Allocating receive & Freeing send

Zero-copy ownership-passing point-to-point communication

Simple Exchange Pattern

```
MPI_Alloc_mem(size,info,buf);
produce(size,buf);
MPI_Send(buf,size,..);
MPI_Free_mem(buf);
```

DATA COPY

```
MPI_Alloc_mem(size,info,buf);
MPI_Recv(buf,size,..);
consume(size,buf);
MPI_Free_mem(buf);
```

Real programs amortize away the allocation and free calls by reusing bounce buffers, but is this the right thing to optimize away?

Simple Exchange Pattern

?

produce(size,buf);

MPI_Xsend(buf,size,..);

?

OWNERSHIP PASS

?

MPI_Xrecv(buf,size,..);

consume(size,buf);

?

Is there a way to optimize away the data movement?

Simple Exchange Pattern

MPI_Alloc_mem(size,info,buf);

produce(size,buf);

MPI_Fsend(buf,size,..);

~~MPI_Free_mem(buf);~~

OWNERSHIP PASS

~~MPI_Alloc_mem(size,info,buf);~~

MPI_Arecv(buf,size,..);

consume(size,buf);

MPI_Free_mem(buf);

Instead of freeing the send buffer,
allow the receiver to take ownership of it.

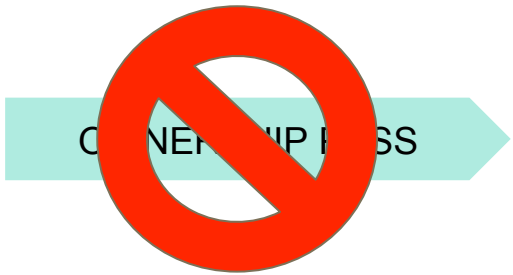
Simple Exchange Pattern

MPI_Alloc_mem(size,info,buf);

produce(size,buf);

MPI_Fsend(buf,size,..);

~~MPI_Free_mem(buf);~~



~~MPI_Alloc_mem(size,info,buf);~~

MPI_Arecv(buf,size,..);

consume(size,buf);

MPI_Free_mem(buf);

This pattern still works for distributed memory,
and should be no worse than the existing approach.

Allocating receive for distributed memory

Squyres and Goodell previously proposed allocating receive to handle NUMA/ NUNA (buffer “close” to NIC) issues.

Implementation can (but is not required to) give the user the eager buffer, which eliminates a data copy in that scenario (it can reclaim it in Free_mem).

Eliminate Probe > Get_count > Alloc_mem > Recv pattern to receive unknown-size buffer (Dinan).

Other optimizations exist, because giving implementation more information/ control is always a good thing.

Limitations

Not easy to do this with default heap manager (e.g. malloc/free).

May need user to specify info keys to Alloc_mem in order to get interprocess shared-memory buffers.

Freeing *part* of buffer is hard/impossible, so messaging buffers cannot be part of the working buffers - how can I free a single face of a 3D cube?

> Doesn't this force a memory copy into/from dedicated message buffers or sub-optimal access patterns during the produce/consume loops?

If MPI processes are OS threads, ownership passing is trivial (prior art).