UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
PRIMEIRO TRABALHO DE IMPLEMENTAÇÃO 2019/2

PROBLEMA DE QUADRATURA PARA INTEGRAÇÃO NUMÉRICA

Autores: Daniel La Rubia 115033904
Paula Macedo 113049909
Professora: Silvana Rossetto

Rio de Janeiro, 28 de outubro de 2019.

1.Descrição do problema

O trabalho consiste em implementar o método de integração numérica retangular usando a estratégia de quadratura adaptativa para definir os subintervalos de forma dinâmica. O programa possui duas versões: sequencial e paralela, onde na versão paralela, o programa concorrente deve garantir a solução com balanceamento de carga entre as threads, objetivando um desempenho máximo.

2.Solução do problema

É criado um programa que calcula a integral definida de uma função não trivial pelo método de integração numérica do ponto médio. A entrada do programa sequencial é a definição do intervalo de integração e o erro máximo tolerado a ser calculado para um conjunto de funções do tipo f(x). Já no programa concorrente, além do intervalo e do erro, é solicitado também o número de threads do programa.

Foram implementados três algoritmos: Sequencial, Concorrente Balanceado e Concorrente Desbalanceado. O algoritmo sequencial foi implementado utilizando a recursividade e o algoritmo concorrente desbalanceado fez uso da mesma função recursiva, sendo a única diferença a subdivisão do intervalo [a, b] em N, que é o número de *threads*. O algoritmo gera uma árvore de recursividade e realiza o somatório dos valores que são aceitos de acordo com o erro passado pelo usuário.

Já no algoritmo de integração concorrente balanceado, foi utilizada uma pilha, onde as *threads* estariam funcionando como produtoras/consumidoras, consumindo o conjunto de dados armazenado no topo da pilha e realizando uma operação para verificar se o erro da área está no limite tolerado. Se não estiver, é realizado um *push* com os dois novos retângulos. A fim de balancear o problema, é realizada uma carga inicial da pilha com N valores (número de threads).

3.Testes realizados e resultados

O ambiente de testes utilizados neste trabalho é baseado em Linux e a versão do compilador gcc utilizado foi a 9.1. A máquina utilizada possui 4 processadores.

O programa foi compilado da seguinte maneira: ***main.c -o main -lm - lpthread***.

Dado um determinado conjunto de testes, é feita a comparação entre os tempos de execução da solução sequencial e da solução paralela.

As funções utilizadas neste trabalho encontram-se abaixo:

a(x) = 1 + x
b(x) = sqrt(1 - x^2), -1 < x < 1
c(x) = sqrt(1 + x^4)
d(x) = sen(x^2)
e(x) = cos(e^-x)
f(x) = cos(e^-x) * x
g(x) = cos(e^-x) * (0.005 * x^3 + 1)

Alguns dos intervalos utilizados, erros máximo tolerados e no caso do programa paralelo, o número de threads são explicitados na tabela a seguir, assim como o tempo de execução de cada teste.

Teste 1	**Intervalo A**	**Intervalo B**	**Erro máximo**					
	-1	**1**	**1,00E-10**					
	SEQUENCIAL			**CONCORRENTE**	**(BALANCEADO)**		**CONCORRENTE**	(DESBALANCEADO)
Funções	**Resultado**	**Tempo (seg)**	**Nº de threads**	**Resultado**	**Tempo (seg)**	**Nº de threads**	**Resultado**	**Tempo (seg)**
a(x)	2,00000	0,000019	8	2	0,001531	8	2	0,001238
b(x)	1,570796	0,002961	8	1,570796	0,006182	8	1,570796	0,002197
c(x)	2,178859	0,001568	8	2,178859	0,002263	8	2,178859	0,001402
d(x)	0,620537	0,002623	8	0,620537	0,001851	8	0,620537	0,001845
e(x)	0,582151	0,002441	8	0,582151	0,002052	8	0,582151	0,001612
f(x)	0,697994	0,002718	8	0,697994	0,002186	8	0,697994	0,001249
g(x)	0,584238	0,002987	8	0,584238	0,002240	8	0,584238	0,000941
	Tempo total :	**0,015465**		**Tempo Total:**	**0,018359**		**Tempo Total:**	**0.010484**

Teste 2	Intervalo A	Intervalo B	Erro máximo					
	-10000	10000	1,00E-5					
	SEQUENCIAL			CONCORRENTE	(BALANCEADO)		CONCORRENTE	(DESBALANCEADO)
Funções	Resultado	Tempo (seg)	Nº de threads	Resultado	Tempo (seg)	Nº de threads	Resultado	Tempo (seg)
a(x)	20000	0,000029	4	20000	0,001286	4	20000	0,000452
b(x)	NAN	-	4	NAN	-	4	NAN	-
c(x)	6,6 x 10^11	0,152283	4	6,6x10^11	0,451610	4	6,6x10^11	0,063056
d(x)	2,253796	61,0588	4	2,253796	131,002710	4	2,253796	25,501674
e(x)	10806,04	0,000005	4	NAN	0,000128	4	NAN	0,000215
f(x)	0	0,000007	4	NAN	0,000084	4	NAN	0,000103
g(x)	10806,046	0,000007	4	NAN	0,904320	4	NAN	0,211490
	Tempo total :	61,211243		Tempo Total:	132,360223		Tempo Total:	24,561483

Teste 3	Intervalo A	Intervalo B	Erro máximo					
	0	10	1,00E-10					
	SEQUENCIAL			CONCORRENTE	(BALANCEADO)		CONCORRENTE	(DESBALANCEADO)
Funções	Resultado	Tempo (seg)	Nº de threads	Resultado	Tempo (seg)	Nº de threads	Resultado	Tempo (seg)
a(x)	60	0,000020	8	60	0,000979	4	60	0,001026
b(x)	NAN	-	8	NAN	-	4	NAN	-
c(x)	334,519383	0,011078	8	334,519383	0,019631	4	334,519383	0,006258
d(x)	0,583671	0,016143	8	0,583671	0,036096	4	0,583671	0,009889
e(x)	9,790508	0,000531	8	9,790508	0,001993	4	9,790508	0,000901
f(x)	49,906568	0,000596	8	49,906568	0,002030	4	49,906568	0,003236
g(x)	22,289965	0,002876	8	22,289965	0,006170	4	22,289965	0,001686
	Tempo total :	0,031347		Tempo Total:	0,066985		Tempo Total:	0,023081

Teste 4	Intervalo A	Intervalo B	Erro máximo					
	100	100000	3,00E-3					
	SEQUENCIAL			CONCORRENTE	(BALANCEADO)		CONCORRENTE	(DESBALANCEADO)
Funções	Resultado	Tempo (seg)	Nº de threads	Resultado	Tempo (seg)	Nº de threads	Resultado	Tempo (seg)
a(x)	5000094900	0,000053	4	5000094900	0,001283	4	5000094900	0,000491
b(x)	NAN	-	4	NAN	-	4	NAN	-
c(x)	3,3 x 10^14	0,152626	4	3,3 x 10^14	0,477806	4	3,3 x 10^14	0,058039
d(x)	-18,264325	9,751366	4	-18,264325	11,594370	4	-18,264325	3,654192
e(x)	99900	0,000004	4	99900	0,000109	4	99900	0,000115
f(x)	4,9x10^9	0,000007	4	4,9x10^9	0,000078	4	4,9x10^9	0,000178
g(x)	1,249x 10^17	1,318027	4	1,249x 10^17	3,252456	4	1,249x 10^17	0,625806
	Tempo total :	11,222174		Tempo Total:	15,326193		Tempo Total:	4,338925

Durante a execução do programa, cada função é calculada respeitando a ordem que foram chamadas. O usuário não tem o poder de escolher executar o cálculo de uma função isolada.

Foram escolhidos intervalos relativamente pequenos e grandes para a e b, bem como para o erro tolerado, permitindo assim observar o desempenho de cada programa para diferentes abordagens.

Foi observado que ao testarmos as funções para intervalos muito grandes ou com erros muito pequenos, o tempo de processamento é elevado.

Todos os resultados dos testes gerados pelo programa foram checados com os resultados obtidos pelo WolframAlpha.

4.Avaliação de desempenho: Sequencial x Paralelo

O algoritmo de integração sequencial é recursivo, portanto apresenta um excelente desempenho. O algoritmo paralelo deveria ser mais rápido quando se trata de intervalos de integração muito grandes ou erros muito mínimos. Quanto aos três algoritmos tratados no programa, o que apresenta melhor velocidade (menor tempo) é o Concorrente Desbalanceado.

Quanto a implementação, o algoritmo concorrente foi o que apresentou o maior tempo no geral (pior desempenho). Os fatores que podem ter ocasionado isso são, principalmente, o problema encontrado em manter as *threads* em aguardo enquanto uma outra *thread* pode ainda gerar novos valores. O problema será melhor descrito na seção “Dificuldades encontradas”.

O algoritmo paralelo recursivo desbalanceado é tão somente a subdivisão do intervalo **[a, b]** em N subintervalos, onde N é o número de *threads*. Cada *thread* ficou responsável por calcular um subintervalo, usando o próprio algoritmo recursivo do programa sequencial. A causa do desbalanceamento é que a necessidade de cálculos de cada subintervalo varia de acordo com o comportamento da função.

5.Modularidade do programa

O programa encontra-se modularizado com a finalidade de garantir ao programa legibilidade e permitir a facilidade em sua manutenção. O programa foi subdividido em arquivos de extensão .c e .h.

O arquivo ControleThreads.c trata exclusivamente do gerenciamento das threads (criação, sincronização, alocação de memória para as threads e encerramento).

Funcoes.c apresenta a implementação das funções calculadas pelo programa.

O arquivo Timer.h, é o mesmo disponibilizado pela professora nas aulas de laboratório e trata da questão de medição de tempo de execução do programa.

QuadraturaAlgorithm.c apresenta de fato a implementação do método, neste arquivo encontram-se as funções relacionadas à solução sequencial e à solução concorrente.

O arquivo main.c, como nome já diz, apresenta a main do nosso programa, bem como toda a implementação do menu do programa e a chamada das funções.

Não há nenhum outro arquivo .h além do Timer.h, pois não houve a necessidade, já que não há outros arquivos que façam uso da mesma estrutura de dados e não há outras implementações para um mesmo protótipo.

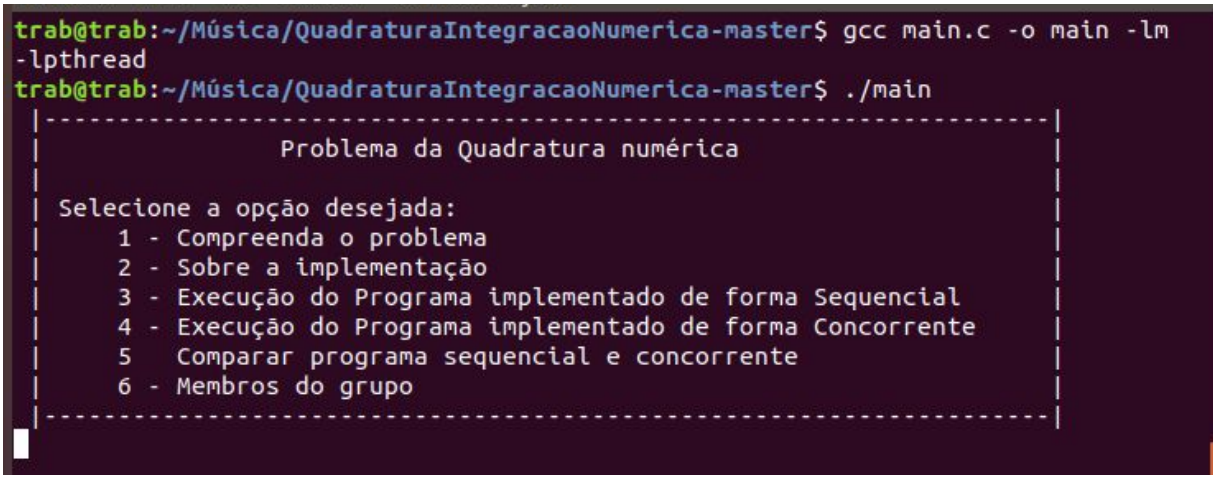
6.Interface usuário

A fim de facilitar a visualização da solução do problema e unificar a chamada para os programas sequenciais e concorrente, o programa apresenta um menu.

Este menu, oferece as opções necessárias para que um usuário entenda a problemática do trabalho, qual abordagem implementada para solução da mesma, além da execução da

versão sequencial e versão concorrente em separado e da execução em conjunto, permitindo assim, um comparativo o resultado das versão em uma única tela.

Adicionalmente, é possível encontrar as informações da dupla, como nome e DRE.

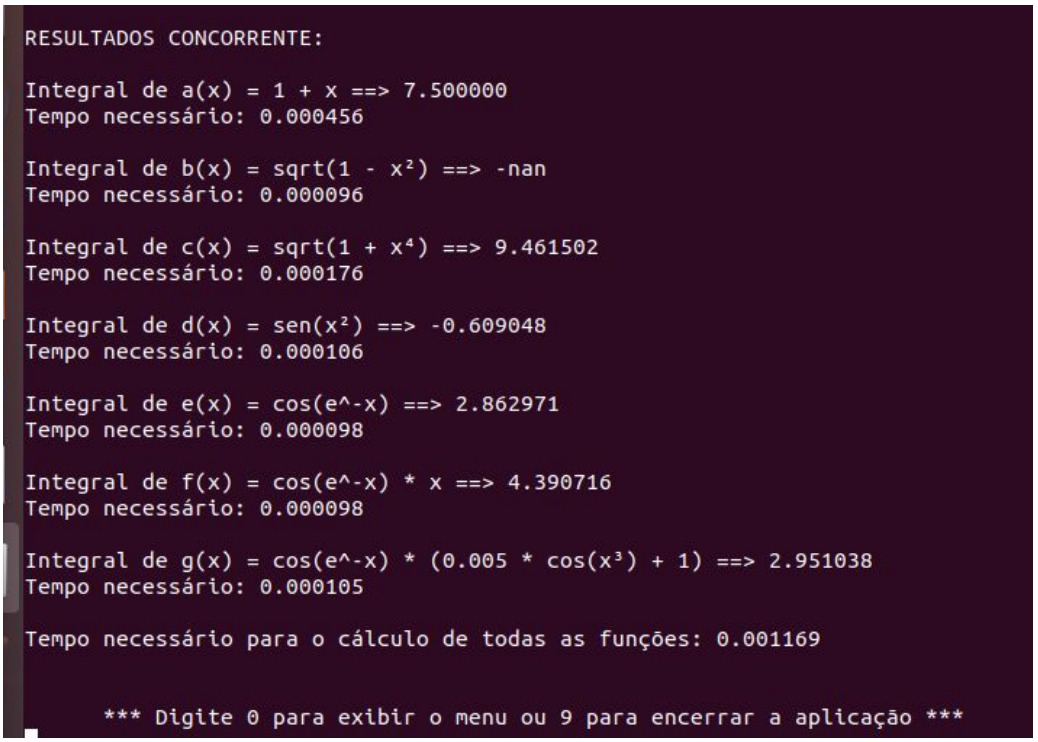


Tela de menu do programa

Ao escolher as opções 1 ou 2, será retornado ao usuário uma descrição do tópico selecionado.

Ao escolher 3, será solicitado ao usuário que digite primeiramente o intervalo [a,b], sendo a [enter], b [enter] e o erro [enter]. Ao escolher 4 ou 5, será solicitado o número de threads [enter] e logo após o mesmo que a opção 3.

Os resultados são impressos com suas respectivas funções, além do tempo de execução para cada função e o tempo de execução total do programa.



Tela de resultado do programa concorrente

Ao final de todas as opções, o usuário tem a opção de retornar ao menu principal ou encerrar o programa.

7.Dificuldades encontradas

Durante a elaboração da resolução deste trabalho encontramos certas dificuldades que serão pontuadas e explicitadas abaixo:

- Entender como funciona a resolução matemática do problema apresentado e saber aplicar o método;
- Terminada a versão sequencial, implementar a versão paralela. A primeira dificuldade foi pensar em que abordagem utilizar, já que apenas dividir o trabalho de cálculo dos subintervalos entre as threads é uma estratégia que não gera um balanceamento de carga. No fim das contas, decidimos implementar essa estratégia já que foi a que nos permitiu atingir o melhor tempo, embora não seja balanceada;
- O balanceamento foi, de forma disparada, a maior dificuldade encontrada na resolução do trabalho. É válido pontuar, inclusive, que uma das possíveis causas para o desempenho ruim do algoritmo concorrente balanceado se dá por conta de que o mesmo não está, de fato, balanceado. É possível que, em alguns casos, principalmente ao se aproximar do final da execução, algumas *threads* sejam encerradas enquanto ainda há alguma(s) *thread(s)* ativa(s), o que pode gerar novos valores na pilha para execução. Nossa ideia para resolução deste problema foi alterar a condição de parada das *threads* para “**parar se não houver threads executando && topo da pilha == vazio**”, fazendo uso de uma variável de condição

para que as *threads* aguardassem até que as que estavam ativas enviassem um sinal para que continuassem (caso novos valores fossem adicionados ao topo da pilha). Entretanto, não conseguimos implementar o controle de *threads ativas*, o que gera o desbalanceamento.

- Na realização dos testes caímos em problemas que, honestamente, fizeram-nos questionar nossa sanidade mental. Situações como o programa travar no intervalo **[-100, 100]**, que não é nem mesmo um intervalo tão grande. Ou em intervalos grandes como na bateria de testes 2, de intervalo **[-10000, 10000]**, onde a partir da função **e(x)** os resultados passavam a serem retornados como *not a number* (NAN). Em intervalos consideravelmente pequenos, o algoritmo apresentou resultados compatíveis se comparados com o *Wolfram Alpha*. Ao crescer os intervalos, ou entrar nos negativos com **[-1000, 0]**, nos deparamos com uma grande demora na função **e(x)** em diante, que já passavam a retornar resultados como NAN.
- Sem sombra de dúvida, a busca pela corretude do próprio algoritmo matemático foi um dos maiores desafios. Às vésperas da entrega do trabalho, cair em situações em que nossos resultados destoava tanto do esperado fizeram com que acreditássemos que toda implementação feita havia sido em vão, que havia algum erro lógico no código, que algum tratamento de condição de corrida havia passado despercebido. Por fim, decidimos colocar nos testes os resultados que encontramos em sua totalidade, mesmo que não estejamos satisfeitos e sabendo que estão incorretos. Também é desejo nosso saber se, no fim das contas, o algoritmo matemático que está incorreto ou se a implementação num geral que foi mal feita.
- Um último problema que é simples de ser resolvido, mas que não foi implementado por questão de tempo, é a utilização do *malloc* / *realloc* para o controle da pilha. Embora não acreditamos ter caído em um caso que a pilha visivelmente estoura (iniciamos a mesma com o valor **102400**), a implementação correta consistiria em fornecer garantias ao usuário de que o mesmo não teria seus cálculos interrompidos por conta do espaço disponível no *buffer*.

8. Conclusão

Para um programa concorrente ter desempenho superior à um programa sequencial, é necessário que sua lógica e implementação assumam estratégias que visem o ganho de velocidade, promovendo o balanceamento de cargas pelas suas threads.

E a elaboração de um programa concorrente apresenta muitos desafios.