Introdução à programação paralela

Gabriel Martins de Miranda – 13/0111350

1. Título do capítulo

Grouping Data for Communication.

Empacotamento de dados para comunicação.Tipos de dados MPI.

2. Objetivo do capítulo

É bem sabido na comunidade de computação que mandar mensagens na geração atual de sistemas paralelos é uma operação custosa. Assim, são apresentados métodos de agrupamento de informação para se mandar o menor número de mensagens possível. São três os mecanismos para isto apresentados no capítulo, através do parâmetro **count**, de **tipos derivados** ou dos métodos **MPI_Pack** e **MPI_Unpack**.

3. Resumo do capítulo

São 3 os métodos para enviar mensagens com mais de um escalar. Para vetores (vários elementos do mesmo tipo), podemos usar métodos comuns de envio com count igual ao tamanho do vetor e datatype o tipo do vetor. Para tipos complexos, podemos construir tipos derivados ou usar os métodos MPI_Pack e MPI_Unpack.

Tipos derivados são structs passados como datatype das funções de comunicação. No mpi, é necessário saber o número de elementos de cada tipo, o tipo dos elementos e suas localizações relativas. No mpi, as funções para criação de tipos derivados são **MPI_Type_contiguous** (para subconjunto de entradas consecutivas num vetor), **MPI_Type_vector** (para elementos de vetor uniformemente espaçados na memória), **MPI_Type_indexed** (para elementos de vetor não espaçados uniformemente) e **MPI_Type_struct** (cujos elementos tem tipos e localizações arbitrários na memória). Antes de usado, o novo tipo deve ser apresentado ao sistema chamando-se a função **MPI_Type_commit**. Formalmente, o tipo derivado é uma sequência de pares, cujo primeiro elemento de cada par é o tipo e o segundo o deslocamento na memória. Aqui vai a sintaxe das funções apresentadas:

```
int MPI_Type_contiguous(
int             count       /*in*/,
MPI_Datatype    old_type    /*in*/,
MPI_Datatype    new_mpi_t   /*out*/)

int MPI_Type_vector(
int             count           /*in*/,
int             block_lenght    /*in*/,
int             stride          /*in*/,
MPI_Datatype    element_type    /*in*/,
MPI_Datatype    new_mpi_t       /*out*/)

int MPI_Type_indexed(
int             count           /*in*/,
int             block_lenghts[] /*in*/,
int             displacements[] /*in*/,
MPI_Datatype    old_type        /*in*/,
MPI_Datatype    new_mpi_t       /*out*/)
```

```
int MPI_Type_struct(
int             count              /*in*/
int             block_lenghts[]    /*in*/
MPI_Aint        displacements[]    /*in*/
MPI_Datatype    typelist[]         /*in*/
MPI_Datatype*   new_mpi_t          /*out*/)

int MPI_Type_commit(
MPI_Datatype*   new_mpi_t          /*in/out*/)
```

Existem ainda as funções **MPI_Pack** (armazena explicitamente dados num buffer definido pelo usuário) e **MPI_Unpack** (para extrair dados deste mesmo buffer). Suas assinaturas são:

```
int MPI_Pack(
void*           pack_data     /*in*/
int             in_count      /*in*/
MPI_Datatype    datatype      /*in*/
void*           buffer        /*out*/
int             buffer_size    /*in*/
int*            position      /*in/out*/
MPI_Comm        comm          /*in*/)

int MPI_Unpack(
void*           buffer        /*in*/
int             size          /*in*/
int*            position      /*in/out*/
void*           unpack_data   /*out*/
int             count         /*in*/
MPI_Datatype    datatype      /*in*/
MPI_Comm        comm          /*in*/)
```

Quando não usar tipos derivados: tipo usado poucas vezes, mensagens em buffers do usuário e não do sistema e quando especificado número de elementos contido.

4. Solução dos exercícios

Foi usado o MPICH, implementação de alta performance e portabilidade do MPI.

1. Programa do trapézio usando Get_data3.

Ok, agora um novo tipo de dados é criado contendo dois floats - a e b e um int – n usando-se MPI_Type_struct. A função de criação do tipo é chamada por todos os processos. Ao final todas as integrais são somadas usando-se o MPI_Reduce.

2. Programa do trapézio usando Get_data4.

Ok, envio de dados através de MPI_Pack e MPI_Unpack usando um buffer. O processo 0 dá Pack nos itens e envia como Broadcast usando a flag MPI_PACKED. Os outros processos recebem o buffer e o desfazem em variaveis usando MPI_Unpack.

3. Mandar uma entrada de matrix de um processo a outro. A entrada é uma estrutura de dois int's e um float.

Ok, o código é bem similar ao do exercício anterior. Apenas que agora em vez de dois floats e um int, usamos dois ints e um float.

5. Trabalhos de programação

1.a. Printa matriz quadrada em blocos de colunas. Todos mandam pra 0 e ele printa. Use MPI_Type_vector e MPI_Recv.

Ok, no programa, a coluna 3 de uma matriz é passada do processo 1 a 0. O processo 0 printa a coluna recebida.

1.b. Processo 0 lê linha de matriz e manda pros outros processos com broadcast. Use MPI_Type_vector e MPI_Send.

Da mesma forma que o exercício anterior, porém usando colunas.

3. Transposta de matriz. Todos mandam uma coluna como uma linha para processo 1.

Ok, manda coluna do processo 1 para linha do processo 0.

4. Mande entradas de um vetor esparso entre processos.

Pack uma linha de subscritos de vetor esparso e envia a outro processo.

6. Conclusão

Foram aprendidas formas de se enviar tipos compostos de dados de uma única vez, sem a necessidade de mandar cada elemento por vez. Isto garante melhor performance que no caso elemento a elemento.

7. Referências consultadas

Parallel Programming with MPI by Peter Pacheco ; Grouping Data for Communication – chapter six.