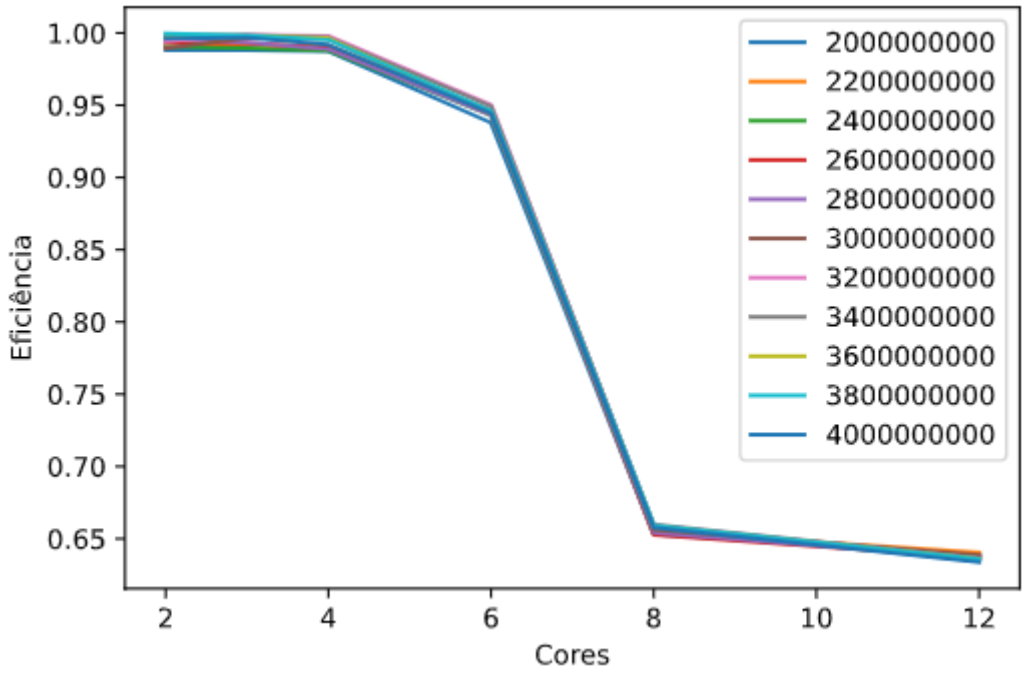


Imagem 9: Assim como no gráfico da eficiência do algoritmo para estimar o PI, a curva obtida foi similar, porém os valores envolvidos diferem. A eficiência continua sendo mais alta no início (2, 3 e 4 cores), diminuindo ao chegar em 6 e, posteriormente, ocorre uma mudança brusca ao utilizar 8 e 12 cores.

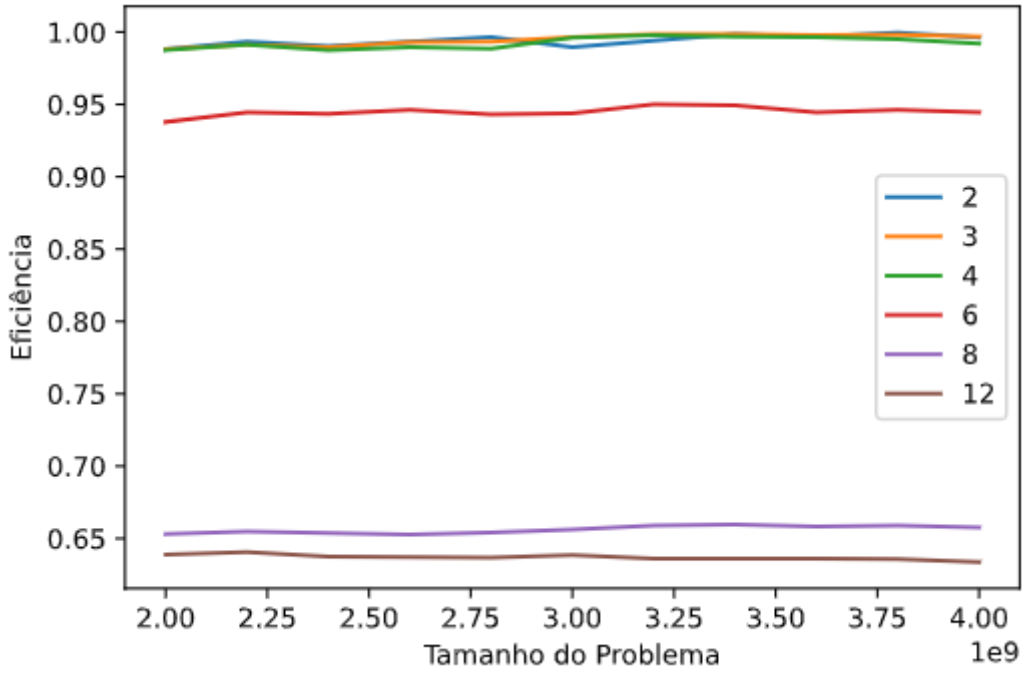


Imagem 10: Na comparação da eficiência por tamanho do problema, de acordo com a quantidade de cores utilizados, as eficiências são bem similares quando usamos 2, 3 e 4 cores, ocorrendo algumas alterações conforme a mudança de tamanho do problema, mas, no geral, é bem próxima a 1.0, enquanto que com outras quantidades de cores a eficiência é bem menor, principalmente com o uso dos cores lógicos, chegando a cerca de 0.65 de eficiência.

Diante desses dados, é concluído que o algoritmo se categoriza como **Fracamente Escalável**, pois aumentando o tamanho do problema e o número de cores em proporções iguais, a eficiência se mantém constante, com algumas pequenas flutuações. Salienta-se, ainda, que essa eficiência diminui consideravelmente ao utilizar os cores lógicos, mas se mantém constante ao redor desses valores reduzidos.

Considerações Finais

Neste relatório foram apresentados dois algoritmos: um que estima PI utilizando o método de Monte Carlo e o outro que integra numericamente funções por meio da regra dos trapézios. Também foi explicado o que é o método de Monte Carlo e como é possível aplicá-lo para estimar o valor de PI, apresentou-se a noção de integração numérica, incluíndo qual a vantagem dela e como utilizar a regra dos trapézios para calcular a integral de uma função. Em seguida foi exposto como foram implementadas as versões seriais e paralelas dos algoritmos para estimar PI, assim como para integrar utilizando a regra dos trapézios.

Após expor qual a configuração do hardware da máquina utilizada nos experimentos, foi realizada uma breve apresentação da corretude dos algoritmos implementados, mostrando a ideia que suporta a corretude dos mesmos. No caso da estimativa de PI, o que suporta a corretude dele são: alguns conceitos matemáticos sobre PI, a aplicação na área de circunferências e a ideia estatística do uso de Monte Carlo para verificar a satisfação da propriedade sobre a proporção entre a área do círculo e o quadrado no qual ele está inscrito. Enquanto que para a corretude da integração numérica, usando a regra dos trapézios, foi apresentado um exemplo visual da aplicação da regra para calcular a área sob a curva de uma função, sendo essa área entendida como a integral definida entre os pontos considerados limites da integração. As versões paralelas desses algoritmos possuem a mesma explicação para a corretude, podendo ter um passo a mais devido a forma como a paralelização impacta na solução, a coleta final das informações, por exemplo. Mas a corretude desses passos é trivialmente justificada, pois sem eles a informação completa não é obtida por nenhum dos processos envolvidos na tarefa, podendo assim, dar continuidade ao cálculo/apresentação do resultado final obtido.

Por conseguinte, são apresentados os tempos obtidos para cada tamanho de problema utilizado em cada configuração analisada. Apresentou-se primeiramente os resultados dos algoritmos da estimativa de PI, sendo comparadas duas implementações que diferem apenas na forma de obtenção dos sorteios utilizados para o método de Monte Carlo. Em seguida foi comparado o tempo de execução serial e paralela dos algoritmos em cada uma das implementações, apresentando os gráficos de speedup e de eficiência deles discutindo os impactos que a forma de obtenção da aleatoriedade causou, assim como as similaridades

entre os gráficos apresentados dado o algoritmo utilizado e por fim indicou-se qual seria a escalabilidade do algoritmo, independente da forma de obtenção dos números aleatórios.
Ao final, foi concluído que no problema da integração numérica pela regra dos trapézios, apenas são apresentados os tempos, os devidos gráficos de speedup e da eficiência do algoritmo implementado, discutiu-se os resultados obtidos, encontrando explicações para os comportamentos dos gráficos em certos pontos e encerrando com a indicação da escalabilidade do algoritmo.

Referências

Introdução a Sistemas Paralelos

Calculando o valor de Pi via método de Monte Carlo

Monte Carlo Method -- from Wolfram MathWorld

Estimating the value of Pi using Monte Carlo

Integração numérica - Ricardo Biloti

Integração Numérica - Paulo J. S. Silva

Regra dos Trapézios

INTEGRAÇÃO NUMÉRICA