MPI

```
int MPI_Init(int *argc, char *argv[]);

int MPI_Comm_size(MPI_Comm comm, int *size);

int MPI_Comm_rank(MPI_Comm comm, int *rank);

int MPI_Finalize();

int MPI_Barrier(MPI_Comm comm);
```

SEND / RECEIVE

```
int MPI_Send(const void *buf, int count, MPI_Datatype datatype, int dest,
  int tag, MPI_Comm comm);

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,
  MPI_Comm comm, MPI_Status *status);

int MPI_Sendrecv(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
  int dest, int sendtag,     void*recvbuf, int recvcount, MPI_Datatype
  recvtype, int source, int recvtag, MPI_Comm comm, MPI_Status *status);

int MPI_Isend(const 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);

int MPI_Test(MPI_Request* request, int* flag, MPI_Status* status); // flag is true if
  the request was ok!
```

COMMUNICATOR

```
MPI_Comm newComm;

int MPI_Comm_create(MPI_Comm comm, MPI_Group group, MPI_Comm *newComm);

(for SPLIT) MPI_Comm newComm;

int MPI_Comm_split(MPI_Comm comm, int color, int key, MPI_Comm *newComm);

int MPI_Comm_rank(MPI_Comm comm, int *rank);
```

DATATYPE

```
MPI_Datatype newtype;

int MPI_Type_vector(int count, int blocklength, int stride,
  MPI_Datatype oldtype, MPI_Datatype *newtype);

int MPI_Type_commit(MPI_Datatype *datatype);

int MPI_Type_create_resized(MPI_Datatype oldtype, MPI_Aint lb, MPI_Aint
extent, MPI_Datatype, *newtype);
```

COLLECTIVE COMM

```
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root,
  MPI_Comm comm);

int MPI_Scatter(const void* sendbuf, int sendcount, MPI_Datatype sendtype,
  void *recvbuf, int recvcount,MPI_Datatype recvtype, int root,
  MPI_Comm comm);

int MPI_Gather(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
  void* recvbuf, int recvcount, MPI_Datatype recvtype, int root,
  MPI_Comm comm);

int MPI_Allgather(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
  void *recvbuf,int recvcount, MPI_Datatype recvtype, MPI_Comm comm);
```

COLLECTIVE CALCULATION

```
int MPI_Reduce(const void *sendbuf, void *recvbuf, int count,
  MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm);

int MPI_Allreduce(const void *sendbuf, void *recvbuf, int count,
  MPI_Datatype datatype, MPI_Op op, MPI_Comm comm);
```

GROUP

```
MPI_Group world_group, new_group;

int MPI_Comm_group(MPI_Comm comm, MPI_Group *world_group);

int MPI_Group_incl(MPI_Group worls_group, int n, const int ranks[],
  MPI_Group *new_group);

int MPI_Comm_create(MPI_Comm old_comm, MPI_Group new_group,
  MPI_Comm *newComm);

int newRank;

int MPI_Group_rank(MPI_Group group, int *rank);
```

CART

```
int dims[2] = {(column) numberOfColumn, (row) numberOfRow};

int periods[2] = {(column) 1, (row) 1} // 1 = true

int reorder = 0; // 0 = false

MPI_Comm newComm;

int MPI_Cart_create(MPI_Comm comm_old, int ndims, int dims[], int periods[],
  int reorder, MPI_Comm *comm_cart);

int MPI_Comm_rank(MPI_Comm comm, int *rank);

int myCoords[2];

int MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int coords[]);

int remain_dims[2] = {false, true} // rows topology

MPI_Comm subgrid_row_comm;

int MPI_Cart_shift(MPI_Comm comm, int direction, int displacement,
  int *rank_source, int *rank_dest);

direction: {0 = columnShift, 1 = rowShift} // Così come è scritto voglio fare lo shift sulle righe

displacement: {>0 = towardsUpperShift, <0 = towardsLowerShift} // di quanti posti voglio fare lo shift
```

CONSTANTS

MPI_ANY_SOURCE	MPI_ANY_TAG	MPI_COMM_WORLD

REDUCE	Function	REDUCE	Function	REDUCE	Function
MPI_MAX	Maximum	MPI_PROD	Product	MPI_LOR	Logical OR
MPI_MIN	Minimum	MPI_LAND	Logical AND	MPI_BOR	Bitwise OR
MPI_SUM	Sun	MPI_BAND	Bitwise AND	MPI_LXOR	Logical exclusive OR
MPI_BXOR	Bitwise exclusive OR	MPI_MAXLOC	Maximum & Location	MPI_MINLOC	Minimum & Location

MPI_CHAR	MPI_WCHAR	MPI_SHORT	MPI_INT	MPI_LONG
MPI_LONG_LONG	MPI_SIGNED_CHAR	MPI_UNSIGNED_CHAR	MPI_UNSIGNED_SHORT	MPI_UNSIGNED_LONG
MPI_FLOAT	MPI_DOUBLE	MPI_LONG_DOUBLE	MPI_LONG_LONG_INT	MPI_UNSIGNED

UTILS

```
FILE *fd = fopen(char *path, char *mode);
fscanf(FILE *fp, char *format, [argumentsi]) // (file, "%d", &m[i*dim2+j])
fseek(fd, new_position, SEEK_SET);
fclose(fd)
```