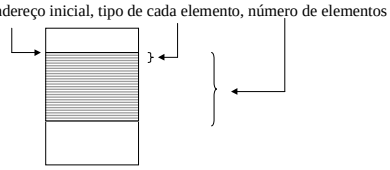
Empacotamento de Dados em MPI

Tópicos:
- Buffer de Mensagem
- Empacotamento/Desempacotamento de Dados
- Comparação entre Métodos

Referência: Pacheco,P.S. *Parallel Programming with MPI* Morgan Kaufmann, San Francisco, 1997.

Buffer de Mensagem

- Até aqui:
  - Buffers de mensagens: áreas contíguas de memória
  - Parâmetros:
    - Endereço inicial, tipo de cada elemento, número de elementos



Buffer de Mensagem (cont.)

- Caso de Dados Não-Contíguos em Memória:
  - Enviar várias mensagens, uma para cada área contígua

    Problema: Custo pode ser alto (custo = $\alpha+\beta n$, com $\alpha >> \beta$)

    Ex: Custo de 5 msgs(20 bytes) >> custo de 1 msg(100bytes)
  - Se os dados forem regularmente espaçados: usar *stride*
  - Dados não-regularmente espaçados:
    - Fazer empacotamento/desempacotamento de dados
    - Criar um novo "tipo" de dado em MPI (tipo *derivado*)

Buffer de Mensagem (cont.)

- Caso de dados regularmente espaçados:

Exemplo: Matriz bidimensional, em linguagem C
```
float A[10][10];
```
Transmitir terceira linha de A entre dois procesadores:
```
if (my_rank==0) MPI_Send(&A[2][0],10,MPI_FLOAT,...)
else  if (my_rank==1) MPI_Recv(&A[2][0],10,MPI_FLOAT,...)
```
$\Rightarrow$ Neste caso, dados estão contíguos

Como transmitir uma coluna de A entre dois processadores?
(Isto é, enviar A[0][2], A[1][2], A[2][2], ..., A[9][2] $\rightarrow$ não-contíguos)

Buffer de Mensagem (cont.)

Solução: Criação de tipo "vetor", com *stride*
```
MPI_Type_vector( int       count;
                 int       block_length;
                 int       stride;
                 MPI_Datatype element_type;
                 MPI_Datatype*  new_mpi_type; )
```
No caso do exemplo:
```
MPI_Datatype novotipo;
MPI_Type_vector(10, 1, 10, MPI_FLOAT, &novotipo);
MPI_Type_commit(&novotipo);
if (my_rank==0) MPI_Send(&A[0][2], 1, novotipo, 1, 0, ...
else if (my_rank==1) MPI_Recv(&A[0][2], 1, novotipo, 0, 0, ...
```


Empacotamento e Desempacotamento de Dados

- Objetivo:
  - Acumular, numa área contígua, dados esparsos
  - Após receber a mensagem, espalhar de novo os dados

```
int  MPI_Pack ( void*      pack_data;
                int        in_count;
                MPI_Datatype  datatype;
                void*      buffer;
                int        buffer_size_bytes;
                int*       position;
                MPI_Comm   comm; )
```
int MPI_Unpack( ) : Função oposta a MPI_Pack( )

Empacotamento e Desempacotamento de Dados

Exemplo: transmitir float* a_ptr, b_ptr ; int* n_ptr;

```
char  buffer[100];  /* Store data in buffer        */
int   position; if (my_rank == 0){
printf("Enter a, b, and n\n");
scanf("%f  %f  %d", a_ptr, b_ptr, n_ptr);
position = 0;
/* Position is in/out */
MPI_Pack(a_ptr, 1, MPI_FLOAT, buffer, 100,
   &position, MPI_COMM_WORLD);
/* Position has been incremented: it now refer- */
/* ences the first free location in buffer.     */
MPI_Pack(b_ptr, 1, MPI_FLOAT, buffer, 100,
   &position, MPI_COMM_WORLD);
/* Position has been incremented again. */
```


Empacotamento e Desempacotamento de Dados

```
   MPI_Pack(n_ptr, 1, MPI_INT, buffer, 100,
  &position,MPI_COMM_WORLD);
   /* Position has been incremented again. */
   /* Now broadcast contents of buffer */
   MPI_Bcast(buffer, 100, MPI_PACKED, 0,MPI_COMM_WORLD);
} else {
   MPI_Bcast(buffer, 100, MPI_PACKED, 0,MPI_COMM_WORLD);
   /* Now unpack the contents of buffer */
   position = 0;
   MPI_Unpack(buffer, 100, &position, a_ptr, 1,
      MPI_FLOAT, MPI_COMM_WORLD);
   /* Once again position has been incremented: */
   /* it now references the beginning of b.     */
  . . .
```

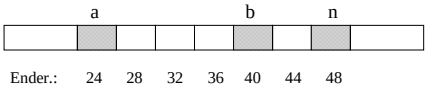

Criação de Tipo Derivado

- Objetivo: Definir um tipo de acordo com os dados

Exemplo: float a,b; int n;

⇒ Mesmo com declarações próximas, não há garantia de que as variáveis serão contíguas na memória

Possível alocação pelo compilador C:

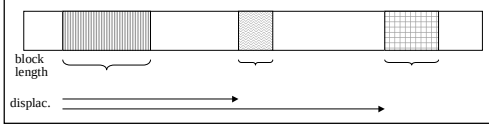


Problema: Como transmitir {a,b,c} numa única mensagem?

Criação de Tipo Derivado (cont.)

- Solução:

```
MPI_Type_struct(
      int           count              /* in */ ,
      int           block_lengths[ ]   /* in */,
      MPI_Aint      displacements[ ]   /* in */,
      MPI_Datatype  typelist[ ]        /* in */,
      MPI_Datatype* novotipo           /* out */ );
```



Criação de Tipo Derivado (cont.)

Implementação do Exemplo, com MPI:

```
void Build_derived_type(
      float*      a_ptr        /* in  */,
      float*      b_ptr        /* in  */,
      int*        n_ptr        /* in  */,
      MPI_Datatype* mesg_mpi_t_ptr  /* out  */) { /* ptr to new type */
  /* The number of elements in each "block" of the   */
  /*     new type.  For us, 1 each.                  */
  int block_lengths[3];
   /* Displacement of each element from start of new  type.  The "d_i's." */
  MPI_Aint displacements[3];
   /* MPI types of the elements.  The "t_i's."       */
  MPI_Datatype typelist[3];
   /* Use for calculating displacements              */
  MPI_Aint start_address, address;
```


Criação de Tipo Derivado (cont.)

```
  block_lengths[0] = block_lengths[1] = block_lengths[2] = 1;
   /* Build a derived datatype consisting of two floats and an int    */
  typelist[0] = MPI_FLOAT;  typelist[1] = MPI_FLOAT; typelist[2] = MPI_INT;
   /* First element, a, is at displacement 0      */
  displacements[0] = 0;
   /* Calculate other displacements relative to a */
  MPI_Address(a_ptr, &start_address);
   /* Find address of b and displacement from a   */
  MPI_Address(b_ptr, &address);  displacements[1] = address - start_address;
   /* Find address of n and displacement from a   */
  MPI_Address(n_ptr, &address);  displacements[2] = address - start_address;
   /* Build the derived datatype */
  MPI_Type_struct(3, block_lengths, displacements, typelist, mesg_mpi_t_ptr);
   /* Commit it -- tell system we'll be using it for communication. */
  MPI_Type_commit(mesg_mpi_t_ptr);    }
```

Criação de Tipo Derivado (cont.)

```
void Get_data3(
        float*  a_ptr    /* out */,
        float*  b_ptr    /* out */,
        int*    n_ptr    /* out */,
        int     my_rank  /* in  */) {
    MPI_Datatype  mesg_mpi_t; /* MPI type corresponding */
                              /* to 2 floats and an int */
        if (my_rank == 0){
        printf("Enter a, b, and n\n");
        scanf("%f %f %d", a_ptr, b_ptr, n_ptr);
    }

    Build_derived_type(a_ptr, b_ptr, n_ptr, &mesg_mpi_t);
    MPI_Bcast(a_ptr, 1, mesg_mpi_t, 0, MPI_COMM_WORLD);
}
```


Comparação entre Métodos

- Com empacotamento:
  - Menor “overhead” para preparar mensagem; porém...
  - A cada nova transmissão, é necessário empacotar e desempacotar dados
- Com tipo derivado:
  - Maior “overhead” para criação do novo tipo; porém...
  - Uma vez criado o tipo, pode ser usado diversas vezes

⇒ Na prática:
- Escolha depende de cada caso
- Não há um método universalmente superior