LANGUAGE BINDING

A.1 Introduction

In this section we summarize the specific bindings for both Fortran and C. We present first the C bindings, then the Fortran bindings. Listings are alphabetical within chapter.

A.2 Defined Constants for C and Fortran

These are required defined constants, to be defined in the files `mpi.h` (for C) and `mpif.h` (for Fortran).

```
/* return codes (both C and Fortran) */
MPI_SUCCESS
MPI_ERR_BUFFER
MPI_ERR_COUNT
MPI_ERR_TYPE
MPI_ERR_TAG
MPI_ERR_COMM
MPI_ERR_RANK
MPI_ERR_REQUEST
MPI_ERR_ROOT
MPI_ERR_GROUP
MPI_ERR_OP
MPI_ERR_TOPOLOGY
MPI_ERR_DIMS
MPI_ERR_ARG
MPI_ERR_UNKNOWN
MPI_ERR_TRUNCATE
MPI_ERR_OTHER
MPI_ERR_INTERN
MPI_ERR_LASTCODE

/* assorted constants (both C and Fortran) */
```

```
MPI_BOTTOM
MPI_PROC_NULL
MPI_ANY_SOURCE
MPI_ANY_TAG
MPI_UNDEFINED
MPI_UB
MPI_LB

/* status size and reserved index values (Fortran) */
MPI_STATUS_SIZE
MPI_SOURCE
MPI_TAG

/* Error-handling specifiers (C and Fortran) */
MPI_ERRORS_ARE_FATAL
MPI_ERRORS_RETURN

/* Maximum sizes for strings */
MPI_MAX_PROCESSOR_NAME
MPI_MAX_ERROR_STRING

/* elementary datatypes (C) */
MPI_CHAR
MPI_SHORT
MPI_INT
MPI_LONG
MPI_UNSIGNED_CHAR
MPI_UNSIGNED_SHORT
MPI_UNSIGNED
MPI_UNSIGNED_LONG
MPI_FLOAT
MPI_DOUBLE
MPI_LONG_DOUBLE
MPI_BYTE
MPI_PACKED



/* elementary datatypes (Fortran) */
MPI_INTEGER
MPI_REAL
MPI_DOUBLE_PRECISION
MPI_COMPLEX
MPI_DOUBLE_COMPLEX
```

```
MPI_LOGICAL
MPI_CHARACTER
MPI_BYTE
MPI_PACKED

/* datatypes for reduction functions (C) */
MPI_FLOAT_INT
MPI_DOUBLE_INT
MPI_LONG_INT
MPI_2INT
MPI_SHORT_INT
MPI_LONG_DOUBLE_INT

/* datatypes for reduction functions (Fortran) */
MPI_2REAL
MPI_2DOUBLE_PRECISION
MPI_2INTEGER


/* optional datatypes (Fortran) */

MPI_INTEGER1
MPI_INTEGER2
MPI_INTEGER4
MPI_REAL2
MPI_REAL4
MPI_REAL8

/* optional datatypes (C) */
MPI_LONG_LONG_INT

/* reserved communicators (C and Fortran) */
MPI_COMM_WORLD
MPI_COMM_SELF

/* results of communicator and group comparisons */

MPI_IDENT
MPI_CONGRUENT
MPI_SIMILAR
MPI_UNEQUAL

/* environmental inquiry keys (C and Fortran) */
MPI_TAG_UB
```

```
MPI_IO
MPI_HOST

/* collective operations (C and Fortran) */
MPI_MAX
MPI_MIN
MPI_SUM
MPI_PROD
MPI_MAXLOC
MPI_MINLOC
MPI_BAND
MPI_BOR
MPI_BXOR
MPI_LAND
MPI_LOR
MPI_LXOR

/* Null handles */
MPI_GROUP_NULL
MPI_COMM_NULL
MPI_DATATYPE_NULL
MPI_REQUEST_NULL
MPI_OP_NULL
MPI_ERRHANDLER_NULL

/* Empty group */
MPI_GROUP_EMPTY

/* topologies (C and Fortran) */
MPI_GRAPH
MPI_CART
```

The following are defined C type definitions, also included in the file `mpi.h`.

```
/* opaque types (C) */
MPI_Aint
MPI_Status

/* handles to assorted structures (C) */
MPI_Group
MPI_Comm
MPI_Datatype
MPI_Request
MPI_Op
```

```
/* prototypes for user-defined functions (C) */
typedef int  MPI_Copy_function(MPI_Comm *oldcomm, *newcomm, int *keyval,
                    void *extra_state)
typedef int  MPI_Delete_function(MPI_Comm *comm, int *keyval,
                          void *extra_state)}
typedef void MPI_Handler_function(MPI_Comm *, int *, ...);
typedef void MPI_User_function( void *invec, void *inoutvec, int *len,
                     MPI_Datatype *datatype);
```

For Fortran, here are examples of how each of the user-defined functions should be declared.

The user-function argument to MPI_OP_CREATE should be declared like this:

```
FUNCTION USER_FUNCTION( INVEC(*), INOUTVEC(*), LEN, TYPE)
<type> INVEC(LEN), INOUTVEC(LEN)
 INTEGER LEN, TYPE
```

The copy-function argument to MPI_KEYVAL_CREATE should be declared like this:

```
FUNCTION COPY_FUNCTION(OLDCOMM, KEYVAL, EXTRA_STATE, ATTRIBUTE_VAL_IN,
                                                  ATTRIBUTE_VAL_OUT, FLAG)
 INTEGER OLDCOMM, KEYVAL, EXTRA_STATE, ATTRIBUTE_VAL_IN, ATTRIBUTE_VAL_OUT
 LOGICAL FLAG
```

The delete-function argument to MPI_KEYVAL_CREATE should be declared like this:

```
FUNCTION DELETE_FUNCTION(COMM, KEYVAL, ATTRIBUTE_VAL, EXTRA_STATE)
 INTEGER COMM, KEYVAL, ATTRIBUTE_VAL, EXTRA_STATE
```

A.3 C Bindings for Point-to-Point Communication

These are presented here in the order of their appearance in the chapter.

```
int MPI_Send(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_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)

int MPI_Bsend(void* buf, int count, MPI_Datatype datatype, int dest,
             int tag, MPI_Comm comm)

int MPI_Ssend(void* buf, int count, MPI_Datatype datatype, int dest,
             int tag, MPI_Comm comm)
```

```
int MPI_Rsend(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm conn)

int MPI_Buffer_attach( void* buffer, int size)

int MPI_Buffer_detach( void** buffer, int* size)

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

int MPI_Ibsend(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm conn, MPI_Request *request)

int MPI_Issend(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm conn, MPI_Request *request)

int MPI_Irsend(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm conn, MPI_Request *request)

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

int MPI_Wait(MPI_Request *request, MPI_Status *status)

int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)

int MPI_Request_free(MPI_Request *request)

int MPI_Waitany(int count, MPI_Request *array_of_requests, int *index,
              MPI_Status *status)

int MPI_Testany(int count, MPI_Request *array_of_requests, int *index,
              int *flag, MPI_Status *status)

int MPI_Waitall(int count, MPI_Request *array_of_requests,
              MPI_Status *array_of_statuses)

int MPI_Testall(int count, MPI_Request *array_of_requests, int *flag,
              MPI_Status *array_of_statuses)

int MPI_Waitsome(int incount, MPI_Request *array_of_requests, int *outcount,
              int *array_of_indices, MPI_Status *array_of_statuses)

int MPI_Testsome(int incount, MPI_Request *array_of_requests, int *outcount,
              int *array_of_indices, MPI_Status *array_of_statuses)

int MPI_Iprobe(int source, int tag, MPI_Comm conn, int *flag,
              MPI_Status *status)

int MPI_Probe(int source, int tag, MPI_Comm conn, MPI_Status *status)

int MPI_Cancel(MPI_Request *request)
```

```
int MPI_Test_cancelled(MPI_Status *status, int *flag)

int MPI_Send_init(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm conn, MPI_Request *request)

int MPI_Bsend_init(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm conn, MPI_Request *request)

int MPI_Ssend_init(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm conn, MPI_Request *request)

int MPI_Rsend_init(void* buf, int count, MPI_Datatype datatype, int dest,
              int tag, MPI_Comm conn, MPI_Request *request)

int MPI_Recv_init(void* buf, int count, MPI_Datatype datatype, int source,
              int tag, MPI_Comm conn, MPI_Request *request)

int MPI_Start(MPI_Request *request)

int MPI_Startall(int count, MPI_Request *array_of_requests)

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

int MPI_Sendrecv_replace(void* buf, int count, MPI_Datatype datatype,
              int dest, int sendtag, int source, int recvtag,
              MPI_Comm comm, MPI_Status *status)

int MPI_Type_contiguous(int count, MPI_Datatype oldtype,
              MPI_Datatype *newtype)

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

int MPI_Type_hvector(int count, int blocklength, MPI_Aint stride,
              MPI_Datatype oldtype, MPI_Datatype *newtype)

int MPI_Type_indexed(int count, int *array_of_blocklengths,
              int *array_of_displacements, MPI_Datatype oldtype,
              MPI_Datatype *newtype)

int MPI_Type_hindexed(int count, int *array_of_blocklengths,
              MPI_Aint *array_of_displacements, MPI_Datatype oldtype,
              MPI_Datatype *newtype)

int MPI_Type_struct(int count, int *array_of_blocklengths,
              MPI_Aint *array_of_displacements,
              MPI_Datatype *array_of_types, MPI_Datatype *newtype)
```

```
int MPI_Address(void* location, MPI_Aint *address)

int MPI_Type_extent(MPI_Datatype datatype, int MPI_Aint *extent)

int MPI_Type_size(MPI_Datatype datatype, int MPI_Aint*size)

int MPI_Type_count(MPI_Datatype datatype, int *count)

int MPI_Type_lb(MPI_Datatype datatype, int* MPI_Aint displacement)

int MPI_Type_ub(MPI_Datatype datatype, int* MPI_Aint displacement)

int MPI_Type_commit(MPI_Datatype *datatype)

int MPI_Type_free(MPI_Datatype *datatype)

int MPI_Get_elements(MPI_Status *status, MPI_Datatype datatype, int *count)

int MPI_Pack(void* inbuf, int incount, MPI_Datatype datatype, void *outbuf,
              int outsize, int *position, MPI_Comm comm)

int MPI_Unpack(void* inbuf, int insize, int *position, void *outbuf,
              int outcount, MPI_Datatype datatype, MPI_Comm comm)

int MPI_Pack_size(int incount, MPI_Datatype datatype, MPI_Comm conn,
              int *size)
```

A.4 C Bindings for Collective Communication

```
int MPI_Barrier(MPI_Comm comm )

int MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root,
              MPI_Comm conn )

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

int MPI_Gatherv(void* sendbuf, int sendcount, MPI_Datatype sendtype,
              void* recvbuf, int *recvcounts, int *displs,
              MPI_Datatype recvtype, int root, MPI_Comm comm)

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

int MPI_Scatterv(void* sendbuf, int *sendcounts, int *displs,
              MPI_Datatype sendtype, void* recvbuf, int recvcount,
              MPI_Datatype recvtype, int root, MPI_Comm conn)
```

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

int MPI_Allgatherv(void* sendbuf, int sendcount, MPI_Datatype sendtype,
                  void* recvbuf, int *recvcounts, int *displs,
                  MPI_Datatype recvtype, MPI_Comm comm)

int MPI_Alltoall(void* sendbuf, int sendcount, MPI_Datatype sendtype,
                  void* recvbuf, int recvcount, MPI_Datatype recvtype,
                  MPI_Comm conn)

int MPI_Alltoallv(void* sendbuf, int *sendcounts, int *sdispls,
                  MPI_Datatype sendtype, void* recvbuf, int *recvcounts,
                  int *rdispls, MPI_Datatype recvtype, MPI_Comm conn)

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

int MPI_Op_create(MPI_User_function *function, int commute, MPI_Op *op)

int MPI_Op_free( MPI_Op *op)

int MPI_Allreduce(void* sendbuf, void* recvbuf, int count,
                  MPI_Datatype datatype, MPI_Op op, MPI_Comm conn)

int MPI_Reduce_scatter(void* sendbuf, void* recvbuf, int *recvcounts,
                  MPI_Datatype datatype, MPI_Op op, MPI_Comm conn)

int MPI_Scan(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype,
                  MPI_Op op, MPI_Comm conn )
```

A.5 C Bindings for Groups, Contexts, and Communicators

```
int MPI_Group_size(MPI_Group group, int *size)

int MPI_Group_rank(MPI_Group group, int *rank)

int MPI_Group_translate_ranks (MPI_Group group1, int n, int *ranks1,
                  MPI_Group group2, int *ranks2)

int MPI_Group_compare(MPI_Group group1,MPI_Group group2, int *result)

int MPI_Comm_group(MPI_Comm comm, MPI_Group *group)

int MPI_Group_union(MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

int MPI_Group_intersection(MPI_Group group1, MPI_Group group2,
                  MPI_Group *newgroup)
```

```
int MPI_Group_difference(MPI_Group group1, MPI_Group group2,
                MPI_Group *newgroup)

int MPI_Group_incl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup)

int MPI_Group_excl(MPI_Group group, int n, int *ranks, MPI_Group *newgroup)

int MPI_Group_range_incl(MPI_Group group, int n, int ranges[][3],
                MPI_Group *newgroup)

int MPI_Group_range_excl(MPI_Group group, int n, int ranges[][3],
                MPI_Group *newgroup)

int MPI_Group_free(MPI_Group *group)

int MPI_Comm_size(MPI_Conn conn, int *size)

int MPI_Comm_rank(MPI_Conn comm, int *rank)

int MPI_Comm_compare(MPI_Conn conn1, conn2, int *result)

int MPI_Conn_dup(MPI_Comm conn, MPI_Conn *newconn)

int MPI_Conn_create(MPI_Conn conn, MPI_Group group, MPI_Conn *newconn)

int MPI_Conn_split(MPI_Comm conn, int color, int key, MPI_Conn *newconn)

int MPI_Conn_free(MPI_Conn *conn)

int MPI_Comm_test_inter(MPI_Conn conn, int *flag)

int MPI_Comm_remote_size(MPI_Conn conn, int *size)

int MPI_Conn_remote_group(MPI_Conn conn, MPI_Group *group)

int MPI_Intercomm_create(MPI_Comm local_conn, int local_leader,
                MPI_Conn peer_conn, int remote_leader, int tag,
                MPI_Conn *newinterconn)

int MPI_Intercomm_merge(MPI_Conn interconn, int high, MPI_Conn *newintraconn)

int MPI_Keyval_create(MPI_Copy_function *copy_fn, MPI_Delete_function
                *delete_fn, int *keyval, void* extra_state)

int MPI_Keyval_free(int *keyval)

int MPI_Attr_put(MPI_Conn conn, int keyval, void* attribute_val)

int MPI_Attr_get(MPI_Conn conn, int keyval, void **attribute_val, int *flag)

int MPI_Attr_delete(MPI_Conn conn, int keyval)
```

A.6 C Bindings for Process Topologies

```
int MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims, int *periods,
              int reorder, MPI_Comm *comm_cart)

int MPI_Dims_create(int nnodes, int ndims, int *dims)

int MPI_Graph_create(MPI_Comm comm_old, int nnodes, int *index, int *edges,
              int reorder, MPI_Comm *comm_graph)

int MPI_Topo_test(MPI_Comm conn, int *status)

int MPI_Graphdims_get(MPI_Comm comm, int *nnodes, int *nedges)

int MPI_Graph_get(MPI_Comm conn, int maxindex, int maxedges, int *index,
              int *edges)

int MPI_Cartdim_get(MPI_Comm conn, int *ndims)

int MPI_Cart_get(MPI_Comm comm, int maxdims, int *dims, int *periods,
              int *coords)

int MPI_Cart_rank(MPI_Comm conn, int *coords, int *rank)

int MPI_Cart_coords(MPI_Comm conn, int rank, int maxdims, int *coords)

int MPI_Graph_neighbors_count(MPI_Comm comm, int rank, int *nneighbors)

int MPI_Graph_neighbors(MPI_Comm comm, int rank, int maxneighbors,
              int *neighbors)

int MPI_Cart_shift(MPI_Comm comm, int direction, int disp, int *rank_source,
              int *rank_dest)

int MPI_Cart_sub(MPI_Comm comm, int *remain_dims, MPI_Comm *newcomm)

int MPI_Cart_map(MPI_Comm comm, int ndims, int *dims, int *periods,
              int *newrank)

int MPI_Graph_map(MPI_Comm comm, int nnodes, int *index, int *edges,
              int *newrank)
```

A.7 C Bindings for Environmental Inquiry

```
int MPI_Get_processor_name(char *name, int *resultlen)

int MPI_Errhandler_create(MPI_Handler_function *function,
              MPI_Errhandler *errhandler)

int MPI_Errhandler_set(MPI_Comm comm, MPI_Errhandler errhandler)

int MPI_Errhandler_get(MPI_Comm comm, MPI_Errhandler *errhandler)
```

```
int MPI_Errhandler_free(MPI_Errhandler *errhandler)

int MPI_Error_string(int errorcode, char *string, int *resultlen)

int MPI_Error_class(int errorcode, int *errorclass)

int double MPI_Wtime(void)

int double MPI_Wtick(void)

int MPI_Init(int *argc, char ***argv)

int MPI_Finalize(void)

int MPI_Initialized(int *flag)

int MPI_Abort(MPI_Comm comm, int errorcode)
```

A.8 C Bindings for Profiling

```
int MPI_Pcontrol(const int level, ...)
```

A.9 Fortran Bindings for Point-to-Point Communication

```
MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR

MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS(MPI_STATUS_SIZE),
    IERROR

MPI_GET_COUNT(STATUS, DATATYPE, COUNT, IERROR)
    INTEGER STATUS(MPI_STATUS_SIZE), DATATYPE, COUNT, IERROR

MPI_BSEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR

MPI_SSEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR

MPI_RSEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERROR

MPI_BUFFER_ATTACH( BUFFER, SIZE, IERROR)
    <type> BUFFER(*)
```

```
    INTEGER SIZE, IERROR

MPI_BUFFER_DETACH( BUFFER, SIZE, IERROR)
    <type> BUFFER(*)
    INTEGER SIZE, IERROR

MPI_ISEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

MPI_IBSEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

MPI_ISSEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

MPI_IRSEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

MPI_IRECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR

MPI_WAIT(REQUEST, STATUS, IERROR)
    INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERROR

MPI_TEST(REQUEST, FLAG, STATUS, IERROR)
    LOGICAL FLAG
    INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERROR

MPI_REQUEST_FREE(REQUEST, IERROR)
    INTEGER REQUEST, IERROR

MPI_WAITANY(COUNT, ARRAY_OF_REQUESTS, INDEX, STATUS, IERROR)
    INTEGER COUNT, ARRAY_OF_REQUESTS(*), INDEX, STATUS(MPI_STATUS_SIZE),
    IERROR

MPI_TESTANY(COUNT, ARRAY_OF_REQUESTS, INDEX, FLAG, STATUS, IERROR)
    LOGICAL FLAG
    INTEGER COUNT, ARRAY_OF_REQUESTS(*), INDEX, STATUS(MPI_STATUS_SIZE),
    IERROR

MPI_WAITALL(COUNT, ARRAY_OF_REQUESTS, ARRAY_OF_STATUSES, IERROR)
    INTEGER COUNT, ARRAY_OF_REQUESTS(*),
    ARRAY_OF_STATUSES(MPI_STATUS_SIZE,*), IERROR

MPI_TESTALL(COUNT, ARRAY_OF_REQUESTS, FLAG, ARRAY_OF_STATUSES, IERROR)
```

```
    LOGICAL FLAG
    INTEGER COUNT, ARRAY_OF_REQUESTS(*),
    ARRAY_OF_STATUSES(MPI_STATUS_SIZE,*), IERROR

MPI_WAITSOME(INCOUNT, ARRAY_OF_REQUESTS, OUTCOUNT, ARRAY_OF_INDICES,
                ARRAY_OF_STATUSES, IERROR)
    INTEGER INCOUNT, ARRAY_OF_REQUESTS(*), OUTCOUNT, ARRAY_OF_INDICES(*),
    ARRAY_OF_STATUSES(MPI_STATUS_SIZE,*), IERROR

MPI_TESTSOME(INCOUNT, ARRAY_OF_REQUESTS, OUTCOUNT, ARRAY_OF_INDICES,
                ARRAY_OF_STATUSES, IERROR)
    INTEGER INCOUNT, ARRAY_OF_REQUESTS(*), OUTCOUNT, ARRAY_OF_INDICES(*),
    ARRAY_OF_STATUSES(MPI_STATUS_SIZE,*), IERROR

MPI_IPROBE(SOURCE, TAG, COMM, FLAG, STATUS, IERROR)
    LOGICAL FLAG
    INTEGER SOURCE, TAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR

MPI_PROBE(SOURCE, TAG, COMM, STATUS, IERROR)
    INTEGER SOURCE, TAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR

MPI_CANCEL(REQUEST, IERROR)
    INTEGER REQUEST, IERROR

MPI_TEST_CANCELLED(STATUS, FLAG, IERROR)
    LOGICAL FLAG
    INTEGER STATUS(MPI_STATUS_SIZE), IERROR

MPI_SEND_INIT(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
    <type> BUF(*)
    INTEGER REQUEST, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

MPI_BSEND_INIT(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
    <type> BUF(*)
    INTEGER REQUEST, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

MPI_SSEND_INIT(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

MPI_RSEND_INIT(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

MPI_RECV_INIT(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR

MPI_START(REQUEST, IERROR)
```

```
    INTEGER REQUEST, IERROR

MPI_STARTALL(COUNT, ARRAY_OF_REQUESTS, IERROR)
    INTEGER COUNT, ARRAY_OF_REQUESTS(*), IERROR

MPI_SENDRECV(SENDBUF, SENDCOUNT, SENDTYPE, DEST, SENDTAG, RECVBUF,
                RECVCOUNT, RECVTYPE, SOURCE, RECVTAG, COMM, STATUS, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER SENDCOUNT, SENDTYPE, DEST, SENDTAG, RECVCOUNT, RECVTYPE, SOURCE,
    RECVTAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR

MPI_SENDRECV_REPLACE(BUF, COUNT, DATATYPE, DEST, SENDTAG, SOURCE, RECVTAG,
                COMM, STATUS, IERROR)
    <type> BUF(*)
    INTEGER COUNT, DATATYPE, DEST, SENDTAG, SOURCE, RECVTAG, COMM,
    STATUS(MPI_STATUS_SIZE), IERROR

MPI_TYPE_CONTIGUOUS(COUNT, OLDTYPE, NEWTYPE, IERROR)
    INTEGER COUNT, OLDTYPE, NEWTYPE, IERROR

MPI_TYPE_VECTOR(COUNT, BLOCKLENGTH, STRIDE, OLDTYPE, NEWTYPE, IERROR)
    INTEGER COUNT, BLOCKLENGTH, STRIDE, OLDTYPE, NEWTYPE, IERROR

MPI_TYPE_HVECTOR(COUNT, BLOCKLENGTH, STRIDE, OLDTYPE, NEWTYPE, IERROR)
    INTEGER COUNT, BLOCKLENGTH, STRIDE, OLDTYPE, NEWTYPE, IERROR

MPI_TYPE_INDEXED(COUNT, ARRAY_OF_BLOCKLENGTHS, ARRAY_OF_DISPLACEMENTS,
                OLDTYPE, NEWTYPE, IERROR)
    INTEGER COUNT, ARRAY_OF_BLOCKLENGTHS(*), ARRAY_OF_DISPLACEMENTS(*),
    OLDTYPE, NEWTYPE, IERROR

MPI_TYPE_HINDEXED(COUNT, ARRAY_OF_BLOCKLENGTHS, ARRAY_OF_DISPLACEMENTS,
                OLDTYPE, NEWTYPE, IERROR)
    INTEGER COUNT, ARRAY_OF_BLOCKLENGTHS(*), ARRAY_OF_DISPLACEMENTS(*),
    OLDTYPE, NEWTYPE, IERROR

MPI_TYPE_STRUCT(COUNT, ARRAY_OF_BLOCKLENGTHS, ARRAY_OF_DISPLACEMENTS,
                ARRAY_OF_TYPES, NEWTYPE, IERROR)
    INTEGER COUNT, ARRAY_OF_BLOCKLENGTHS(*), ARRAY_OF_DISPLACEMENTS(*),
    ARRAY_OF_TYPES(*), NEWTYPE, IERROR

MPI_ADDRESS(LOCATION, ADDRESS, IERROR)
    <type> LOCATION(*)
    INTEGER ADDRESS, IERROR

MPI_TYPE_EXTENT(DATATYPE, EXTENT, IERROR)
    INTEGER DATATYPE, EXTENT, IERROR

MPI_TYPE_SIZE(DATATYPE, SIZE, IERROR)
```

```
    INTEGER DATATYPE, SIZE, IERROR

MPI_TYPE_COUNT(DATATYPE, COUNT, IERROR)
    INTEGER DATATYPE, COUNT, IERROR

MPI_TYPE_LB( DATATYPE, DISPLACEMENT, IERROR)
    INTEGER DATATYPE, DISPLACEMENT, IERROR

MPI_TYPE_UB( DATATYPE, DISPLACEMENT, IERROR)
    INTEGER DATATYPE, DISPLACEMENT, IERROR

MPI_TYPE_COMMIT(DATATYPE, IERROR)
    INTEGER DATATYPE, IERROR

MPI_TYPE_FREE(DATATYPE, IERROR)
    INTEGER DATATYPE, IERROR

MPI_GET_ELEMENTS(STATUS, DATATYPE, COUNT, IERROR)
    INTEGER STATUS(MPI_STATUS_SIZE), DATATYPE, COUNT, IERROR

MPI_PACK(INBUF, INCOUNT, DATATYPE, OUTBUF, OUTSIZE, POSITION, COMM, IERROR)
    <type> INBUF(*), OUTBUF(*)
    INTEGER INCOUNT, DATATYPE, OUTSIZE, POSITION, COMM, IERROR

MPI_UNPACK(INBUF, INSIZE, POSITION, OUTBUF, OUTSIZE, DATATYPE, COMM,
               IERROR)
    <type> INBUF(*), OUTBUF(*)
    INTEGER INSIZE, POSITION, OUTCOUNT, DATATYPE, COMM, IERROR

MPI_PACK_SIZE(INCOUNT, DATATYPE, COMM, SIZE, IERROR)
    INTEGER INCOUNT, DATATYPE, COMM, SIZE, IERROR
```

A.10 Fortran Bindings for Collective Communication

```
MPI_BARRIER(COMM, IERROR)
    INTEGER COMM, IERROR

MPI_BCAST(BUFFER, COUNT, DATATYPE, ROOT, COMM, IERROR)
    <type> BUFFER(*)
    INTEGER COUNT, DATATYPE, ROOT, COMM, IERROR

MPI_GATHER(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT, RECVTYPE, ROOT,
               COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR

MPI_GATHERV(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNTS, DISPLS,
                RECVTYPE, ROOT, COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
```

```
    INTEGER SENDCOUNT, SENDTYPE, RECVCOUNTS(*), DISPLS(*), RECVTYPE, ROOT,
    COMM, IERROR

MPI_SCATTER(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT, RECVTYPE,
                ROOT, COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR

MPI_SCATTERV(SENDBUF, SENDCOUNTS, DISPLS, SENDTYPE, RECVBUF, RECVCOUNT,
                RECVTYPE, ROOT, COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER SENDCOUNTS(*), DISPLS(*), SENDTYPE, RECVCOUNT, RECVTYPE, ROOT,
    COMM, IERROR

MPI_ALLGATHER(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT, RECVTYPE,
                COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, IERROR

MPI_ALLGATHERV(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNTS, DISPLS,
                RECVTYPE, COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER SENDCOUNT, SENDTYPE, RECVCOUNTS(*), DISPLS(*), RECVTYPE, COMM,
    IERROR

MPI_ALLTOALL(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT, RECVTYPE,
                COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, IERROR

MPI_ALLTOALLV(SENDBUF, SENDCOUNTS, SDISPLS, SENDTYPE, RECVBUF, RECVCOUNTS,
                RDISPLS, RECVTYPE, COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER SENDCOUNTS(*), SDISPLS(*), SENDTYPE, RECVCOUNTS(*), RDISPLS(*),
    RECVTYPE, COMM, IERROR

MPI_REDUCE(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, ROOT, COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER COUNT, DATATYPE, OP, ROOT, COMM, IERROR

MPI_OP_CREATE( FUNCTION, COMMUTE, OP, IERROR)
    EXTERNAL FUNCTION
    LOGICAL COMMUTE
    INTEGER OP, IERROR

MPI_OP_FREE( OP, IERROR)
    INTEGER OP, IERROR
```

```
MPI_ALLREDUCE(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER COUNT, DATATYPE, OP, COMM, IERROR

MPI_REDUCE_SCATTER(SENDBUF, RECVBUF, RECVCOUNTS, DATATYPE, OP, COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER RECVCOUNTS(*), DATATYPE, OP, COMM, IERROR

MPI_SCAN(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, COMM, IERROR)
    <type> SENDBUF(*), RECVBUF(*)
    INTEGER COUNT, DATATYPE, OP, COMM, IERROR
```

A.11 Fortran Bindings for Groups, Contexts, etc.

```
MPI_GROUP_SIZE(GROUP, SIZE, IERROR)
    INTEGER GROUP, SIZE, IERROR

MPI_GROUP_RANK(GROUP, RANK, IERROR)
    INTEGER GROUP, RANK, IERROR

MPI_GROUP_TRANSLATE_RANKS(GROUP1, N, RANKS1, GROUP2, RANKS2, IERROR)
    INTEGER GROUP1, N, RANKS1(*), GROUP2, RANKS2(*), IERROR

MPI_GROUP_COMPARE(GROUP1, GROUP2, RESULT, IERROR)
    INTEGER GROUP1, GROUP2, RESULT, IERROR

MPI_COMM_GROUP(COMM, GROUP, IERROR)
    INTEGER COMM, GROUP, IERROR

MPI_GROUP_UNION(GROUP1, GROUP2, NEWGROUP, IERROR)
    INTEGER GROUP1, GROUP2, NEWGROUP, IERROR

MPI_GROUP_INTERSECTION(GROUP1, GROUP2, NEWGROUP, IERROR)
    INTEGER GROUP1, GROUP2, NEWGROUP, IERROR

MPI_GROUP_DIFFERENCE(GROUP1, GROUP2, NEWGROUP, IERROR)
    INTEGER GROUP1, GROUP2, NEWGROUP, IERROR

MPI_GROUP_INCL(GROUP, N, RANKS, NEWGROUP, IERROR)
    INTEGER GROUP, N, RANKS(*), NEWGROUP, IERROR

MPI_GROUP_EXCL(GROUP, N, RANKS, NEWGROUP, IERROR)
    INTEGER GROUP, N, RANKS(*), NEWGROUP, IERROR

MPI_GROUP_RANGE_INCL(GROUP, N, RANGES, NEWGROUP, IERROR)
    INTEGER GROUP, N, RANGES(3,*), NEWGROUP, IERROR

MPI_GROUP_RANGE_EXCL(GROUP, N, RANGES, NEWGROUP, IERROR)
    INTEGER GROUP, N, RANGES(3,*), NEWGROUP, IERROR
```

```
MPI_GROUP_FREE(GROUP, IERROR)
    INTEGER GROUP, IERROR

MPI_COMM_SIZE(COMM, SIZE, IERROR)
    INTEGER COMM, SIZE, IERROR

MPI_COMM_RANK(COMM, RANK, IERROR)
    INTEGER COMM, RANK, IERROR

MPI_COMM_COMPARE(COMM1, COMM2, RESULT, IERROR)
    INTEGER COMM1, COMM2, RESULT, IERROR

MPI_COMM_DUP(COMM, NEWCOMM, IERROR)
    INTEGER COMM, NEWCOMM, IERROR

MPI_COMM_CREATE(COMM, GROUP, NEWCOMM, IERROR)
    INTEGER COMM, GROUP, NEWCOMM, IERROR
MPI_COMM_SPLIT(COMM, COLOR, KEY, NEWCOMM, IERROR)
    INTEGER COMM, COLOR, KEY, NEWCOMM, IERROR
MPI_COMM_FREE(COMM, IERROR)
    INTEGER COMM, IERROR

MPI_COMM_TEST_INTER(COMM, FLAG, IERROR)
    INTEGER COMM, IERROR
    LOGICAL FLAG

MPI_COMM_REMOTE_SIZE(COMM, SIZE, IERROR)
    INTEGER COMM, SIZE, IERROR

MPI_COMM_REMOTE_GROUP(COMM, GROUP, IERROR)
    INTEGER COMM, GROUP, IERROR

MPI_INTERCOMM_CREATE(LOCAL_COMM, LOCAL_LEADER, PEER_COMM, REMOTE_LEADER, TAG,
                NEWINTERCOMM, IERROR)
    INTEGER LOCAL_COMM, LOCAL_LEADER, PEER_COMM, REMOTE_LEADER, TAG,
    NEWINTERCOMM, IERROR

MPI_INTERCOMM_MERGE(INTERCOMM, HIGH, INTRACOMM, IERROR)
    INTEGER INTERCOMM, INTRACOMM, IERROR
    LOGICAL HIGH

MPI_KEYVAL_CREATE(COPY_FN, DELETE_FN, KEYVAL, EXTRA_STATE, IERROR)
    EXTERNAL COPY_FN, DELETE_FN
    INTEGER KEYVAL, EXTRA_STATE, IERROR

MPI_KEYVAL_FREE(KEYVAL, IERROR)
    INTEGER KEYVAL, IERROR
```

```
MPI_ATTR_PUT(COMM, KEYVAL, ATTRIBUTE_VAL, IERROR)
    INTEGER COMM, KEYVAL, ATTRIBUTE_VAL, IERROR

MPI_ATTR_GET(COMM, KEYVAL, ATTRIBUTE_VAL, FLAG, IERROR)
    INTEGER COMM, KEYVAL, ATTRIBUTE_VAL, IERROR
    LOGICAL FLAG

MPI_ATTR_DELETE(COMM, KEYVAL, IERROR)
    INTEGER COMM, KEYVAL, IERROR
```

A.12 Fortran Bindings for Process Topologies

```
MPI_CART_CREATE(COMM_OLD, NDIMS, DIMS, PERIODS, REORDER, COMM_CART, IERROR)
    INTEGER COMM_OLD, NDIMS, DIMS(*), COMM_CART, IERROR
    LOGICAL PERIODS(*), REORDER

MPI_DIMS_CREATE(NNODES, NDIMS, DIMS, IERROR)
    INTEGER NNODES, NDIMS, DIMS(*), IERROR

MPI_GRAPH_CREATE(COMM_OLD, NNODES, INDEX, EDGES, REORDER, COMM_GRAPH, IERROR)
    INTEGER COMM_OLD, NNODES, INDEX(*), EDGES(*), COMM_GRAPH, IERROR
    LOGICAL REORDER

MPI_TOPO_TEST(COMM, STATUS, IERROR)
    INTEGER COMM, STATUS, IERROR

MPI_GRAPHDIMS_GET(COMM, NNODES, NEDGES, IERROR)
    INTEGER COMM, NNODES, NEDGES, IERROR

MPI_GRAPH_GET(COMM, MAXINDEX, MAXEDGES, INDEX, EDGES, IERROR)
    INTEGER COMM, MAXINDEX, MAXEDGES, INDEX(*), EDGES(*), IERROR

MPI_CARTDIM_GET(COMM, NDIMS, IERROR)
    INTEGER COMM, NDIMS, IERROR

MPI_CART_GET(COMM, MAXDIMS, DIMS, PERIODS, CODRDS, IERROR)
    INTEGER COMM, MAXDIMS, DIMS(*), COORDS(*), IERROR
    LOGICAL PERIODS(*)

MPI_CART_RANK(COMM, CODRDS, RANK, IERROR)
    INTEGER COMM, COORDS(*), RANK, IERROR

MPI_CART_COORDS(COMM, RANK, MAXDIMS, COORDS, IERROR)
    INTEGER COMM, RANK, MAXDIMS, CODRDS(*), IERROR

MPI_GRAPH_NEIGHBORS_COUNT(COMM, RANK, NNEIGHBORS, IERROR)
    INTEGER COMM, RANK, NNEIGHBORS, IERROR

MPI_GRAPH_NEIGHBORS(COMM, RANK, MAXNEIGHBORS, NEIGHBORS, IERROR)
    INTEGER COMM, RANK, MAXNEIGHBORS, NEIGHBORS(*), IERROR
```

```
MPI_CART_SHIFT(COMM, DIRECTION, DISP, RANK_SOURCE, RANK_DEST, IERROR)
    INTEGER COMM, DIRECTION, DISP, RANK_SOURCE, RANK_DEST, IERROR

MPI_CART_SUB(COMM, REMAIN_DIMS, NEWCOMM, IERROR)
    INTEGER COMM, NEWCOMM, IERROR
    LOGICAL REMAIN_DIMS(*)

MPI_CART_MAP(COMM, NDIMS, DIMS, PERIODS, NEWRANK, IERROR)
    INTEGER COMM, NDIMS, DIMS(*), NEWRANK, IERROR
    LOGICAL PERIODS(*)

MPI_GRAPH_MAP(COMM, NNODES, INDEX, EDGES, NEWRANK, IERROR)
    INTEGER COMM, NNODES, INDEX(*), EDGES(*), NEWRANK, IERROR
```

A.13 Fortran Bindings for Environmental Inquiry

```
MPI_GET_PROCESSOR_NAME(NAME, RESULTLEN, IERROR)
    CHARACTER*(*) NAME
    INTEGER RESULTLEN, IERROR

MPI_ERRHANDLER_CREATE(FUNCTION, HANDLER, IERROR)
    EXTERNAL FUNCTION
    INTEGER ERRHANDLER, IERROR

MPI_ERRHANDLER_SET(COMM, ERRHANDLER, IERROR)
    INTEGER COMM, ERRHANDLER, IERROR

MPI_ERRHANDLER_GET(COMM, ERRHANDLER, IERROR)
    INTEGER COMM, ERRHANDLER, IERROR

MPI_ERRHANDLER_FREE(ERRHANDLER, IERROR)
    INTEGER ERRHANDLER, IERROR

MPI_ERROR_STRING(ERRORCODE, STRING, RESULTLEN, IERROR)
    INTEGER ERRORCODE, RESULTLEN, IERROR
    CHARACTER*(*) STRING

MPI_ERROR_CLASS(ERRORCODE, ERRORCLASS, IERROR)
    INTEGER ERRORCODE, ERRORCLASS, IERROR

DOUBLE PRECISION MPI_WTIME()

DOUBLE PRECISION MPI_WTICK()

MPI_INIT(IERROR)
    INTEGER IERROR

MPI_FINALIZE(IERROR)
    INTEGER IERROR
```

```
MPI_INITIALIZED(FLAG, IERROR)
    LOGICAL FLAG
    INTEGER IERROR

MPI_ABORT(COMM, ERRORCODE, IERROR)
    INTEGER COMM, ERRORCODE, IERROR
```

A.14 Fortran Bindings for Profiling

```
MPI_PCONTROL(level)
    INTEGER LEVEL, ...
```

MPI FUNCTION INDEX

THE INTERNATIONAL JOURNAL OF SUPERCOMPUTER APPLICATIONS AND HIGH PERFORMANCE COMPUTING—

INFORMATION FOR CONTRIBUTORS

SCOPE OF ARTICLES

The International Journal of Supercomputer Applications and High Performance Computing published quarterly by the MIT Press, is directed to researchers in computational science, as well as to educators, programmers, designers and users of computers who have particular interests in the application and development of supercomputers.

The goal of the journal is to provide a lively forum for the communication of original research papers and timely review articles on the use of supercomputers to solve complex modeling problems in a spectrum of disciplines. The emphasis will be on experiences with the use of supercomputers rather than on the exposition of computational results peculiar to a specific research topic. Software techniques that apply to classes of problems often cross disciplines; articles should focus on the exchange of such techniques, as well as present methods for analyzing, measuring and applying algorithms and solution schemes related to particular application areas.

The scope of the journal is reflected by the specialties of the board of contributing editors. Sample topic areas include aerospace engineering, artificial intelligence and knowledge processing, astro physics, atmospheric research and meteorological forecasting, automotive design and production, computational aerodynamics, computer graphics and imaging, cryptographic analysis, economic modeling, implementation techniques and pragmatic software and architectural considerations, integrated circuit design, molecular biology, motion-picture graphics, nuclear fusion research, performance studies, petroleum reservoir engineering and hydrology simulations, pharmaceutical research, structural analysis and computer-aided design, and theoretical and experimental physics.

SUBMISSION OF MANUSCRIPTS

Send four copies of individual manuscripts, with a brief biography of each author, along with a suggestion of the appropriate subject area editor for review of the article, to:

Dr. Jack Dongarra
Department of Computer Science
104 Ayres Hall
University of Tennessee
Knoxville, TN 37996-1301

SPECIAL ISSUES

Proposals for issues dedicated to a particular topic of interest to the journal's audience will be considered. Once accepted, a guest editor will be appointed and the special issue will be coordinated through the editorial office of the journal. Submit four copies of special issue proposals, with a brief biography of each submitter, to:

Dr. Joanne L. Martin
IBM Corporation
Neighborhood Road
Kingston, NY 12401

Authors wishing to provide manuscript by electronic transmittal are requested to contact the editors for information about formats and procedures. Only original papers will be considered. Manuscripts are accepted for review with the understanding that the same work has not and will not be submitted for consideration elsewhere, and that its submission for publication has been approved by all of the authors, the institution where the work was performed, and the appropriate funding agencies. It is further understood that any person cited as a source of personal communications has approved such citation; written authorization may be required at the editor's discretion. Articles and any other material published in *The International Journal of Supercomputer Applications and High Performance Computing*, represent the opinions of the authors and should not be construed to reflect the opinions of the editors or the publisher.

To comply with the U.S. Copyright law, authors are required to sign a copyright transfer form before publication. It is the policy of MIT Press to own the copyright to the contributions it publishes, and to facilitate the appropriate reuse of this material by others; the transfer of copyright grants authors and their employers full rights to reuse their own material. Authors submitting a manuscript do so on the understanding that if the manuscript is accepted for publication, copyright for the article, including the right to reproduce the article in all forms and media, shall be assigned exclusively to the publisher. Authors are responsible for obtaining permissions to use photographs, drawings, tables, and other previously published material in their articles.

ORGANIZATION OF MANUSCRIPTS

Publication time can be minimized if the manuscript is carefully prepared according to the following style rules:

Form of Manuscript: Submit typed manuscripts in quadruplicate, including tables and illustrations, preferably triple-spaced, but at least double-spaced, on one side of 8½ × 11-in. white paper. Number each page. Page 1 should contain the article title, authors' names, and complete affiliations. Page 2 should contain a proposed running head (abbreviated form of the title) of less than 30 characters and the name, mailing address and telephone number of the author to whom the queries should be addressed. Page 2 should also contain a description of any special symbols used in the manuscript.

Summary: Page 3 should contain a short summary (no more than 150 words) that describes the problem to be solved, at a level that will appeal to interested readers across disciplines. The summary should also define the significance of the work within a general scientific framework, and establish its computational requirements. Computational methods and conclusions reached should be described briefly.

Introduction: A concise introduction should present the subject of the paper in the context of previous studies in the discipline, and describe models or hypotheses tested.

Results and Discussion: This section may be subdivided by further headings or may be combined. Use section numbers for the first three levels of headings: 1., 1.1, 1.1.1.

Tables: Tables should be numbered consecutively with Arabic numerals in order of appearance in the text, and grouped at the end of the manuscript. Each table should be typed double-spaced on a separate page. A short descriptive caption should be typed directly above each table and any necessary footnotes (indicated by superscript lowercase letters) should be typed directly below the table. Extensive tables to be reproduced as photoengravings should be typed carefully in the exact camera-ready format desired.

Figures and Illustrations: Provide camera-ready figures and illustrations. Number and label concisely all illustrations on the back or bottom corners of the front, and group all at the end of the manuscript. A sharp image and good contrast are essential for quality reproductions; submit photographs for detailed halftones. Show only essential information on charts and graphs (i.e., coordinate axes, major grid lines, and lines of points of interest). A list of all legends should be typed consecutively on separate pages numbered at the end of the manuscript. Figures should be planned to fit the proportions of the printed page. Lettering on the original figure should be large enough to be legible after reduction (approximately 50% to 60%). Differences in type size within a single figure should be no more than approximately 15%. Characters should be approximately 3⁄16-in. high. (Leroy lettering size 140). Symbols used to identify points within a graph should be large enough that they will be easily distinguishable after reduction. Graphs should be plotted with generous margins in black ink on stiff white paper no larger than 8½ × 11 in. Grid lines that are to show must be inked in black. Halftones should be submitted not larger than 6½ × 4½ in.

For color illustrations, submit transparencies (35 mm) or glossy photographs. Do not submit photographs with silk or canvas finishes, or mounted on art boards. Make photographs the same size or larger than to be used in the journal, but no larger than 8½ × 11 in. If several illustrations are to be used, submit artwork all of the same size to be reduced at the same percentage. To indicate cropping, put slash marks using a black marking pen outside the area where the image is to be cropped. For transparencies, these marks can be made on the cardboard holder or on masks placed over the slide. Cost for printing color art is $1,200 per page.

Mathematics: Type mathematics exactly as it should appear. Numbers for displayed equations should be placed in parentheses at the right margin. Indicate best breaks for equations in case they will not fit on one line. In typewritten manuscripts, it will be assumed that all letters in displayed equations are to appear in italics unless marked as boldface or roman. Mark symbols in text as italics or boldface. If not marked, it will be assumed that letter symbols in text are to be set in roman type.

References: Only articles that have been published or are in press may be included in the references. Unpublished results (including articles submitted for publication) or personal communications should be cited as such within the text. Alphabetize the reference list by first author's last name. Cite references in text by author and date, for example: (Doe et al., 1982). Include the following information in the references:

1. Books: Authors' last names and initials, year, title, city, publisher, page numbers (if any). Example:
   Arnold, V. I. 1978. *Mathematical methods of classical mechanics*. New York: Springer-Verlag.
2. Chapters in book/papers in published proceedings: Authors' last names and initials, year, title of chapter/paper, title of book/proceedings, editors' initials and last names, city, publisher, page numbers (if any). Example:
   Edelman, G. M. 1978. Group selection and phasic reentrant signalling: a theory of higher brain function. In *The mindful brain*, edited by G. M. Edelman and V. B. Mountcastle. Cambridge: MIT Press, pp. 51–100.
3. Journal articles; Authors' last names and initials, year, title of article, name of journal, volume number, issue number (if any), page numbers. Example:
   Brooks, R. A. 1982. Symbolic error analysis and robot planning. *Internat. J. Robotics Res.* 1(4):29–36.
4. Technical reports: Authors' last names and initials, year, title of report, number of report, location and name of institution. Example:
   Mason, M. T. 1982. Manipulator grasping and pushing operations. AI-TR-690. Cambridge: Massachusetts Institute of Technology, Artificial Intelligence Laboratory.

For journal names, follow "Abbreviations of the Names of Serials," reviewed in *Mathematical Reviews* (reprinted from the December index, updated annually by the American Mathematical Society). In general authors should be guided by *A Manual for Authors of Mathematical Papers*, published in 1962 (revised 1984) by the American Mathematical Society, P.O. Box 6248, Providence, Rhode Island 02940.

SPECIAL SECTIONS

Correspondences: This section is open to succinct reports of important research findings. They should not exceed six double-spaced typewritten pages including tables and illustrations. Do not use any section headings except for Acknowledgments and References. All correspondence articles must have summaries. Illustrative materials should be kept to a minimum (usually not more than two or three tables and figures). Literature should be cited as for full-length papers. This section does not guarantee an accelerated means of publication, but is designed to offer the opportunity to present in brief form noteworthy results of works in progress and descriptions of evolving techniques and methods.

Industrial Applications: Designed to highlight the needs and interests of computational scientists in industrial settings, this section is open to full papers or brief reports covering work performed in an applied research environment.

Perspectives: At the discretion of the editors, this section will enable the publication of papers describing concepts, philosophical issues, or summary reports concerning advances being made in the field of scientific computation. Contributions should not exceed six double-spaced typewritten pages including tables and illustrations. Do not use any section headings except for Acknowledgment and References. Do not include a summary. Literature should be cited as for full-length papers.

PROOFS AND TIME OF PUBLICATION

Decisions on papers will be made as rapidly as possible. After an initial examination by the editors to confirm suitability of the article for the interdisciplinary readership of *Supercomputer Applications and High Performance Computing*, the editors will strive to have manuscripts reviewed promptly by an editorial board member and at least one other peer reviewer. When papers are accepted subject to revision, only a single revised version will be considered.

Proofs will be reviewed by the editors in consultation with the author within six months of acceptance of an article. It is necessary to regard accepted manuscripts as final texts to which no changes should be made. Any substantive changes made in the manuscripts to conform with the conventions of *Supercomputer Applications and High Performance Computing* will be submitted to authors for approval before the proof stage.