UFRJ - UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
CCMN - Centro de Ciências Matemáticas e da Natureza
DCC - Departamento de Ciência da Computação
MAB117 - Computação Concorrente

INTEGRAÇÃO NUMÉRICA

Rio de Janeiro
Janeiro/2016

Alunos:
Douglas Quintanilha Barbosa Ferreira
Julia Anne de Souza Alves

Professora:
Silvana Rossetto

Índice

1 Introdução

1.1 A Solução

Através da estratégia de quadratura adaptiva, utilizando o metódo de integração numérica retangular com ponto médio foi implementado um algoritmo de cujo objetivo é aproximar o valor de uma dada integral definida com os subintervalos de forma dinâmica das funções a seguir:

1) $f(x) = 1 + x$
2) $f(x) = \sqrt{1 - x^2}, -1 < x < 1$
3) $f(x) = \sqrt{1 + x^4}$
4) $f(x) = \text{sen}(x^2)$
5) $f(x) = \cos(e^{-x}) * (0.005 * x^3 + 1)$

Foram implementadas duas versões, uma concorrente e uma sequencial, de solução para cada problema citado.

As entradas do programa sequencial são: limite superior de integração, limite inferior de integração e erro máximo.

As entradas do programa paralelo são: limite superior de integração, limite inferior de integração, erro máximo e número de threads. O número de threads poderá variar de 0 a 8 sendo criticado, no momento da compilação, os valores diferentes destes.

Na execução da função dois também há a critica, no momento da compilação, a limites de integração inferiores a -1 ou superiores a 1.

A saída do programa segue o padrão abaixo:

> – FUNÇÃO *NOME VERSAO* –
> Número de threads é: <*#threads*>
> Resultado integral de NOME é: <*resultado*>
>
> Tempo de inicialização é: <*inicialização*>
> Tempo de execução é: <*execução*>
> Tempo total é: <*inicialização+execução*>

Onde NOME é a o nome identificador da função e VERSAO é paralela ou sequencial.
A linha que sinaliza o numero de threads apenas aparece na versão paralela, obviamente.

1.2 Equipe

A equipe composta por Julia Anne e Douglas Quintanilha.

1.2.1 Divisão de Tarefas e Comunicação

A divisão de tarefas seguiu-se juntamente como fluxo de execução das atividades combinado com a disponibilidade de cada membro da equipe.

Inicialmente, marcamos reuniões em dias consecutivos para entendermos o problema e discutir a melhor estratégia de resolução.

Com a necessidade de comunicação frequente, foi utilizado aplicativos mobile para melhor controle de tarefas.

Além disso, todo o código foi versionado e pode ser visto em

https://github.com/juliaanne/IntegracaoNumerica

1.2.2 Compreensão e Dificuldades do Problema

Na solução sequencial foi importante entender como o metódo do ponto médio funcionava e pensar em como o programa executaria por meio recursivo as diversas iterações.

Após a implementação da solução sequencial e o entendimento de como funciona o método, a dificuldade foi projetar o modelo concorrente para a solução do problema.

As partes mais críticas foram casos específicos de como distribuir o calculo a divisão do intervalo de integração pelo numero de threads para que nenhuma ficasse ociosa.

Outra parte crítica foi a modularização do programa para que o código pudesse ficar organizado e limpo facilitando o debug do mesmo. Como o problema é muito complexo foi necessário uma atenção especial a essa parte para que não houvesse o comprometimento da execução do problema.

Outro ponto a ressaltar foi um erro que podia ser facilmente cometido e poderia passar despercebido foi o calculo de erro que deveria sempre ser calculado em valor absoluto.

IMPREVISTOS ENCONTRADOS

Começamos o desenvolvimento do trabalho no dia 21 de dezembro, pensando com calma a melhor maneira de solucionar o problema e buscando fazê-lo com o mais adequado desempenho. Como pode ser visto no nosso repositório do trabalho no GitHub.

Com o trabalho já terminado e o relatório pronto, descobrimos no dia da entrega 05 de janeiro que tinhamos interpretado o problema de outra maneira. Embora as nossas soluções se propusessem a terminar o problema com o valor correto, não resolvíamos conforme desejado.

Por essa razão, no mesmo dia fizemos uma nova versão sequencial e tentamos desenvolver uma versão concorrente por dois dias consecutivos. Infelizmente, não tivemos tempo hábil para terminá-la e preferimos entregar a antiga versão para não sofrer maiores penalidades.

Os testes quanto ao **ganho** presentes neste relatório são comparações entre as versões sequenciais e paralelas da primeira resolução. Acreditamos não fazer sentido comparar versões sequenciais e concorrentes que resolvem o problema por métodos diferentes.

Nos restringimos a colocar neste relatório, na devida parte de testes, apenas o **tempo sequencial** gasto e sua **saída real** e saída esperada referente a nova versão, sendo estes referidos como *Tempo Sequencial por Partições* e *Saída Sequencial por Partições*, respectivamente.

Pedimos desculpa pelo interpretação errada do problema e agradecemos a compreensão.

2 Desenvolvimento

2.1 Solução Sequencial

Os argumentos da entrada são utilizados no programa respectivamente como limite inferior, limite superior e erro máximo. Caso nenhuma dessas entradas seja respeitada, o programa se encerra com erro.

2.1.1 Das Estruturas de Dados

Foi utilizada variáveis do tipo inteiro e double para os calculos das funções.

2.1.2 Da Solução Projetada

2.1.2.1 Solução com erro de toda função

A função de calculo da integral é feita em etapas:

Definição do intervalo sendo o (limite superior – limite inferior) dividido pelo numero de partições, que na primeira iteração vale 1.

Define-se o limite inferior da partição como o limite inferior da integração.

Através de um laço de 0 até o numero de partições é calculado o valor de cada partição sendo intervalo vezes função no ponto (limite inferior da partição + limite superior da partição) dividido por 2. Onde limite superior da partição é limite inferior da partição + tamanho do intervalo.

Com o resultado acima, é calculado o erro subtraindo-se do resultado da iteração anterior o resultado atual. Na primeira iteração, o resultado da iteração anterior é zero.

Caso o erro obtido acima seja maior que o erro informado na entrada do programa, não é considerado um erro aceitável e é calculado novamente todos os passos acima com o numero de partições dobrado. Caso o erro seja menor, retorna-se o valor desejado.

2.1.2.2 Solução com erro de cada partição

A função de calculo da integral é feita em etapas:

Definição do intervalo sendo o (limite superior – limite inferior). Após isso, dividi-se este intervalo em dois e calcula-se a area destes dois intervalos, sendo area da esquerda um quarto do intervalo vezes metade do intervalo e a area da direita sendo três quartos do intervalo vezes metade do intervalo.

Calcula-se o erro somando os valores das duas áreas e comparando com o resultado da iteração anterior, que seria a área do pai.

Se este erro não for aceitável, ou seja, se ele for maior que o erro máximo, recursivemente chamamos a função para cada área (esquerda e direita) calculada.

Se este erro for aceitável, retornarmos o valor das integrais somadas.

2.1.3 Da Saída do Programa

-- FUNÇÃO NOME SEQUENCIAL --
O valor da integral de NOME de LIMITE SUPERIOR até LIMITE INFERIOR é: RESULTADO
Tempo de inicialização é: TEMPO1
Tempo de execução é: TEMPO2
Tempo total é: TEMPO1+TEMPO2
Onde nome é a identificação da função, exemplo: SENO.

2.2 Solução Paralela

Os argumentos da entrada são utilizados no programa respectivamente como limite inferior, limite superior, erro máximo e número de threads. Caso nenhuma dessas entradas seja respeitada, o programa se encerra com erro. Caso o número de threads seja menor que 1 ou maior que 8, o programa se encerra com erro.

2.2.1 Das Estruturas de Dados

Foi utilizada variáveis do tipo double para os calculos das funções e controle do fluxo de execução. Também foram utilizados dois vetores principais, um que guardava o resultado parcial da integral, calculado por cada thread e o outro com o fim dos intervalos de integração, para que estes valores pudessem ser consumidos pelas threads, se baseando na estratégia Produtor / Consumidor.

2.2.2 Da Solução Projetada

A solução projetada para resolver o problema concorrente foi de dividir a carga de calcular a integral em partições, que são pedaços menores do intervalo de integração, e distribuí-las em blocos para as threads, com o objetivo de reduzir o erro numérico.

Como o método da quadratura adaptativa deveria ser utilizado, foi necessário ter várias iterações para o cálculo da integral, sempre verificando se o erro desejado havia sido obtido. Com isto em mente, adotamos a estratégia de utilizar uma barreira ao final de cada iteração, para que todas as

threads esperassem as outras finalizarem seus respectivos pedaços do cálculo da integral. Dentro da barreira também aproveitávamos para atualizar os valores do erro, número de partições para a próxima iteração e atualizar o valor final da integral com o valor obtido na iteração atual.

Para a divisão do trabalho em blocos, utilizamos como inspiração a estratégia Produtor / Consumidor, em que a cada iteração dividimos o intervalo de integração em partições menores, e estas partições são agrupadas em pequenos blocos, que são identificados pelo número da última partição de seu bloco. Estes blocos são salvos em vetor de partições que serve como um buffer. Este buffer é consumido quando as threads iniciam seu cálculo de integral, removendo um bloco do buffer, calculando as partições iniciais e finais e realizando o cálculo da integral nestas pequenas partições.

Como exemplo desta divisão, vamos supor que rodemos o programa utilizando 5 threads e em determinada iteração, dividimos o intervalo de integração em 128 partições. Como queremos obter o balanceamento de carga, dividimos as partições entre nossas 5 threads, arredondando pra baixo. Obtemos 25 partições por thread nessa iteração, como cada bloco é identificado pela sua última partição, preenchemos o vetor de partições da seguinte maneira: [24][49][74][99][127]. Vale atentar para o último elemento do vetor, que é identificado pela partição 127, se tornando assim o maior bloco, já que temos uma divisão com resto e o nosso ultimo bloco incorpora as partições restantes.

2.2.3 Da Saída do Programa

-- FUNÇÃO NOME PARALELA --
Número de threads é: #THREADS
Resultado da integral de NOME é: RESULTADO

Tempo de inicialização é: TEMPO1
Tempo de execução é: TEMPO2
Tempo total é: TEMPO1+TEMPO2
Onde nome é a identificação da função, exemplo: SENO.

3 Testes

Foi desenvolvido um script que calcula o ganho de desempenho e pode ser visto nos arquivos de código fonte.

3.1 Função 1

A função f (x) = 1 + x, apelidada como função linear, pode ser vista na solução sequencial em linearSequencial.c e na solução concorrente em linearParalela.c.

3.1.1 Saída Esperada e Saída Real

Caso 1

Limite de integração: [0, 10]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 60
Situação: **APROVADO**
Saída Sequencial: 60.000000
*Saída Sequencial por Partições**: 60.00000000*
Saída Paralela para de 1 até 8 threads: 60.000000

Caso 2

Limite de integração: [50, 130]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 7280
Situação: **APROVADO**
Saída Sequencial: 7280.000000
*Saída Sequencial por Partições**: 7280.00000000*
Saída Paralela para de 1 até 8 threads: 7280.000000

Caso 3

Limite de integração: [-30, 70]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 2100
Situação: **APROVADO**
Saída Sequencial: 2100.000000
*Saída Sequencial por Partições**: 2100.00000000*
Saída Paralela para de 1 até 8 threads: 2100.000000

3.1.2 Ganho de Desempenho

Caso 1

Limite de integração: [0, 10]
Erro: 0.00001
Tempo Sequencial: 0.00001216
*Tempo Sequencial por Partições**: 0.00003099*
Ganho Real para 1 thread: 0.04860998737823230525
Ganho Real para 2 thread: 0.04549271048011602167
Ganho Real para 3 thread: 0.04381391618207339081
Ganho Real para 4 thread: 0.01891151067974554795
Ganho Real para 5 thread: 0.01891151067974554795
Ganho Real para 6 thread: 0.01870846198126537233
Ganho Real para 7 thread: 0.02870030016533170676
Ganho Real para 8 thread: 0.02006402006402006402

Caso 2

Limite de integração: [50, 130]
Erro: 0.00001
Tempo Sequencial: 0.00001979
*Tempo Sequencial por Partições**: 0.00000882*
Ganho Real para 1 thread: 0.03706816844582667899
Ganho Real para 2 thread: 0.02835027510957754359
Ganho Real para 3 thread: 0.03481732709397673120
Ganho Real para 4 thread: 0.03974801869538711643
Ganho Real para 5 thread: 0.02476987033554389785
Ganho Real para 6 thread: 0.02232962421790116518
Ganho Real para 7 thread: 0.02812435477686340441
Ganho Real para 8 thread: 0.03062945937998173685

Conclusão: Como a função é linear e seu erro por ser simetrico será compensado, é necessário apenas 1 iteração, tornando quanto maior o numero de threads menor o ganho pois se torna muito custoso a criação e manutenção das mesmas.

3.2 Função 2

A função $f(x) = \sqrt{1 - x^2}$, $-1 < x < 1$, apelidada como função parabola, pode ser vista na solução sequencial em parabolaSequencial.c e na solução concorrente em parabolaParalela.c.

3.2.1 Saída Esperada e Saída Real

Caso 1

Limite de integração: [-1, 1]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 1.5708
Situação: **APROVADO**
Saída Sequencial: 1.570802
*Saída Sequencial por Partições**: 1.57088585*
Saída Paralela para de 1 até 8 threads: 1.570802

Caso 2

Limite de integração: [0, 1]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 0.78540
Situação: **APROVADO**
Saída Sequencial: 0.785401
*Saída Sequencial por Partições**: 0.78544292*
Saída Paralela para de 1 até 8 threads: 0.785401

3.2.2 Ganho de Desempenho

Caso 1

Limite de integração: [-1, 1]
Erro: 0.00001
Tempo Sequencial: 0.00009584
*Tempo Sequencial por Partições**: 0.00006318*
Ganho Real para 1 thread: 0.15556673754605807781
Ganho Real para 2 thread: 0.08757790083520660854
Ganho Real para 3 thread: 0.05824935433230364105
Ganho Real para 4 thread: 0.06937812561806006837
Ganho Real para 5 thread: 0.12216504875359436144
Ganho Real para 6 thread: 0.03827456864216054013
Ganho Real para 7 thread: 0.08415063082962198240
Ganho Real para 8 thread: 0.06280181772878822035

Caso 2

Limite de integração: [0, 1]
Erro: 0.00001
Tempo Sequencial: 0.00004005
*Tempo Sequencial por Partições**: 0.00003791*
Ganho Real para 1 thread: 0.09887179993581356308
Ganho Real para 2 thread: 0.07884721383763353368
Ganho Real para 3 thread: 0.05461520280711961165
Ganho Real para 4 thread: 0.03746036321706543672
Ganho Real para 5 thread: 0.06162209236992920552
Ganho Real para 6 thread: 0.03071334763020963210

Ganho Real para 7 thread: 0.03919428725410940447
Ganho Real para 8 thread: 0.02432368968672318442

3.3 Função 3

A função f (x) = √1 + x^4, apelidada como função exponencial, pode ser vista na solução sequencial em exponencialSequencial.c e na solução concorrente em exponencialParalela.c.

3.3.1 Saída Esperada e Saída Real

Caso 1

Limite de integração: [0, 20]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 2667.88
Situação: **APROVADO**
Saída Sequencial: 2667.87771398
*Saída Sequencial por Partições**:* 2667.87707642
Saída Paralela para de 1 até 8 threads: 2667.87771398

Caso 2

Limite de integração: [-50, 50]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 83335.8
Situação: **APROVADO**
Saída Sequencial: 83335.78543169
*Saída Sequencial por Partições**: 83335.78049421*
Saída Paralela para de 1 até 8 threads: 83335.78543169

Caso 3

Limite de integração: [30, 100]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 324333
Situação: **APROVADO**
Saída Sequencial: 324333.34499833
*Saída Sequencial por Partições**: 324333.34329630*
Saída Paralela para de 1 até 8 threads: 324333.34499833

3.3.2 Ganho de Desempenho

Caso 1

Limite de integração: [0, 20]
Erro: 0.00001
Tempo Sequencial: 0.00481009
*Tempo Sequencial por Partições**: 0.00111103*
Ganho Real para 1 thread: 0.97723352599981715307
Ganho Real para 2 thread: 1.04059553503169957605
Ganho Real para 3 thread: 1.04034968408233628822
Ganho Real para 4 thread: 1.32995042658666836875
Ganho Real para 5 thread: 1.01563073695253081563
Ganho Real para 6 thread: 0.98728392456305277900
Ganho Real para 7 thread: 1.14780063452169719067
Ganho Real para 8 thread: 1.22357881364903489890

Caso 2

Limite de integração: [-50, 50]
Erro: 0.00001
Tempo Sequencial: 0.04423904
*Tempo Sequencial por Partições**: 0.00472903*
Ganho Real para 1 thread: .79330798079906413716
Ganho Real para 2 thread: 1.39837061385623452093
Ganho Real para 3 thread: 2.13515797035640444977
Ganho Real para 4 thread: 2.25279632552918718288
Ganho Real para 5 thread: 1.89073759236251879945
Ganho Real para 6 thread: 1.55709567358897490932
Ganho Real para 7 thread: 1.63241565127865479260
Ganho Real para 8 thread: 1.62857759214142721565

Conclusão: Quanto maior o valor da entrada consequentemente haverão mais partes a serem calculadas, então quanto mais thread estão em execução, melhor é o desempenho. Mas há o limite de 4 processadores na maquina usada e por isso a melhor opção são 4 threads.

3.4 Função 4

A função f (x) = sen(x^2), apelidada como função seno, pode ser vista na solução sequencial em senoSequencial.c e na solução concorrente em senoParalela.c.

3.4.1 Saída Esperada e Saída Real

Caso 1

Limite de integração: [0, 10]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 0.583671
Situação: **APROVADO**
Saída Sequencial: 0.58366983
*Saída Sequencial por Partições**: 0.58365890*
Saída Paralela para de 1 até 8 threads: 0.58366983

Caso 2

Limite de integração: [-30, 0]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 0.625544
Situação: **APROVADO**
Saída Sequencial: 0.62554149
*Saída Sequencial por Partições**: 0.62550130*
Saída Paralela para de 1 até 8 threads: 0.62554149

Caso 3

Limite de integração: [-50, 100]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 1.25048
Situação: **APROVADO**
Saída Sequencial: 1.25047960
*Saída Sequencial por Partições**: 1.25053121*
Saída Paralela para de 1 até 8 threads: 1.25047960

3.4.2 Ganho de Desempenho

Caso 1

Limite de integração: [0, 10]
Erro: 0.00001
Tempo Sequencial: 0.00086093
*Tempo Sequencial por Partições**: 0.00104403*
Ganho Real para 1 thread: 0.65416733152491888729
Ganho Real para 2 thread: 1.07090738593171716947
Ganho Real para 3 thread: 0.60506471405598663564
Ganho Real para 4 thread: 1.10775096909756185921
Ganho Real para 5 thread: 0.64840442338072669826
Ganho Real para 6 thread: 0.46352353379509167979
Ganho Real para 7 thread: 0.42051500433931853832
Ganho Real para 8 thread: 0.46307249329535768487

Caso 3

Limite de integração: [-50, 100]
Erro: 0.00001
Tempo Sequencial: 0.04709697
*Tempo Sequencial por Partições**: 0.01944709*
Ganho Real para 1 thread: 1.10507493399628374432
Ganho Real para 2 thread: 1.21785849337576591138
Ganho Real para 3 thread: 1.34835852977289461411
Ganho Real para 4 thread: 2.77469337710776982727
Ganho Real para 5 thread: 2.06751266735045442019
Ganho Real para 6 thread: 1.59765718611658419440
Ganho Real para 7 thread: 1.41858223973243388792
Ganho Real para 8 thread: 1.70434457015075437491

3.5 Função 5

A função f (x) = cos(e^{-x}) * (0.005 * x^3 + 1), apelidada como função elefante, pode ser vista na solução sequencial em elefanteSequencial.c e na solução concorrente em elefanteParalela.c.

3.5.1 Saída Esperada e Saída Real

Caso 1

Limite de integração: [0, 20]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 219.759
Situação: **APROVADO**
Saída Sequencial: 219.75925433
*Saída Sequencial por Partições**: 219.75889635*
Saída Paralela para de 1 até 8 threads: 219.75923511

Caso 2

Limite de integração: [10, 100]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 125077
Situação: **APROVADO**

Saída Sequencial: 125077.49999708
*Saída Sequencial por Partições**: 125077.49687971*
Saída Paralela para de 1 até 8 threads: 125077.49998833

Caso 3
Limite de integração: [-10, 40]
Erro: 0.00001
Saída Esperada (de acordo com o Wolfram): 3239.42
Situação: **APROVADO**
Saída Sequencial: 3239.42245489
*Saída Sequencial por Partições**: 3239.48072597*
Saída Paralela para de 1 até 8 threads: 3239.42245938

3.5.2 Ganho de Desempenho

Caso 2
Limite de integração: [10, 100]
Erro: 0.00001
Tempo Sequencial: 0.03894901
*Tempo Sequencial por Partições**: 0.00737309*
Ganho Real para 1 thread: 0.92737725332209499727
Ganho Real para 2 thread: 1.27982814985665394204
Ganho Real para 3 thread: 1.77156115926801505997
Ganho Real para 4 thread: 1.62543339530148845463
Ganho Real para 5 thread: 1.59226783726969209842
Ganho Real para 6 thread: 1.70850844609533917042
Ganho Real para 7 thread: 1.76829728771209750050
Ganho Real para 8 thread: 1.89147951890771425339

Caso 2
Limite de integração: [-10, 40]
Erro: 0.00001
Tempo Sequencial: 0.65455103
*Tempo Sequencial por Partições**: 0.04113102*
Ganho Real para 1 thread: 0.92126304845554514822
Ganho Real para 2 thread: 1.24641002549244750740
Ganho Real para 3 thread: 1.71045404196595274444
Ganho Real para 4 thread: 2.07416488898905156785
Ganho Real para 5 thread: 1.86071912596469920223
Ganho Real para 6 thread: 2.01478774076465175861
Ganho Real para 7 thread: 2.04377787043300127377
Ganho Real para 8 thread: 2.12424916262638742193

Conclusão: Quanto maior o numero de threads, melhor é o desempenho do programa, visto que temos uma função complexa.

4 Considerações Finais

O projeto foi desenvolvido parte a parte sempre pensando na melhor forma de execução e melhor desempenho do software.

O código está completamente comentado sendo de fácil entendimento. Além disso, há uma variante de debug para a versão concorrente, que mostra a execução dos programas passo a passo,

imprimindo os valores dos vetores de partição e de integral, o cálculo que cada thread realiza em cada partiçao , quando cada thread é criada e quando se trava, ficando bem mais fácil acompanhar e entender o fluxo do programa.