Trabalho de Computação Concorrente

UFRJ

Rodrigo da Costa Passos – 115196299

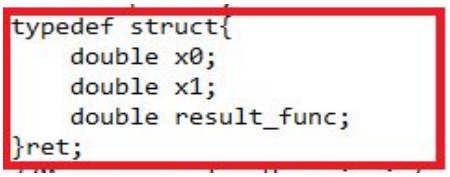
Yago Alves da Costa – 115212477

Lógica do Algoritmo

A. Sequencial

A ideia do Algoritmo Sequencial é que o usuário ao executar o código, ele irá informar o valor do intervalo inicial e final e o valor do erro que ele quer que a função seja calculada. Em seguida, o usuário também vai informar qual será a função que ele deseja que o código calcule a integral com aqueles intervalos e erro informados anteriormente. E o programa irá retornar o valor da integral naquele intervalo.

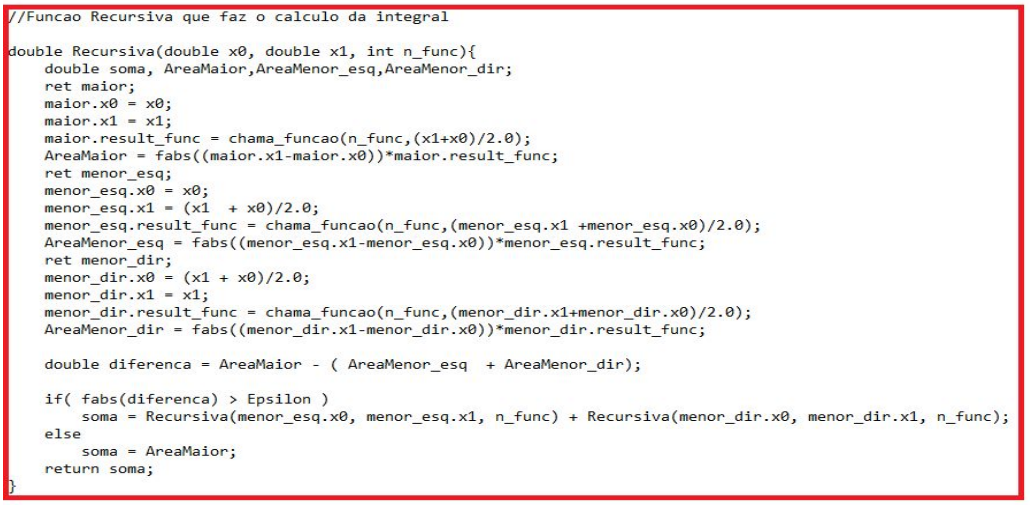
Para calcular a integral de forma mais simples e prática, utilizamos uma struct (conforme a Figura 1) que vai armazenar as informações de um retângulo que será formado à medida que a função recursiva implementada seja chamada.



```
typedef struct{
    double x0;
    double x1;
    double result_func;
}ret;
```

Figura 1 – Struct ret

Durante a execução da função Recursiva (ilustrada na Figura 2 abaixo), utilizamos essa "struct" apresentada com três variáveis, um x0 (intervalo inicial), um x1 (intervalo final) e um "result_func" que é usado para calcular o resultado da função naquele ponto que será passado. Criamos três variáveis do tipo dessa "struct", cada uma delas calcula o valor da área de cada um dos três retângulos que serão formados a cada chamada da função recursiva, um retângulo menor que vai do intervalo inicial até o ponto médio, um que vai do ponto médio até o intervalo final e o outro que vai do intervalo inicial até o intervalo final.



```
//Funcao Recursiva que faz o calculo da integral

double Recursiva(double x0, double x1, int n_func){
    double soma, AreaMaior,AreaMenor_esq,AreaMenor_dir;
    ret maior;
    maior.x0 = x0;
    maior.x1 = x1;
    maior.result_func = chama_funcao(n_func,(x1+x0)/2.0);
    AreaMaior = fabs((maior.x1-maior.x0))*maior.result_func;
    ret menor_esq;
    menor_esq.x0 = x0;
    menor_esq.x1 = (x1  + x0)/2.0;
    menor_esq.result_func = chama_funcao(n_func,(menor_esq.x1 +menor_esq.x0)/2.0);
    AreaMenor_esq = fabs((menor_esq.x1-menor_esq.x0))*menor_esq.result_func;
    ret menor_dir;
    menor_dir.x0 = (x1 + x0)/2.0;
    menor_dir.x1 = x1;
    menor_dir.result_func = chama_funcao(n_func,(menor_dir.x1+menor_dir.x0)/2.0);
    AreaMenor_dir = fabs((menor_dir.x1-menor_dir.x0))*menor_dir.result_func;

    double diferenca = AreaMaior - ( AreaMenor_esq  + AreaMenor_dir);

    if( fabs(diferenca) > Epsilon )
        soma = Recursiva(menor_esq.x0, menor_esq.x1, n_func) + Recursiva(menor_dir.x0, menor_dir.x1, n_func);
    else
        soma = AreaMaior;
    return soma;
}
```

Figura 2 – Função Recursiva

A ideia é que os três retângulos informados no parágrafo anterior funcionaram da mesma forma que os retângulos estão funcionando nesse link a seguir: https://pt.wikipedia.org/wiki/Ficheiro:Riemann_sum_(middlebox).gif . E a cada vez que a diferença do retângulo maior pelos dois menores for maior do que o erro, novos retângulos estarão sendo adicionados ao valor dessa integral. A função recursiva irá parar de executar quando essa diferença for menor ou igual ao erro e dessa forma, ela retornará o valor da integral que será a soma da área de cada um desses retângulos encontrados no decorrer da execução, será uma aproximação.

Ao final da execução do programa, a função main (ilustrada na Figura 3) irá imprimir na tela o valor da variável global "soma" que será o valor da integral calculado de forma recursiva.

```
int main( int argc, char *argv[] ){
    //inicializações
    double intervalo_inicial, intervalo_final;
    int numero_funcao;
    double resposta, inicio, fim, delta1;

    if(argc < 3) {
        printf("Use: %s <intervalo inicial> <intervalo final> <Erro>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    Epsilon = atof(argv[3]);
    intervalo_inicial = atof(argv[1]);
    intervalo_final = atof(argv[2]);

    /*printf("Entre o invervalo inicial: ");
    scanf("%lf", &intervalo_inicial);
    printf("Entre o invervalo final: ");
    scanf("%lf", &intervalo_final);
    printf("Entre o valor de Epsilon: ");
    scanf("%lf", &Epsilon);*/

    numero_funcao = menu();
    GET_TIME(inicio);
    //Chamada da Função Recursiva
    resposta = Recursiva(intervalo_inicial, intervalo_final, numero_funcao);
    GET_TIME(fim);
    delta1=fim-inicio;
    printf("O valor da Integral é : %lf.\n", resposta);
    printf("O Tempo total durante todo o processo eh de: %lf segundos.\n",delta1 );
    return 0;
}
```

Figura 3 – Função Main

B. Concorrente

O programa concorrente está modularizado e implementado de forma diferente do sequencial. A primeira ideia é que o usuário ao executar o programa, terá que passar por linha de comando o número de threads, o intervalo inicial, o intervalo final e o erro (ilustrado na Figura 4 em conjunto com as inicializações da Função Main).

```
//Inicialização
double inicio, fim, delta1, delta2;
GET_TIME(inicio);
//valida e recebe os valores de entrada
if(argc < 4) {
    printf("Use: %s <numero de threads> <intervalo inicial> <intervalo final> <Erro>\n", argv[0]);
    exit(EXIT_FAILURE);}
//converte valores recebidos
nThreads = atoi(argv[1]);
Epsilon = atof(argv[4]);
double intervalo_inicial = atof(argv[2]);
double intervalo_final = atof(argv[3]);
pthread_t tid[nThreads];
//aloca inialmente espaço para o buffer
buffer = malloc(sizeof(ret) * (nThreads*2)); if(!buffer) exit(-1);
tam_buffer = nThreads*2;
//recebe do usuario qual funcao ele quer analizar
int numero_funcao = menu();
int *t;
//divide o dominio da funcao de acordo com o numero de threads
for (int i =0; i<nThreads;i++){
    ret aux;
    if(i==nThreads-1){
        aux.x0=intervalo_final - ((intervalo_final-intervalo_inicial)/nThreads);
        aux.x1=intervalo_final;}
    else{
        aux.x0 = intervalo_inicial + ((intervalo_final-intervalo_inicial)/nThreads)*i;
        aux.x1 = aux.x0 + ((intervalo_final-intervalo_inicial)/nThreads);}
    aux.n_func = numero_funcao;
    buffer[i]=aux;
    tam_ocupado++;}
//inicializa as variaveis de condição e mutex
pthread_mutex_init(&mutex, NULL);
pthread_cond_init(&cond, NULL);
GET_TIME(fim);
delta1 = fim-inicio;
```

Figura 4 – Inicializações da Função Main

Assim como o programa sequencial utilizamos uma função chamada menu, que ajuda o usuário na escolha de qual função ele gostaria que o programa calculasse integral, conforme a figura 5 abaixo:

```
int menu(){
    int numero_funcao;
    printf("Escolha qual das seguintes funções abaixo você gostaria de utilizar:\n");
    printf("Se for a funcao f(x) = 1 + x -> escreva 1 \n ");
    printf("Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2 \n ");
    printf("Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3 \n ");
    printf("Se for a funcao f(x) = sen(x^2) -> escreva 4 \n ");
    printf("Se for a funcao f(x) = cos(e^-x) -> escreva 5 \n ");
    printf("Se for a funcao f(x) = cos(e^-x)*x -> escreva 6 \n ");
    printf("Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7 \n ");
    scanf("%d", &numero_funcao);

    return numero_funcao;

}
```

Figura 5 – Função menu

Uma estratégia também utilizada pelo programa concorrente, é a ideia de dividir o domínio da função de acordo com o número de threads. Isso ocorre logo após as inicializações e garante que inicialmente exista no Buffer um valor igual ao número de threads, de forma que na primeira iteração de cada thread, elas não tenham que esperar uma terminar para outra começar. Essa ideia está ilustrada na Figura 4 acima, dentro do “for” que a figura apresenta.

Agora as threads são criadas e o programa passa a executar a parte concorrente do problema, conforme a Figura 6 abaixo apresenta:

```
    //Parte que vai ser paralelizada

    //cria as nThreads threads
    GET_TIME(inicio);
    for(int i =0;i<nThreads; i++){
        t = malloc(sizeof(int)); if(t==NULL) return -1;
        *t = i;
        if (pthread_create(&tid[i], NULL,concorrente,(void *)t)) {
            printf("--ERRO: pthread_create()\n"); exit(-1); }
    }
    for (int i=0; i<nThreads; i++) {
        if (pthread_join(tid[i], NULL)) {
            printf("--ERRO: pthread_join() \n"); exit(-1);
        }

    }
    GET_TIME(fim);
    delta2 = fim-inicio;

    //Finalização

    //printa o valor da integral de acordo com esse metodo do ponto medio
    printf("O valor da integral da funcao numero %d \nNo intervalo inicial = %lf e no intervalo final = %lf será igual a: %lf\n",
                numero_funcao, intervalo_inicial, intervalo_final, soma);

    printf("O tempo das inicializações foi de: %lf segundos\n", delta1);
    printf("O tempo da parte paralelizada foi de: %lf segundos\n", delta2);
    //destroi todas as variaveis de condição e mutex
    pthread_mutex_destroy(&mutex);
    pthread_cond_destroy(&cond);

    pthread_exit(NULL);

}
```

Figura 6 – Parte Concorrente da Função Main

A parte concorrente possui a chamada de diversas funções que facilitam o programa durante a leitura do código. A primeira é a função *aloca_buffer* (ilustrado na Figura 7 abaixo) que tem como objetivo adicionar mais memória ao Buffer no caso que ele esteja cheio. Nessa função percebemos uma oportunidade de melhoria, que seria implementar de forma que quando o Buffer estiver cheio a função *aloca_buffer* não tenha que copiar cada elemento do Buffer antigo para o novo Buffer com o dobro de tamanho. Algo de relevante sobre o Buffer, é que ele funciona como uma pilha, isso significa que durante a execução ele sempre armazena os elementos novos na primeira posição de memória livre que ele encontra, evitando o caso de o programa indicar que o Buffer está cheio e existir uma posição de memória com o valor NULL.

```
//Funcao chamada quando uma thread percebe que o buffer encheu (tam_ocupado = tam_buffer)
//Tracar exclusao mutua antes de chamar a funcao
void aloca_buffer ( ) {
    int new_tam_buffer = tam_buffer*2;
    ret *new_buffer;

    new_buffer = malloc(sizeof(ret) * new_tam_buffer); if(!new_buffer) exit(-1);

    for (int i = 0; i < tam_buffer; i++) {
        new_buffer[i] = buffer[i];
    }

    tam_buffer = new_tam_buffer;
    buffer = new_buffer;

}
```

Figura 7 – Função Aloca_buffer

Cada a thread pega um elemento do Buffer quando ele não estiver vazio, onde cada elemento adicionado ao Buffer é uma struct do tipo “ret” (semelhante a struct apresentada no programa sequencial, com diferença no último elemento), que contém um elemento x0 (intervalo inicial), um elemento x1 (intervalo final) e um “n_func” que será o número da função que foi informado pelo usuário. Cada thread adiciona no Buffer utilizando a função *adiciona_Buffer* que também utiliza a variável do tipo “ret” (struct criada anteriormente) e está ilustrada na Figura 8 abaixo:

```
void adiciona_Buffer(double x0, double x1, int n_func){

        if(tam_ocupado == tam_buffer){
            aloca_buffer();
        }

        double delta = (x0 + x1)/2;

        ret aux;
        aux.x0= x0;
        aux.x1 = delta;
        aux.n_func = n_func;

        ret aux2;
        aux2.x0= delta;
        aux2.x1 = x1;
        aux2.n_func = n_func;

        buffer[tam_ocupado]= aux;
        tam_ocupado++;
        buffer[tam_ocupado]=aux2;
        tam_ocupado++;

}
```

Figura 8 – Função adiciona_Buffer

Então a ideia é que cada thread fique pegando um elemento do Buffer até que o Buffer esteja vazio e o programa tenha terminado a execução. Então cada thread faz os cálculos das áreas dos três retângulos utilizando a função “calculaAreas”(olhar a Figura 9 abaixo), a partir de um x0 e x1 pertencente ao elemento que eles retiraram do Buffer e utilizando uma nova struct chamada “ajuda”(olhar a figura 10 abaixo) que é utilizada como uma struct auxiliar no cálculo dessas áreas.

```
Areas calculaAreas(double x0, double x1, int n_func){
    double AreaMaior,AreaMenor_esq,AreaMenor_dir;
    ajuda maior;
    Areas area;
    maior.x0 = x0;
    maior.x1 = x1;
    maior.result_func = chama_funcao(n_func,(x1+x0)/2.0);
    AreaMaior = fabs((maior.x1-maior.x0)) * maior.result_func;
    ajuda menor_esq;
    menor_esq.x0 = x0;
    menor_esq.x1 = (x1 -x0)/2.0;
    menor_esq.result_func = chama_funcao(n_func,(menor_esq.x1 + menor_esq.x0)/2.0);
    AreaMenor_esq = fabs((menor_esq.x1-menor_esq.x0)) * menor_esq.result_func;
    ajuda menor_dir;
    menor_dir.x0 = (x1 -x0)/2.0;
    menor_dir.x1 = x1;
    menor_dir.result_func = chama_funcao(n_func,(menor_dir.x1 + menor_dir.x0)/2.0);
    AreaMenor_dir = fabs((menor_dir.x1-menor_dir.x0))*menor_dir.result_func;
    area.menor_esq = AreaMenor_esq;
    area.menor_dir = AreaMenor_dir;
    area.maior =   AreaMaior;
    return area;
}
```

Figura 9 – Função CalculaAreas

```
typedef struct{
    double x0;
    double x1;
    double result_func;
}ajuda;
```

Figura 10 – Struct ajuda

Logo após é verificado se a diferença das áreas do maior retângulo pela dos dois menores é maior que o erro, caso seja maior que o erro ele adiciona os elementos das duas áreas dos retângulos menores no buffer, caso contrário ele adiciona o valor da área maior na variável soma que estará o valor da integral, valor esse que será retornado ao fim da execução do programa e vai representar o valor da integral naqueles intervalos. Essa implementação está feita na função concorrente, conforme mostra abaixo a Figura 11:

```
void *concorrente(void *arg){
    double soma_local, AreaMaior, AreaMenor_esq, AreaMenor_dir;Areas area_return;
    while ( 1 ) {
        //Parte consumidora ( retira elemento do buffer )
        pthread_mutex_lock(&mutex);
        while ( tam_ocupado == 0) {
            count++;
            if ( count == nThreads){ // Finalizacao em cadeia, se for a ultima thread a ficar em espera e nao tiver nada no buffer
                pthread_cond_signal (&cond);
                pthread_mutex_unlock(&mutex);
                pthread_exit(NULL);}
            pthread_cond_wait(&cond, &mutex);
            count--;}
        tam_ocupado = tam_ocupado - 1;
        ret area_local = buffer[tam_ocupado];
        pthread_mutex_unlock(&mutex);
        //Parte em paralelo
        area_return = calculaAreas(area_local.x0, area_local.x1, area_local.n_func);
        AreaMenor_esq = area_return.menor_esq;
        AreaMenor_dir = area_return.menor_dir;
        AreaMaior = area_return.maior;
        double diferenca = AreaMaior - (AreaMenor_esq + AreaMenor_dir);
        if ( fabs(diferenca) > Epsilon ) {
            //Parte produtora ( insere dois elementos no buffer)
            pthread_mutex_lock(&mutex);
            adiciona_Buffer(area_local.x0, area_local.x1, area_local.n_func);
            pthread_cond_signal (&cond);
            pthread_cond_signal (&cond);
            pthread_mutex_unlock(&mutex);
        } else {
            soma_local = AreaMaior;
            //adiciona o valor da soma no valor total;
            pthread_mutex_lock(&mutex);
            soma = soma + soma_local;
            pthread_mutex_unlock(&mutex);}}
    pthread_exit(NULL);}
```

Figura 11 – Função Concorrente

Testes Realizados

Primeiramente calculamos a integral de uma das função no site: pt.symbolab.com/solver/definite-integral-calculator . Depois calculamos com mesmo intervalo em nosso programa sequencial e em seguida no concorrente. Fazendo 5 execuções e pegando o menor tempo com uma 1 thread, 2 threads e 4 threads, no caso do concorrente.

Para apresentação dos testes calculamos as funções 1, 2 e 4 com intervalos distintos para cada função. Realizamos os testes em uma máquina com um processador de 4 núcleos.

Entradas dos valores no código:

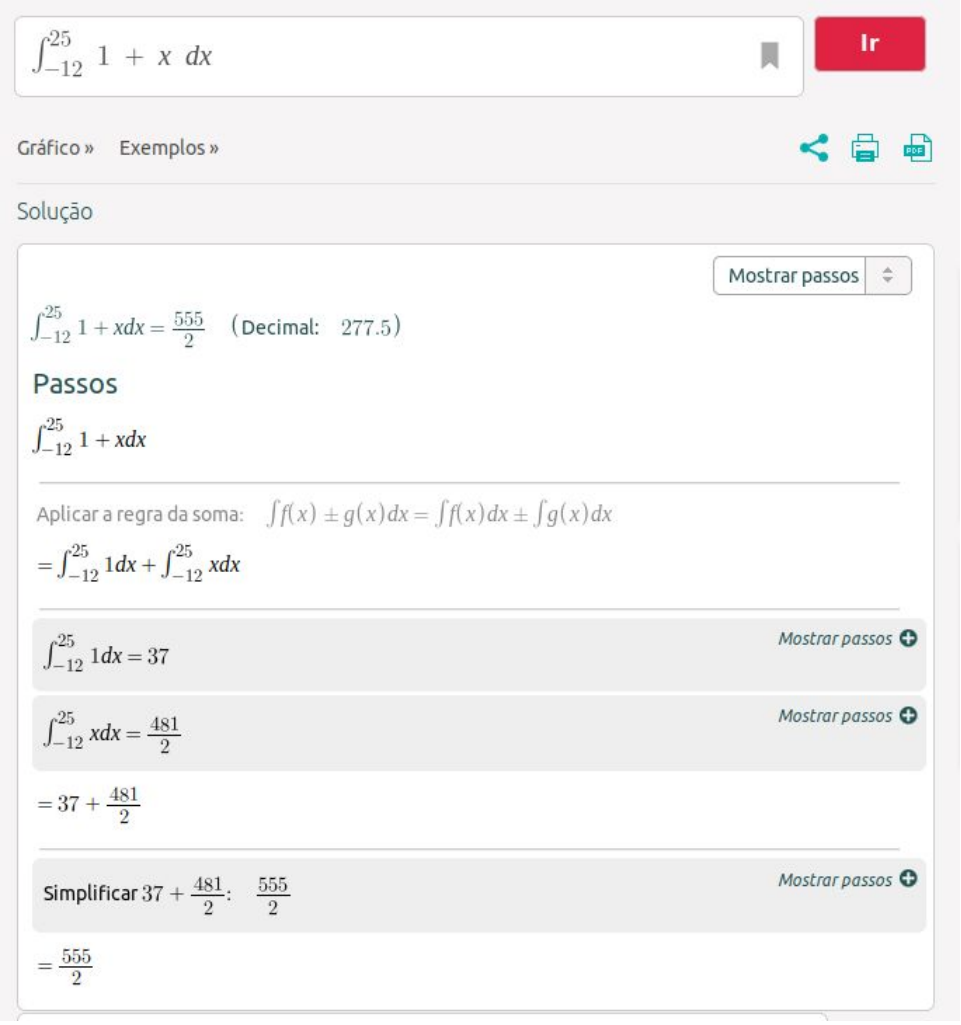
>>Sequencial

```
yago@yagoadc:~/Downloads/Codigos $ ./saida_seq
Use: ./saida_seq <intervalo inicial> <intervalo final> <Erro>
```

>>Concorrente

```
yago@yagoadc:~/Downloads/Codigos $ ./saida
Use: ./saida <numero de threads> <intervalo inicial> <intervalo final> <Erro>
```

Função 1)



>> Sequencial

```
yago@yagoadc:~/Downloads/Codigos $ ./saida_seq -12 25 0.000001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 1
O valor da Integral é : 277.500000.
O Tempo total durante todo o processo eh de: 0.000001 segundos.
```

>> Concorrente

> Com 1 thread

```
yago@yagoadc:~/Downloads/T1 - Definitivo$ ./saida 1 -12 25 0.0001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 1
O valor da integral da funcao numero 1
No intervalo inicial = -12.000000 e no intervalo final = 25.000000 será igual a: 277.500000
O tempo das inicializações foi de: 0.000012 segundos
O tempo da parte paralelizada foi de: 0.000660 segundos
O tempo da finalização foi de: 0.000001 segundos
```

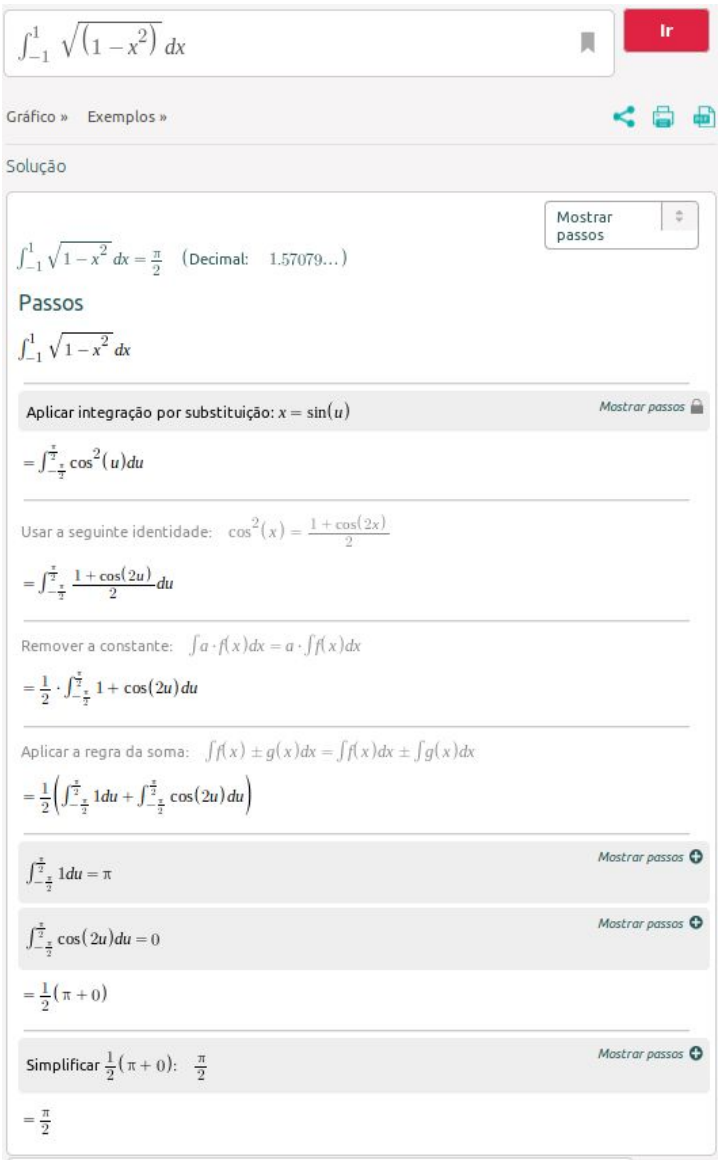
> Com 2 threads

```
yago@yagoadc:~/Downloads/T1 - Definitivo$ ./saida 2 -12 25 0.0001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 1
O valor da integral da funcao numero 1
No intervalo inicial = -12.000000 e no intervalo final = 25.000000 será igual a: 277.500000
O tempo das inicializações foi de: 0.000009 segundos
O tempo da parte paralelizada foi de: 0.000785 segundos
O tempo da finalização foi de: 0.000001 segundos
```

> Com 4 threads

```
yago@yagoadc:~/Downloads/T1 - Definitivo$ ./saida 4 -12 25 0.0001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 1
O valor da integral da funcao numero 1
No intervalo inicial = -12.000000 e no intervalo final = 25.000000 será igual a: 277.500000
O tempo das inicializações foi de: 0.000013 segundos
O tempo da parte paralelizada foi de: 0.001031 segundos
O tempo da finalização foi de: 0.000000 segundos
```

Função 2)



>> Sequencial

```
yago@yagoadc:~/Downloads/Códigos $ ./saida_seq -1 1 0.000001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 2
O valor da Integral é : 1.570865.
O Tempo total durante todo o processo eh de: 0.000131 segundos.
```

>> Concorrente

> Com 1 thread

```
yago@yagoadc:~/Downloads/Códigos $ ./saida 1 -1 1 0.000001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 2
O valor da integral da funcao numero 2
No intervalo inicial = -1.000000 e no intervalo final = 1.000000 será igual a: 1.570865
O tempo das inicializações foi de: 0.000010 segundos
O tempo da parte paralelizada foi de: 0.000791 segundos
O tempo da finalização foi de: 0.000000 segundos
```

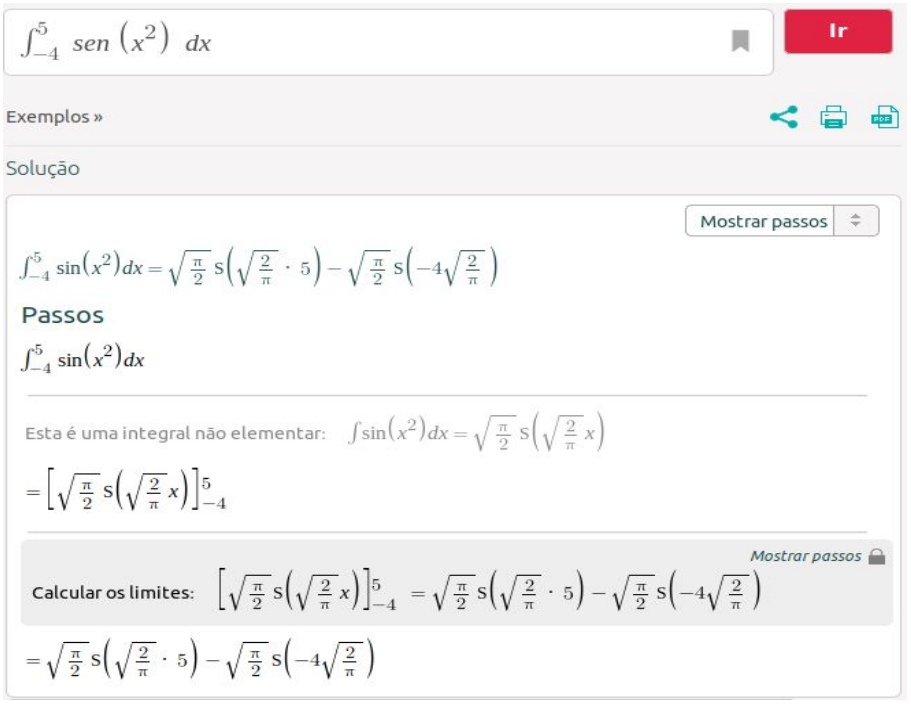
> Com 2 threads

```
yago@yagoadc:~/Downloads/Códigos $ ./saida 2 -1 1 0.000001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 2
O valor da integral da funcao numero 2
No intervalo inicial = -1.000000 e no intervalo final = 1.000000 será igual a: 1.570865
O tempo das inicializações foi de: 0.000021 segundos
O tempo da parte paralelizada foi de: 0.001043 segundos
O tempo da finalização foi de: 0.000001 segundos
```

> Com 4 threads

```
yago@yagoadc:~/Downloads/Códigos $ ./saida 4 -1 1 0.000001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 2
O valor da integral da funcao numero 2
No intervalo inicial = -1.000000 e no intervalo final = 1.000000 será igual a: 1.570865
O tempo das inicializações foi de: 0.000021 segundos
O tempo da parte paralelizada foi de: 0.001371 segundos
O tempo da finalização foi de: 0.000002 segundos
```

Função 4)



>> Sequencial

```
yago@yagoadc:~/Downloads/Códigos $ ./saida_seq -4 5 0.0000014
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 4
O valor da Integral é : 1.275107.
O Tempo total durante todo o processo eh de: 0.001007 segundos.
```

>> Concorrente

> Com 1 thread

```
yago@yagoadc:~/Downloads/Códigos $ ./saida 1 -4 5 0.000001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 4
O valor da integral da funcao numero 4
No intervalo inicial = -4.000000 e no intervalo final = 5.000000 será igual a: 1.275034
O tempo das inicializações foi de: 0.000010 segundos
O tempo da parte paralelizada foi de: 0.001992 segundos
O tempo da finalização foi de: 0.000002 segundos
```

> Com 2 threads

```
yago@yagoadc:~/Downloads/Códigos $ ./saida 2 -4 5 0.000001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 4
O valor da integral da funcao numero 4
No intervalo inicial = -4.000000 e no intervalo final = 5.000000 será igual a: 1.275034
O tempo das inicializações foi de: 0.000010 segundos
O tempo da parte paralelizada foi de: 0.003723 segundos
O tempo da finalização foi de: 0.000002 segundos
```

> Com 4 threads

```
yago@yagoadc:~/Downloads/Códigos $ ./saida 4 -4 5 0.000001
Escolha qual das seguintes funções abaixo você gostaria de utilizar:
Se for a funcao f(x) = 1 + x -> escreva 1
 Se for a funcao f(x) = raiz(1 - x^2) -> escreva 2
 Se for a funcao f(x) = raiz(1 + x^4) -> escreva 3
 Se for a funcao f(x) = sen(x^2) -> escreva 4
 Se for a funcao f(x) = cos(e^-x) -> escreva 5
 Se for a funcao f(x) = cos(e^-x)*x -> escreva 6
 Se for a funcao f(x) = cos(e^-x)*(0,005*x^3 + 1) -> escreva 7
 4
O valor da integral da funcao numero 4
No intervalo inicial = -4.000000 e no intervalo final = 5.000000 será igual a: 1.275034
O tempo das inicializações foi de: 0.000014 segundos
O tempo da parte paralelizada foi de: 0.003753 segundos
O tempo da finalização foi de: 0.000001 segundos
```

Avaliação de Desempenho

Após os testes realizados, percebemos que o tempo de execução do código sequencial foi menor do que o tempo de execução do código concorrente em todas as funções, então não tivemos um ganho na versão concorrente mesmo executando com uma thread. E o que deve ter ocasionado esse tempo maior, foi a necessidade de se aplicar a técnica de exclusão mútua quando as threads precisam acessar um recurso compartilhado (seção crítica), que nesse caso é o *buffer* e a variável *tam_ocupado* (que serve como um stack pointer do *buffer*), principalmente quando uma thread for adicionar no buffer as áreas para serem calculadas posteriormente e nesse momento ainda precisar alocar mais espaço no *buffe*r, fazendo com que as outras threads fiquem esperando por um tempo considerável quando precisarem acessar passar por alguma seção crítica.

Outra possível melhoria para tentar diminuir o tempo de execução da parte paralela no programa que poderia ser aplicada na função que calcula as áreas, seria a adaptação da struct que ela retorna, fazendo ela armazenar em alguma variável os cálculos das áreas dos retângulos que vão ser adicionadas no buffer . Se pensarmos que para calcular uma função e em algum ponto e se gastou um tempo considerável nesse cálculo, seria interessante guardar o resultado desse cálculo para as outras threads não precisarem calcular novamente, visando assim um ganho no tempo de execução.