Nome: João Vitor de Freitas Barbosa
DRE: 117055449
Nome: Leonardo Emerson André Alves.
DRE: 117062624.

Computação Concorrente

Trabalho 1 - Relatório

Cálculo de integrais definidas

Descrição do problema:

Soma de Riemann: A integral de $f(x)$ no intervalo $[a, b]$ é igual ao limite do somatório de cada um dos valores que a função $f(x)$assume, de 0 a n, multiplicados por Δx. O que se espera é que quando n for muito grande o valor da soma se aproxime do valor da área abaixo da curva e, portanto, da integral de $f(x)$ no intervalo. Ou seja, que o limite esteja definido. A definição de integral aqui apresentada é chamada de soma de Riemann e é a abordagem que iremos utilizar.

$$\int_a^b f(x)dx = \lim_{\Delta x \to 0} \sum_{i=0}^{n} f(x_i^*)\Delta x$$

onde

$\Delta x = \frac{b-a}{n}$ é o comprimento dos pequenos subintervalos nos quais se divide o intervalo $[a, b]$. Os extremos destes intervalos são os números $x_0(= a)$, x_1, ..., $x_n(= b)$.

onde $x_i^* = \lim_{\Delta x \to 0} i \cdot \Delta x + a$ equivale a um ponto no intervalo de a até $b$da função quando o valor do número de termos n tende a infinito ou equivalentemente quando o valor de Δx tende a 0.

onde

$f(x_i^*)$ é o valor (“altura”) da função $f(x)$ quando xé igual ao ponto amostral x_i^*, definido como um ponto que está no subintervalo $[x_{i-1}, x_i]$(podendo até mesmo ser um destes pontos extremos do subintervalo.

A soma é dada pela divisão da região a ser calculada em formas (retângulos, trapézios, parábolas ou cubos) que juntos formam uma região que é similar àquela a ser medida, então calcula-se a área de cada uma das formas, e finalmente soma-se todas essas áreas menores juntas. Essa abordagem pode ser usada para encontrar uma aproximação numérica para a integral definida mesmo se o teorema fundamental do cálculo não ajudar a encontrar uma forma fechada.

Tendo em vista que a região preenchida pelas formas menores geralmente não corresponde a exata forma da região a ser medida, a Soma de Riemann será diferente desta. Esse erro pode ser reduzido se a região for mais dividida, usando formas cada vez menores. Ao passo que as formas ficam menores, a soma se aproxima a Integral de Riemann.

Como queremos uma boa aproximação para o cálculo de nossa integral, teremos uma entrada N bem grande, então a carga de trabalho para apenas uma thread seria grande demais. Intercalando a carga de trabalho para mais de 1 thread, sendo que cada uma pegaria pedaços diferentes de $(b - a)/N$ para calcular.,

O método utilizado por ser visualizado neste link.

Dados de entrada do nosso programa concorrente:

./trabalho1 <a> <b> <subintervalos N> <número de threads>

onde a e b são os limites de integração, sendo a < b

Dados de entrada do nosso programa sequencial:
./trabalho1Sequencial <a> <b> <subintervalos N>

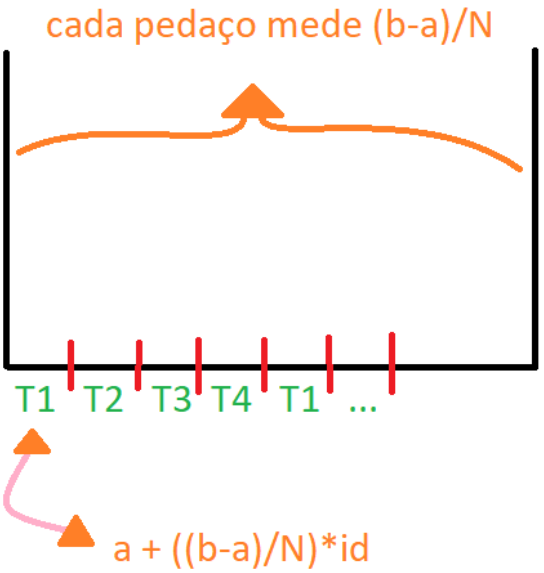
A saída esperada de ambos os programas é o valor da integração e o tempo que foi gasto no cálculo.

Projeto da Solução Concorrente
Temos no escopo global o número de threads - $nThreads$
O valor N que é a divisão da região a ser calculada - N
As variáveis tipo $double\ a,\ b$ representando o intervalo em que a integral deve ser calculada
Temos também um método chamado $funcao$ que retorna o valor em $double$ da função que queremos calcular no ponto x.

As threads receberão como parâmetro apenas o identificador para calcular seus respectivos "quadrados" e irá retornar um valor chamado $somaLocal$ que consiste no somatório do valor da função desejada naquele ponto vezes (b - a)/N.
Algo como y * (b-a)/N. Não há diferenciação no segundo fator da multiplicação pois todas as threads multiplicam pelo mesmo tamanho da base do quadrado, o que muda é apenas o valor da função no ponto determinado, que é dado em i = a + ((b-a)/N)*id, tornando assim possível saber em qual parte a thread está calculando e saber o valor da função naquele ponto específico. Por isso, logo após faço uma variável x receber esse i e depois passo para $y\ =\ funcao(x)$, recebendo assim, o valor da função naquele ponto.
Já na main, há um variável que recebe o retorno das threads em $somaConc$ que é o resultado de todas threads, ou seja, o resultado da nossa integral desejada.

Na solução sequencial teríamos um loop começando em a e indo até b. Com nossa variável de iteração sendo somada de (b-a)/N
Algo como: for (double i = a; i < b; i += (b - a) / N)

Já na solução concorrente, cada thread deve começar na sua respectiva área e ir calculando as outras de acordo com a quantidade de threads.
Teríamos algo como: for(double i = a+ ((b-a)/N)*id; i < b; i+= ((b-a)/N)*nThreads)
onde id representa o identificador da thread atual e nThreads é o número de threads da execução do programa.



Uma outra forma de dividir a tarefa de maneira concorrente, seria atribuir um pedaço contínuo de (b-a)/N para cada thread, ao invés de fazê-las calcular pedaços intercalados.

Casos de teste

Os testes foram realizados em um sistema ubuntu 20.04.2 LTS com processador intel core i5-4440 com 3.10GHZ x 4

Consideramos 8 valores para o número de subintervalos que o método utiliza, sendo n = 10^3, 10^4, 10^5, 10^6, 10^7, 10^8, 10^9, 10^10. Com isso, executamos o código 5 vezes e registramos o menor tempo. A seguir temos um quadro com os valores, e após o quadro realizamos uma investigação sobre o desempenho do programa concorrente em relação ao programa sequencial.

O programa será executado com os seguintes casos:

$\int_a^b cos(5x)\,dx$, com a = 2 e b = 5

$\int_a^b e^{(1-x^{2})}\,dx$, com a = 2 e b = 5

$\int_a^b \sqrt{1 + cos(4x)}\,dx$, com a = 0 e b = pi

Em todos os casos usaremos a biblioteca math.h para calcular o valor da função no ponto em questão. E também verificaremos a corretude dos valores obtidos no wolfram e symbolab.
Usaremos algo como $cos(5 * x)$, $exp(1 - (x * x))$ e $sqrt(1 + cos(4 * x))$ para calcular o valor da função no ponto desejado usando a biblioteca math.h

Avaliação de desempenho com relação ao tempo

Usaremos a Lei de Amdahl para estimar o ganho de desempenho, que consiste em $\frac{T_{sequencial}}{T_{concorrente}}$,

onde $T_{sequencial}$ é o tempo total de execução do programa sequencial
e $T_{concorrente}$ é o tempo total de execução do programa concorrente.

$\int_{a}^{b} cos(5x)\, dx$, com a = 2 e b = 5

Corretude: 0.0823338721583194 (valor obtido no wolfram)

Alguns dos valores obtidos variando o número de threads e o valor de N:
0.082333870886072, 0.082333865191473, 0.082333865191472, 0.082333865191481, 0.082333865028385, 0.082333878455364

	Sequencial	1 Thread	2 Threads	3 Threads	4 Threads
10^3	0.0000447	0.001271	0.001360	0.001651	0.002097
10^4	0.000718	0.001630	0.001551	0.001372	0.002160
10^5	0.00296	0.003910	0.002508	0.002098	0.002006
10^6	0.03134	0.026487	0.0208	0.01690	0.01725
10^7	0.3118	0.3449	0.1718	0.1243	0.1143
10^8	3.10333	3.6975	1.8405	1.2384	0.9728
10^9	31.3287	38.1514	18.632	12.358	9.7052
10^10	311.4633	373.8969	202.7465	149.3058	107.4242

Tabela de ganhos:

	1 Thread	2 Threads	3 Threads	4 Threads
10^3	0,0351	0,0328	0,0270	0,0213
10^4	0.44049	0.4629	0.5233	0.33240
10^5	0.7570	1.1802	1.4108	1.4755
10^6	1.1832	1.5067	1.8544	1.81681
10^7	0.9040	1.8149	2.5084	2.7279
10^8	0.83930	1.6861	2.5059	3.1901
10^9	0.82116	1.6814	2.53509	3.22803
10^10	0,83301	1,5362	2,086	2,8993

Podemos ver que para os valores de N menor que 10^5, os ganhos significativos no corrente passam a ser irrelevantes.

$\int_{a}^{b} e^{(1-x^{2})} dx$, com a = 2 e b = 5

Corretude: 0.011268731614284087991 (valor obtido no wolfram)

Alguns dos valores obtidos variando o número de threads e o valor de N: 0.011269478435171, 0.011268739084111, 0.011268806295129, 0.011268732429814, 0.011268732408121

	Sequencial	1 Thread	2 Threads	3 Threads	4 Threads
10^5	0.00234	0.0068	0.00273	0.00172	0.00233
10^6	0.028728	0.0236	0.01554	0.01394	0.01251
10^7	0.23558	0.2100	0.1091	0.0793	0.0672
10^8	2.3448	2.0693	1.0336	0.6887	0.5576
10^9	22.2733	20.4836	10.2409	6.9249	5.4817
10^10	226.1559	209.0831	104.5796	71.0283	53.4909

Tabela de ganhos:

	1 Thread	2 Threads	3 Threads	4 Threads
10^5	0.34411	0.85714	1.3604	1.0042
10^6	1.2172	1.8486	2.0608	2.2964
10^7	1.1218	2.1593	2.9707	3.5056
10^8	1.1331	2.2685	3.40467	4.2051
10^9	1.0873	2.17493	3.21640	4.0632
10^10	1.08165	2.16252	3.18402	4.2279

$\int_a^b \sqrt{1 + cos(4x)}\, dx$, com a = 0 e b = pi

Corretude: 2.82842712474619 (valor obtido no wolfram)

Alguns dos valores obtidos variando o número de threads e o valor de N:
2.828471552642511, 2.828431567613391, 2.828427568797326, 2.828427129481422, 2.828427056432002, 2.828426491340839, 2.828427128726428, 2.828426491340839

	Sequencial	1 Thread	2 Threads	3 Threads	4 Threads
10^5	0.00519	0.00966	0.00518	0.00198	0.00211
10^6	0.03255	0.0392	0.02224	0.01657	0.01524
10^7	0.32329	0.3369	0.17059	0.1354	0.1106
10^8	3.35532	3.3031	1.6655	1.1487	0.9309
10^9	32.2086	32.87986	16.32902	11.3205	8.7181
10^10	321.3424	330.6901	164.0940	110.9336	101.3710

Tabela de ganhos:

	1 Thread	2 Threads	3 Threads	4 Threads
10^5	0.53726	1.0019	2.6212	2.4597
10^6	0.8303	1.4635	1.9643	2.1358
10^7	0.9596	1.8951	2.3876	2.9230
10^8	1.0158	2.0146	2.9209	3.6043
10^9	0.9795	1.9724	2.8451	3.6944

10^10	0.97173	1.9582	2.8967	3.1699

Discussão

Estes quadros de ganhos exemplificam muito que o objetivo de se implementar algoritmos concorrentemente é benéfico quando temos carga de processamento impactante. No primeiro exemplo, onde temos quantidades de subintervalos não muito grandes como 10^3 e 10^4 subintervalos é perceptível que o aumento do número de threads ao invés de auxiliar no aumento de desempenho, faz piorar, pois nestes casos o overhead de criação das threads é maior do que o ganho de implementar o código de forma concorrente. Porém quando aumentamos a quantidade de subintervalos para 10^5 é perceptível que começa a ocorrer um pequeno aumento de desempenho em decorrência do aumento de threads, isto é, a implementação concorrente começa a fazer sentido. Por fim, ao aumentarmos a quantidade de subintervalos até 10^10, vemos que a aceleração ao comparar o tempo sequencial e o concorrente (com 4 threads) é igual a 3.1699 (último caso de teste), chegando até a ter 4.2279 de ganho em relação ao sequencial no segundo caso de teste, isto é, temos uma aceleração condizente com o que se espera do ganho de desempenho do algoritmo.

Referências bibliográficas

acessado em 19 de abril - khanacademy.org - link 1

acessado em 19 de abril - ecalculo if usp - link 2

acessado em 19 de abril - Reimann sum - link 3

acessado em 19 de abril - Mathinsight - link 4

acessado em 20 de abril - sfu.ca - link 5