COL380

Introduction to
Parallel & Distributed Programming

Agenda

- MPI non-blocking
- MPI datatypes
- MPI collectives

Message Semantics

- In order (per pair and tag)
  - ➡ Multi-threaded applications need to be coordinate
- Progress
  - ➡ For a matching send/Recv pair, at least one of these two will complete
- Fairness not guaranteed
  - ➡ A Send or a Recv may starve because all matches are satisfied by others
- Resource limitation can cause deadlocks
- Ready/Synchronous sends requires the least resources
  - ➡ Also used for debugging

Example

If (rank == 0):

```
Send(sbuffer0, to 1);
Recv(rbuffer0, from 1);
```

match

match

else:

```
Send(sbuffer1, to 0);
Recv(rbuffer1, from  0);
```

Deadlock
if neither send can copy out its sbuffer

int MPI_Isend(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)

int MPI_Irecv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

Non-blocking calls

Returns even before buf copied out; caller must not use.

Example

If (rank == 0):

MPI_Isend(sbuffer0, to 1);

MPI_Irecv(rbuffer0, from 1);

Wait for earlier calls to finish

Will not deadlock

else:

MPI_Isend(sbuffer1, to 0);

MPI_Irecv(rbuffer1, from 0);

Wait for earlier calls to finish

Detecting Completion

- MPI_Wait(&request, &status)
  - ➡ status similar to MPI_recv
  - ➡ Blocks as per the blocking version's semantics
    - Send: message was copied out, Recv was started, etc.
    - Recv: Wait for data to fill
  - ➡ Request is freed as a side-effect
- MPI_Test(&request, &flag, &status)
  - ➡ Non-blocking poll
  - ➡ flag indicates whether operation is complete
  - ➡ Request is freed as a side-effect

Use MPI_Request_get_status to retain request
Later MPI_Request_free

Subodh Kumar

Detecting Completion

- **MPI_Wait**(&request, &status)
  - ➡ status similar to MPI_recv
  - ➡ Blocks as per the blocking version's semantics
    - Send: message was copied out, Recv was started, etc.
    - Recv: Wait for data to fill
  - ➡ Request is freed as a side-effect
- **MPI_Test**(&request, &flag, &status)
  - ➡ Non-blocking poll
  - ➡ flag indicates whether operation is complete
  - ➡ Request is freed as a side-effect

Also see:

MPI_Waitany, MPI_Waitall, MPI_Waitsome

MPI_Testany, MPI_Testall, MPI_Testsome

Subodh Kumar

- Send - Recv is point-to-point
  - ➡ Call-to-call matching
  - ➡ Integer tag to control matching
  - ➡ Wildcard matching: MPI_ANY_SOURCE and MPI_ANY_TAG
- Recv buffer must contain enough space for message
  - ➡ Receiving fails otherwise
  - ➡ Can query the actual count received (MPI_Get_count)
    - Send determines the actual number sent
  - ➡ type parameters determines data structure ↔ message buffer copying

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	

Objects of type MPI_Datatype

Subodh Kumar

- MPI does not understand language's layout (struct, e.g.)
  - ➡ Too system architecture dependent

MPI_INT, MPI_FLOAT ..

- Typemap:
  - ➡ (type_0, disp_0), ..., (type_n, disp_n)
  - ➡ i^{th} entry is of type_i and starts at byte base + disp_i

- MPI does not understand language's layout (struct, e.g.)
  - ➡ Too system architecture dependent

MPI_INT, MPI_FLOAT ..

- Typemap:
  - ➡ (type_0, disp_0), ..., (type_n, disp_n)
  - ➡ i^{th} entry is of type_i and starts at byte base + disp_i

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

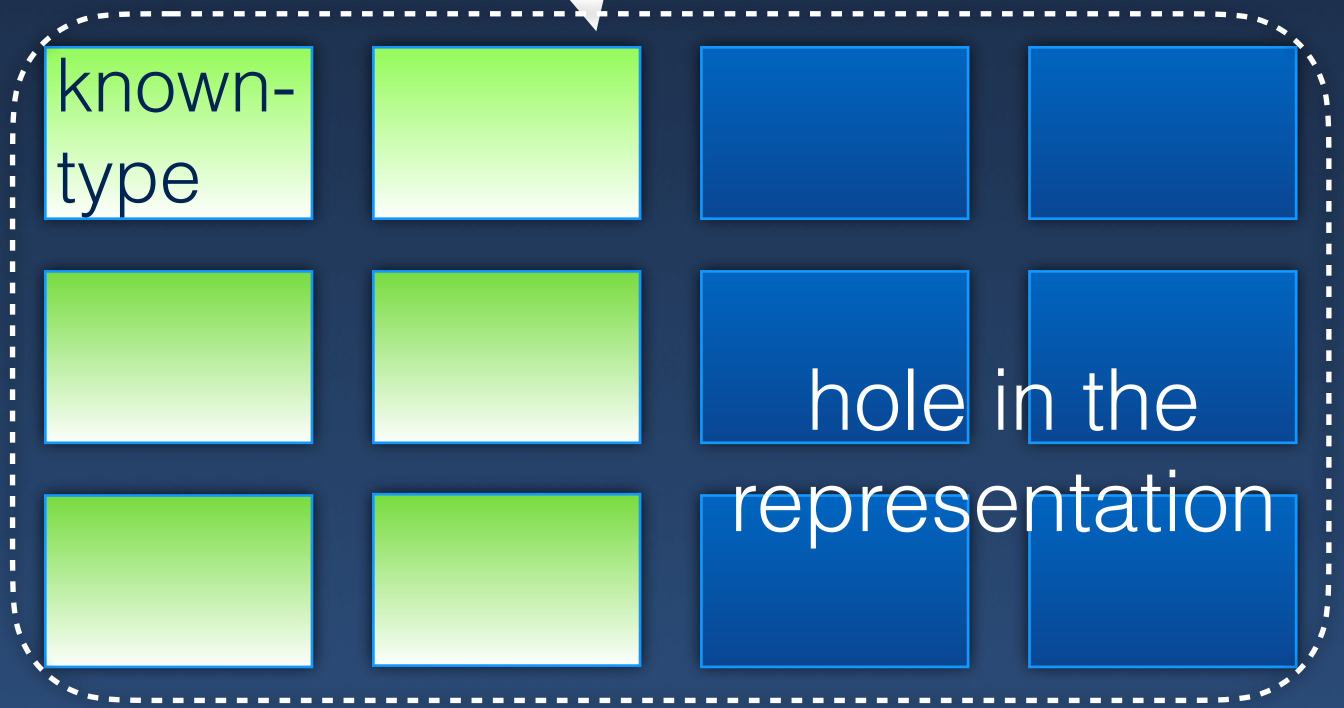
Blocks

- Equally-spaced blocks of the known datatype

3 2 4

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

  - Assume contiguous copies of 'knowntype'
  - Stride between blocks specified in units of knowntype
  - All picked blocks are of the same length



(row major)

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

Gap between blocks is in bytes

5 2,1,1,1,2

- MPI_Type_indexed(count, array_of_blocklengths,
  array_of_offsets, knowntype, &newtype);

  0,4,6,8,10

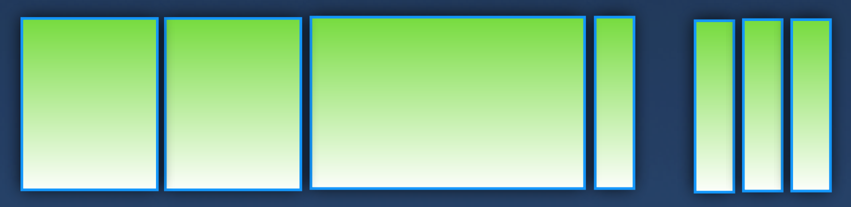
  - ➡ Blocks can contain different number of copies
  - ➡ And may have different strides
  - ➡ But the same data type

Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_byteoffsets, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)},
    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26};
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

Struct

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_byteoffsets, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)}, 
    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26}; 
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

- MPI_Type_create_struct(count, array_of_blocklengths, array_of_byteoffsets, array_of_knowntypes, &newtype)
  - ➡ Example:
    - Suppose Type0 = {(double, 0), (char, 8)}, 
    - int BL[] = {2, 1, 3}, Disp[] = {0, 16, 26}; 
    - MPI_Datatype Typ[] = {MPI_FLOAT, Type0, MPI_CHAR)
  - ➡ MPI_Type_create_struct(3, BL, Disp, Typ, &newtype):
    - (float, 0), (float, 4), (double, 16), (char, 24), (char, 26), (char, 27), (char, 28)

MPI_Type_get_contents(..)

Data Type Functions

- MPI_Type_commit(&datatype)
  - ➡ A datatype object must be committed before communication
- MPI_Type_size(datatype, &size)
  - ➡ Total size in bytes
- MPI_Type_get_extent(datatype, &beg, &extent);
- MPI_Type_create_resized(datatype, beg, extent, &newtype);
- MPI_Get_address(data, &Address[0]);
- MPI_BOTTOM

Data Type Functions

- MPI_Type_commit(&datatype)
  - ➡ A datatype object must be committed before communication

- MPI_Type_size(datatype, &size)
  - ➡ Total size in bytes

```
MPI_Datatype atype;
MPI_Type_contiguous(4, MPI_CHAR, &atype);
int asize;
MPI_Type_size(atype, &asize);
MPI_Type_commit(&atype);
MPI_Send(buf, nItems, atype, dest, ..);
MPI_Recv(...);
```

- MPI_Type_get_extent(datatype, &b
- MPI_Type_create_resized(datatype, beg, extent, &newtype);
- MPI_Get_address(data, &Address[0]);
- MPI_BOTTOM

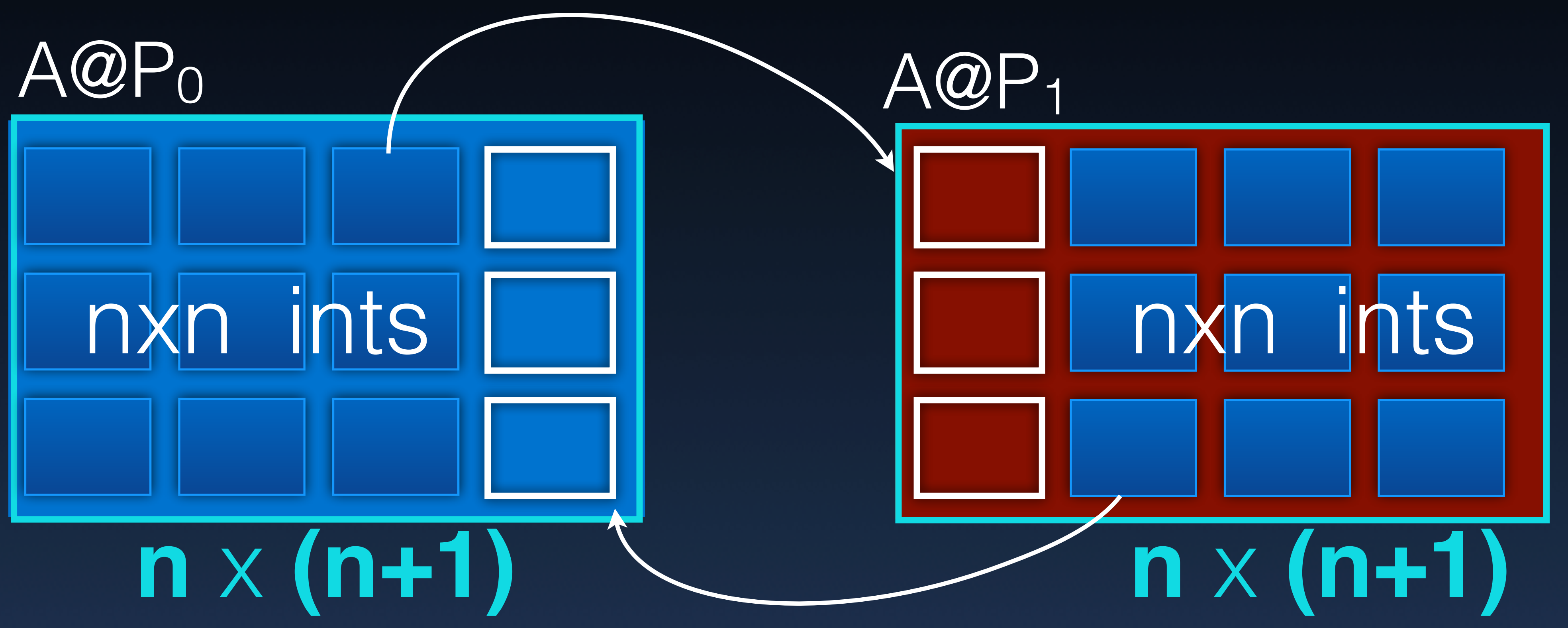
Derived Type Example

```
sendParticles(struct Particle particle[], int N):
    MPI_Datatype Particletype;
    MPI_Datatype types[3] = {MPI_INT, MPI_DOUBLE, MPI_CHAR};
    int blockcount[3] = {1, 6, 7};

    /* compute displacements of structure components */
    MPI_Aint disp[3];
    MPI_Address(particle, disp);
    MPI_Address(particle[0].d, disp+1);
    MPI_Address(particle[0].b, disp+2);
    for (int i=2; i >= 0; i--) disp[i] -= disp[0];

    MPI_Type_struct(3, blockcount, disp, types, &Particletype);
    MPI_Type_commit( &Particletype);
    MPI_Send(particle, N, Particletype, dest, tag, comm);
```

```
struct Particle
{
   int class;        // particle class
   double d[6];  // particle coordinates
   char b[7];      // some additional info
};
```

Subodh Kumar

Data Transfer




```
MPI_Status status;
MPI_Datatype column;
MPI_Type_vector(n, 1, n+1, MPI_INT, &column);
MPI_Type_commit(&column);
if(rank == 0) {
   MPI_Send(A+n-1, 1, column, 1, tag, MPI_COMM_WORLD);
   MPI_Recv(A+n, 1, column, 1, tag, MPI_COMM_WORLD, &status);
}
if(rank == 1) {
   MPI_Recv(A, 1, column, 0, tag, MPI_COMM_WORLD, &status);
   MPI_Send(A+1, 1, column, 0, tag, MPI_COMM_WORLD);
}
```

Collective Communication

- MPI_Barrier
  - Barrier synchronization across all members of a group
- MPI_Bcast
  - Broadcast from one member to all members of a group
- 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_Reduce, MPI_Allreduce,
  - Reduction operations
- MPI_Reduce_Scatter
  - Combined reduction and scatter operation
- MPI_Scan, MPI_Exscan
  - Prefix

Barrier

- Synchronization of the calling processes
  – the call blocks until all of the processes have placed the call

```
MPI_Barrier(comm);
```

Broadcast

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

pointer on all

number & type

identified sender

can be intercommunicator

- All participants must call, match by comm and root
- No implicit synchronization

Broadcast

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

- mesg: pointer on all
- count, MPI_INT: number & type
- root: identified sender
- comm: can be intercommunicator

- All participants must call, match by comm and root
- No implicit synchronization

	Data index 0-2		
rank A	A0	A1	A2
rank B	?	?	?
rank C	?	?	?
rank D	?	?	?

→

	Data index 0-2		
A	A0	A1	A2
B	A0	A1	A2
C	A0	A1	A2
D	A0	A1	A2

- Difference in usage of blocking and non-blocking send
- Making MPI types suitable for data transfer
    - ➡ Types include ‘untransferred’ holes
- Introduction to group-collective calls