Primeiro Trabalho - Relatório Parcial

Computação Concorrente (MAB-117) – 2021/1 Remoto

Paralelização de integração pelo método de Simpson

Christopher Ciafrino, Igor Vilhalba

August 29, 2021

1 Descrição do problema

O *método de Simpson* é um método de integração numérica onde utilizamos um *polinômio de Lagrange* $P(x)$ para aproximar uma função $f(x)$ em um subintervalo $[a, b]$ tal que:

$$\int_a^b f(x)\,dx \approx \int_a^b P(x)\,dx \tag{1}$$

$P(x)$ aproxima $f(x)$ utilizando três pontos distintos: $(a, f(a))$, $(m, f(m))$ e $(b, f(b))$, onde $m = \frac{a+b}{2}$.
Isso nos dá a seguinte expressão para $P(x)$

$$P(x) = f(a)\frac{(x-m)(x-b)}{(a-m)(a-b)} + f(m)\frac{(x-a)(x-b)}{(m-a)(m-b)} + f(b)\frac{(x-a)(x-m)}{(b-a)(b-m)} \tag{2}$$

Dada essa aproximação, podemos encontrar integral de $P(x)$ no intervalo $[a, b]$

$$\int_a^b P(x)\,dx = \frac{b-a}{6}\left[f(a) + 4f(m) + f(b)\right] \tag{3}$$

Entretanto, essa aproximação só é boa para intervalos pequenos. Uma solução para isso é: dada uma função $f(x)$ definida em um intervalo $[a, b]$, dividiremos o intervalo em um número n (par) de subintervalos $[x_i, x_{i+1}]$, para $i = 0, 1, ..., n-1$ e calcularemos a integral desses subintervalos.

Ou seja, primeiramente precisamos calcular o valor de $f(x)$ em $n+1$ pontos x_k, com $k = 0, 1, ..., n$, de tal forma que $x_k = a + kh$ e $h = \frac{a-b}{n}$ é o tamanho de cada um desses subintervalos.

Com essa abordagem, obtemos a seguinte nova igualdade:

$$\int_a^b P(x)\,dx = \frac{h}{3}\left[f(x_0) + 2\sum_{i=1}^{\frac{n}{2}-1} f(x_{2i}) + 4\sum_{i=1}^{\frac{n}{2}} f(x_{2i-1}) + f(x_n)\right] \tag{4}$$

Dada essa descrição do método, a entrada esperada do nosso programa é uma expressão matemática, que calculará os valores para f(x), o limite inferior e superior do intervalo de integração, a quantidade de subintervalos (que deve ser par!) e a quantidade de threads que se deseja executar. A saída é a soma obtida em cada thread.

Por exemplo, vamos considerar uma função qualquer $f(x)$. No nosso programa, criaremos um vetor *vet* (não necessariamente com esse nome), de tamanho $n + 1$.
Cada elemento $vet[i]$ desse vetor deve corresponder a $f(x_i)$, para $i = 0, 1, ..., n$

Como podemos ver em (4), há dois somatórios, que podem ser facilmente divididos. Além disso, os resultados são independentes, o que torna possível dividir essa tarefa entre múltiplos fluxos de execução.

2 Projeto inicial da solução concorrente

2.1 Estratégia de paralelização

Esse problema pode ser paralelizado de duas formas diferentes. Para falarmos de cada uma, vamos chamar de Σ_{2i} e Σ_{2i-1}, respectivamente, os somatórios $\sum_{i=1}^{\frac{n}{2}-1} f(x_{2i})$ e $\sum_{i=1}^{\frac{n}{2}} f(x_{2i-1})$ que aparecem em (4).

1. Uma forma possível é dividir as tarefas em um número par de sub-tarefas. Isso porque dividiríamos cada um dos somatórios em um número igual fluxos de execução, onde cada fluxo calcularia exclusivamente ou Σ_{2i}, ou Σ_{2i-1}.
   Por exemplo, se o usuário desejar que a integral seja calculada utilizando 6 threads, 3 calcularão *somente* o somatórios Σ_{2i} e 3 calcularão *somente* Σ_{2i-1}, totalizando as 6 threadssolicitada.

2. A outra forma consiste em calcular uma parte de Σ_{2i} e de Σ_{2i-1} em cada fluxo de execução. Nessa abordagem, o número de threads não fica restrito a um número par.
   Por exemplo, vamos supor que o usuário queira que a integral seja calculada utilizando 3 threads, com n subintervalos, e seja $m = n/2$.

   - Uma thread calculará:
     - Σ_{2i} com $i = 1, 2, ..., (m-1)/3$
     - e Σ_{2i-1} com $i = 1, 2, ..., m/3$
   - outra thread calculará:
     - Σ_{2i} com $i = \frac{m-1}{3} + 1, \frac{m-1}{3} + 2, ..., (2m-2)/3$
     - e Σ_{2i-1} com $i = (m/3) + 1, (m/3) + 2, ..., 2m/3$
   - e a thread restante calculará:
     - Σ_{2i} com $i = \frac{2m-2}{3} + 1, \frac{2m-2}{3} + 2, ..., m-1$
     - e Σ_{2i-1} com $i = (2m/3) + 1, (2m/3) + 2, ..., m$

Nós optamos pela segunda forma, já que o usuário terá maior liberdade para escolher o número de fluxos de execução. Além disso, acreditamos que essa estratégia faça melhor uso da *cache*, pois como um somatório utiliza os índices ímpares e o outro os índices pares do vetor que guarda os valores de $f(x)$, na prática esses índices posições intercaladas na memória e pertencem à mesma porção da memória. Ou seja, se uma porção do vetor for colocada na *cache*, a thread aproveitará totalmente essa porção que foi carregada, não só os índices pares ou ímpares.

2.2 Planejamento de implementação

Não precisaremos retornar nenhuma *struct* de cada thread. Basta retornarmos a soma dos resultados das porções de Σ_{2i} e Σ_{2i-1} calculados nas threads.
Basta somarmos esses resultados na função que chama as threads, então teremos o valor total da integral.

O vetor onde serão armazenados os valores de $f(x_i)$, para $i = 0, 1, ..., n$ terá escopo global, assim como a variável que armazenará quantas threads serão executadas. Dessa forma, apenas precisaremos passar para as threads um *id* para que saibamos que porção dos somatórios precisamos calcular nela.

3 Testes

3.1 Teste de corretude

Para testar a corretude nós vamos escolher diversos tipos de funções, desde casos mais simples (integrais que sabemos na mão) até casos mais complexos (um exemplo seria uma integral sem primitiva analítica), e testá-los com diversos intervalos (em torno de 10, com tamanhos distintos). Para determinar se o valor encontrado está correto, usaremos a fórmula do erro do próprio método. Que é dada como: a integral calculada pelo método deve possuir erro inferior a

$$\frac{M(b-a)h^4}{180}$$

em que M é o valor máximo de $f^{(4)}(x)$ no intervalo $[a, b]$ e h é a quantidade de blocos em que dividimos o intervalo, ou seja, $h = \frac{(b-a)}{n}$.

3.2 Teste de desempenho

Na parte de desempenho teremos uma abordagem um pouco distinta, daremos mais ênfase em testar intervalo com magnitudes diferentes em busca de variações na performance (intervalos muito grandes e muito pequenos, por exemplo). Vamos pegar todas funções testadas anteriormente e adicionar alguns casos extremos adicionais, para ver como o programa concorrente se comporta.

4 Referências Bibliográficas

1. Simpson's Rule
2. Apostila de Cálculo Númerico do professor Collier (fornecida na disciplina)
3. expr – biblioteca utilizada para interpretar expressões matemáticas