C-DAC Four Days Technology Workshop

ON

Hybrid Computing – Coprocessors/Accelerators
Power-Aware Computing – Performance of Applications Kernels

hyPACK-2013
(Mode-1:Multi-Core)

Lecture Topic:
Multi-Core Processors:
MPI 1.0 Overview (Part-IV)

Venue : CMSD, UoHYD ; Date : October 15-18, 2013

Lecture Outline

- ❖ MPI advanced point-to-point communication
- ❖ MPI Communication modes
- ❖ Grouping Data for Communication –Derived Data Types
- ❖ Conclusions

Source : Reference : [4], [11], [12],[14], [25], [26]

MPI Basic Datatypes

(Contd…)

MPI Basic Datatypes - C

MPI Datatype	C datatype
MPI_CHAR	Signed char
MPI_SHORT	Signed short int
MPI_INT	Signed int
MPI_LONG	Signed long int
MPI_UNSIGNED_CHAR	Unsigned char
MPI_UNSIGNED_SHORT	Unsigned short int
MPI_UNSIGNED	Unsigned int
MPI_UNSIGNED_LONG	Unsigned long int
MPI_FLOAT	Float
MPI_DOUBLE	Double
MPI_LONG_DOUBLE	Long double
MPI_BYTE	
MPI_PACKED	

MPI Derived Datatypes

Message type

- A message contains a number of elements of some particular datatype
- MPI datatypes:
  - Basic types
  - Derived Data types (Vectors; Structs; Others)
- Derived types can be built up from basic types
- C types are different from Fortran types

MPI Derived Types - `MPI_Type_Struct`

- Background :
  - How to distribute `a float value 'a; a float value 'b;` and `integer 'n'` to other process `?`
  - How to distribute `struct in which values a, b,` and `n` are stored to other process ?
- Efficient handle of Memory Management (Contiguous Memory)

```
typedef strucut {
    float a;
    float b;
    int a;
    }
    INPUT_TYPE;
```

the variable definition

INPUT_TYPE input;

Source : Reference : [11], [12], [25], [26]

MPI Derived Types - `MPI_Type_Struct`

Question :

What happens to distribute **struct** using MPI_Bcast to other process ?

❖ It won't work ! and Compiler Should scream at you

❖ **MPI provides MPI_Datatype**

❖ (How to define variables ?) For example Select a=2.0; b=3.0; n=2040.

The first element is a `float`.
The second element is a `float`
The `thrid` element is `integer`

The first element has address `&a`
The second element has address `&b`
The third element has integer `&c`

Variable	Address	Contents
`a`	24	2.0
`b`	40	3.0
`n`	48	2040

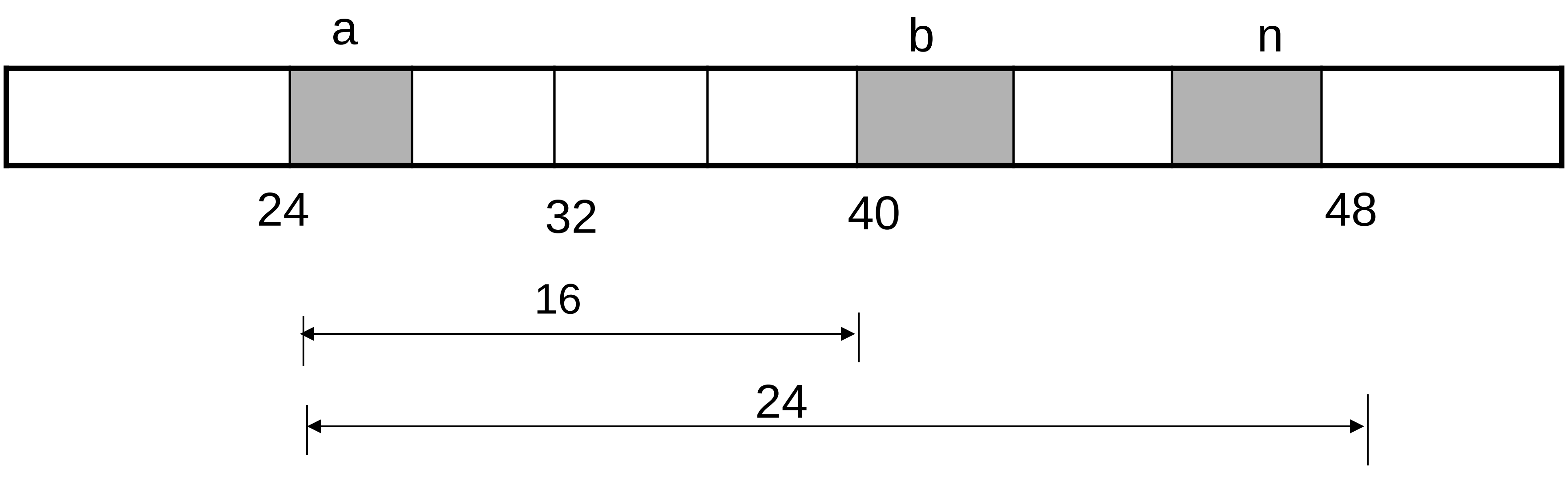
MPI Derived Types -`MPI_Type_Struct`

❖Only address (&`a`) is provided

❖MPI uses the concept of relative addresses or displacements of '`b`' and '`n`' from '`a`'

❖Only address (&`a`) is provided

Construct Memory layout with displacements



MPI Derived Types - `MPI_Type_Struct`

<u>Information can be provided to Communication subsystems</u>

❖ There are three elements to be transmitted

- The first element is a `float`.
- The second element is a `float`.
- The `third` element is `integer`

❖ **Displacement Information**

- The first element is displayed 0 bytes from the beginning of the message
- The second element is displayed 16 bytes from the beginning of the message
- The third element is displayed 24 bytes from the beginning of the message..

MPI Derived Types - `MPI_Type_Struct`

General MPI datatype or derived datatype

- It is sequence of pairs $\{(t_o, d_o), (t_1,d_1);,,,,, t_{n-1}, d_{n-1})\}$; .

  where each t_i is a basic MPI datatype and each d_i is a displacement in bytes.

Building Derived Datatype :

To incorporate `a,b,`and `n` into single message

- Build general derived datatypes : Use

  int MPI_Type_struct (…………………………………….)

MPI Derived Types - `MPI_Type_Struct`

How to build derived datatypes ?

```
Void Build_derived_type(
  float *            a_ptr              /* in….  */,
  float *            b_ptr              /* in ….*/,
  int *              n_ptr              /* in…..*/,
  MPI_Datatype  mesg_mpi_t_ptr          /*out */  )
 {
   /* Pointer to new MPI type */

  /* The number of elements in each "block" of the new type is 1   */
  Int block_lengths[3];
```

MPI Derived Types - MPI_Type_Struct

How to build derived datatypes ?

/* Displacement of each element from start of new type. The “d _1’s”, */

/* MPI_Aint (“address int”) is an MPI_defined C type. Usually an `int` or a long `int` */

MPI_Aint displacements[3];

/* MPI types of the elements, The “t_I’s,” */

MPI_Datatype tylelist[3];

/* Use for calculating displacements */

MPI_Ainit start_Addres;

MPI_Aint address; .

Block_lengths[0] = block_lengths[1]=block_lengths[2] =1;

MPI Derived Types - MPI_Type_Struct

How to build derived datatypes ?

```
/* 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  */
displacement[0] = 0;

/* Calculate other displacements relative to a    */
MPI_Address (a_ptr, &start_address);
```

How to build derived datatypes ?

```
/* Find Address of b and displacement from a */
MPI_Address (b_ptr, &address);
displacement[1] = address – start_address;

/* Find Address of n and displacement from a */
MPI_Address (n_ptr, &address);
displacement[2] = address – start_address;

/* Build the derived datatype */
MPI_type_struct(3, block_lengths, displacements, typelist,
    mesg_mpi_t_ptr);
```

How to build derived datatypes ?

```
/* Commit - Using for Communication                    */
MPI_Type_Commit (mesg_mpi_t_ptr);
} /* Build_derived_type */

Void  Get data3 (
  float *            a_ptr                       /* out….  */,
  float *            b_ptr                        /* out….*/,
  int *              n_ptr                        /*  in…...*/,
  MPI_Datatype  mesg_mpi_t;       /* MPI  type corresponding  */
                                  /* to a, b and n            */ )
```

MPI Derived Types -`MPI_Type_Struct`

How to build derived datatypes ?

```
If (my_rank == 0)
   printf("Enter a, b, and n  \ n");
    scanf (" %f %f %d", a_ptr, b_ptr, n_ptr);

      Build_derived_datatype(a_ptr, b_ptr, n_ptr, &mesg_mpi_t);
      MPI_Bcast(a_ptr,1, mesg_mpi_t, Root, MPI_COMM_WORLD);
  }/* Get_data3 */
```

MPI Derived Data types

Committing a datatype

- Once a datatype has been constructed, it needs to be committed before it is used.
- This is done using `MPI_Type_Commit`
- **C**

  int MPI_Type_Commit (MPI_Datatype *datatype);

- **Fortran**

  MPI_Type_Commit (datatype, ierror)

  integer datatype, ierror

Source : Reference : [11], [12], [25], [26]

MPI Derived Types : MPI_Type_Struct

Build general derived datatypes

```
Int MPI_Type_Struct (
    Int              count               /* in….  */,
    Int              block_lengths [ ]   /* in ….*/,
    MPI_Aint         displacements[  ]   /* in…..*/,
    MPI_Datatype     typelist [  ]       /* in……*/,
    MPI_Datatype*    new_mpi_t           /* out…*/   );
```

MPI Derived Data types

Constructing a Struct Datatype

- C :

```
int MPI_Type_struct (int count, int array_of_blocklengths,
    MPI_Aint *array_of_displacements,
    MPI_Datatype *array_of_types,
    MPI_Datatype *newtype);
```

- Fortran :

```
MPI_Type_Struct (count, array_of_blocklengths,
        array_of_displacements,  array_of_types, newtype, ierror)
```

MPI Derived Datatype Constructors :Type Matching

Contiguous Data

- The simplest derived `datatype` consists of a number of contiguous items of the same `datatype`

- **C :**

  int MPI_Type_contiguous (int count, MPI_Datatype oldtype,MPI_Datatype *newtype);

- **Fortran :**

  MPI_Type_contiguous (count, oldtype, newtype)

  integer count, oldtype, newtype

MPI Derived Data types

Example 1: Write a C-program to read 5th row of a square matrix of size (10 X10) from the process with Rank 0 and send the message to process with Rank 1 without using MPI Derived Datatypes library calls. Assume that the entries of matrix are single precision float values.

- The simplest derived `datatype` consists of a number of contiguous items of the same `datatype`
- Recall that C Stores in row-major order

Input :

float A[10][10]

	0	1	2	3	4	5	6	7	8	9
0	22	14	61	43	15	34	21	8	7	32
1	22	34	2	8	17	58	9	8	4	93
2	32	8	81	52	47	63	7	83	65	23
3	17	29	28	72	24	19	61	27	45	18
4	**14**	**4**	**31**	**2**	**67**	**71**	**8**	**9**	**7**	**11**
5	26	24	21	9	17	14	9	8	4	66
6	53	8	81	2	9	43	7	83	8	13
7	63	24	31	73	4	79	31	33	5	56
8	81	4	31	2	13	55	8	9	45	21
9	48	54	55	71	83	96	39	58	74	34

MPI Derived Data types

```
int Count=10;
int Count; Destination, Destination_tag, Source, Source_tag;
    /*……..Sending the 5th row of two-dimensional array  ……*/
 If (my_rank ==0)
{
    /*……..Get the row and send to the process with rank = 1  …. */
    Destination = 1; Destination_tag =0;
    MPI_Send(&A[5][0],  Count,  MPI_FLOAT,
            Destination, Destination_tag, MPI_COMM_WORLD);}
else  ( */  …..My Rank =1 ……*/) {
    Source = 0; Source_tag =0;
    /* …….Process with rank 1 receives the data   …..*/
    MPI_Recv(&A[5][0],  Count,  MPI_FLOAT,
            source, source_tag, MPI_COMM_WORLD,  &status);
}
```

Data is contiguous

MPI Other Derived Data type Constructors

Example 2: Write a C-program to read 5th row of a square matrix of size (10 X10) from the process with Rank 0 and send the same values to process with Rank 1 using MPI Derived Datatypes (Use MPI_Type_contiguous library calls) Assume that the entries of matrix single precision float values.

- The simplest derived `datatype` consists of a number of contiguous items of the same `datatype`
- Recall that C Stores in row-major order

Input :

float A[10][10]

	0	1	2	3	4	5	6	7	8	9
0	22	14	61	43	15	34	21	8	7	32
1	22	34	2	8	17	58	9	8	4	93
2	32	8	81	52	47	63	7	83	65	23
3	17	29	28	72	24	19	61	27	45	18
4	**14**	**4**	**31**	**2**	**67**	**71**	**8**	**9**	**7**	**11**
5	26	24	21	9	17	14	9	8	4	66
6	53	8	81	2	9	43	7	83	8	13
7	63	24	31	73	4	79	31	33	5	56
8	81	4	31	2	13	55	8	9	45	21
9	48	54	55	71	83	96	39	58	74	34

MPI Other Derived Data types Constructors

```
int count=10;
float A[10][10], B[10,10];
MPI_Datatype   input_row_mpi_t;
MPI_Type_contiguous(count, MPI_FLOAT, &input_row_mpi_t);
MPI_Type_Commit(&input_row_mpi_t)

If (my_rank ==0)
   MPI_Send  ( &A[5][0], Count, input_row_mpi_t,  1, 0, MPI_COMM_WORLD);
  else  ( */  …..My Rank =1 ……*/)
   MPI_Recv (&B[5][0], Count, input_row_mpi_t, 0,0, MPI_COMM_WORLD,
             &status);

 MPI_Type_free( &input_row_mpi_t);
```

MPI Other Derived Data types Constructors

Example 3: Write a C-program to send structure from process with Rank 0 to process with Rank 1 using MPI Derived Datatypes (MPI_Type_contiguous).

```
int count=4;
MPI_Datatype particle_type;
MPI_Type_contiguous( 4, MPI_DOUBLE, &particle_type);
MPI_Type_Commit ( &particle_type)
```

```
typedef struct {
   double x,y,z;
   double mass
} PARTICLE_T
Variable Def
PARTCILE_T particle_type
```

```
If (my_rank == 0)
     MPI_Send ( partcile, count, particle_type, 1, 0, MPI_COMM_WORLD);
 else
     MPI_Recv (particle  count,  particle_type,  0,0, MPI_COMM_WORLD,  &status);
MPI_Type_free( &particle_type);
```

MPI Other Derived Data types Constructors

Example 4: Write a C-program to read 5th column of a square matrix of size (10 X10) from the process with Rank 0 and send the same values to process with Rank 1 using MPI Derived Datatypes (Use MPI_Type_vector library calls) Assume that the entries of matrix are single precision float values.

- The simplest derived `datatype` consists of a number of contiguous items of the same `datatype`
- Recall that C Stores in row-major order

Input :

float A[10][10]

	0	1	2	3	4	5	6	7	8	9
0	22	14	61	43	15	**34**	21	8	7	32
1	22	34	2	8	17	**58**	9	8	4	93
2	32	8	81	52	47	**63**	7	83	65	23
3	17	29	28	72	24	**19**	61	27	45	18
4	14	4	31	2	67	**71**	8	9	7	11
5	26	24	21	9	17	**14**	9	8	4	66
6	53	8	81	2	9	**43**	7	83	8	13
7	63	24	31	73	4	**79**	31	33	5	56
8	81	4	31	2	13	**55**	8	9	45	21
9	48	54	55	71	83	**96**	39	58	74	34

MPI Other Derived Data types Constructors

```
int MPI_Type_vector(
                int             count,          /*  in */,
                int             block_length[ ] /*  in  */,
                int             stride          /*  in  */,
                MPI_Datatype    element_type    /*  in  */,
                MPI_Datatype*   new_mpi_t       /*  out */ )

   /* …column_mpi_t is declared to have MPI_Datatype */

   MPI_Type_vector (10,1,10, MPI_FLOAT, &column_mpi_t);

   MPI_Type_Commit(&column_mpi_t)

 If (my_rank ==0)
      MPI_Send ( &A[0][5], 1, coulmn_mpi_t, 1, 0, MPI_COMM_WORLD);

  else

     MPI_Recv (&A[0][5], 1,  column_mpi_t,  0,0, MPI_COMM_WORLD,  &status);

 MPI_Type_free( &column_mpi_t);
```

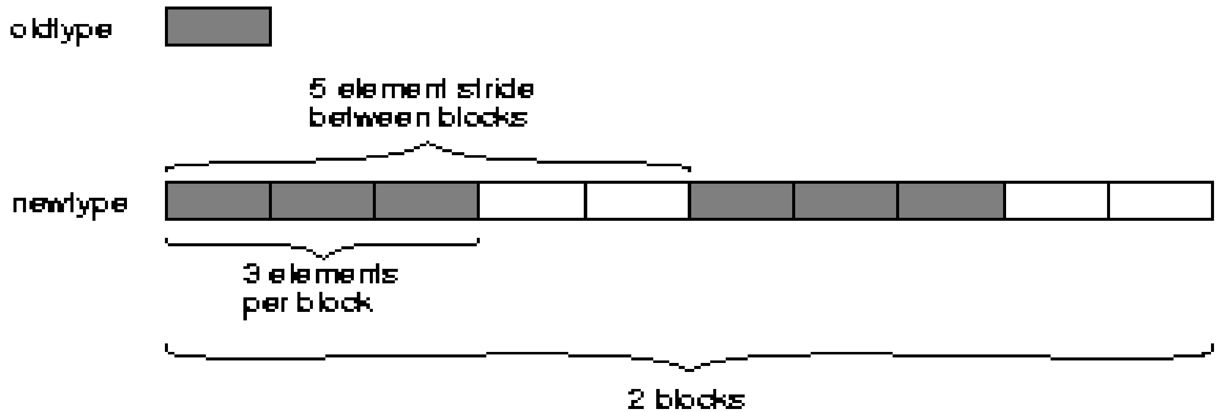
MPI Other Derived Data type Constructors

Vector Datatype

Count = 2;

Stride = 5;

block length = 3;



Struct Datatype

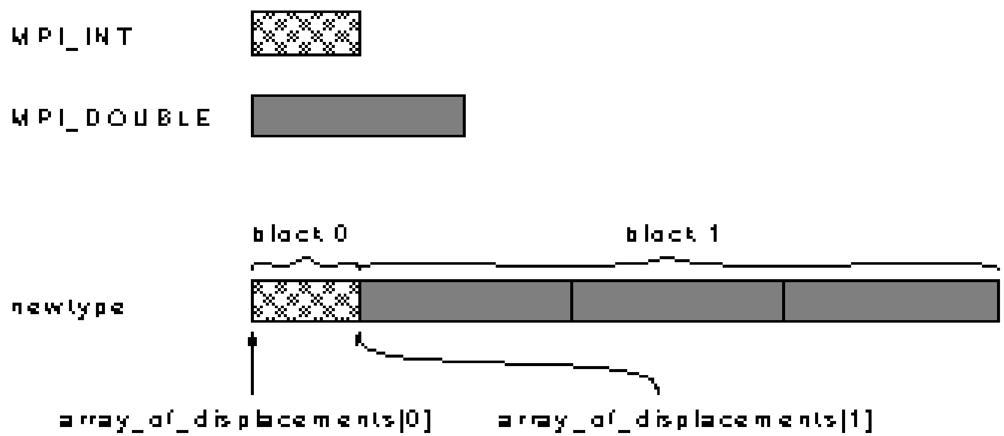
Count = 2;

Array_of_blocklengths[0] =1

Array_of_types[0] = MPI_INT;

Array_of_blocklengths[1] =3

Array_of_types[1] = MPI_DOUBLE;



MPI Other Derived Data type Constructors

Constructing a Vector Datatype

- **C**

  int MPI_Type_vector (int count, int blocklength, int stride, MPI_Datatype oldtype, MPI_Datatype *newtype);

- **Fortran**

  MPI_Type_vector (count, blocklength, stride, oldtype, newtype, ierror)

Extent of a Datatype

- **C**

  int MPI_Type_extent (MPI_Datatype datatype, int *extent);

- **Fortran**

  MPI_Type_extent(datatype, extent, ierror)

  integer datatype, extent, ierror

MPI Other Derived Data types Constructors

Example 4: Write a C-program to send upper triangular portion of a square matrix **A** of size (10 X10) from the process with Rank 0 to the process with Rank 1 using MPI Derived Datatypes (Use MPI_Type_indexed calls) Assume that the entries of matrix are single precision float values.

- The simplest derived `datatype` consists of a number of contiguous items of the same `datatype`
- Recall that C Stores in row-major order

Input :

float A[10][10]

	0	1	2	3	4	5	6	7	8	9
0	**22**	14	61	43	15	34	21	8	7	32
1	0	**34**	2	8	17	58	9	8	4	93
2	0	0	**81**	52	47	63	7	83	65	23
3	0	0	0	**72**	24	19	61	27	45	18
4	0	0	0	0	**67**	71	8	9	7	11
5	0	0	0	0	0	**14**	9	8	4	66
6	0	0	0	0	0	0	**7**	83	8	13
7	0	0	0	0	0	0	0	**33**	5	56
8	0	0	0	0	0	0	0	0	**45**	21
9	0	0	0	0	0	0	0	0	0	**34**

MPI Other Derived Datatype Constructors

(Contd…)

- Building a derived type consisting of copies of `oldtype` with a variety of block lengths and displacements.
- **C :**

```
int MPI_Type_indexed
 (
     int                  count,
     int                  block_lengths[ ],
     int                  displacements[ ],
     MPI_Datatype         oldtype,
     MPI_Datatype*        newtype_t
 )
```

MPI Other Derived Data types Constructors

```
float A[n][n],
float U [n,n];
Int displacements [n]; block_lenghts[n];
MPI_Datatype   index_mpi_t;

for (i = 0; i < n;  i++) {Block_lengths[i] = n-1;  Displacements[i] = (n+1)*1; }

MPI_Type_indexed(n, block_lenghts, displacements, MPI_FLOAT, &index_mpi_t);
MPI_Type_Commit ( &index_mpi_t);


If (my_rank == 0)
  MPI_Send  (A, 1, index_mpi_t, 1,  0,  MPI_COMM_WORLD);
else  ( */  …..My Rank == 1 ……*/)
   MPI_Recv ( U,  1, index_mpi_t, 1, 0, MPI_COMM_WORLD,  &status);
MPI_Type_free( &index_mpi_t);
```

MPI Derived Datatype Constructors :Type Matching

```
If (my_rank ==0)
  MPI_Send  ( message, sent_count, send_mpi_t, 1,0, MPI_COMM_WORLD);
 else  ( */  …..My Rank =1 ……*/)
 MPI_Recv             (message,recv_count,recv_mpi_t,1,0,MPI_COMM_WORLD,
&status);
```

- Is send_mpi_t be identical to recv_mpi_t ?
- What about send_count and recv_count ?

General MPI datatype or derived datatype

- It is sequence of pairs $\{(t_o, d_o), (t_1,d_1);,,,,, t_{n-1}, d_{n-1})\}$; .
  where each t_i is a basic MPI datatype and each d_i is a displacement in bytes.

MPI Derived Datatype Constructors :Type Matching

The sequence of basic type $\{(t_o, t_1, , , , , t_{n-1})\}$, is called **type Signature**.

- It is sequence of types specified by a derived datatype
- Fundamental rule for type matching in MPI is that the type signatures specified by the **sender** and the **receiver** must be compatible.
- Carry the Communication using MPI_Send & MPI_Recv
- Collective communications : the type signatures specified by all the processes must be identical

Source : Reference : [11], [12], [25], [26]

MPI Derived Datatype Constructors :Type Matching

- Communication using MPI_Send & MPI_Recv

The type signature specified by arguments passed to MPI_Send is

$$\{ (t_o, t_1,,,,, t_{n-1}) \},$$

And the type signature specified by the arguments to MPI_Recv is

$$\{ (d_o, d_1,,,,, d_{m-1}) \},$$

Then *n* must be less than or equal to *m* and t_i must equal for *i=0,…, m-1*

Example 5: Write a C-program to send the fist column of a square matrix A of size (10 X10) from the process with *Rank 0* and to the process with *Rank 1* using MPI Derived Datatypes and the concepts of MPI Type Matching. Assume that the entries of matrix are single precision float values.

Source : Reference : [11], [12], [25], [26]

MPI Derived Datatype Constructors :Type Matching

- Create Column_mpi_t : A column of 10 X 10 array of floats

  Type : ((MPI_FLOAT,0), (MPI_FLOAT, 10*sizeof(float)), ……..,

  (MPI_FLOAT, 20*sizeof(float)), ... (MPI_FLOAT, 90*sizeof(float)))

  and its type signature is (repeated 10 times

  (MPI_FLOAT, MPI_FLOAT,MPI_FLOAT, MPI_FLOAT, …..MPI_FLOAT)

MPI Derived Datatype Constructors :Type Matching

- MPI_Send to send a message consisting of one copy of column_mpi_t
- MPI_Recv provided the type signature specified by the receive consists of at least 10 floats

```
float A[10][10];
If (my_rank ==0)
   MPI_Send ( &A[0][0], 1, column_mpi_t, 1, 0, MPI_COMM_WORLD);
   MPI_Type_free( &column_mpi_t);
 else ( */ .....My Rank =1 ……*/)
 MPI_Recv ( &A[0][0], 10, MPI_FLOAT ,0, 0, MPI_COMM_WORLD, &status);
```

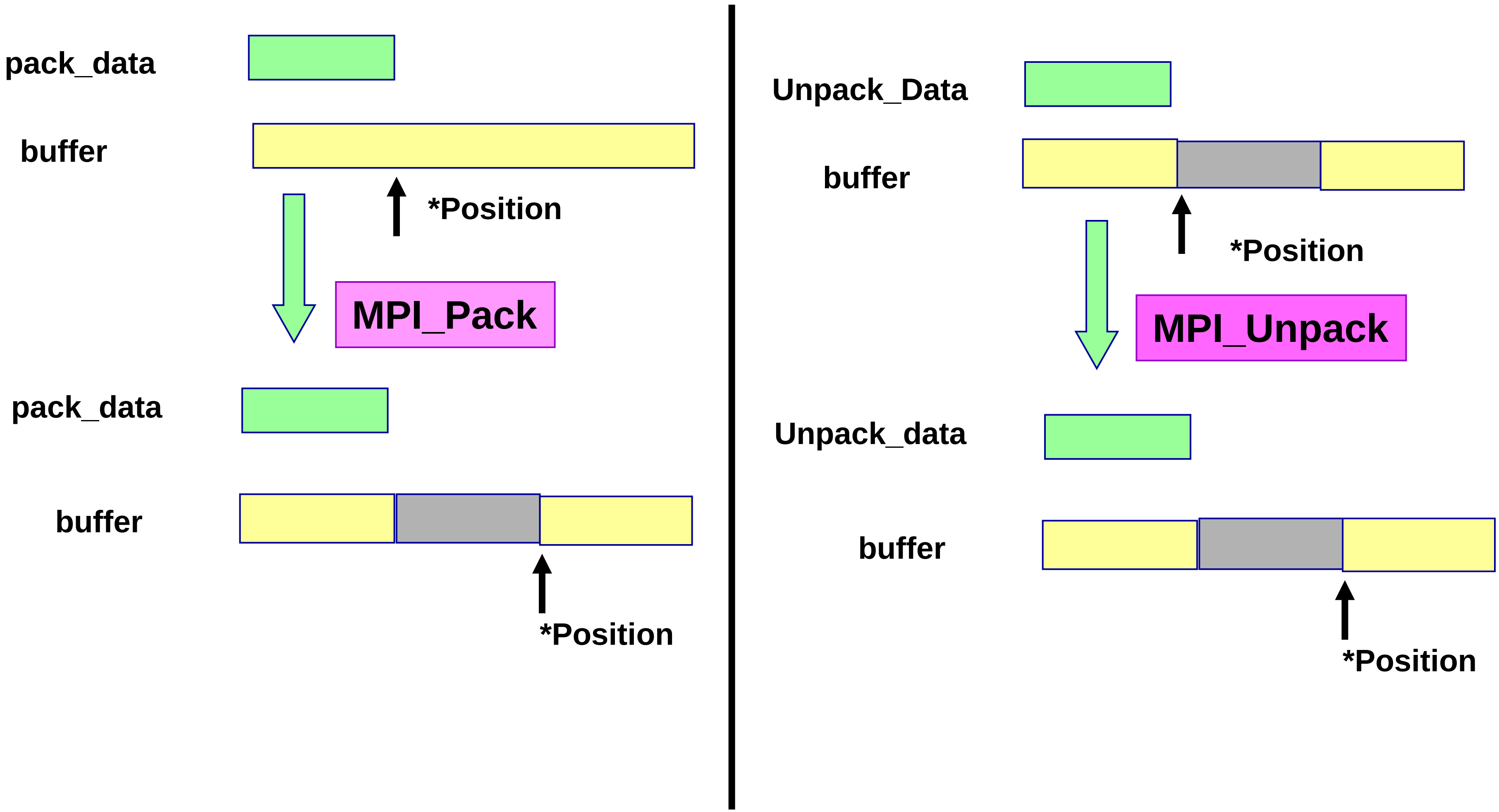
MPI : Derived Data types Constructors : Pack/Unpack

❖ Alternative approach to grouping data is provided by the MPI functions **MPI_Pack** and **MPI_Unpack**.

- MPI_Pack allows one to explicitly store noncontiguous data in contiguous locations

- MPI_Unpack can be used to copy data from a contiguous buffer into noncontiguous memory locations

MPI : Derived Data types Constructors : Pack/Unpack



MPI : Derived Data types Constructors

(Contd…)

Deciding Which Method to Use : Performance Issues

- Handing non-contiguous data;
- Test of 1000 element vector of doubles with stride of 24 doubles.
  - “MPI_Type_vector” and “MPI_Type_struct(.*,*)”;
  - “User packs and unpacks by hand
- Performance very dependent on implementation; should improve with time
- Collect many small messages into a single large message;
- Use of collective when many copies bcast /gather

Derived Datatype Constructors :Pack/Unpack

(Contd…)

Int MPI_Pack (

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

- MPI_Pack can be used to explicitly store data in a user-defined buffer
- MPI_Unpack can be used to extract data from a buffer that was constructed using MPI_Pack

Other Derived Datatype Constructors

Deciding Which Method to Use : Performance Issues

- Overheads- Calling Derived datatype (Creation of functions & Calls to MPI_Pack /MPI_Unpack
- Data to be sent – Consecutive entries of an array; data may have same type and stored at regular intervals in memory (Use either derived data types or MPI_Pack /Un_pack
- Data to be sent – data my have same type & stored in irregularly spaced locations in memory- Use MPI_Type_indexed
- Performance Issues :
  - "Sending heterogeneous data
  - Collect many small messages into a single large message;

MPI Process Topologies

General Topology Functions

```
int MPI_Topo_test(
    MPI_Comm        comm            /* in……*/,
    int             top_type        /* out….. */)
```

MPI_Topo_test: Determine whether comm has a topology and its type.

MPI supports

- Cartesian Topology Management
- Graph Topology Management

MPI : Working with Groups, Contexts, and Communicators

❖ Illustrate Basics & Example :

MPI implementation of matrix-matrix multiplication on 9 process

❖ **Objective :** Create a communicator whose underlying group consists of the processes in the first row of our virtual grid

- Assume that that MPI_COMM_WORLD consists of *p* processes where $q^2 = p$ (condition is necessary) and first row of the process consist of the processes with ranks *0,1,……, q-1*
- To create new communicator, the following steps are required

  - Make a list of the processes in the new communicator
  - Get the group underlying MPI_COMM_WORLD
  - Create the new group
  - Create the new communicator

3X3 grid processes		
0	1	2
3	4	5
6	7	8

MPI : Working with Groups, Contexts, and Communicators

The following code can be executed

```
MPI_Group   group_world;
MPI_Group   first_row_group;
MPI_Comm  first_row_comm;
int              proc,  q;
int*             process_ranks;

/*……..Make a list of the processes in the new communicator  ……*/
   process_ranks =  (int *) malloc (q*sizeof(int) );
   for (proc = 0; proc < q; proc++ )  process_ranks[proc] = proc;
/*……..Get the group underlying MPI_COMM_WORLD    …. */
   MPI_Comm_group(MPI_COMM_WORLD, &group_world);

/* …….Create the new group  …..*/
 MPI_Group_incl(group_wrold, q ,porcess_ranks, &first_row_group);
/* …….Create the new communicator …*/
 MPI_Comm_create (MPI_COMM_WORLD, first_row_group, &first_row_world);
```

MPI : Working with Groups, Contexts, and Communicators

```
int MPI_Comm_group(
    MPI_Comm       comm            /* in……*/,
    MPI_Group*     group           /* out …..*/)
```

MPI_Comm_group : Returns the group underlying the communicator *comm*

```
int MPI_Group_incl(
    MPI_Group      old_group               /* in……*/,
    int            new_group_size          /* in……*/,
    int            ranks_in_old_group[ ]   /* in …...*/,
    MPI_Group*     new_group               /* out….*/)
```

MPI_Group_incl : It creates a new group from a list of processes in existing group, old_group. The number of processes in the new group is new_group_size.

MPI : Working with Groups, Contexts, and Communicators

➡ int MPI_Comm_create(

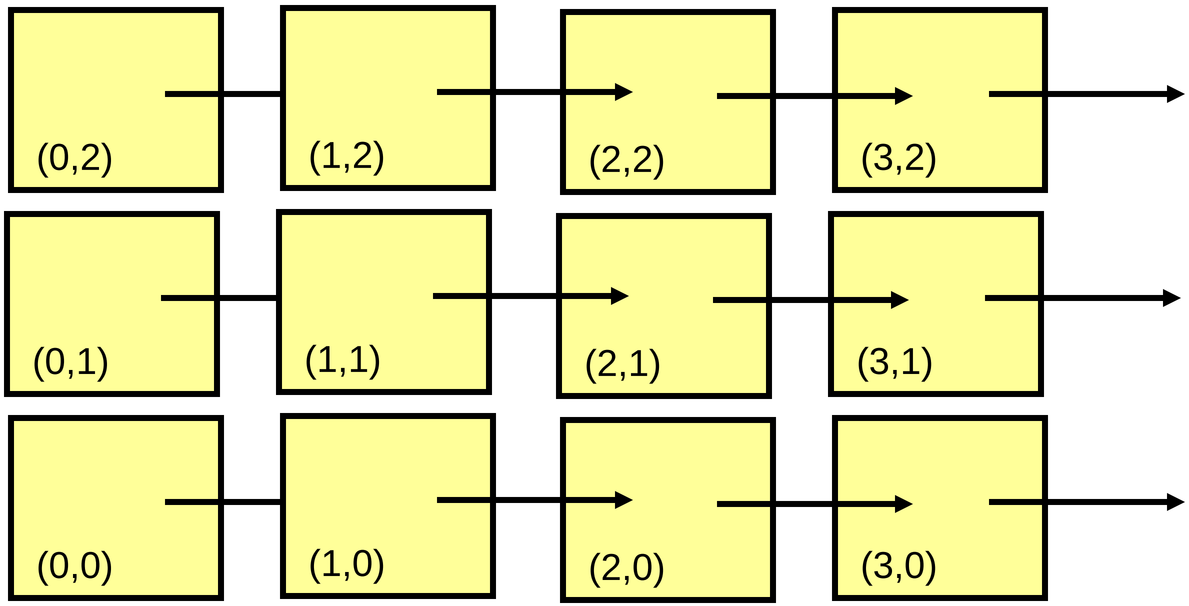
MPI_Comm	old_comm	/* in……*/,
MPI_Group	new_group	/* in……*/,
MPI_Comm*	new_comm	/* out…*/)

MPI_Comm_create : Associates a context with the group new_group and creates the communicator new_comm. All of the process in new_group belong to the group underlying old_comm.

- MPI_Comm_group and MPI_Group_incl are both local operations
- MPI_Comm_create is a collective operation
- If several communicators are being created, they must be created in the same order on all the processes.

MPI Process Topologies - Cartesian topology

- A two dimensional Cartesian Decomposition
- MPI provides a collection of routines for defining, examining, and manipulating Cartesian topologies



- For example, the **second** process from the ***left*** and the third from the ***bottom*** is labeled as (1,2)

MPI Process Topologies - Cartesian topology

- Creates a Cartesian decomposition of the processes with the number of dimensions given by the number_of_dims argument.
- Each element of the decomposition (rectangles in the figure) is labeled by a coordinate tuple indicating the position of the element in each of the coordinate directions.
- It is collective operation

```
int MPI_Cart_create (
    MPI_Comm        old_comm          /* in……*/,
    int             number_of_dims    /* in….. */,
    int             dim_sizes [ ]     /* in …..*/,
    int             wrap_around[  ]   /* in…...*/,
    int             reorder           /* in……*/,
    MPI_Comm*       Cart_Comm         /* out…*/ )
```

Number_of_dims =2; dim_sizes[1]= 4; dim_sizes[2]=3;

Wrap_around[1] = .false. ; wrap_around[1] =.true. ;

MPI Process Topologies - Cartesian topology

```
int MPI_Cart_get (
    MPI_Comm        comm            /* in……*/,
    int             max_dims        /* in….. */,
    int             dims [ ]        /* out …..*/,
    int             periods[  ]     /* out…...*/,
    int             coords[ ]       /* out……*/)
  printf (“( %d : %d  ) \n”, coords[0], coord[1] );
```

- ❖ Print the coordinates of the calling process in the communicator ‘comm’
- ❖ Returns both values of the dims[ ], and periods[ ],argument used in MPI_Cart_create

MPI Process Topologies - Cartesian topology

```
int MPI_Cart_rank(
    MPI_Comm      comm          /* in……*/,
    int           coords[ ]     /* in….. */,
    int*          rank          /* out …..*/)
 printf ("%d : %d  ) \n", rank );
```

returns the rank in the Cartesian communicator comm of the process with Cartesian coordinates.

```
int MPI_Cart_coords(
    MPI_Comm      comm             /* in……*/,
    int           rank             /* in……*/,
    int           number_of_dims   /* in …...*/,
    int*          coords[ ]        /* out….*/)
```

returns the coordinates of the process with rank 'rank' in the Cartesian communicator comm. It is the inverse to MPI_Cart_rank.

MPI Process Topologies - Cartesian topology

```
int MPI_Cart_sub(
    MPI_Comm     cart_comm          /* in……*/,
    int          free_coords[ ]     /* in….. */,
    MPI_comm*    new_comm           /* out …..*/)
```

It partitions the process in cart_comm into a collection of disjoint communicators whose union is cart_comm.

```
int MPI_Comm_split(
    MPI_Comm     comm               /* in……*/,
    int          split_key          /* in……*/,
    int          rank_key           /* in …...*/,
    MPI_comm*    new_comm           /* out….*/)
```

It creates a new communicator for each value of split_key. Process with same split_key form a new group.

- Several communicators can be created simultaneously using MPI_Comm_split

MPI Communicators

- Two types of Communicators

  Intra-communicators:Collection of processes that can send messages to each other in engage in collective communication operations

  Inter-communicators :Used for sending messages between processes belonging to disjoint intra-communicators

- Why should we bother about inter-communicators ?

- A minimal (intra) communicator is composed of

  - **a group** (is an ordered collection of processes. If a group consists of *p* process, each process in the group is assigned a unique **rank)**

  - **a context** (is a system defined object that uniquely identifies a communicator)

Source : Reference : [11], [12], [25], [26]

MPI Communicators

MPI Communicators : Remarks

❖ Two distinct communicators will have different contexts, even if they have identical underlying groups

❖ A context can be thought of as a system-defined tag that is associated with a group in communicator.

❖ Contexts are used to ensure that messages are received correctly.

3X3 grid processes		
0	1	2
3	4	5
6	7	8

Example : **MPI** implementation of matrix-matrix multiplication on 9 process (virtual grid = 3 X 3 square grid)

Group :

- Create a communicator for each row of the grid.
- Second row communicator -Processes {3,4,5} from MPI_COMM_WORLD
- Second row communicator –Process {0,1,2} is same as {3,4,5}

MPI Communicators : Remarks

Example : MPI implementation of matrix-matrix multiplication on 9 process

Contexts

- Define Context to be an int
- Each process can define list of available contexts
- When new communicator is created, the processes participating in the creation could "negotiate" the choice of a context that is available to each process.

Remarks

1. Construction of communicators, groups, and contexts are purely hypothetical
2. Vendor implementation of each object is system dependent
3. Vendor system an use something very different

Conclusions

- An Overview of Grouping Data for Communication – Derived Data Types

References

1. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Programming, Boston, MA : Addison-Wesley

2. Butenhof, David R **(1997),** Programming with POSIX Threads , Boston, MA : Addison Wesley Professional

3. Culler, David E., Jaswinder Pal Singh **(1999),** Parallel Computer Architecture - A Hardware/Software Approach , San Francsico, CA : Morgan Kaufmann

4. Grama Ananth, Anshul Gupts, George Karypis and Vipin Kumar **(2003),** Introduction to Parallel computing, Boston, MA : Addison-Wesley

5. Intel Corporation, **(2003),** Intel Hyper-Threading Technology, Technical User's Guide, Santa Clara CA : Intel Corporation Available at : http://www.intel.com

6. Shameem Akhter, Jason Roberts **(April 2006),** Multi-Core Programming - Increasing Performance through Software Multi-threading , Intel PRESS, Intel Corporation,

7. Bradford Nichols, Dick Buttlar and Jacqueline Proulx Farrell **(1996),** Pthread Programming O'Reilly and Associates, Newton, MA 02164,

8. James Reinders, Intel Threading Building Blocks – (**2007**) , O'REILLY series

9. Laurence T Yang & Minyi Guo (Editors), (**2006**) *High Performance Computing - Paradigm and Infrastructure* Wiley Series on Parallel and Distributed computing, Albert Y. Zomaya, Series Editor

10. Intel Threading Methodology ; Principles and Practices Version 2.0 copy right **(March 2003),** Intel Corporation

References

11. William Gropp, Ewing Lusk, Rajeev Thakur **(1999),** Using MPI-2, Advanced Features of the Message-Passing Interface, The MIT Press..

12. Pacheco S. Peter, **(1992),** Parallel Programming with MPI, , University of Sanfrancisco, Morgan Kaufman Publishers, Inc., Sanfrancisco, California

13. Kai Hwang, Zhiwei Xu, (**1998**), Scalable Parallel Computing (Technology Architecture Programming), McGraw Hill New York.

14. Michael J. Quinn (**2004**), Parallel Programming in C with MPI and OpenMP McGraw-Hill International Editions, Computer Science Series, McGraw-Hill, Inc. Newyork

15. Andrews, Grogory R. **(2000),** Foundations of Multithreaded, Parallel, and Distributed Progrmaming, Boston, MA : Addison-Wesley

16. SunSoft. Solaris multithreaded programming guide. SunSoft Press, Mountainview, CA, **(1996)**, Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill,

17. Chandra, Rohit, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon, **(2001)**,Parallel Programming in OpenMP San Fracncisco Moraan Kaufmann

18. S.Kieriman, D.Shah, and B.Smaalders **(1995),** Programming with Threads, SunSoft Press, Mountainview, CA. 1995

19. Mattson Tim, **(2002),** Nuts and Bolts of multi-threaded Programming Santa Clara, CA : Intel Corporation, Available at : http://www.intel.com

20. I. Foster **(1995,** Designing and Building Parallel Programs ; Concepts and tools for Parallel Software Engineering, Addison-Wesley (1995)

21. J.Dongarra, I.S. Duff, D. Sorensen, and H.V.Vorst **(1999),** Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools) SIAM, 1999

References

22. OpenMP C and C++ Application Program Interface, Version 1.0". **(1998),** OpenMP Architecture Review Board. October 1998

23. D. A. Lewine. *Posix Programmer's Guide:* **(1991),** Writing Portable Unix Programs with the Posix. 1 Standard. O'Reilly & Associates, 1991

24. Emery D. Berger, Kathryn S McKinley, Robert D Blumofe, Paul R.Wilson, *Hoard : A Scalable Memory Allocator for Multi-threaded Applications* ; The Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX). Cambridge, MA, November (**2000)**. Web site URL : http://www.hoard.org/

25. Marc Snir, Steve Otto, Steyen Huss-Lederman, David Walker and Jack Dongarra, (**1998**) *MPI-The Complete Reference: Volume 1, The MPI Core, second edition* [MCMPI-07].

26. William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir, and Marc Snir (**1998**) *MPI-The Complete Reference: Volume 2, The MPI-2 Extensions*

27. A. Zomaya, editor. Parallel and Distributed Computing Handbook. McGraw-Hill, **(1996)**

28. OpenMP C and C++ Application Program Interface, Version 2.5 (**May 2005**)", From the OpenMP web site, URL **: http://www.openmp.org/**

29. Stokes, Jon 2002 Introduction to Multithreading, Super-threading and Hyper threading *Ars Technica*, October **(2002)**

30. Andrews Gregory R. 2000, Foundations of Multi-threaded, Parallel and Distributed Programming, Boston MA : Addison – Wesley (**2000)**

31. Deborah T. Marr , Frank Binns, David L. Hill, Glenn Hinton, David A Koufaty, J . Alan Miller, Michael Upton, "Hyperthreading, Technology Architecture and Microarchitecture", Intel (**2000-01)**

Thank You

Any questions ?