2️⃣

MPI 1

Files	PP_MPI1.pdf
Link	
Status	Done
Type	Lecture

Questions

- [ ] Check my understanding of the difference between a group and communicator

Intro

- Message Passing Interface, helps us run parallel services **using the same program and code, this is called Single Program Multiple Data (SPMD)**
- Specific behaviour is done using programmatically the rank of the process, usually a "master" with rank 0 (but not necessarily)
- Returns 0 if successful or aborts if there is an issue
- Must initialise MPI and finalise
- Groups vs communicators: groups contain ordered processes from 0 to X, and a communicator is made for the group to enable us to call MPI functions on it, so you decide which processes you want in a group, a processes can belong to any number of groups, therefore by extensions a processes can belong to any number of communicators

MPI Data Types

MPI Type	C Type
MPI_CHAR	signed char
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED	unsigned int
MPI_FLOAT	float
MPI_DOUBLE	double

Rules for sending/receiving

- Sender/receiver specify valid destination/source
- Same communicator
- Matching tags & data types
- Receiver buffer is large enough (could also cause problems like buffer overflow)

Example Receiving Message Unknown Size

Receiving message of unknown size

Commands

MPI Commands

Aa Function definition	Notes	Diagram
MPI_COMM_WORLD	Includes all started processes	
int MPI_Comm_size(MPI_Comm comm, int *nprocs)	Number of processes within a group	
int MPI_Comm_rank(MPI_Comm comm, int *myrank)	Gives this process' rank	

Function definition	Notes	Diagram
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)		
MPI_Recv(void *buf, int count, MPI_Datatype datatype, int src , int tag, MPI_Comm comm, MPI_Status *status)		
MPI_ANY_SOURCE	Receive from any source, actual source in status variable (int src = status.MPI_SOURCE)	
MPI_ANY_TAG	Receive with any tag, actual tag in status variable (int tag = status.MPI_TAG)	
MPI_Get_count(*status, mpi_datatype, *count)	Get message size	
int MPI_Probe(int source, int tag, MPI_Comm comm, MPI_Status *status)	Returns only after a matching message has been found	
int MPI_Barrier(MPI_Comm comm)	All processes must reach and wait, usually used for debugging	
double MPI_Wtime(void)	Number of seconds representing elapsed wall-clock time since some time in the past, all the processes have the same start wall time so delta should give how long a process takes	
int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)	All processes must call, one process sends the same message to all others	
int MPI_Scatter(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)	Must be divisible nicely, or add padding, there is another function that takes care of that nicely.	

Function definition	Notes	Diagram
int MPI_Gather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount,MPI_Datatype recvtype, int root, MPI_Comm comm)	Opposite of scatter	
int MPI_Reduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)	Gather with an arithmetic operation	
int MPI_Allreduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)	Same as reduce but all processes have the result	
int MPI_Iprobe(int src, int tag, MPI_Comm comm, int* flag, MPI_Status* status)	(From second lecture) Non-blocking flag: non-zero, if there is a matching message, status: source, tag and size of the message	