Relatório do Capítulo 04 de Introdução à Programação Paralela

Lucas Sousa de Oliveira (10/59491)

2 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 númericos 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 metologia 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 processor. 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 pratica 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, &
                   status);
38             total = total + integral;
39         }
40     } else {
41         MPI_Send(&integral, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
42     }
43
44     if (my_rank == 0) {
45         printf("With n = %d trapezoids, our estimate\n", n);
46         printf("of the integral from %f to %f = %f\n", a, b, total);
47     }
48
49     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 modificando a linha indicada 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     float Trap(float local_a, float local_b, int local_n, float h);
23
24     MPI_Init(&argc, &argv);
25     MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
26     MPI_Comm_size(MPI_COMM_WORLD, &p);
27
28     Get_data(&a, &b, &n, my_rank, p);
29
30     h = (b-a)/n;
31     local_n = n/p;
32
33     local_a = a + my_rank*local_n*h;
34     local_b = local_a + local_n*h;
35     integral = Trap(local_a, local_b, local_n, h);
36
37     if (my_rank == 0) {
38         total = integral;
39         for (source = 1; source < p; source++) {
40             MPI_Recv(&integral, 1, MPI_FLOAT, source, tag, MPI_COMM_WORLD, &
                   status);
41             total = total + integral;
42         }
43     } else {
44         MPI_Send(&integral, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
45     }
46
47     if (my_rank == 0) {
48         printf("With n = %d trapezoids, our estimate\n", n);
49         printf("of the integral from %f to %f = %f\n", a, b, total);
50     }
```

```
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, &
                    status);
57              total = total + integral;
58          }
59      } else {
60          MPI_Send(&integral, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
61      }
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
        , int p) {
72      int source = 0;
73      int dest;
74      int tag;
75      MPI_Status status;
76
77      if (my_rank == 0){
78                  printf("Functions (f):\n0. x**2\n1. x**2+1\n2. x**2+2")
79          printf("Enter a, b, n, and f\n");
80          scanf("%f %f %d %d", a_ptr, b_ptr, n_ptr, f_ptr);
81          for (dest = 1; dest < p; dest++){
82              tag = 0;
83              MPI_Send(a_ptr, 1, MPI_FLOAT, dest, tag, MPI_COMM_WORLD);
84              tag = 1;
85              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*.

Figura 3: Resultado do programa modificado.

Quando o programa é rodado em apenas um processador espera-se que as respostas passem a ficar ordenadas, uma vez que os fatores externos que geravam o desordenamento, tais como atrasos conexão e cargas de utilização dos outros processadores, deixam de existir.

6 Conclusão

Conclui-se que a biblioteca MPI possui as ferramentas necessárias para o desenvolvimento de programas paralelos, inclusive com seus processos distribuidos entre várias máquinas. Através do uso desta biblioteca pode-se construir programas segundo o modelo SPMD que trocavam informações de variadas formas.

7 Referências

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