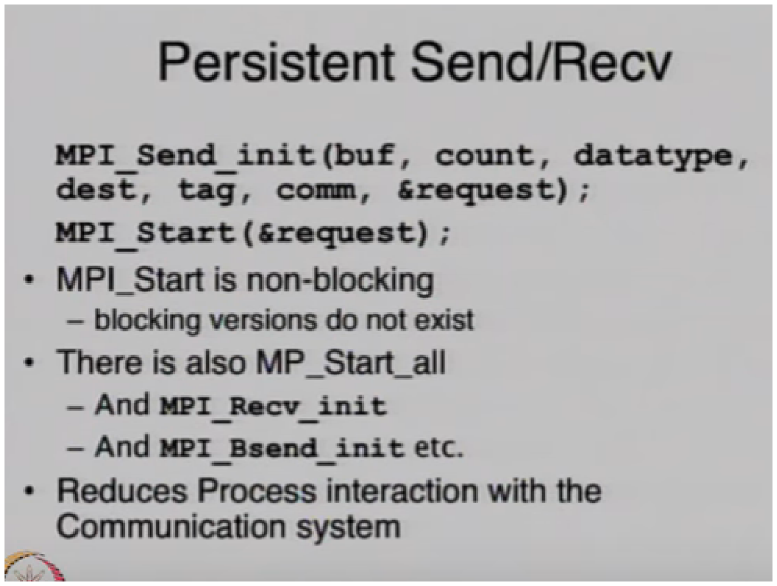
Parallel Computing
Prof. Subodh Kumar
Department of Mechanical Engineering
Indian Institute of Technology- Delhi

Module No # 04
Lecture No # 16
MPI (Contd..)

(Refer Slide Time: 00:26)



Persistent Send/Recv

```
MPI_Send_init(buf, count, datatype,
dest, tag, comm, &request);
MPI_Start(&request);
```

- MPI_Start is non-blocking
  - blocking versions do not exist
- There is also MP_Start_all
  - And `MPI_Recv_init`
  - And `MPI_Bsend_init` etc.
- Reduces Process interaction with the Communication system

Variant, where you do not necessarily need all the information. Sender provide information right in the beginning, so then the under underlying can start to set up its local tables. For example start to make connections and it is later on it is similar to non-blocking sent, where you wanted to say, here is the data I want to send, give me the handle identifier in the request variable. I am going to come back later and check weather that request is complete or not.

Here I am saying here is the piece of the parameters that I need to send. Here the buffers are the tax and the destinations. But do not send it just here, give me a request and which identifies this send request and sometimes later, I will come back and say now you are ready to send MPI. This is initially this essential to get MPI started on business of sending early, before you have to get data ready.

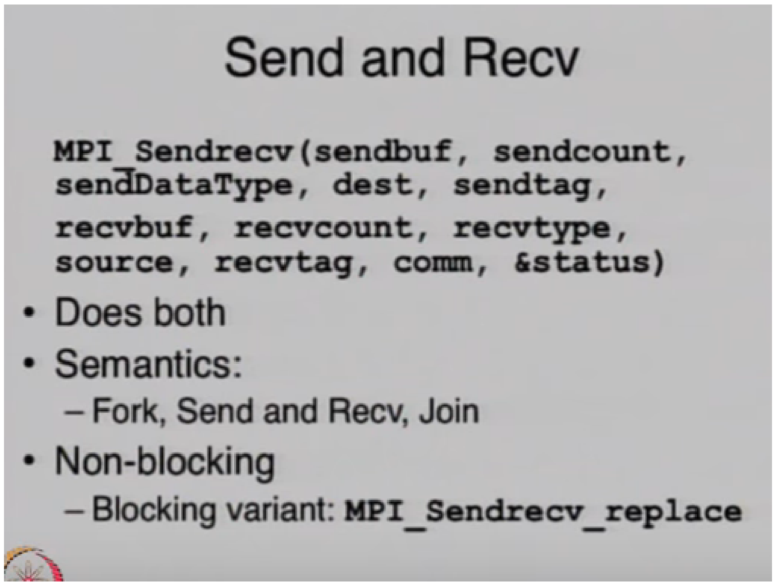
If you do MPI send, you have to make sure that buffer is complete with the data you wanted to send. In this case it does not happen, all you need to know is where you are sending it. What is the tag? Enough to get MPI sending process to begin and at later stage you can start when you are finished the number. So that at this point or the MPI process has to do mostly is just to start to send the data.

Professor student conversation starts sir it is just MPI between that it is essentially pipelining, you are saying that you get started with setting things up early and setting things. So it takes there is some computation and all it is not that much computation. It is more getting connections, making sure that socket is available or rather a port is available and things of that sort copying it to buffer that is not always done.

But yes copying it buffer here, we do not specify the data. You specify the buffer here, send it as the buffer. So the point is known in later on, you only need to say I have told you about this request. Now you can begin there is not persistent connection for each it does use persistent connection. It is reduced, but you do not know whether it there is no guarantee connection has been persistent.

Since your last sent it may also be that first time you are sending to that particular machine. So there is persistent connection but that all is hidden in a typical implementation is not keeping it persistent for too long each connection maintaining each connection requires some resources. So there will be some small number of connections you can keep persistent and if you ask for more then you just have to recycle some of that, Professor Student conversation ends.

(Refer Slide Time: 04:25)


Send and Recv

```
MPI_Sendrecv(sendbuf, sendcount,
sendDataType, dest, sendtag,
recvbuf, recvcount, recvtype,
source, recvtag, comm, &status)
```

- Does both
- Semantics:
  - Fork, Send and Recv, Join
- Non-blocking
  - Blocking variant: `MPI_Sendrecv_replace`

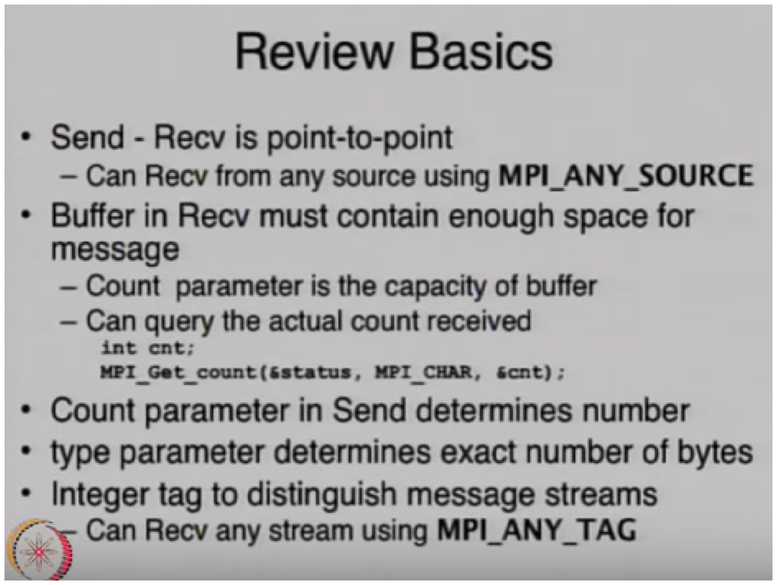
There is a send, receive variant. This is the last variant of point to point communication, we are going to talk about which basically says send this piece of data and receive a response back. It is essentially the same thing, is sent followed by a receive involve function. Again minor over it removed. In this case you have to send provide the address to the data, you are sending as well as the address to the data where you received again complete sent time from received time can be different.

The source from where you may be receiving can be different from destination you are sending. The type of data, the count of data would make sense always generally be different. It is probably the response of something and it allows basically combining that to function cross into one. Professor student conversation starts on from there do not mean. What for semantics? Semantics not actually physically doing, it is the semantics for join.

Because you may be sending to someone and receiving from someone if this send has to physically happen first and that receiver is ready to receive to match with some other send that has already happened. But it was send to happen right. So it is like this two send and receive calls are happening concurrently, it is possible then you receive, unless you are doing basic non blocking.

The receive will have to start only when send finishes. In this case it is not applied even if it is blocking, both will block independently. Professor student conversation ends. Alright non blocking variant has slightly different name, but the same

(Refer Slide Time: 07:01)



Review Basics

- Send - Recv is point-to-point
  - Can Recv from any source using **MPI_ANY_SOURCE**
- Buffer in Recv must contain enough space for message
  - Count parameter is the capacity of buffer
  - Can query the actual count received

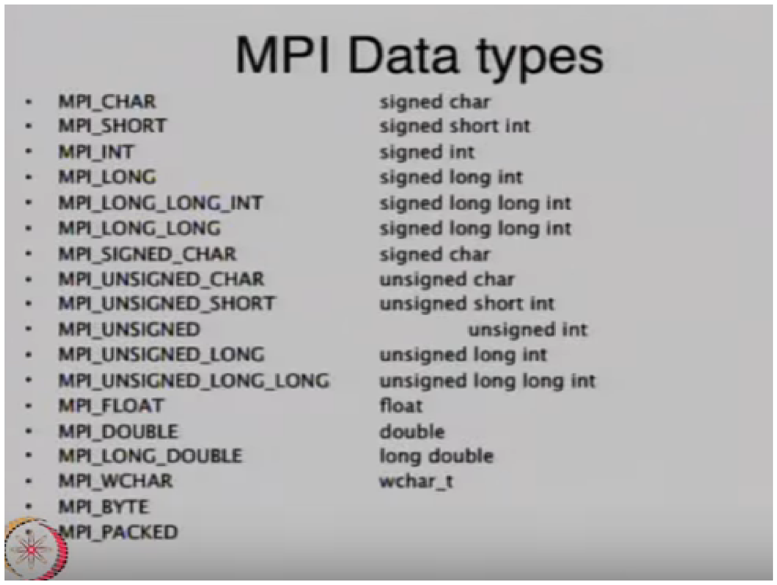
```
int cnt;
MPI_Get_count(&status, MPI_CHAR, &cnt);
```

- Count parameter in Send determines number
- type parameter determines exact number of bytes
- Integer tag to distinguish message streams
  - Can Recv any stream using **MPI_ANY_TAG**

Parameters, so basically everything we have talked about is point to point source sending to one destination at a time. It remains point to point even when you want to receive from potentially multiple sources. You will be receiving from one of them. It is like select and sockets you can receive from any source thing in the value of recipient, value of source, MPI source. Any source buffer must contain enough space.

The parameter as I had earlier mentioned the tag can also be, any tag. So you do not have to say will only receive now messages of five, three, sixty nine. You can say send me any message, send it from any source and I am going to be ready to receive it. There are these standard types that you can send.

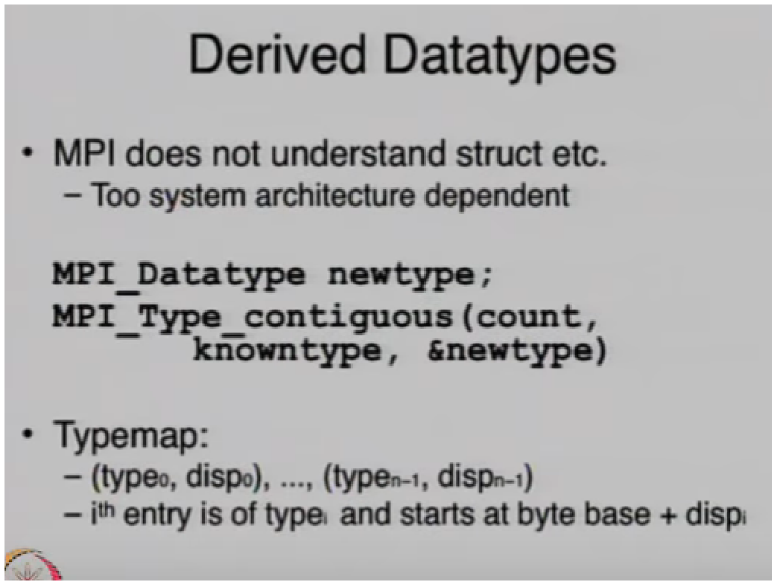
(Refer Slide Time: 08:16)


MPI Data types

MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT	signed long long int
MPI_LONG_LONG	signed long long int
MPI_SIGNED_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wchar_t
MPI_BYTE	
MPI_PACKED	

The standard C type essentially you can also create your own types. So that you do not have to bother feeling your strut on the other side. You can say here is the strut, send one copy or hundred copies. You are destructs to the destination, so the detail can be taken care by the underlying system difference. Whatever the architecture difference and it is the end here and there, so those are the standard and the ones that

(Refer Slide Time: 08:59)


Derived Datatypes

- MPI does not understand struct etc.
  - Too system architecture dependent

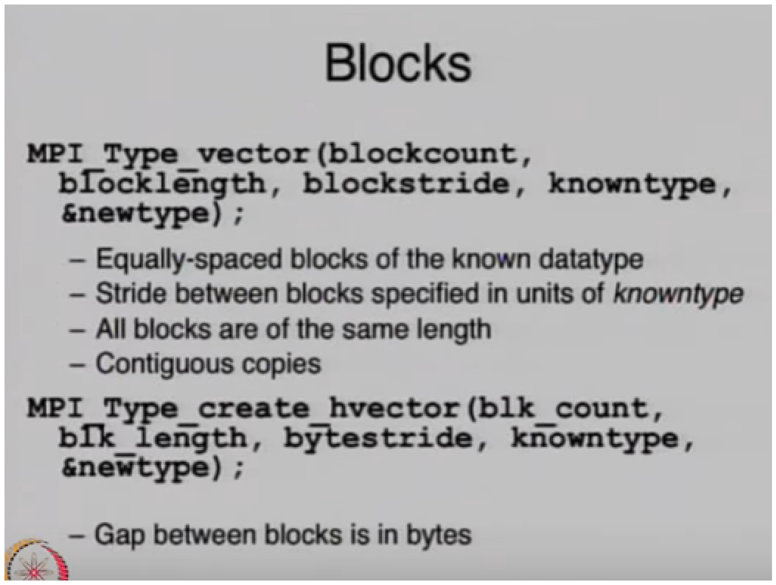
```
MPI_Datatype newtype;
MPI_Type_contiguous(count,
        knowntype, &newtype)
```

- Typemap:
  - $(type_0, disp_0), \ldots, (type_{n-1}, disp_{n-1})$
  - i^{th} entry is of $type_i$ and starts at byte base + $disp_i$

You generate are called derived data types and I am not going to go through the details of all of it. But essentially in the beginning for every strut, you may want to send you create a data type give it a name provide, how it is arranged that strut and later on simply call that derived data, type name in your send function calls. This is not the only way to do it.

But here is one example where there is MPI defined data type and you declare some types and variables which is of that type and you set that variable in variety of different ways. One is simply called this as continuous that method makes an array type for an already pre know type, that either is standard or something. You are going to say that this type is a ten integer element array of ten integer arrays of ten struts or whatever there are other variants.

(Refer Slide Time: 10:24)



Blocks

```
MPI_Type_vector(blockcount,
  blocklength, blockstride, knowntype,
  &newtype);
```

- Equally-spaced blocks of the known datatype
- Stride between blocks specified in units of *knowntype*
- All blocks are of the same length
- Contiguous copies

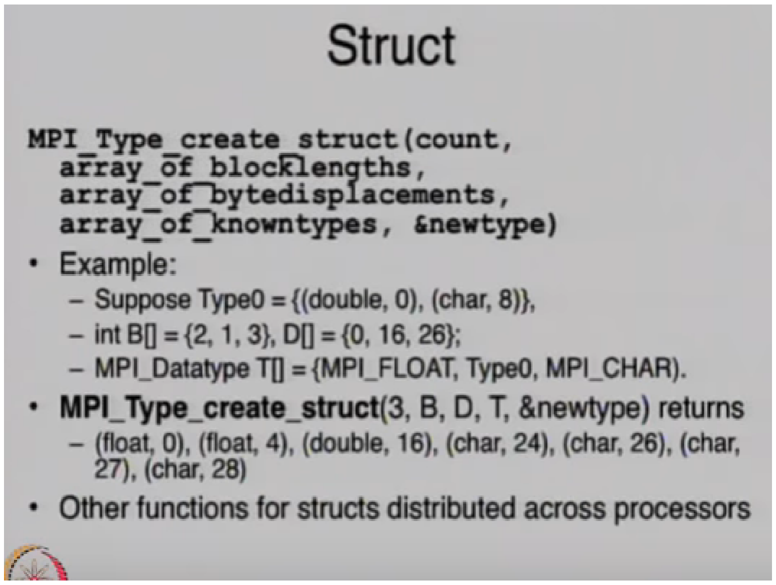
```
MPI_Type_create_hvector(blk_count,
  blk_length, bytestride, knowntype,
  &newtype);
```

- Gap between blocks is in bytes

Then again as I said you should go look up the details of it, do not need to talk about all of these details in the class. You can make a vector and in fact you can get quite creative with how your struts are locally arranged and what it looks likes to an MPI. You can create type by saying here is the strut and I say command bunch of things that I do not really care to send some other things that I do not really care to send.

You can create type only sending these this command fields and that is being done in vector and half vector variants, where you say jump so much and then pick up one more piece of data jumps. So pick up one more piece of data of a known type. We can also create blocks of such strided memory fields, there is also type strut.

(Refer Slide Time: 11:37)



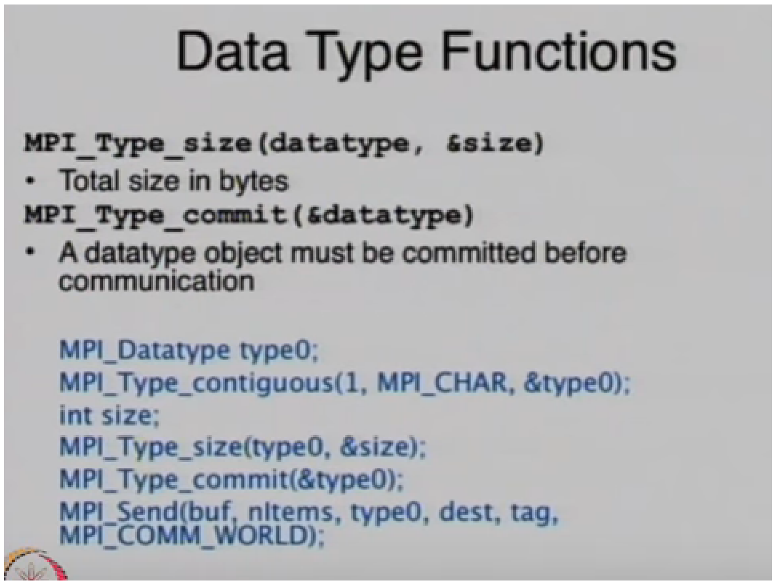
Struct

```
MPI_Type_create_struct(count,
  array_of_blocklengths,
  array_of_bytedisplacements,
  array_of_knowntypes, &newtype)
```

- Example:
  - Suppose Type0 = {(double, 0), (char, 8)},
  - int B[] = {2, 1, 3}, D[] = {0, 16, 26};
  - MPI_Datatype T[] = {MPI_FLOAT, Type0, MPI_CHAR).
- **MPI_Type_create_struct**(3, B, D, T, &newtype) returns
  - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)
- Other functions for structs distributed across processors

So you can create a strut by giving it some of the details of the strut and this will of course look at more details an MPI manual also, but all of this will be on the slides. So you can take a quick look just to see a how to get these things started.

(Refer Slide Time: 12:07)



Data Type Functions

```
MPI_Type_size(datatype, &size)
```

- Total size in bytes

```
MPI_Type_commit(&datatype)
```

- A datatype object must be committed before communication

```
MPI_Datatype type0;
MPI_Type_contiguous(1, MPI_CHAR, &type0);
int size;
MPI_Type_size(type0, &size);
MPI_Type_commit(&type0);
MPI_Send(buf, nItems, type0, dest, tag,
MPI_COMM_WORLD);
```

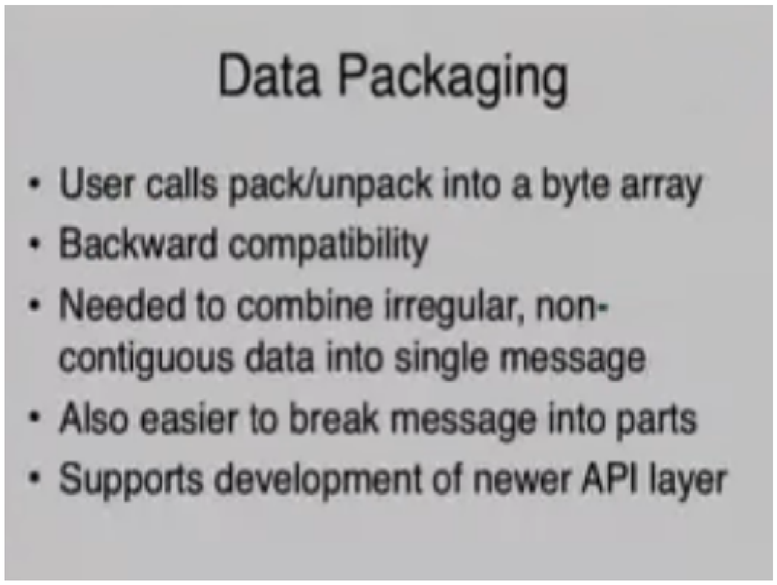
There are other variants and let me just skip all of this. I will point out this thing for most types, you need to commit them diverse, just not to create it. You commit that type before you start using it in subsequent send and receive call and this is something, professor student conversation starts problem commit is everything like that, so how compiling the code is not actually compiling the code.

You are basically storing some values side. The type says that when I am going to start, send sixty things of this type. How do you interpret the manual? So that information is stored in this type variable that you are creating and if you go back to this original example, you create MPI data type. New type which is a strut internally, it is going to store lots of information and as you may define this strut all this information is stored in there.

So it is not compiler generated, so you have said this strut means first four bytes followed by the sixteenth to twentieth byte and all that stuff. So all the all that goes into this type definition variable later on when you call send with it MPI routine will be interpreted and packed the data out of the buffer. Professor student conversation starts, sir what is coming? I do not really know commit is doing.

You probably need to tell MPI that this type is not going to change anymore or something of that sort. I am not totally said in fact MPI commit I do not believe is a part of standard either it is open MPI thing. If I am not mistaken, professor student conversation ends. There is an example of how to generate something.

(Refer Slide Time: 14:34)

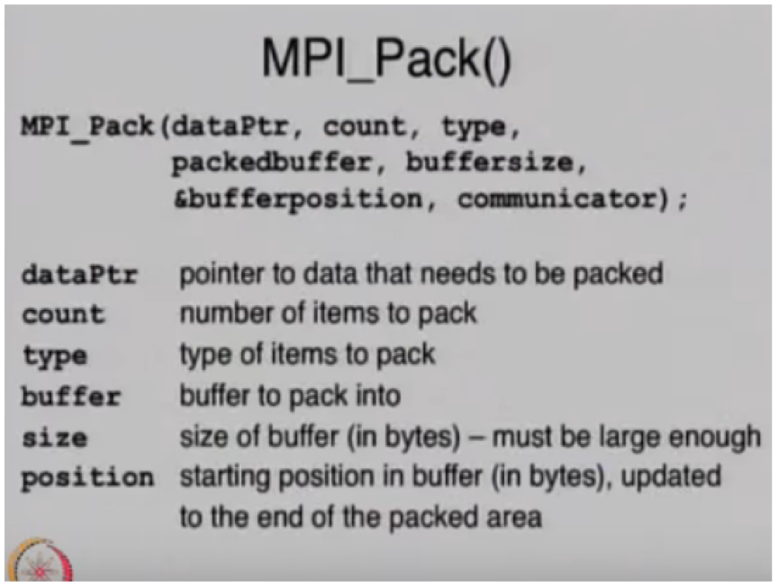


Data Packaging

- User calls pack/unpack into a byte array
- Backward compatibility
- Needed to combine irregular, non-contiguous data into single message
- Also easier to break message into parts
- Supports development of newer API layer

There are variants where you explicitly call pack and unpack. This is also true of almost communication routines including PBN that I had mentioned earlier, where you explicitly provide pack and pack routine which says, here is some list of variables pack them into the string of bytes. You send that string of bytes and on the other side you unpack it and original version of MPI standard use the packing and unpacking style, that continue to be there and it is little more generic. There is no understanding that there will be some strike. That there is some even steps you will jump to figure out. How much data to send?

(Refer Slide Time: 15:36)



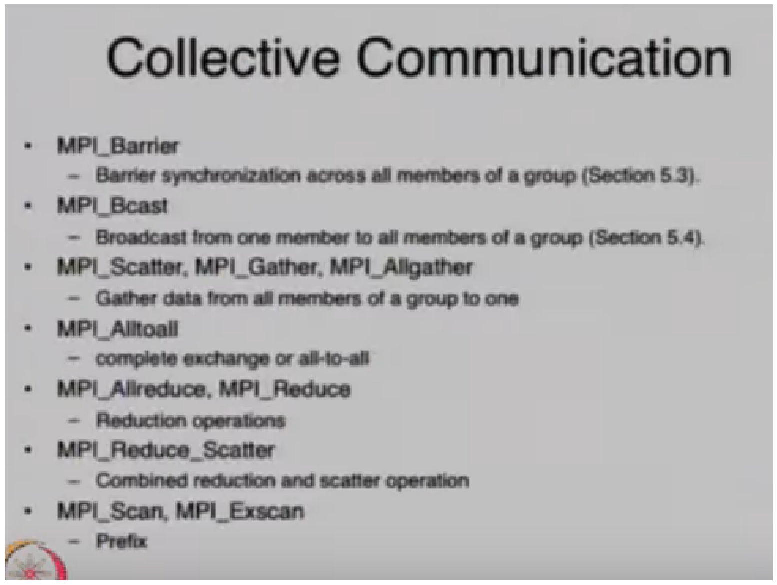
MPI_Pack()

```
MPI_Pack(dataPtr, count, type,
         packedbuffer, buffersize,
         &bufferposition, communicator);
```

dataPtr	pointer to data that needs to be packed
count	number of items to pack
type	type of items to pack
buffer	buffer to pack into
size	size of buffer (in bytes) – must be large enough
position	starting position in buffer (in bytes), updated to the end of the packed area

So pack can be a bit more generic and so there is also another and pack version for it. But it is basically saying here is data pointer. How many elements are going to sent? The type that it contains cannot create this pack buffer.

(Refer Slide Time: 16:16)



Collective Communication

- MPI_Barrier
  - Barrier synchronization across all members of a group (Section 5.3).
- MPI_Bcast
  - Broadcast from one member to all members of a group (Section 5.4).
- MPI_Scatter, MPI_Gather, MPI_Allgather
  - Gather data from all members of a group to one
- MPI_Alltoall
  - complete exchange or all-to-all
- MPI_Allreduce, MPI_Reduce
  - Reduction operations
- MPI_Reduce_Scatter
  - Combined reduction and scatter operation
- MPI_Scan, MPI_Exscan
  - Prefix

I am not going to go through packing example, so it is rather simple and not important, so now we get to non point to point communication. The examples are listed here, some of them will make sense to you by just there names, some of them will make sense to you very soon. Because you will be doing lot of MPI barrier. Everybody in the group must get there which means everybody must call this function and when everybody has called this function you can proceed.

MPI broadcast is one sending everybody in the group. MPI scatter, does everybody know, what scatter and gather operations are? Some of you know what is scatter? Professor student conversation starts protection of and even number or one, two, three, four. So when get one goes one to two right. So it is richer version of broadcast. In some ways, instead of saying here is a piece of data and send it to everybody.

It says here is an array of data, send the first to the first guy, second to the second guy, third to the third guy, fourth to the fourth guy. It is not broadcasting the same element. But itself it is broadcasting an array. But it is not sending the array to everybody, it is sending one section of array to one person. You can use it in anyways, in fact you see you will use it. So instead of sending one to one to everybody, you basically saying all in one primitive send.

So now there is no sequence involved. You are in pair for send join, typically it is the entire communicator. There is no order of recipients, but we have one. No there is no rank, you know that one is going to specific person and two is going to specific person. But then there is no order, they have to receive it. So you do not have to say that to one before, you send two matching of data is from index and rank are going to get matched.

But there is no notion that one has to receive and only once you have figured out that one has received can be sent to two. So there is no sequence among the sent. So it is like work send to all in the join. What is the blocking policy? In this as in each of them receive, I will ask you that question very soon. Let me explain all the different versions of it, different primitives. So to speak, sir the number of places in the array, why did it divide equally?

Those are some of the detail which we will talk about. But there is no higher level question to be asked. Gather is the inverse of scatter, so you are saying you produce something, instead of me receiving from you, then, I will say gather everybody results. I will provide one array, the first person result will appear in the first slot, second person in the second slot, last person in the last slot.

Even that are in general operation, because one of them you ask, which would be true even if you did it. Otherwise, other case you can see particular machine is taken, then something about that which you can do even now. You can say unblocking gather and then keep waiting and you say at some point I will decide that that machine is stuck. So all of that will apply, it is just combining the primitive into one at all.

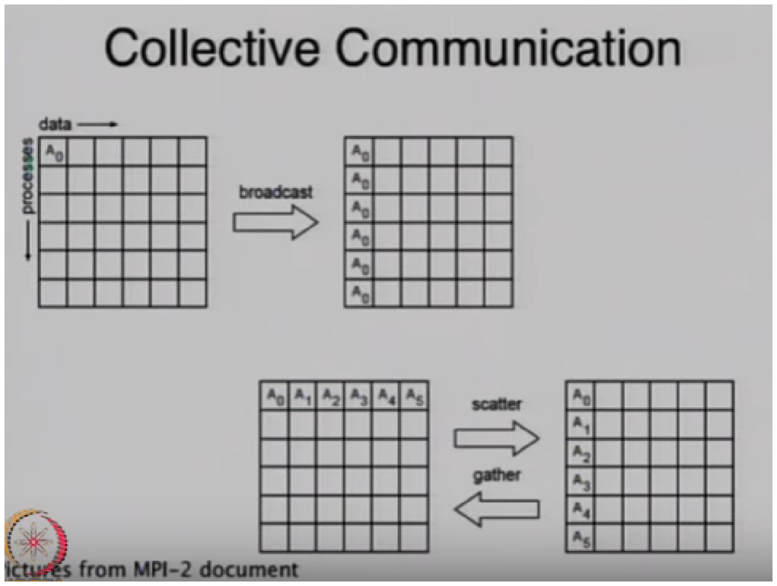
Professor student conversation ends. There MPI all gather which is gathering multiple things, so it is variant of gathers we will talk about it. But does not have to be a detailed out yet. MPI all to all is sending a whole bunch of things to everybody by everybody. So it is essentially the lots of scatter and gather happening together. MPI reduce, you should know what will reduce open reduce?

So you got everybody in at every location, have some piece of data and you combine them using some associated committed evaporator. Professor student conversation starts, if we gather up some operational you can think of it like that way gather is simply put it in the memory. Professor student conversation ends and all reduce again is just like all version of everything else where typically you reduce and get one result given to say node zero.

Reduce result is going to everybody, getting reduced. MPI reduce scatter, is reduced and then scatter MPI scan. Another thing that you will do lot. What is a scan? It is a prefix sum. What is the prefix sum? So it is not necessarily, just a sum but some operator where in reduce you may basically apply a reduction of the entire set and produce one result, which you can imagine is the result for last sum.

If you apply the same operation from N minus one, first N minus one pieces of data. Ignore the last piece of data, then you have generated a smaller reduce a sub set of smaller reduce and that becomes the result on the N minus. First you can think of everybody reducing, only reducing up to it starting from zero. Professor student conversation starts, if this is scanning gradually automatically decides upon the division based upon the number.
There is no guarantee that will be done in parallel just like for an open MPI, professor student conversation ends. Scan can be done in parallel, sounds like, let us take prefix sum, you are adding everything up to you, so everybody is is the result of the previous guy plus one more thing. Let us assume that we are adding the first one at itself, there is nothing second one takes the previous guys result and at store it on the value the third one takes the previous guys result adds to it is on value.

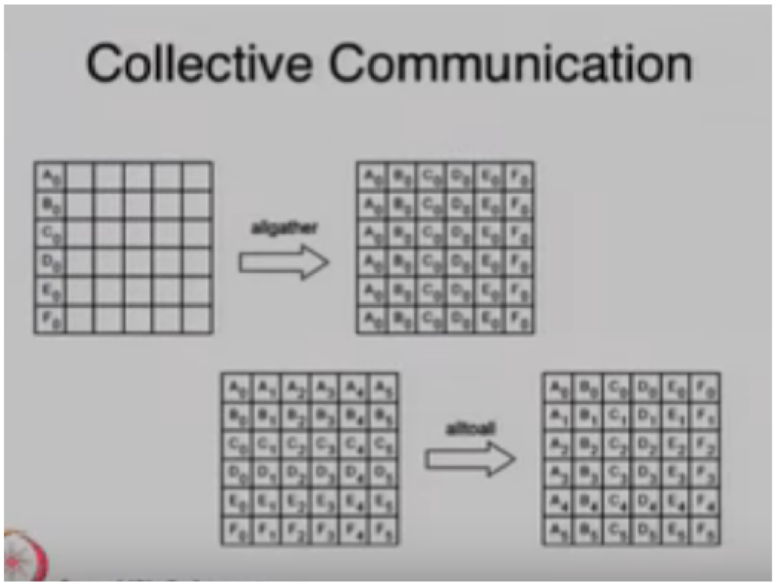
Professor student conversation starts if it is not parallel that is the effect we want. But there are other way to do it so anybody in broadcast, everybody can receive. We will come back to it, we we have talked reduce and together you can be reducing many things. That is the question I was going to ask you professor student conversation ends. Let us do barrier, first how do you do a barrier? Very easy everybody has to call it.
(Refer Slide Time: 25:54)



I have got you the slides explaining I will go through them very quickly. The other part, I have already told you. But broadcast as what you see in this table, here every processor has a row and processes or threads grow from the top row to the bottom and every processor may have array of data. So processor zero has array, that the top row and one array is the next row and so on. So broadcast basically says one piece of data and that piece of data has to go to everybody.

Every processor scatter says, I have an array of data. So processor zero has an array and that array gets distributed among all the processor. So that is scatter and the rivers is gather, everybody has one piece of data and it is all going to get collected into one array or processor zero here is all.

(Refer Slide Time: 27:05)



Collective Communication

Gather version where you got everybody in a gather. You are taking everybody, is piece of data and putting it in processor zero array. But in all gather everybody is gathering it. Processor zero is getting the gathered result. Processor one is getting the gathered result and so on. Professor student conversation starts sir, how do you call it? Same question, same answer. Can you explain this?

The gather, it is basically everybody what happens in gather? One processor is taking everybody result and putting it in one array. All gather it like whatever that processor was getting it is not divided, everybody is gathering from the one element that every had, So at the end all gather. It is not only processor zero just that had the gathered results. But everybody that is the only difference, everybody is gathering and all to all is taking it to the other dimensions.

Also all gather required scatter. Why everybody has a piece of data? So all of you have something and I say gather means you give it to me, I am going to get a collection, everybody has the assignment and you give it to me and got the list of assignments. So to that to do I have to scatter. How no scattering gather means, I will take everybody's piece of data and gather it to one pile. I was saying scattered it pair is not a pair.

There is one unique fellow, who is going to gather it and everybody's assignments gets collected into one pile in place. All gather basically, making a copy of this pile and having it in every place that is the only thing. So it is like everybody calling gather, we will talk about that shortly. We have to first figure out, how to call gather in the first place. So only then it will make sense in all gather context, not very clear how gather happen plus if you think about how gather happens?

You will think about how all the gather happens, all to all there is a complete here. Because all to all every process should have all the data, if you already had. So each processor should have a zero two s zero, then a one. All to all is essentially not quite transparent, it is all to all. I actually need to check on this. It is all to all, does not mean that all data has to go to all elements or all processes.

All to all, we all gather when that everybody was gathering one element. Now everybody starts to get different way that is at least what based on the slide. But I will verify that to make sure that is the correct interpretation of all to all. It was one direction, one gather all the other is two directional. All to all could be, I am not sure I will take a look at them. It means that everyone is accurate that has to do which is the same thing what I said earlier.

But it may be that there is scattering, but everybody's data is going to everybody, it is like all broadcast which I do not think is, but I will check on that should be a 0. This whole will be easier, only a 0 go to everybody. I think there is the right hand side probably this is everyone has everyone, no it is not just the copy of a 0, not a copy of P0. This copy is available, every person. This table is the same table as before meaning that, each row corresponds to the processor and that processor has the elements in that row to begin with.

Professor student conversation ends. So this is saying processor zero had elements A0, A1, A2, A3, A4, A5 which got scattered to everybody. But these processor, the second processes elements also got scattered to everybody. So how would broadcast work? So we will talk about the functions. Professor student conversation starts, broadcast which are there I mean in MPI or no it is not a network broadcast, it is network broadcast into communicator.

Broadcast will definitely take communicator as a parameter, but in case network broadcast, all of the first place, so it does not have to do network broadcast. In fact it would not be implemented as network broadcast, if it does not brought. It would sent on each lap, but then sending a network broadcast that means I am taking one, if I am sending a broadcast in this communicator, then there are may be checking.

If they are multiple Host on the same land it can it can optimize standard does not say anything about it is not definitely blindly doing a broadcast. It is also increasing, it has to be broadcast if it relies on all the implementation. Do you rely on communication? But that is not the standard says and India is going to provide standard communication. If MPI does not, it is going to provide to you as an application.

What is the multi cast? So that multicast is not well supported it is not available in most collections, professor student conversation ends. Let us get to the business of figuring out how this may happen at as a function? How do you call the broadcast function? Such that one fellow is going to send, everybody is going to receive. Professor student conversation starts, rank is some is that parallel.

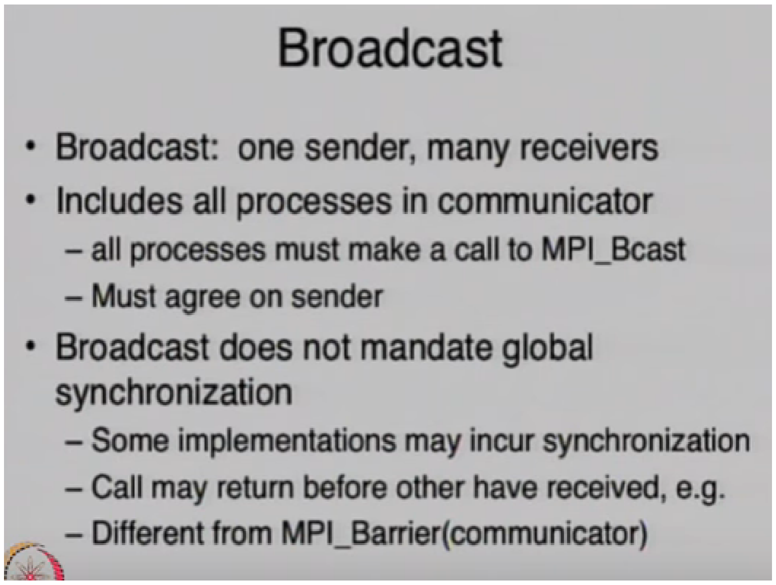
So would it work? If you said one fellow said, broadcast instead of send you are basically replacing a send the broadcast and everybody else doing the same broadcast is like you want everybody may not want using here. Actually you do that is the standard says broadcast means, everybody have to receive it because it is also synchronization not as strict as barrier. But there is loose synchronization associated with it.

But those that not received is broadcast, you send back right. So if a receiver, then you have to send to a everybody which means you basically want a folks and send in each of the folks with the folk knowing which one is trying to send to and then I receive which is not the most efficient way of doing it. Because then there is all this independent copy is going their own setups on copies in the buffer.

You may want to change the buffer that is why the broadcast function is there right. But somehow you literally need send this to happen, you do not want to be sequence. You should have processor and you also have including processor on each node for alteration that is a lot potentially.

Instead, that is why important to implement it as a primitive together and one efficient way of doing it is everybody to call broadcast instead of calling you might have thought of this as broadcast send and broadcast receive. There is no reason to separate that out, professor student conversation ends. Everybody calls a broadcast and in the broadcast.

(Refer Slide Time: 39:13)

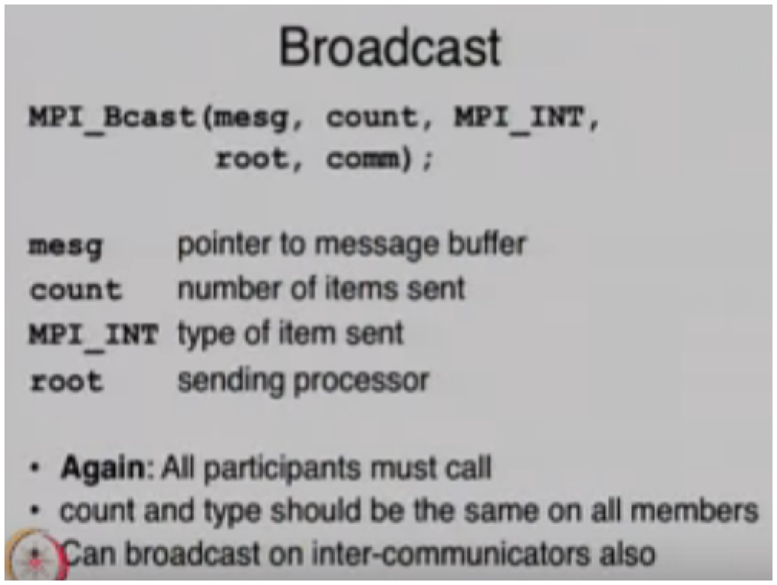


Broadcast

- Broadcast: one sender, many receivers
- Includes all processes in communicator
  - all processes must make a call to MPI_Bcast
  - Must agree on sender
- Broadcast does not mandate global synchronization
  - Some implementations may incur synchronization
  - Call may return before other have received, e.g.
  - Different from MPI_Barrier(communicator)

So if you look at, let us go through slides. First broadcast means it is going to be sent to everybody in your communicator by one of the people that is the part of this communicator and it is not exactly the same as MPI. Typically there will be synchronization involved, but it is not guaranteed that we have returned from broadcast has received it already.

(Refer Slide Time: 39:21)



Broadcast

```
MPI_Bcast(mesg, count, MPI_INT,
          root, comm);
```

`mesg`	pointer to message buffer
`count`	number of items sent
`MPI_INT`	type of item sent
`root`	sending processor

- **Again**: All participants must call
- count and type should be the same on all members
- Can broadcast on inter-communicators also

So if you return, if in barriers what happens? Everybody calls barrier and you have received it. If you have return back you can be sure that everybody has called that, if you had returned from the barrier here that guarantee is not provided. It is not necessary if you have returned from broadcast, everybody has called the broadcast function and so the broadcast takes the message primitive of course, you need to send something, you need to send so many copies of type.

You have to send broadcast is in reference to communicator. It can be a part of many communicator, specify that just like all the other point to point functions. There is also one more function which says who is going to be broadcasting eternally? You know who the broadcaster is, everybody must agree that he is the broadcaster. If everybody calls it broadcast method with that variable equal to three, then three data get send.

Everybody else buffer gets filled, everybody sending pointer, everybody is saying five ends. One of them is filled, others will get filled as a result of this operation. Professor student conversation starts, but how will that common? Everybody is agreeing to the same value, it is part of the program, you should know who receive broadcast from who. So how will that change? If it was they do not match with each other. I need to use barrier, so that for a broadcast first what we get and then there may be a voting process.

You figure out who is going to be broadcasting and then we are going to receive from that recipient. How long will it consist? If you did not have this, let us look at alternative. I think we have in fact, let me close this. We will continue this discussion, because we are going to run way over time on the result. We will do a short discussion now, but we will continue at next class also. Professor student conversation ends.

We were talking about broadcast and asked to go think about all the different ways you might do broadcast gather and the other group communication primitives. Broadcast is what we had already look at and if you just to remind you broadcast is something that everybody has to do meaning that recipient and the sender has to call and they have to be consistent respect to each other. So for example message is a pointer to some buffer, the sender has data.

When you call the sender recipients can need to allocate the space. But can have whatever they like in it will get over return and they all need to agree on the amount of data array in broadcast. You see here there is no getting the request back, there is no getting the status back which means you cannot tell how many elements actually came like you do in send and receive which is better. How many elements came?

Everybody has to agree on who is broadcasting that route and then this context of the communicator. We have already talked about the fact that groups are associated with communicators. So far the business of communicator was just to give you a local rank in your group, you said you got one, you got two, and you got three. There was no real need for communicator except to say I am sending to two did not necessarily have to think it in terms of communicator.

But now communicated became important because that is a group communication. So whoever is on that communicator has to participate in this group communication. There is a question? Professor student conversation starts, is the why is the need to have size to fix or by that is discussed those things it is not necessary that everybody expects same count of same type. But what is expected that everybody reads the entire message. So the count times, size of the data type has to be known, this is more of design decisions more than any logical.

Listen for it because anytime you allow partial returns to happen, you got to have a way to vary how much was returned and by design all return values are either success or failure. Nothing about the data which means you need to have additional way to query something. So all the receive had this will get information in this. One does not have had that the reason, it does not very strictly, very clear at the same time broadcast is the bit more synchronous.

It is not perfectly synchronized, a bit more synchronized than some receive meaning that you are expecting all the broadcast to be similar. They need understand that we are on this primitive. Professor student conversation ends. So that is the rough reason for why and status lag is not happening. The other things not here which you would have expected from send and receive tag is not here.

Basically the same reason, we are not really expected for lots of this broadcast to be in flied. This is not non blocking, you cannot say broadcast later check if your broadcast happen or not. Because these are relatively synchronous, you really do not need to say, I have got these many in flight and I need to figure out which of those messages I am reading them out. So that is also not provided again, essentially the same reason very similar reason.

What prevents you from non blocking? Is bit more complicated to implement. Professor student conversation starts, it is non blocking right, it is blocking it is not synchronous meaning perfectly synchronous like barrier. So you are not guaranteed that fellow has already called the broadcast, but there is enough guarantee that when I have returned from broadcast it has been matched. There are things that have been send or sent from my buffer.

There will somewhere in the system, so it is blocking of receiving a data for blocking synchronization for send also blocking for everybody. What are you blocking? What do you mean by when we say that you are expecting a right type of recipient and blocking? No that would be barrier. What would you mean by blocking means that I know this will happen, I can go ahead and change my buffer message.

Buffer either typically it is synchronization for example open MPI implements synchronous way. But it is not have synchronous way, it can copy this message somewhere else, copy all the information you are sending set things up and return. So before broadcast, any need of receiver before and after it I return from broadcast. Any send or receive that I made earlier it would have been an issue, that is not necessarily true.

There is guarantee in the standard that broadcast messages do not get confused with send messages meaning that, when I am broadcasting something and somebody is receiving something, this broadcast messages not getting matched with the receive. It is only going to get matched with the broadcast one and other thing is messages are remaining or not what I send first to a given recipient will reach there first.

Beyond that there is no other, but are the broadcast messages are ordered between the rest of the pair, between the barriers and typically broadcast messages may be implemented as loop and send. Because in the standard there is no guarantee for what kind of implementation one make sure and that is done in some implementation. But the idea is, you can have more efficient broadcast. For example if you have largest size of group, then some kind of log replace broadcast send to few people, they send few more.

So you can get there much faster. Sir we tell so that will change the head of the tag to receive the first message. So the tag comes early, so it is does not have to check for the time. You just want to know whether your message is being received or not then that is enough to say that this is the receive recipient. You have sent or if you had a wild card then you do not match it. This is the tag you have sent if you had a wild card.

Then you do not match it, but when you are testing whether your message has been received then that message, what I am asking is when I am go for a particular tag. So when I asking whether you will take my buffer or not it only change the first, it will change the tag also for that there is test whether your request is completed or not. Probe is not for completion, probe is like an early interpretation.

So you do not have to look, what are the messages? You partially look up for the tag, how you want to assess, whether probe will look up for all the messages. So there is that one way, it might implemented. There is no need for that, there be a buffer where all the messages are, for example you have set receive provided above the MPI could directly put it in your buffer. So there is no ordering among buffers at least from the standards point of view. There is no first message in the buffer, professor student conversation ends.

There is also question about, what if I want to broadcast and what everybody wants prove it? I will just say broadcast and whoever wants to read it. Why is that not easy thing? Why is there need for matching? So one question is how does the system know whether you are going to read or not? Like the system infinitely keep all broadcast messages until everybody has read it. There is no way to know.

So either you will say subscribing to this message and that means if people are not subscribed to it, then do not give it back which is essentially the same thing as subscribe and read versus just read. What about blocking versus non blocking? Professor student conversation starts, I think change the blocking versus should be what that the addition of all the process name it should not collapse broadcasting. No, so question is what would be required if you had to provide a support for non blocking broadcast?

Assuming that the everybody is expecting to read it for every broadcast. Everybody in the group supposed to read it. There is no tag, that is one issue. Now those two are kind of dependent on each other. You add tag rank of the recipient. Rank of the recipient is not there meaning, how will we ensure that it is non blocking?

So if you ensure that non blocking is that you immediately return. I mean that when you say broadcast, it takes down the construction that there is broadcast on this communicator and it returns and then you have to come back and query that broadcast succeeded or not. So request will turn to you just like non blocking reeds send and receive buffer. Somehow broadcast suffering, because let us say I am a receipt of the broadcast, then I have not really reached the broadcast instruction here.

But the messages are received, so there should be some buffering in the store which is going to happen send and receive also. Then there should be basically, which is not right. There should not be it's not there currently, that would be needed. Why it is not? Why is it any harder? Then just send and received it may be harder. Because we need to purpose and I receive making sure that everyone has same.

So what we have? So there is increased buffering requirement send receive messages also require when you are sending to somebody and that somebody has not come to the received call. Yet there is some buffering going to happen provided. But now everybody has to come to this receive meaning, now everybody has to come to a broadcast. Now you still want to maintain that good communication.

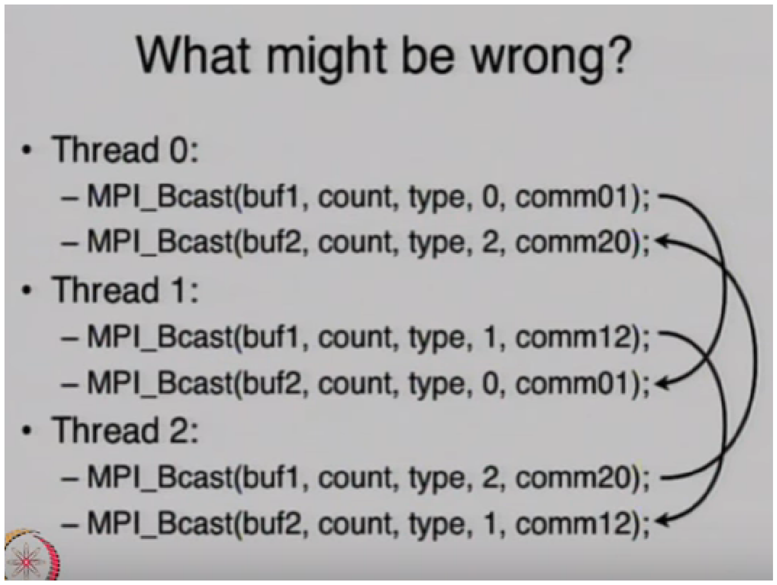
Do not get confused with point to point communication, so broadcast you do a bunch of broadcast which are in the flight and corresponding broadcast needs to get matched in sent and received. You are send good match with the first receiver from whoever and receive would match the first send from whoever. But now everybody broadcast must match to be kept in order.

So I make a three broadcast cost, somebody else makes three broadcast calls which gets matched which requires a bit. What if you have the same time? Also tagging would help when you want to differentiate one type of broadcast from another type broadcast. But you generally broadcasting stream of messages on the same time. The same probe multiple send, so if we have three send, the same recipient, the same time tag, the same problem, you do have the same problem.

But it is a point to point one and you are going as far. As the that is the programmers problem, at that point programmer has to make sure that if I wanted this send to match this receiver, I somehow ensure it system is able to meet some systems, problem is easier. But if you doing it for broadcasting, these two problems becomes harder. Because everybody’s broadcast must be matched.

There is array semantics there is added semantics for that. But why do not we change it with one to one group by sending recipient processes. There has to be m point, so let us take another example, I allow multiple broadcast to happen in first broadcast. Processor zero was the route, second broadcast processor one was the route, third broadcast processor two was the route. Then we have to make sure that zeroth route has to be called.

(Refer Slide Time: 59:39)



What might be wrong?

- Thread 0:
  - MPI_Bcast(buf1, count, type, 0, comm01);
  - MPI_Bcast(buf2, count, type, 2, comm20);
- Thread 1:
  - MPI_Bcast(buf1, count, type, 1, comm12);
  - MPI_Bcast(buf2, count, type, 0, comm01);
- Thread 2:
  - MPI_Bcast(buf1, count, type, 2, comm20);
  - MPI_Bcast(buf2, count, type, 1, comm12);

Everywhere first, otherwise there is no matching, there is deadlock once route has to be called everywhere and. So let us look at this example, oh wait this is non blocking. This is for synchronization, this example does not quit answer that question. We have zero, these are communicators, zero one is the group for the communicator, zero and one and for thread number zero and one the group contains those two threads on two zero against thread looses zero. New broadcast always within a communicator.

But we will come to this is a deadlock, because synchronization. But it also even if it were non blocking, it would require the similar level of synchronization or reordering inside the system. Let us in fact see, what is going to happen? Are going to broadcast in at two places, two groups. In each of the threads zero said, there is broadcast on form zero one, which is the first. I only see in my, so the first broadcast of thread zero is going to have to be matched another broadcast with the same route in the same communicator.

So that is going to get matched with this one. What is the communicator? Lost by communicator is essentially the group within which communication happens. So far we have been using world as a communicator, that is we have created a member of that communicator. But you can create some news. So this is of thread one broadcast messages, sent only to thread one.

Because thread zero is already here and in the meantime thread one has not called this because it earlier called another broadcast which gets matched with thread two and that communicator on one, two. But it cannot get matched yet, because thread two had earlier called another broadcast that got matched with thread 2, 0 with communicator 2, 0 on thread 0. But that cannot be called yet because the first broadcast does not matched yet.

So if it is coming, it is synchronous mean this cannot received. So whoever is blocking, but why does the same probe sender blocks for many reasons. The only thing that as a user you are guaranteed is, if a blocking call returns then your call has been made by that, you mean you are actually everywhere in a group. It does not guarantee you anything of that sort. So then I am not able to understand why thread 0?

First broadcast will be for what it is saying it will be made for match. If it is synchronous, so how can I know that it has matched all the plane within it is as well as synchronous. You are saying synchronous so likewise well barrier is synchronous, you are not required to provide barrier facility in broadcast, that does not mean that system will make sure that you do not synchronize system.

It may not synchronize. So if it becomes a double heart, double user because it can be as bad as a synchronous and on the other hand when you rely on synchronous way it is not provide. So we are taking the worst case of it yes standard that is not specifically that it could have happened. No it is possible for that certain circumstance. So does not even true that it is always going to have happen for a given implementation.

Open MPI does this but even if open MPI will return before actually synchronizing with the matching broadcast it is not guaranteed that it will return to you. In this case because there may be this buffer, maybe full. So it cannot return until it has cleaned out. Your buffer cannot clean out your buffer, because it can copy it into your buffer and it cannot copy into your buffer it cannot copy into its own buffer.

Because it is waiting for some other thread to get matched and empty some space. So just because it is able to return does not mean this situation, when you get this part it was in a position to return. So what synchronization? It is to receive all the buffer. There is no synchronization been provided by the standard. There is synchronization, weak synchronization says that when all of them are going to make broadcast call.

So this is going to get match with them and so that that is that level of synchronization that they would get matched with all of them. You do not know when that is going to happen. You have seen there are two thread calls, because of blockers. But other one does broadcast will be matched. But then what this is normal? But send receive is point to point, but here you have said something about the entire thing.

So send receive is also weak synchronization. So not straight, so you think I should synchronize? No I mean I thought that there was something that resynchronization mean, I had said that earlier also. For send and receive here, there is weak synchronization for the group. There is between the points you have created translation. So many in this case that does not have to synchronizing in this case it is very much like send and receives.

So it ends point or something what becomes synchronic in this getting a route. There is also one zero effectively right. But that is not necessarily, if you implemented the broadcast using a bunch of receives, you would probably get a slightly different behavior from the implementation. If you implemented broadcast as something that most synchronized, which is what most implementation do, meaning that when you have returned there is no standard that guarantees that anything has happened on the other end.

But on most implementation something would actually happen at the other end in open MPI. That is the case the parameter indicates, the group. So how do we write the parameter? You create a communicator, so that that is the constant it is like 0 file descriptive. 0 means something it is like that a constant communicator which exist in the MPI is pre built communicator which automatically is filled in with the members that you create in the group.

Any other communicator you have to physically call functions to build alright. So question about broadcast why is it that you have to say, I want to only receive broadcast from routs number zero. So in send in receive you could have said any sender, any tag. Here you have to be very precise and this is another reason why it is little bit more synchronized then receive. But why cannot we say because everybody is calling broadcast, nobody is calling send or receive.

So need somebody to the chosen as, so what if good? I say let us do a broadcast which is actually in asymmetric one, sends and some receive. So why should not I just say, we cast sent, we cast receive. So we cast send would be automatically would means send to everybody. We cast receive, we cast from everybody.

Why do we need a new route? Or this group mechanism, here same function is called by everybody with some parameters meaning different things at different places. Professor student conversation ends. Let me stop here take just a quick break and continue the same question after the break.