Dear all,

I use a PC with following specification:

Windows 7 OS, 64 Bit, 4 GB of RAM, Intel® Core(TM)2Duo CPU 3.34 GHz

I have gotten an Email from MPICH Mailing list. In that Email Prof. Kovács has discussed about sending and receiving structured data type of data type in C programming language. The Email and discussion about it is stated at the end of this letter. I also tried to compiling and running the program of Prof. Kovács on my PC. It worked without any problem with MPICH 1.3 and also Intel MPI.

In my small project, I want to do similar work but in Fortran programming language. In the following, I have explained my small project. Source code is attached. I defined ***Particle*** data type that contains 4 INTEGER4, 11 INTEGER2, 2 LOGICAL and 2 INTEGER4 array with size 6. To define an MPI data type for the ***particle*** data type, first I defined ***columntype*** type for INTEGER4 array with size 6. Then I structured ***particletype*** and committed it. I defined two array (***p***,***particles***) of type ***particletype*** `and allocated them (ALLOCATE(particles(0:NELEM));ALLOCATE(p(0:1010))). I initialized` ***particles*** `array. For sending specific segments of` ***particles*** `array from server process (rank=0) to array` ***P*** `on Client processes, I defined` ***indextype*** `MPI data type. I want to get the array on client processes without gap and continuancely. In sending with`

```
 call MPI_SEND(particles, 1008, particletype, i, tag, MPI_COMM_WORLD, ierr)

Client processes get first 521 elements truly and for the rest, get garbage.
In following sending case:

  call MPI_SEND(particles, 1, indextype, i, tag, MPI_COMM_WORLD, ierr)

Client processes get first 412 elements truly and for the rest they get wrong numbers.
I got above results for both Win32 and x64 projects. I also tried Intel MPI. But the
results are the same.
Is problem from definition of MPI data type or its origin is huge array size?
For sizes smaller than 1008, also the situation is same and last cells of the array
does not send truly.
I examined buffer sending.

CALL MPI_PACK_SIZE(1, particletype, MPI_COMM_WORLD, SS1, IERR)
bufsize = MPI_BSEND_OVERHEAD + (ss1)*1008
ALLOCATE(particles_buf(bufsize))
CALL MPI_Buffer_attach( particles_buf, bufsize, IERR)
CALL MPI_BSEND(particles,1008,particletype,i,TAG,MPI_COMM_WORLD,IERR)
CALL MPI_BUFFER_DETACH(particles_buf, bufsize, IERR)

But the result is the same.
The following is my small project and discussion about Email of Prof. Zoltán is
narrated.

program ping
implicit none
include 'mpif.h'
integer :: numtasks, rank, dest, source, tag, ierr
integer :: i
integer :: NELEM
integer :: stat(MPI_STATUS_SIZE)
type Particle
```

```fortran
INTEGER(4) :: IW1
INTEGER(4) :: IP1
INTEGER(4) :: IW2
INTEGER(4) :: IP2
INTEGER(4) :: IPWF
INTEGER(4) :: IP
INTEGER(4) :: IL10
INTEGER(2) :: IMP4
INTEGER(2) :: IMP5
INTEGER(2) :: LIB1
INTEGER(2) :: LIB2
INTEGER(2) :: LIB3
INTEGER(2) :: IVOL
INTEGER(2) :: ITARG
INTEGER(2) :: ISEG
INTEGER(2) :: NFAU
INTEGER(2) :: NMAX
INTEGER(2) :: NTOT
LOGICAL :: nomix
LOGICAL :: TRcorr
INTEGER(4) :: JZ(6)
INTEGER(4) :: IJ(6)
end type Particle
type (Particle),DIMENSION(:),POINTER :: p, particles, particles_buf
integer particletype, oldtypes(0:3), blockcounts(0:3), offsets(0:3), extent
integer columntype,SIZE,id
integer NUMSEG,indextype,SS1,SS2,SS3,ss4,bufsize
integer,allocatable,dimension(:) :: blocklengths,displacements
tag = 1
call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numtasks, ierr)
NELEM=10000
ALLOCATE(particles(0:NELEM))
ALLOCATE(p(0:1010))
! Setup description of the 4 MPI_REAL fields x, y, z, velocity
offsets(0) = 0
oldtypes(0) = MPI_INTEGER4
blockcounts(0) = 7
call MPI_TYPE_EXTENT(MPI_INTEGER4, extent, ierr)
offsets(1) = 7 * extent
oldtypes(1) = MPI_INTEGER2
blockcounts(1) = 11
! Setup description of the 2 MPI_INTEGER fields n, type
! Need to first figure offset by getting size of MPI_REAL
call MPI_TYPE_EXTENT(MPI_INTEGER2, extent, ierr)
offsets(2) = 11 * extent
oldtypes(2) = MPI_LOGICAL
blockcounts(2) = 2
SIZE=6
call MPI_TYPE_CONTIGUOUS(SIZE, MPI_INTEGER4, columntype, ierr)
call MPI_TYPE_COMMIT(columntype, ierr)
call MPI_TYPE_EXTENT(MPI_LOGICAL, extent, ierr)
offsets(3) = 2 * extent
oldtypes(3) = columntype
blockcounts(3) = 2
! Now define structured type and commit it
call MPI_TYPE_STRUCT(4, blockcounts, offsets, oldtypes, particletype, ierr)
call MPI_TYPE_COMMIT(particletype, ierr)
! Initialize the particle array and then send it to each task
tag = 1
```

```
if (rank .eq. 0) then
do 10 i=0, NELEM-1
particles(i)%IW1=i+1
particles(i)%IP1=i+2
particles(i)%IW2=i+3
particles(i)%IP2=i+4
particles(i)%IPWF=i+5
particles(i)%IP=i+6
particles(i)%IL10=i+7
particles(i)%IMP4=i+8
particles(i)%IMP5=i+9
particles(i)%LIB1=i+10
particles(i)%LIB2=i+11
particles(i)%LIB3=i+12
particles(i)%IVOL=i+13
particles(i)%ITARG=i+14
particles(i)%ISEG=i+15
particles(i)%NFAU=i+16
particles(i)%NMAX=i+17
particles(i)%NTOT=i+18
particles(i)%nomix=.FALSE.
particles(i)%TRcorr=.FALSE.
particles(i)%JZ=i
particles(i)%IJ=i+1
10 continue
NUMSEG = 4
ALLOCATE(blocklengths(NUMSEG), displacements(NUMSEG))
blocklengths(1)=798; displacements(1)=0
blocklengths(2)=84; displacements(2)=7798
blocklengths(3)=114; displacements(3)=4800
blocklengths(4)=12; displacements(4)=8952
CALL MPI_TYPE_INDEXED(NUMSEG, blocklengths, displacements, particletype,indextype,
ierr)
call MPI_TYPE_COMMIT(indextype, ierr)
do 20 i=1, numtasks-1
! call MPI_SEND(particles, 1, indextype, i, tag, MPI_COMM_WORLD, ierr)
call MPI_SEND(particles, 1008, particletype, i, tag, MPI_COMM_WORLD, ierr)
!!CALL MPI_PACK_SIZE(7, MPI_INTEGER4, MPI_COMM_WORLD, SS1, IERR)
!!CALL MPI_PACK_SIZE(11, MPI_INTEGER2, MPI_COMM_WORLD, SS2, IERR)
!!CALL MPI_PACK_SIZE(2, MPI_LOGICAL, MPI_COMM_WORLD, SS3, IERR)
!!CALL MPI_PACK_SIZE(2, columntype, MPI_COMM_WORLD, SS4, IERR)
!!bufsize = MPI_BSEND_OVERHEAD + (ss1 + ss2 + ss3 + ss4)*1008
!
!CALL MPI_PACK_SIZE(1, particletype, MPI_COMM_WORLD, SS1, IERR)
!bufsize = MPI_BSEND_OVERHEAD + (ss1)*1008
!
!ALLOCATE(particles_buf(bufsize))
!CALL MPI_Buffer_attach( particles_buf, bufsize, IERR)
!CALL MPI_BSEND(particles,1008,particletype,i,TAG,MPI_COMM_WORLD,IERR)
!CALL MPI_BUFFER_DETACH(particles_buf, bufsize, IERR)
20 continue
endif
id=522
if (rank>0) then
source = 0
call MPI_RECV(p, 1008, particletype, source, tag, MPI_COMM_WORLD, stat, ierr)
print *, 'rank= ',rank,' p=',p(id)%nomix,p(id)%TRcorr,p(id)%IJ
end if
call MPI_TYPE_FREE(particletype, ierr)
call MPI_FINALIZE(ierr)
end program ping
```

--- On **Mon, 11/15/10, Jayesh Krishna <*jayesh@mcs.anl.gov*>** wrote:

Also you might want to add the sender proc id in your dbg stmt,

printf("Proc %d: recieving %u.th portion from proc %d ....\n", rank, i);

TO

printf("Proc %d: recieving %u.th portion from proc %d ....\n", rank, i, proc);

Regards,
Jayesh
----- Original Message -----
From: Jayesh Krishna <jayesh@mcs.anl.gov>
To: mpich-discuss@mcs.anl.gov

Hi,
And you can try using MPI_Gather() for getting the chunks from the different processes to rank=0 . Similarly you can use MPI_AllGather() if you want all the processes to receive the chunks (instead of just rank 0).

Regards,
Jayesh
----- Original Message -----
From: Jayesh Krishna <jayesh@mcs.anl.gov>
To: mpich-discuss@mcs.anl.gov

Hi,
What is the version of MPICH2 that you are using (From the error message it looks like you are using MPICH instead of MPICH2. Are you running your job on a heterogeneous cluster ?)? We recommend that you use the latest stable release of MPICH2 (http://www.mcs.anl.gov/research/projects/mpich2/downloads/index.php?s=downloads).
I tried compiling/running your job on a single node and it worked without any errors with MPICH2 1.3 .

Regards,
Jayesh
----- Original Message -----
From: zkovacs <zkovacs@hku.hk>
To: mpich-discuss@mcs.anl.gov

Dear all,

I defined a structure with the name "st1", created an MPI datatype for it and commited it in the function crt_dt(). I also defined another structure with the name "st2" containing 4 structures of the first type (st1). Its MPI data type is also defined and commited it in the function crt_dt(). After allocating an 1D data array of structures st2 with the size N x M, I initialized each structure member of all the structures in the array by calling loc_init(). Then I tried to send chunks of the array with the size of M from the process with ranks higher than 0

to the process 0. The offsets of the chunks for a process with the rank r were set to (r + i x n) x M < N*(M-1) with i =1,2,... , where n is the total number of the processes. That would cover the whole array for different porcesses. However, when the processes send the very first chunks the error message 'p4_error: interrupt SIGSEGV: 11; is generated.
I wonder if there is some bug in the code since the size of the chunks cannot so big. For M = 1500 it is about M*sizeof(st2) = 1500 x 140 bytes. I also tested it with N=10, and M=15 but it does not work.
Has anybody worked with huge nested structure arrays before? Is there any efficient algorithm to redistribute the results stored in the families of chunks to all the processes?

Thanks a lot,
Zoltán

The code is simple:

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define  N 10
#define  M 15


typedef struct st1 {
  double m0, m1, m2, m3;
} st1;

typedef struct st2 {
  int m0;
  double m1;
  st1 m2, m3, m4, m5;
} st2;

st2 * data;

MPI_Datatype st1_type, st2_type;

int crt_dt()
{
  int i, base;

    MPI_Datatype type1[4] =
{MPI_DOUBLE,MPI_DOUBLE,MPI_DOUBLE,MPI_DOUBLE};
    MPI_Datatype type2[6]={MPI_INT,MPI_DOUBLE};
    int blk_len[6] = {1,1,1,1,1,1};
    MPI_Aint disp[6];
    st1 s1;
    st2 s2;

    MPI_Address( &s1.m0, disp );
    MPI_Address( &s1.m1, disp+1 );
    MPI_Address( &s1.m2, disp+2 );
    MPI_Address( &s1.m3, disp+3 );
    base = disp[0];
    for( i = 0; i < 4; i++ )
        disp[i] -= base;

    MPI_Type_struct( 4, blk_len, disp, type1, &st1_type );
    MPI_Type_commit(&st1_type);

    type2[2] = st1_type;
    type2[3] = st1_type;
    type2[4] = st1_type;
    type2[5] = st1_type;

    MPI_Address( &s2.m0, disp );
    MPI_Address( &s2.m1, disp+1 );
    MPI_Address( &s2.m2, disp+2 );
    MPI_Address( &s2.m3, disp+3 );
    MPI_Address( &s2.m4, disp+4 );
    MPI_Address( &s2.m5, disp+5 );
    base = disp[0];
    for( i = 0; i < 6; i++ )
        disp[i] -= base;

    MPI_Type_struct( 6, blk_len, disp, type2, &st2_type );
    MPI_Type_commit(&st2_type);

    return 0;
}


int loc_init( int rank )
{
  unsigned int i, j;

   for( i = 0; i < N; i ++ )
    {
      for( j = 0; j < M; j++ )
    {
      (data+i*M+j)->m0 = rank + i*M + j;
      (data+i*M+j)->m1   = (double)(rank + i*M + j);

      (data+i*M+j)->m2.m0 = (double)(rank + i*M + j);
      (data+i*M+j)->m2.m1 = 0;
      (data+i*M+j)->m2.m2 = 0;
      (data+i*M+j)->m2.m3 = 0;

      (data+i*M+j)->m3.m0 = (double)(rank + i*M + j);
      (data+i*M+j)->m3.m1 = 0;
      (data+i*M+j)->m3.m2 = 0;
      (data+i*M+j)->m3.m3 = 0;
      (data+i*M+j)->m4.m0 = (double)(rank + i*M + j);
      (data+i*M+j)->m4.m1 = 0;
      (data+i*M+j)->m4.m2 = 0;
      (data+i*M+j)->m4.m3 = 0;

      (data+i*M+j)->m5.m0 = (double)(rank + i*M + j);
      (data+i*M+j)->m5.m1 = 0;
      (data+i*M+j)->m5.m2 = 0;
      (data+i*M+j)->m5.m3 = 0;
    }
    }

  return 0;
}


int main (int argc, char *argv[])
{
  int num_proc, rank, proc;
  unsigned int i, j;
  MPI_Status stat;

  /*** Initializations ***/
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &num_proc );
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  printf ("MPI proc %d has started...\n", rank);

  /* local memory allocations for the data array */
  if( ( data = (st2 *)malloc(N*M*sizeof(st2)) ) == NULL )
    {
      fprintf( stderr, "Proc %d: Not enough memory. Exit.\n", rank );
      exit(1);
    }

  /* local initializiation of the data array  */
  loc_init(rank);

  /* create user defined data type  */
  crt_dt();

  MPI_Barrier(MPI_COMM_WORLD);


  /* data transfer */
  if( rank > 0  )  /* Proc 0 does not send data */
    {
       /* send each row in the portion */
      for( i = rank; i < N; i += num_proc )
    {
      printf( "Proc %d: sending %u.th portion to proc 0.\n", rank, i );
      /* tagged by the i  */
      MPI_Send( &data[i*M], M, st2_type, 0, i, MPI_COMM_WORLD );
        }

    }
    else    /* Proc 0 recieves each portion */
    {
      for( proc = 1; proc < num_proc; proc++ )
    {
      for( i = proc; i < N; i += num_proc )
        {
      printf( "Proc %d: recieving %u.th portion from proc %d ....\n",
          rank, i );

          /* tagged by the i */
          MPI_Recv( &data[i*M], M, st2_type, proc, i, MPI_COMM_WORLD,
            &stat );

        }
    }

    }

/* MPI_Barrier(MPI_COMM_WORLD);*/
MPI_Finalize();
free(data);

}
```