UNIVERSIDADE FEDERAL
DO RIO DE JANEIRO

Relatório Integração Numérica

Computação Concorrente-2019.2

Grupo:

-Marcelo Campanelli Nóbrega, DRE: 118067978

-Nathan Vieira Magalhães, DRE: 118038131

Descrição:

Neste trabalho, iremos implementar o método de integração numérica retangular adaptativo, de forma com que o usuário escolha os intervalos, as funções e os erros. Faremos isso de forma sequencial e também de forma concorrente.

1) Sequencial

Primeiro, importamos as bibliotecas.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <time.h>
```

Para implementar a integração de uma forma sequencial, só foi preciso usar o método do ponto médio de forma recursiva, respeitando o erro fornecido pelo usuário. Segue um print da função implementada:

```
double Integral(double a, double b, double err)
{
    double pmedio= (a+b)/2;
    double areaRetMaior= area(a,b);
    double ret1= area(a,pmedio);
    double ret2=area(pmedio,b);
    double error =  areaRetMaior-  (ret1+ret2) ;
    if( module(error) <err )
    {
        return ret1+ret2;
    }else
     return Integral (a,pmedio,err)+Integral(pmedio,b,err);


}
```

Fazemos isso de forma com que a área do retângulo se comporta de acordo com a função “area” descrita abaixo, que recebe um intervalo e multiplica o tamanho dele com a “altura”:

```
double area (double p1,double p2){
    return (p2-p1)*f(p2);
}
```

Usamos uma variável global para ser um char e definir a função escolhida pelo usuário, criamos o “escolhedor de funções” dessa forma descrita pela imagem:

```c
double f(double x)
{
    switch (letter)
    {
    case 'a':
        return 1+x;
        break;
    case 'b':
        if(x<-1 ||x>1)
        {
            printf("A fora do dominio permitido \n");
            exit(-1);
        }
        return sqrt(1-pow(x,2));
        break;
    case 'c':
        return sqrt(1+pow(x,4));
        break;
    case 'd':
        return sin(pow(x,2));
        break;
    case 'e':
        return cos(exp(-x));
        break;
    case 'f':
        return cos(exp(-x))*x;
        break;
    case 'g':
        return cos(exp(-x))*(0.005*pow(x,3)+1);
        break;


    }

    return pow(x,3);
}
```

A função principal só usa essas funções e faz as restrições para o usuário utilizar o programa, com avisos de erros. Os casos de testes

serão demonstrados a seguir, usamos os intervalos de 5 a 1000, um erro de 10 elevado a -5 para os testes:

```
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out a 5 1000 0.00001
O valor da integral 500983.444161
Tempo total de integracao 79 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out b 5 1000 0.00001
A fora do dominio permitido

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out b -0.6 0.7 0.00001
O valor da integral 1.199294
Tempo total de integracao 0 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out c 5 1000 0.00001
O valor da integral 333333325.686849
Tempo total de integracao 9525 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out d 5 1000 0.00001
O valor da integral -0.951950
Tempo total de integracao 8402 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out e 5 1000 0.00001
O valor da integral 995.000000
Tempo total de integracao 0 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out f 5 1000 0.00001
O valor da integral 499988.444099
Tempo total de integracao 548 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out g 5 1000 0.00001
O valor da integral 1250001063.909807
Tempo total de integracao 51744 milisegundos
```

Percebemos que as funções c, d e g demoraram bastante, queremos melhorar o desempenho com a versão concorrente.

2) Concorrente

Para implementar a função de forma concorrente, usamos as mesmas bibliotecas, mas pensamos em usar uma pilha para balancear o trabalho entre as threads. Usamos duas estruturas, uma que armazena os valores da integração, e outra que define a pilha:

```c
typedef  struct
{
    double a,b,err;
}integradata ;

struct Stack {
    int top;
    unsigned capacity;
    integradata* array;
};
```

As funções da pilha vem em seguida:

```c
struct Stack* createStack(unsigned capacity)
{
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (integradata*)malloc(stack->capacity * sizeof(integradata));
    return stack;
}

int isFull(struct Stack* stack)
{
    return stack->top == stack->capacity - 1;
}

int isEmpty(struct Stack* stack)
{
    return stack->top == -1;
}

void push(struct Stack* stack, integradata item)
{
    if (isFull(stack))
        {
            printf("Estourou a pilha \n");exit(-1);
        }
    stack->array[++stack->top] = item;
}
```

Para dividir os trabalhos, é necessário um compartilhamento de acesso a dados, para isso, usamos essas variáveis globais, onde N é uma constante definida no início do código como 5000:

```
integradata Buffer[N];
pthread_mutex_t mutex;
double integralTotal=0;
struct Stack *pilha;
```

Dividimos a função de integrar duas partes, a primeira é análoga a sequencial, só se diferencia porque utiliza dados da estrutura “integradata”. A segunda funciona com acesso a pilha e com restrições do uso das variáveis globais, só mostraremos a segunda porque a lógica da primeira parte já foi demonstrada.

```
void* Integral(void* data)
{
    integradata* integraldt = ( integradata*) data;

    pthread_mutex_lock(&mutex);
    push(pilha,*integraldt);
    pthread_mutex_unlock(&mutex);

   while(1){
       pthread_mutex_lock(&mutex);
       if(peek(pilha)!=NULL){
        integradata dt = *pop(pilha);
       // printf("unpopou &lf\n",dt.a);
        pthread_mutex_unlock(&mutex);
        Integra(dt.a,dt.b,dt.err);
        }else{
            pthread_mutex_unlock(&mutex);
            break;
        }
    }
   pthread_exit(NULL);

}
```

Dividimos em blocos o trabalho das threads, que varia de acordo com quantas threads o usuário desejar. A função a seguir cria as threads de acordo com a estrutura integradata e vai somando o trabalho delas a uma variável.

```
double IntegraConcorrente(double a, double b, double err, int nthreads)
{
    pthread_t threads[nthreads];
    pthread_mutex_init(&mutex,NULL);

    double blocksize = (b-a)/nthreads;

    for(int i=0;i<nthreads;i++)
    {
        integradata* data= (integradata*) malloc(sizeof(integradata));
        data->a=a+(i*blocksize);
        data->b=a+((i+1)*blocksize);
        data->err=err;
        if (pthread_create(&threads[i], NULL, Integral,(void*) data))
        {
            printf("--ERRO: pthread_create()\n"); exit(-1);
        }

    }
    for(int i=0;i<nthreads;i++)
    {
        if (pthread_join(threads[i],NULL)) {
            printf("--ERRO: pthread_join()\n"); exit(-1);
        }
    }
    return integralTotal;

}
```

A main e a função que define as funções escolhidas pelo usuário também são análogas ao caso sequencial, então, vamos para os casos de testes, onde usamos diferentes números de threads, mas mantendo os parâmetros que usamos na versão sequencial.

Função a:

```
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out a 5 1000 0.00001 1
O valor da integral 500983.444161
Levando tempo de 117 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out a 5 1000 0.00001 2
O valor da integral 500983.444161
Levando tempo de 94 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out a 5 1000 0.00001 3
O valor da integral 500983.758882
Levando tempo de 202 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out a 5 1000 0.00001 4
O valor da integral 500983.444161
Levando tempo de 313 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out a 5 1000 0.00001 5
O valor da integral 500984.010658
Levando tempo de 232 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out a 5 1000 0.00001 6
O valor da integral 500983.758882
Levando tempo de 287 milisegundos
```

No geral, o código sequencial funcionou melhor para essa função, já que seu tempo de trabalho para “a” é menor do que qualquer execução do algoritmo concorrente, mesmo testando com várias threads.

Função b:

```
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out b -0.6 0.7 0.00001 6
O valor da integral 1.199291
Levando tempo de 3 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out b -0.6 0.7 0.00001 6
O valor da integral 1.199291
Levando tempo de 0 milisegundos
```

Em duas execuções usando o mesmo intervalo que foi usado na versão linear, percebemos que o tempo é bem parecido com o caso sequencial.

Função c:

Chegamos em uma das funções em que o tempo na versão sequencial foi bem demorado, vamos ver se melhora.

```
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out c 5 1000 0.00001 1
O valor da integral 333333325.686808
Levando tempo de 8691 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out c 5 1000 0.00001 2
O valor da integral 333333325.686860
Levando tempo de 6469 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out c 5 1000 0.00001 3
O valor da integral 333333325.969566
Levando tempo de 11317 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out c 5 1000 0.00001 4
O valor da integral 333333325.686802
Levando tempo de 12634 milisegundos
```

Percebemos que quando usamos 3 ou mais, o tempo levado é superior, porém o desempenho foi consideravelmente melhor quando usamos duas threads.

O padrão continua se aumentarmos o intervalo, primeiro mostramos o tempo da sequencial, depois do concorrente usando 2 e 3 threads:

```
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out c 5 2000 0.00001
O valor da integral 2666666718.527892
Tempo total de integracao 28169 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out c 5 2000 0.00001 2
O valor da integral 2666666718.526979
Levando tempo de 19342 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out c 5 2000 0.00001 3
O valor da integral 2666666718.699975
Levando tempo de 32819 milisegundos
```

Função d:

```
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out d 5 1000 0.00001 1
O valor da integral -0.951950
Levando tempo de 7932 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out d 5 1000 0.00001 2
O valor da integral -0.951950
Levando tempo de 5754 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out d 5 1000 0.00001 4
O valor da integral -0.951950
Levando tempo de 10092 milisegundos
```

Percebemos, de novo, que com duas threads o tempo melhora em comparação ao sequencial, vamos testar aumentando o intervalo:

```
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out d 5 2000 0.00001
O valor da integral 0.634980
Tempo total de integracao 23203 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out d 5 2000 0.00001 2
O valor da integral 0.634980
Levando tempo de 15527 milisegundos
```

Aqui confirmamos a melhora com o uso de 2 threads.

Função e:

Essa função já é bem otimizada, então é difícil obter resultados melhores que na versão sequencial.

```
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out e 5 1000 0.00001
O valor da integral 995.000000
Tempo total de integracao 0 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out e 5 1000 0.00001 1
O valor da integral 995.000000
Levando tempo de 0 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out e 5 1000 0.00001 2
O valor da integral 995.000000
Levando tempo de 0 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out e 5 1000 0.00001 3
O valor da integral 995.000000
Levando tempo de 1 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out e 5 1000 0.00001 4
O valor da integral 995.000000
Levando tempo de 1 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out e 5 1000 0.00001 5
O valor da integral 995.000000
Levando tempo de 0 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out e 5 1000 0.00001 6
O valor da integral 995.000000
Levando tempo de 1 milisegundos
```

Aqui isso fica bem claro.

Função f:

```
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out f 5 1000 0.00001
O valor da integral 499988.444099
Tempo total de integracao 549 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out f 5 1000 0.00001 2
O valor da integral 499988.444098
Levando tempo de 343 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out f 5 1000 0.00001 3
O valor da integral 499988.758820
Levando tempo de 325 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out f 5 1000 0.00001 4
O valor da integral 499988.444098
Levando tempo de 514 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out f 5 1000 0.00001 5
O valor da integral 499989.010596
Levando tempo de 342 milisegundos
```

Com 2 ou mais threads, o tempo melhorou bastante, o pior caso foi com 4 threads, que ainda assim o tempo ficou melhor.

Função g:

Essa função demorou muito no sequencial, então vai ser muito fácil de ver a melhora com as threads:

```
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>SeqInteg.out g 5 1000 0.00001
O valor da integral 1250001063.909807
Tempo total de integracao 51539 milisegundos

C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out g 5 1000 0.00001 2
O valor da integral 1250001063.909619
Levando tempo de 23845 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out g 5 1000 0.00001 3
O valor da integral 1250001061.991454
Levando tempo de 21790 milisegundos
C:\Users\celoc\facul\trabs\Comp Conc\TRAB>ConcIntegBal.out g 5 1000 0.00001 4
O valor da integral 1250001063.910055
Levando tempo de 27494 milisegundos
```

Aqui fica muito claro que a versão concorrente melhora o tempo de execução, a versão sequencial demora 51 segundos, enquanto o pior caso das execuções concorrentes demora apenas 27 segundos, que representa um ganho muito absurdo.

3) Conclusão

Após a explicação do método utilizado, implementado de duas maneiras diferentes, explicitadas anteriormente, fica muito claro que a versão concorrente de integração numérica é bem mais otimizada que a sequencial, a diminuição do tempo é considerável e essa conclusão era o nosso objetivo.