

# Relatório do Capítulo 04 de Introdução à Programação Paralela

Lucas Sousa de Oliveira (10/59491)

9 de setembro de 2013

## 1 Título do Capítulo

*An Application: Numerical Integration*

## 2 Objetivo

Familiarizar o aluno com a biblioteca MPI através da utilização de métodos numéricos de integração para calcular a integral definida de uma função arbitrária.

## 3 Resumo

Para o desenvolvimento deste capítulo, nenhum novo comando precisou ser apresentado. Ainda assim, a metodologia de típica de escrita de um programa paralelo foi usada. Esta metodologia inclui:

1. Construir um programa serial para resolver um problema. Neste caso o problema estudado foi a regra trapezoidal para a estimação de uma integral definida.
2. De forma a paralelizar o algoritmo serial, simplesmente particiona-se o conjunto de dados entre os processadores. No caso apresentado, cada processo integrou a função sobre parte do intervalo desejado.
3. Os cálculos locais produzidos pelos processos individuais são combinados para produzir o resultado final. Aqui, cada processo enviou os resultados da sua integração para o processo 0, que somou tudo e imprimiu o resultado.

Note que separar as variáveis pelo seu escopo, i.e. globais, aquelas com importância para todos os processos, e locais, aquelas com importância apenas para cada processo individualmente, é uma prática importante. Outro ponto que deve ser considerado fortemente é o comportamento imprevisível de funções de entrada e saída. Para programas paralelos, estas funções podem ou não interagir com o terminal de qualquer máquina que estiver executando os processos, potencialmente gerando bloqueios. Uma solução para tal problema é construir o programa de tal forma que apenas o processo 0 possa executar uma função de E/S.

## 4 Exercícios

- 4.1 Escreva a primeira versão do programa de cálculo paralelizado da regra trapezoidal. Defina  $f(x)$  como uma função cuja integral pode ser facilmente calculada a mão, e.g.  $f(x) = x^2$ . Compile e rode o programa para diferentes números de processos. O que acontece se o programa for rodado em apenas um processo?

A primeira versão do programa é a apresentada abaixo.

```
1 #include <stdio.h>
2 #include "mpi.h"
3
4 int main(int argc, char** argv) {
5     int my_rank;
6     int p;
7     float a = 0.0;
8     float b = 1.0;
9     int n = 1024;
10    float h;
11    float local_a;
12    float local_b;
13    int local_n;
14    float integral;
15    float total;
16    int source;
17    int dest = 0;
18    int tag = 0;
19    MPI_Status status;
20
21    float Trap(float local_a, float local_b, int local_n, float h);
22
23    MPI_Init(&argc, &argv);
24    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
25    MPI_Comm_size(MPI_COMM_WORLD, &p);
26
27    h = (b-a)/n;
28    local_n = n/p;
29
30    local_a = a + my_rank*local_n*h;
31    local_b = local_a + local_n*h;
32    integral = Trap(local_a, local_b, local_n, h);
33
34    if (my_rank == 0) {
35        total = integral;
36        for (source = 1; source < p; source++) {
37            MPI_Recv(&integral, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &
38                status);
39            total = total + integral;
40        }
41    } else {
42        MPI_Send(&integral, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
43    }
44
45    if (my_rank == 0) {
46        printf("With n = %d trapezoids, our estimate\n", n);
47        printf("of the integral from %f to %f = %f\n", a, b, total);
48    }
49
50    MPI_Finalize();
```

```

50 }
51
52 float Trap(float local_a, float local_b, int local_n, float h) {
53     float integral;
54     float x;
55     int i;
56
57     float f(float x);
58
59     integral = (f(local_a) + f(local_b))/2.0;
60     x = local_a;
61     for (i = 1; i <= local_n-1; i++) {
62         x = x + h;
63         integral = integral + f(x);
64     }
65     integral = integral*h;
66     return integral;
67 }
68
69 float f(float x) {
70     float return_val;
71     return_val = x*x;
72     return return_val;
73 }

```

Note na figura 1 que quando o programa é executado com um número de processos igual a  $2^n$ , sua integral é exata. Isto acontece devido ao cálculo simplificado da divisão de dados. A única diferença notável quando o programa é rodado com apenas um processo é a velocidade de computação mais elevada, uma vez que o *overhead* de comunicação foi eliminado.

```

lucas@crunchbang:~/documents/ipp/exercises/c04/ex1$ mpirun -np 10 ./trap
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.329442
lucas@crunchbang:~/documents/ipp/exercises/c04/ex1$ mpirun -np 1 ./trap
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333333
lucas@crunchbang:~/documents/ipp/exercises/c04/ex1$ mpirun -np 2 ./trap
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333333
lucas@crunchbang:~/documents/ipp/exercises/c04/ex1$ mpirun -np 4 ./trap
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333333
lucas@crunchbang:~/documents/ipp/exercises/c04/ex1$ mpirun -np 8 ./trap
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333333
lucas@crunchbang:~/documents/ipp/exercises/c04/ex1$ mpirun -np 16 ./trap
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333333
lucas@crunchbang:~/documents/ipp/exercises/c04/ex1$ mpirun -np 128 ./trap
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333334
lucas@crunchbang:~/documents/ipp/exercises/c04/ex1$ mpirun -np 100 ./trap
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.310441

```

Figura 1: Resultado do teste para diversas quantidades de processos.

#### 4.2 Modifique o programa anterior de forma que a, b e c sejam lidos e distribuídos pelo processo 0 - use a função Get\_data. Onde a função Get\_data deve ser colocada? Seriam necessárias outras modificações para permitir que o programa funcione, além das relacionadas com Get\_data?

Podemos modificar o código (apresentado no final da seção 3) da forma solicitada como está apresentado abaixo.

```
1  #include <stdio.h>
2  #include "mpi.h"
3
4  main(int argc, char** argv) {
5      int          my_rank;
6      int          p;
7      float        a;
8      float        b;
9      int          n;
10     float        h;
11     float        local_a;
12     float        local_b;
13     int          local_n;
14     float        integral;
15     float        total;
16     int          source;
17     int          dest = 0;
18     int          tag = 0;
19     MPI_Status    status;
20
21     void Get_data(float* a_ptr, float* b_ptr, int* n_ptr, int my_rank, int p
22                 );
23     float Trap(float local_a, float local_b, int local_n, float h);
24
25     MPI_Init(&argc, &argv);
26     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
27     MPI_Comm_size(MPI_COMM_WORLD, &p);
28
29     Get_data(&a, &b, &n, my_rank, p);
30
31     h = (b-a)/n;
32     local_n = n/p;
33
34     local_a = a + my_rank*local_n*h;
35     local_b = local_a + local_n*h;
36     integral = Trap(local_a, local_b, local_n, h);
37
38     if (my_rank == 0) {
39         total = integral;
40         for (source = 1; source < p; source++) {
41             MPI_Recv(&integral, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &
42                     status);
43             total = total + integral;
44         }
45     } else {
46         MPI_Send(&integral, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
47     }
48
49     if (my_rank == 0) {
50         printf("With n = %d trapezoids, our estimate\n", n);
51         printf("of the integral from %f to %f = %f\n", a, b, total);
52     }
```

```

51
52     MPI_Finalize();
53 }
54
55 void Get_data(float* a_ptr, float* b_ptr, int* n_ptr, int my_rank, int p) {
56
57     int source = 0;
58     int dest;
59     int tag;
60     MPI_Status status;
61
62     if (my_rank == 0){
63         printf("Enter a, b, and n\n");
64         scanf("%f %f %d", a_ptr, b_ptr, n_ptr);
65         for (dest = 1; dest < p; dest++){
66             tag = 0;
67             MPI_Send(a_ptr, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
68             tag = 1;
69             MPI_Send(b_ptr, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
70             tag = 2;
71             MPI_Send(n_ptr, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
72         }
73     } else {
74         tag = 0;
75         MPI_Recv(a_ptr, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &status);
76         tag = 1;
77         MPI_Recv(b_ptr, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &status);
78         tag = 2;
79         MPI_Recv(n_ptr, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
80     }
81 }
82
83 float Trap(float local_a, float local_b, int local_n, float h) {
84     float integral;
85     float x;
86     int i;
87
88     float f(float x);
89
90     integral = (f(local_a) + f(local_b))/2.0;
91     x = local_a;
92     for (i = 1; i <= local_n-1; i++) {
93         x = x + h;
94         integral = integral + f(x);
95     }
96     integral = integral*h;
97     return integral;
98 }
99
100 float f(float x) {
101     float return_val;
102     return_val = x*x;
103     return return_val;
104 }

```

Note que nenhuma outra modificação foi necessária, apesar de se ter utilizado a função *Trap* auxiliar para isolar o *loop* de cálculo.

```

lucas@crunchbang:~/documents/ipp/exercises/c04/ex2$ mpirun -np 1 ./get_data
Enter a, b, and n
0 1 1024
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333333
lucas@crunchbang:~/documents/ipp/exercises/c04/ex2$ mpirun -np 8 ./get_data
Enter a, b, and n
0 1 1024
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333333
lucas@crunchbang:~/documents/ipp/exercises/c04/ex2$ mpirun -np 10 ./get_data
Enter a, b, and n
0 1 1000
With n = 1000 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333333
lucas@crunchbang:~/documents/ipp/exercises/c04/ex2$ mpirun -np 40 ./get_data
Enter a, b, and n
0 1 2000
With n = 2000 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333334

```

Figura 2: Resultado ao se usar diversos números de processos e intervalos.

## 5 Trabalho de programação

### 5.1 Modifique o programa paralelo com a regra trapezoidal de forma que diversas funções possam ser escolhidas e passadas para a função *Trap*. Apresente ao usuário uma menu de funções possíveis.

O código com as devidas modificações está mostrado abaixo. A saída do programa está na figura 3.

```

1  #include <stdio.h>
2  #include "mpi.h"
3
4  main(int argc, char** argv) {
5      int          my_rank;
6      int          p;
7      float        a;
8      float        b;
9      int          n;
10     int          f;
11     float        h;
12     float        local_a;
13     float        local_b;
14     int          local_n;
15
16     float        (*local_f)(float);
17     float        integral;
18     float        total;
19     int          source;
20     int          dest = 0;
21     int          tag = 0;
22     MPI_Status   status;
23
24     void Get_data(float* a_ptr, float* b_ptr, int* n_ptr, int* f_ptr, int
        my_rank, int p);
25     float Trap(float local_a, float local_b, int local_n, float (*f)(float),
        float h);
26
27

```

```

28     MPI_Init(&argc, &argv);
29     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
30     MPI_Comm_size(MPI_COMM_WORLD, &p);
31
32     Get_data(&a, &b, &n, &f, my_rank, p);
33
34     float k(float);
35     float l(float);
36     float m(float);
37
38     switch(f){
39         case 0: local_f = k;break;
40         case 1: local_f = l;break;
41         case 2:
42             default: local_f = m;break;
43     }
44
45     h = (b-a)/n;
46     local_n = n/p;
47
48
49     local_a = a + my_rank*local_n*h;
50     local_b = local_a + local_n*h;
51     integral = Trap(local_a, local_b, local_n,local_f, h);
52
53     if (my_rank == 0) {
54         total = integral;
55         for (source = 1; source < p; source++) {
56             MPI_Recv(&integral, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &
57                 status);
58             total = total + integral;
59         }
60     } else {
61         MPI_Send(&integral, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
62     }
63
64     if (my_rank == 0) {
65         printf("With n = %d trapezoids, our estimate\n", n);
66         printf("of the integral from %f to %f = %f\n", a, b, total);
67     }
68
69     MPI_Finalize();
70 }
71 void Get_data(float* a_ptr, float* b_ptr, int* n_ptr,int* f_ptr, int my_rank
72     , int p) {
73     int source = 0;
74     int dest;
75     int tag;
76     MPI_Status status;
77
78     if (my_rank == 0){
79         printf("Functions (f):\n0. x**2\n1. x**2+1\n2. x**2+2\n")
80         printf("Enter a, b, n, and f\n");
81         scanf("%f %f %d %d", a_ptr, b_ptr, n_ptr, f_ptr);
82         for (dest = 1; dest < p; dest++){
83             tag = 0;
84             MPI_Send(a_ptr, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
85             tag = 1;
86             MPI_Send(b_ptr, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);

```

```

86         tag = 2;
87         MPI_Send(n_ptr, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
88         tag = 3;
89         MPI_Send(f_ptr, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
90     }
91 } else {
92     tag = 0;
93     MPI_Recv(a_ptr, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &status);
94     tag = 1;
95     MPI_Recv(b_ptr, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &status);
96     tag = 2;
97     MPI_Recv(n_ptr, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
98     tag = 3;
99     MPI_Recv(f_ptr, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
100 }
101 }
102 float Trap(float local_a, float local_b, int local_n, float (*local_f)(float
    ), float h) {
103     float integral;
104     float x;
105     int i;
106
107     integral = ((*local_f)(local_a) + (*local_f)(local_b))/2.0;
108     x = local_a;
109     for (i = 1; i <= local_n-1; i++) {
110         x = x + h;
111         integral = integral + (*local_f)(x);
112     }
113     integral = integral*h;
114     return integral;
115 }
116 float k(float x) {
117     float return_val;
118
119
120     return_val = x*x;
121     return return_val;
122 }
123 float l(float x) {
124     float return_val;
125     return_val = x*x+1;
126     return return_val;
127 }
128 float m(float x) {
129     float return_val;
130     return_val = x*x+2;
131     return return_val;
132 }

```

Note que cada processo deve primeiro enviar a mensagem para depois esperar, uma vez que se o contrário for feito, todos os processos esperarão indefinidamente uns pelos outros, o que caracteriza um *deadlock*.



```

lucas@crunchbang:~/documents/ipp/exercises/c04/expl$ mpirun -np 8 ./get_data
Functions (f):
0. x**2
1. x**2+1
2. x**2+2
Enter a, b, n, and f
0 1 1024 0
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 0.333333
lucas@crunchbang:~/documents/ipp/exercises/c04/expl$ mpirun -np 8 ./get_data
Functions (f):
0. x**2
1. x**2+1
2. x**2+2
Enter a, b, n, and f
0 1 1024 1
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 1.333333
lucas@crunchbang:~/documents/ipp/exercises/c04/expl$ mpirun -np 8 ./get_data
Functions (f):
0. x**2
1. x**2+1
2. x**2+2
Enter a, b, n, and f
0 1 1024 2
With n = 1024 trapezoids, our estimate
of the integral from 0.000000 to 1.000000 = 2.333333

```

Figura 3: Resultado do programa modificado.

- 5.2 Uma alternativa mais precisa que a regra trapezoidal é a regra de Simpson. A idéia básica é de aproximar a curva de  $f(x)$  por arcos de parábolas ao invés de segmentos de linhas. Suponha que  $p < q$  são números reais, e suponha que  $r$  é o ponto médio do segmento  $[p, q]$ . Se nós fizermos  $h = (q - p)/2$ , então a equação da parábola passando pelos pontos  $(p, f(p))$ ,  $(r, f(r))$  e  $(q, f(q))$  é:

$$y = \frac{f(p)}{2h^2}(x - r)(x - q) - \frac{f(r)}{h^2}(x - p)(x - q) + \frac{f(q)}{2h^2}(x - p)(x - r).$$

Se integrarmos esta forma de  $p$  a  $q$ , teremos

$$\frac{h}{3}[f(p) + 4f(r) + f(q)].$$

Assim, se usarmos a mesma notação que usamos na discussão da regra trapezoidal e assumirmos que  $n$ , o número de subintervalos de  $[a, b]$ , é par, podemos aproximar

$$\int_a^b f(x)dx = \frac{h}{3}[f(x_0) + 4f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)].$$

Assumindo que  $n/p$  é par, escreva:

- a. um programa serial que use a regra de Simpson para estimar  $\int_a^b f(x)dx$ .

```

1 | #include <stdio.h>
2 |
3 | main() {

```

```

4     float   integral;
5     float   a, b;
6     int     n;
7     float   h;
8     float   x;
9     int     i;
10
11    float f(float x);
12
13    printf("Enter a, b, and n\n");
14    scanf("%f %f %d", &a, &b, &n);
15
16    h = (b-a)/n;
17    integral = (f(a) + f(b));
18    x = a;
19    for (i = 1; i <= n-1; i++) {
20        x = x + h;
21        integral = integral + (i%2?4:2)*f(x);
22    }
23    integral = integral*h/3.0;
24
25    printf("With n = %d trapezoids, our estimate\n", n);
26    printf("of the integral from %f to %f = %f\n", a, b, integral);
27 }
28
29 float f(float x) {
30     float return_val;
31     return_val = x*x;
32     return return_val;
33 }

```

b. um programa paralelo que use a regra de Simpson para estimar  $\int_a^b f(x)dx$ .

```

1  #include <stdio.h>
2  #include "mpi.h"
3
4  main(int argc, char** argv) {
5      int         my_rank;
6      int         p;
7      float       a;
8      float       b;
9      int         n;
10     int         f;
11     float       h;
12     float       local_a;
13     float       local_b;
14     int         local_n;
15     float       (*local_f)(float);
16     float       integral;
17     float       total;
18     int         source;
19     int         dest = 0;
20     int         tag = 0;
21     MPI_Status  status;
22
23     void Get_data(float* a_ptr, float* b_ptr, int* n_ptr, int* f_ptr, int
        my_rank, int p);
24     float Trap(float local_a, float local_b, int local_n, float (*f)(float),
        float h);
25
26     MPI_Init(&argc, &argv);

```

```

27     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
28     MPI_Comm_size(MPI_COMM_WORLD, &p);
29
30     Get_data(&a, &b, &n, &f, my_rank, p);
31
32     float k(float);
33     float l(float);
34     float m(float);
35
36     switch(f){
37         case 0: local_f = k;break;
38         case 1: local_f = l;break;
39         case 2:
40             default: local_f = m;break;
41     }
42
43     h = (b-a)/n;
44     local_n = n/p;
45
46     local_a = a + my_rank*local_n*h;
47     local_b = local_a + local_n*h;
48     integral = Trap(local_a, local_b, local_n,local_f, h);
49
50     if (my_rank == 0) {
51         total = integral;
52         for (source = 1; source < p; source++) {
53             MPI_Recv(&integral, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &
54                 status);
55             total = total + integral;
56         }
57     } else {
58         MPI_Send(&integral, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
59     }
60
61     if (my_rank == 0) {
62         printf("With n = %d trapezoids, our estimate\n", n);
63         printf("of the integral from %f to %f = %f\n", a, b, total);
64     }
65
66     MPI_Finalize();
67 }
68
69 void Get_data(
70     float* a_ptr,
71     float* b_ptr,
72     int* n_ptr,
73     int* f_ptr,
74     int my_rank,
75     int p) {
76
77     int source = 0;
78     int dest;
79     int tag;
80     MPI_Status status;
81
82     if (my_rank == 0){
83         printf("Enter a, b, n, and f\n");
84         scanf("%f %f %d %d", a_ptr, b_ptr, n_ptr, f_ptr);
85         for (dest = 1; dest < p; dest++){
86             tag = 0;

```

```

86         MPI_Send(a_ptr, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
87         tag = 1;
88         MPI_Send(b_ptr, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
89         tag = 2;
90         MPI_Send(n_ptr, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
91         tag = 3;
92         MPI_Send(f_ptr, 1, MPI_INT, dest, tag, MPI_COMM_WORLD);
93     }
94 } else {
95     tag = 0;
96     MPI_Recv(a_ptr, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &status);
97     tag = 1;
98     MPI_Recv(b_ptr, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &status);
99     tag = 2;
100    MPI_Recv(n_ptr, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
101    tag = 3;
102    MPI_Recv(f_ptr, 1, MPI_INT, source, tag, MPI_COMM_WORLD, &status);
103 }
104 }
105
106 float Trap(
107     float local_a,
108     float local_b,
109     int local_n,
110     float (*local_f)(float),
111     float h
112 ) {
113
114     float integral;
115     float x;
116     int i;
117
118     integral = ((*local_f)(local_a) + (*local_f)(local_b));
119     x = local_a;
120     for (i = 1; i <= local_n-1; i++) {
121         x = x + h;
122         integral = integral + (i%2?4:2)*(*local_f)(x);
123     }
124     integral = integral*h/3.0;
125     return integral;
126 }
127 float k(float x) {
128     float return_val;
129     return_val = x*x;
130     return return_val;
131 }
132 float l(float x) {
133     float return_val;
134     return_val = x*x+1;
135     return return_val;
136 }
137 float m(float x) {
138     float return_val;
139     return_val = x*x+2;
140     return return_val;
141 }

```

## 6 Conclusão

Conclui-se que a biblioteca MPI permite a distribuição de processamento entre diversos computadores, podendo assim aumentar o poder computacional disponível para um programa. As ferramentas apresentadas aqui se mostraram extremamente úteis, apesar de simples, para as tarefas de computação paralela e distribuída.

## 7 Referências

**Pacheco, P.S.,** (1997) *Parallel Programming with MPI*. Morgan Kaufmann.