Relatório do Capítulo 06 de Introdução à Programação Paralela

Lucas Sousa de Oliveira (10/59491)

12 de outubro de 2013

1 Título do Capítulo

Grouping Data For Communication

2 Objetivo

Foi mencionado no capítulo 3 que na geração atual de sistemas paralelos, o envio de uma mensagem é uma operação custosa. Uma consequencia natural disto é que, via de regra, quanto menos mensagens são enviadas, melhor o desempenho geral do programa. Entretanto, em cada um dos nossos programas de cálculo da regra trapezoidal, quando distribuímos os dados, separamos "a", "b"e "n"em mensagens diferentes - seja com MPI_Send e MPI_Recv ou MPI_Bcast. Desta forma, podemos melorar a performance do nosso programa enviando vários dados em uma única mensagem. MPI disponibiliza mecanismos para o agrupamento de dados individuais em uma única mensagem: o parametro *count* em diversas rotinas de comunicação, tipos de dados derivados, e MPI_Pack/MPI_Unpack. Examinaremos cada uma dessas opções a seguir.

3 Resumo

Parei em 6.6

4 Exercícios

4.1 Edite o programa da regra trapezoidal de forma que ele use Get_data3.

O programa com as modificações solicitadas encontra-se abaixo. Note que o uso do tipo derivado com variáveis soltas não é muito bom uma vez que este tipo só poderá ser usado para aquela configuração específica de memória. Se uma outra função quiser utilizar este tipo derivado, ela precisará redefinir o vetor de distancias (*displacements*).

```
1  #include <stdio.h>
2  #include "mpi.h"
3
4  /* Constroi o tipo derivado */
5  void Build_derived_type(
6           float*          a_ptr            /* entrada */,
7           float*          b_ptr            /* entrada */,
8           int*            n_ptr            /* entrada */,
9           MPI_Datatype*   mesg_mpi_t_ptr   /* saida   */);
10
```

```
11 /* Recebe os valores de a, b e n e envia para todos os processos */
12 void Get_data3(
13         float*          a_ptr           /* saida   */,
14         float*          b_ptr           /* saida   */,
15         int*            n_ptr           /* saida   */,
16         int             my_rank         /* entrada */);
17
18 /* Calcula o trapezoide local */
19 float Trap(
20         float           local_a         /* entrada */,
21         float           local_b         /* entrada */,
22         int             local_n         /* entrada */,
23         float           h               /* entrada */)/
24
25 int main(int argc, char** argv) {
26     int         r, p, n, local_n, source, dest, tag;
27     float       a, b, h, local_a, local_b, integral, total;
28     MPI_Status  status;
29
30     MPI_Init(&argc, &argv);
31     MPI_Comm_rank(MPI_COMM_WORLD, &r);
32     MPI_Comm_size(MPI_COMM_WORLD, &p);
33
34     Get_data3(&a, &b, &n, r);
35
36     h = (b-a)/n;
37     local_n = n/p;
38
39     local_a = a + r*local_n*h;
40     local_b = local_a + local_n*h;
41     integral = Trap(local_a, local_b, local_n, h);
42
43     MPI_Reduce(&integral, &total, 1, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);
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();
51 }
52
53 void Build_derived_type(float* a_ptr, float* b_ptr, int* n_ptr, MPI_Datatype
       * mesg_mpi_t_ptr) {
54     int block_lengths[3];
55     MPI_Aint displacements[3], start_address, address;
56     MPI_Datatype typelist[3];
57
58     /* Definindo o numero de elementos de cada bloco. */
59     block_lengths[0] = block_lengths[1] = block_lengths[2] = 1;
60
61     /* Definicao dos tipos que serao usados */
62     typelist[0] = MPI_FLOAT;
63     typelist[1] = MPI_FLOAT;
64     typelist[2] = MPI_INT;
65
66     /* Definindo a distancia do primeiro elemento como 0.
67      * Em seguida calcula-se a posicao de 'a' e de todos os
68      * elementos com relacao a 'a'. */
69     displacements[0] = 0;
```

```
70      MPI_Address(a_ptr, &start_address);
71      MPI_Address(b_ptr, &address);
72      displacements[1] = address - start_address;
73      MPI_Address(n_ptr, &address);
74      displacements[2] = address - start_address;
75
76      /* Construcao do tipo derivado. */
77      MPI_Type_struct(3, block_lengths, displacements, typelist,
            mesg_mpi_t_ptr);
78
79      /* Indicando para o sistema que este tipo derivado sera usado */
80      MPI_Type_commit(mesg_mpi_t_ptr);
81  }
82
83  void Get_data3(float* a_ptr, float* b_ptr, int* n_ptr, int my_rank) {
84      MPI_Datatype  mesg_mpi_t;
85
86      if (my_rank == 0){
87          printf("Enter a, b, and n\n");
88          scanf("%f %f %d", a_ptr, b_ptr, n_ptr);
89      }
90
91      Build_derived_type(a_ptr, b_ptr, n_ptr, &mesg_mpi_t);
92      MPI_Bcast(a_ptr, 1, mesg_mpi_t, 0, MPI_COMM_WORLD);
93  }
94
95  float Trap(float local_a, float local_b, int local_n, float h) {
96      float x, integral;
97      int i;
98
99      /* Prototipo da funcao a ser integrada */
100     float f(float x);
101
102     integral = (f(local_a) + f(local_b))/2.0;
103     x = local_a;
104     for (i = 1; i <= local_n-1; i++) {
105         x = x + h;
106         integral = integral + f(x);
107     }
108     integral = integral*h;
109     return integral;
110 }
111
112 float f(float x) {
113     float return_val;
114     return_val = x*x;
115     return return_val;
116 }
```

4.2 Edite o programa da regra trapezoidal de forma que ele use Get_data4.

4.3 Escreva um programa que cria um tipo derivado de dados para representar uma matrix esparça. Um entrada da matriz é uma estrutura contendo um ponto flutuante e dois inteiros. Os inteiros representam a linha e coluna de uma entrada cujo valor é dado pelo ponto flutuante. Teste seu tipo derivado usando um curto programa que o mande uma entrada de uma matriz de um processo para outro.

O programa com as modificações solicitadas encontra-se abaixo. Note que, como mencionado na questão 1, o uso do tipo derivado com variáveis soltas não é muito bom uma vez que este tipo só poderá ser usado para aquela configuração específica de memória. Se uma outra função quiser utilizar este tipo derivado, ela precisará redefinir o vetor de distancias (*displacements*). Por causa disso, ao invés de lidar com variáveis soltas, elas foram agrupadas em uma estrutura. O C garante que as informações contidas numa estrutura são armazenadas continuamente, o que nos garante que esta estrutura pode ser usadas por diversas funções de forma mais genérica que a anterior.

```
1  #include "mpi.h"
2
3  typedef struct {
4      int x;
5      int y;
6      float v;
7  } sparce_matrix;
8
9  void build_derived_type(sparce_matrix* mat,MPI_Datatype* msg_mpi_t_ptr) {
10     int block_lengths[] = {1,1,1};
11     MPI_Aint displacements[3], start_address, address;
12     MPI_Datatype typelist[3];
13
14     typelist[0] = MPI_INT;
15     typelist[1] = MPI_INT;
16     typelist[2] = MPI_FLOAT;
17
18     displacements[0] = 0;
19     MPI_Address(&(mat->x), &start_address);
20     MPI_Address(&(mat->y), &address);
21     displacements[1] = address - start_address;
22     MPI_Address(&(mat->v), &address);
23     displacements[2] = address - start_address;
24
25     MPI_Type_struct(3, block_lengths, displacements, typelist, msg_mpi_t_ptr
           );
26     MPI_Type_commit(msg_mpi_t_ptr);
27 }
```

4.4 Tendo em vista a regra de equivalencia de tipos, é possível para muitos tipos diferentes especificados por um remetente corresponderem a um dado tipo especificado pelo receptor. Considere um programa com as definições abaixo. Descreva brevemente a memória no processo 0 e no processo 1 referenciados por cada envio/recebimento. Quais recebimentos recebem que envios?

```
1  float        B[5][5];
2  float        x[5];
```

```
3  MPI_Datatype first_mpi_t;
4  MPI_Datatype second_mpi_t;
5  MPI_Datatype third_mpi_t;
6  int          blocklengths[5] = {1,1,1,1,1};
7  int          displacements[5];
8
9  MPI_Type_contiguous(5, MPI_FLOAT, &first_mpi_t);
10 MPI_Type_vector(5, 1, 5, MPI_FLOAT, &second_mpi_t);
11 for (i = 0; i <5; i++)
12     displacements[i] = 6*i;
13 MPI_Type_indexed(5, block_lengths, displacements, MPI_FLOAT, &third_mpi_t);
14
15 ...
16
17 Process 0:
18     MPI_Send(x, 5, MPI_FLOAT, 1, 0, MPI_COMM_WORLD);
19     MPI_Send(&(B[1][0]), 5, MPI_FLOAT, 1, 0, MPI_COMM_WORLD);
20     MPI_Send(x, 1, first_mpi_t, 1, 0, MPI_COMM_WORLD);
21     MPI_Send(&(B[0][3]), 1, second_mpi_t, 1, 0, MPI_COMM_WORLD);
22     MPI_Send(&(B[0][0]), 1, third_mpi_t, 1, 0, MPI_COMM_WORLD);
23 Process 1:
24     MPI_Recv(x, 5, MPI_FLOAT, 0, 0, MPI_COMM_WORLD, &status);
25     MPI_Recv(&(B[1][0]), 5, MPI_FLOAT, 0, 0, MPI_COMM_WORLD, &status);
26     MPI_Recv(x, 1, first_mpi_t, 0, 0, MPI_COMM_WORLD, &status);
27     MPI_Recv(&(B[0][1]), 1, second_mpi_t, 0, 0, MPI_COMM_WORLD, &status);
28     MPI_Recv(&(B[0][0]), 1, third_mpi_t, 0, 0, MPI_COMM_WORLD, &status);
```

O primeiro envio do processo 0 envia os 5 elementos de x para o x do processo 1. O segundo envio do processo 0 envia os primeiros 5 elementos da segunda linha de B para o mesmo lugar da matriz B no processo 1. O terceiro envio do processo 0 envia os 5 elementos de x para o x do processo 1. O quarto envio do processo 0 envia 5 elementos da quarta coluna de B para a segunda coluna da B do processo 1. O quinto envio do processo 0 envia os 5 elementos (i, j), onde $i = j$, para as mesmas posições na matriz B do processo 1.

5 Trabalhos de Programação

5.1

5.2

5.3

6 Conclusão

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

7 Referências

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