

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 consequência 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 melhorar 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.

(Já posso fazer) 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* msg_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
54      * msg_mpi_t_ptr) {
55      int block_lengths[3];
56      MPI_Aint displacements[3], start_address, address;
57      MPI_Datatype typelist[3];
58
59      /* Definindo o numero de elementos de cada bloco. */
60      block_lengths[0] = block_lengths[1] = block_lengths[2] = 1;
61
62      /* Definicao dos tipos que serao usados */
63      typelist[0] = MPI_FLOAT;
64      typelist[1] = MPI_FLOAT;
65      typelist[2] = MPI_INT;
66
67      /* Definindo a distancia do primeiro elemento como 0.
68       * Em seguida calcula-se a posicao de 'a' e de todos os
69       * elementos com relacao a 'a'. */
70      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     /* Construção do tipo derivado. */
77     MPI_Type_struct(3, block_lengths, displacements, typelist,
78                     mesg_mpi_t_ptr);
79
80     /* Indicando para o sistema que este tipo derivado será usado */
81     MPI_Type_commit(mesg_mpi_t_ptr);
82 }
83
84 void Get_data3(float* a_ptr, float* b_ptr, int* n_ptr, int my_rank) {
85     MPI_Datatype mesg_mpi_t;
86
87     if (my_rank == 0){
88         printf("Enter a, b, and n\n");
89         scanf("%f %f %d", a_ptr, b_ptr, n_ptr);
90     }
91
92     Build_derived_type(a_ptr, b_ptr, n_ptr, &mesg_mpi_t);
93     MPI_Bcast(a_ptr, 1, mesg_mpi_t, 0, MPI_COMM_WORLD);
94 }
95
96 float Trap(float local_a, float local_b, int local_n, float h) {
97     float x, integral;
98     int i;
99
100     /* Prototipo da função a ser integrada */
101     float f(float x);
102
103     integral = (f(local_a) + f(local_b))/2.0;
104     x = local_a;
105     for (i = 1; i <= local_n-1; i++) {
106         x = x + h;
107         integral = integral + f(x);
108     }
109     integral = integral*h;
110     return integral;
111 }
112
113 float f(float x) {
114     float return_val;
115     return_val = x*x;
116     return return_val;
117 }

```

- 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

(Já posso fazer)

5 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.

6 Referências

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