Multi Cell Simulator Library API Reference Manual

1

Author: Vishakha Gupta

Generated by Doxygen 1.4.7

Mon Apr 7 20:53:09 2008

Contents

Chapter 1

Multi Cell Simulator Library

Author:

Vishakha Gupta (vishakha@cc.gatech.edu)

1.1 About the project

The proliferation of cell based servers being in nascent stages, IBM has taken care of the developers by providing the IBM Cell Simulator codenamed Mambo. IBM Mambo is an excellent simulator platform for the IBM Cell processor, which provides the choice for running cycle accurate simulations as well as faster functional simulations. The need for proving the feasibility of generating and running parallel code on multiple interconnected Cell processors recommends the development of a functional network simulator linking multiple instances of the Mambo simulator, thus providing simulation support for a network of Cell processors.

The multi cell simulator(MCS) can be of tremendous importance to the research community that aims at developing and testing parallel applications on Cell based multiprocessor systems. It can also be of great advantage to the compiler research groups as a way to simulate different techniques and test their effectiveness.

The idea behind this project was to develop a multi-cell simulator taking into account the special hardware features and software functionalities that have been provided with the multi-core Cell architecture. An extension to this project would be the ability in the simulator to do latency analysis in order to enable developers to model the simulator statistics to the expected real world performance of the multi-cell simulator.

Apart from developing a tool for simulating an assembly of Cell processors, the goal behind the project has also been the design and development of a uniform library interface to enable efficient programming of this new architecture. The design goal has always been the simplicity of use of this new API so that programmers can abstract themselves away from the idiosyncracies of programming heterogeneous architectures.

More details about the project can be found at "http://www.cc.gatech.edu/∼vishakha/research/projects.php#MCS"

1.2 About the API

The API has been designed to take care of multiple communication possibilities exposed by the introduction of tightly coupled cores on the Cell processor. Although the nomenclature in this version of the API seems to derive completely from the socket communication paradigm, the send, receive and connect functions are essentially to copy data over, read data from and establish a direct channel between elements using the most suitable communication medium available. The API does take care of local fast path communication using DMA mechanisms if the communicating entities are on the same Cell board. The result of this simplication is that the programmer sees a uniform function space available for use irrespective of where the source and destination exist.

1.3 Future Work and Challenges

Implementing group communication operations in this system can prove to be a challenging task but it has already been addressed by previous research while designing the MPI group communication primitives and even prior to that in the references presented in the CCL work by researchers from IBM and Kendall Research Corp.

Another task of interest and use in the emerging network technologies would be introducing InfiniBand support as a communication backend in MCS. This would be a more useful tool in case latency analysis is provided based on interconnect.

For someone interested in optimization of algorithms, there is a lot of scope in collective operation primitives since this version of MCS, intended to be a proof of concept, implements the simplest algorithms for these tasks.

Chapter 2

Multi Cell Simulator Library API Documentation

2.1 cleanup.h File Reference

Function declarations for cleaning of data structures used in MCS.

Functions

- INT **mcs_destroy** ()
  Function to clean up all MCS data structures.

2.1.1 Detailed Description

Function declarations for cleaning of data structures used in MCS.

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.1.2 Function Documentation

2.1.2.1 INT mcs_destroy ()

Function to clean up all MCS data structures.

Once this function is called, all the state and other structures created by the library are destroyed. Use of any of the functions after calling this one can lead to undefined results. If an application wants to reuse other functions, it must call mcs_init again.

Returns:

SUCCESS if successful, error otherwise

2.2 config.h File Reference

Config constants and definitions required by the library.

Data Structures

- struct **el_info_t**

  Structure to store information about all network elements.

- **struct elems_count_t**
- **union conns_t**

  Union for master-slave connection information.

- **struct per_sim_info_t**

  Structure to store per simulator information for use by master etc. The instances can send their information using this struct to the master.

Defines

- #define **ELEM_TABLE_SIZE** 37
- #define **CONN_SOCKETS** 1
- #define **CONN_INFINIBAND** 2
- #define **MIN_SYS_PORT** 2000
- #define **MAX_SYS_PORT** 9050
- #define **IP_ADDR_LEN** 12

Typedefs

- typedef **el_info_t elem_info_t**

  Structure to store information about all network elements.

Functions

- **Q_NEW_HEAD** (elem_q, **el_info_t)**
- **H_NEW_TABLE (elem_table, elem_q, ELEM_TABLE_SIZE)**
- **INT config_init ()**

  Function to initialize all the config data structures.

- **INT get_all_elements ()**

  Find information about all elements in the cluster system.

- **INT get_config ()**

  Get simulation parameters.

- **elem_info_t $*$ query_elem_info (ELEMENT_ID element)**

  Find information about a specific element.

- **CHAR isValidElement (ELEMENT_ID element)**

  Returns if the given element id is valid in the system.

2.2.1 Detailed Description

Config constants and definitions required by the library.

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.2.2 Function Documentation

2.2.2.1 INT config_init ()

Function to initialize all the config data structures.

Returns:

SUCCESS or $<$ 0 when failed

2.2.2.2 INT get_all_elements ()

Find information about all elements in the cluster system.

This function populates a table consisting of information about all the communicable elements in the system using elem.info copied in root directory by the host.

If this code were to be run on real blades, this would have to be found by broadcast or there could still be a host code running which would function exactly like the one in the simulated system.

Returns:

SUCCESS or $<$ 0 when failed

2.2.2.3 INT get_config ()

Get simulation parameters.

This function queries the config parameters like number of threads simulator id etc. using sim.info copied in root directory by the simulator host.

If this were to be run on real cluster system, this could be asked again as user input and conveyed to other members participating in the operation

Returns:

SUCCESS or $<$ 0 when failed

2.2.2.4 elem_info_t∗ query_elem_info (ELEMENT_ID *element*)

Find information about a specific element.

This function uses the table populated by querying the host for information regarding an element, its address etc.

Parameters:

element Id of the element

Returns:

Address of element information, NULL if error

2.2.2.5 CHAR isValidElement (ELEMENT_ID *element*)

Returns if the given element id is valid in the system.

Parameters:

element Element to be verified

Returns:

TRUE or FALSE

2.3 connection.h File Reference

Header declaring connection management functions and structures.

Data Structures

- struct **c_info_t**

  Structure to store information about a particular connection.

Defines

- #define **CONN_TABLE_SIZE** 37
- #define **STAT_CREATED** 0x1
- #define **STAT_ESTABLISHED** 0x3
- #define **STAT_CLOSED** 0xffffffff
- #define **STAT_WAITING** 0x7
- #define **STAT_ERROR** 0xeeeeeeee
- #define **STAT_SENDING** 0xb
- #define **STAT_RECEIVING** 0x13
- #define **STAT_INPROGRESS** 0x1b
- #define **STAT_IDLE** 0x23

Typedefs

- typedef **c_info_t conn_info_t**

  Structure to store information about a particular connection.

Functions

- **Q_NEW_HEAD** (conn_q, **c_info_t)**
- **H_NEW_TABLE (conn_table, conn_q, CONN_TABLE_SIZE)**
- **INT init_conn ()**

  Initialize connection subsystem.

- **CONNECTION_ID create_conn_info (ELEMENT_ID receiver, INT recv_port, ELEMENT_-ID sender, INT send_port, void ∗proto_info, INT type)**

  Create connection entry in hashtable.

- **INT get_conn_info (CONNECTION_ID conn_id, conn_info_t ∗∗conn_info)**

  Function to query the status of a connection.

- **INT wait_for_conn (CONNECTION_ID conn_id)**

  Function to make an app wait explicitly for a communication to get over.

- **INT query_data_on_conn (CONNECTION_ID conn_id)**

  Query if data from a particular element/group has arrived.

- **INT close_conn (CONNECTION_ID conn_id)**
  Function to close a connection.
- **INT is_conn_alive (CONNECTION_ID conn_id)**
  See if a connection exists or not.
- **INT update_conn_stat (struct w_info_t ∗work, INT new_stat)**

2.3.1 Detailed Description

Header declaring connection management functions and structures.

Connection information is maintained for all active connections opened by the application to make the access faster

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.3.2 Typedef Documentation

2.3.2.1 typedef struct c_info_t conn_info_t

Structure to store information about a particular connection.

Todo

Add functions to access and modify members of this structure instead of allowing direct access to the application

2.3.3 Function Documentation

2.3.3.1 INT init_conn ()

Initialize connection subsystem.

Returns:

SUCCESS on successful connection, error otherwise

2.3.3.2 CONNECTION_ID create_conn_info (ELEMENT_ID *receiver*, INT *recv_port*, ELEMENT_ID *sender*, INT *send_port*, void ∗ *proto_info*, INT *type*)

Create connection entry in hashtable.

After connection has been established successfully it will be entered into the hashtable for future reference.

Parameters:

receiver Element id of the receiving element on some Cell processor

recv_port The port at which the element process/thread is listening

sender Element id of the sender element

send_port Port to be used for sending data

proto_info Protocol specific structure - will be null for LOCAL

status Status of connection

type Type of connection

Returns:

Id of the new connection

2.3.3.3 INT get_conn_info (CONNECTION_ID *conn_id*, conn_info_t ** *conn_info*)

Function to query the status of a connection.

Parameters:

conn_id Id for querying a specific connection

conn_info Pointer to structure which will be filled with relevant information

Returns:

SUCCESS if successful, error value otherwise

2.3.3.4 INT wait_for_conn (CONNECTION_ID *conn_id*)

Function to make an app wait explicitly for a communication to get over.

This function applies only to the non-blocking type of communication. The app can call this function to explicitly wait until some non-blocking communication completes. Can be thought of something like a barrier.

Parameters:

conn_id Id of connection on which the app should wait

Returns:

SUCCESS if the data send/recv completed, Error value otherwise

2.3.3.5 INT query_data_on_conn (CONNECTION_ID *conn_id*)

Query if data from a particular element/group has arrived.

This function checks to see if there is data from some element that has arrived. A value of ANY_-ELEMENT will indicate data from any element

Parameters:

conn_id ID of connection on which data is expected. Could specify ANY

Returns:

SUCCESS if arrived, Error otherwise

2.3.3.6 INT close_conn (CONNECTION_ID *conn_id*)

Function to close a connection.

This function takes care of closing down a connection and making sure the data structures involved are freed. If there is some pending communication related to this connection, the app needs to wait

Parameters:

conn_id Id of connection to be closed

Returns:

SUCCESS if connection closed, Error value otherwise

2.3.3.7 INT is_conn_alive (CONNECTION_ID *conn_id*)

See if a connection exists or not.

This function checks to see if the queried connection is present in the hashtable and alive

Parameters:

conn_id ID of connection on which data is expected

Returns:

TRUE if connection alive, FALSE otherwise

2.4 defines.h File Reference

Constants required by applications to pass as arguments.

Defines

- #define **LOCAL_SPE** 0xffffffea
- #define **LOCAL_PPE** 0xffffffeb
- #define **ANY_SPE** 0xfffffffa
- #define **ANY_PPE** 0xfffffffb
- #define **ALL_PPES** 0xfffffffc
- #define **ALL_SPES** 0xfffffffd
- #define **NONE 0xffffffff**
- **#define ELEM_PPE 1**
- **#define ELEM_SPE 2**
- **#define TYPE_RELIABLE 0x1**
- **#define TYPE_UNRELIABLE 0x2**
- **#define TYPE_STREAM 0x4**
- **#define TYPE_BLOCK 0x8**
- **#define TYPE_LOCAL 0x100**
- **#define TYPE_REMOTE 0x200**
- **#define DATA_INT 0x0**
- **#define DATA_CHAR 0x1**
- **#define DATA_FLOAT 0x2**
- **#define DATA_LONG 0x4**
- **#define DATA_DOUBLE 0x8**
- **#define DATA_COMPLEX 0x10**
- **#define BLOCKING 0x1**
- **#define NON_BLOCKING 0x2**
- **#define SPLIT_SEND 0x4**
- **#define SPLIT_RECV 0x40**
- **#define COMPLETE_SEND 0x8**
- **#define COMPLETE_RECV 0x800**
- **#define IN_ORDER 0x10**
- **#define OUT_OF_ORDER 0x20**
- **#define RELIABLE 0x100**
- **#define UNRELIABLE 0x200**
- **#define SIMPLE_STREAM 0x80**
- **#define RT_STREAM 0x400**
- **#define STREAM_LIB_ALLOC 0x1**
- **#define STREAM_APP_ALLOC 0x2**
- **#define STREAM_SIMPLE_ALLOC 0x3**
- **#define NO_TIMEOUT 0**
- **#define ALL_GROUPS 0x1**
- **#define ANY_GROUP 0x2**
- **#define GROUP_PPE 0x4**
- **#define GROUP_SPE 0x8**
- **#define GROUP_MIXED 0x10**
- **#define OPERATION_ADDSPE 0x1**

- **#define OPERATION_ADDPPE 0x2**
- **#define OPERATION_DELSPE 0x4**
- **#define OPERATION_DELPPE 0x8**
- **#define HUGE_PAGE_THRESHOLD (1024 ∗ 1024)**
- **#define RANK_MY_SPE 0x1**
- **#define RANK_MY_PPE 0x2**
- **#define RANK_MY_SIM 0x3**

2.4.1 Detailed Description

Constants required by applications to pass as arguments.

This header defines the constants that an application programmer may need while passing arguments to certain functions in the API

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.4.2 Define Documentation

2.4.2.1 #define NONE 0xffffffff

This option could be used in case user did not want to specify a value

2.4.2.2 #define ELEM_PPE 1

For app to specify that element to be used is a PPE

2.4.2.3 #define ELEM_SPE 2

For app to specify that element to be used is a SPE

2.4.2.4 #define TYPE_RELIABLE 0x1

The app expects a reliable connection (e.g. TCP)

2.4.2.5 #define TYPE_UNRELIABLE 0x2

The library need not worry about reliable delivery of data (e.g. UDP)

2.4.2.6 #define TYPE_STREAM 0x4

The connection will be used for sending a stream of data

2.4.2.7 #define TYPE_BLOCK 0x8

The connection will be used for sending a block of data

2.4.2.8 #define TYPE_LOCAL 0x100

Transfer to SPE or PPE on same machine (simulator)

2.4.2.9 #define TYPE_REMOTE 0x200

Transfer to SPE or PPE on remote machine

2.4.2.10 #define DATA_INT 0x0

Integer

2.4.2.11 #define DATA_CHAR 0x1

Char

2.4.2.12 #define DATA_FLOAT 0x2

Float

2.4.2.13 #define DATA_LONG 0x4

Long

2.4.2.14 #define DATA_DOUBLE 0x8

Double

2.4.2.15 #define DATA_COMPLEX 0x10

Data complex..This can be defined further in future

2.4.2.16 #define BLOCKING 0x1

Blocking call. A timeout could be specified to limit the time of blocking

2.4.2.17 #define NON_BLOCKING 0x2

Non-Blocking call. Worker threads can take care of the work and the app can later on query the status

2.4.2.18 #define SPLIT_SEND 0x4

Split the buffer data to create a stream instead of sending it in entirety

2.4.2.19 #define SPLIT_RECV 0x40

Receive a buffer in splits i.e. stream

2.4.2.20 #define COMPLETE_SEND 0x8

Send the entire buffer as one block

2.4.2.21 #define COMPLETE_RECV 0x800

Receive the entire buffer as one block

2.4.2.22 #define IN_ORDER 0x10

Used in case of stream communication to indicate that order of messages is important

2.4.2.23 #define OUT_OF_ORDER 0x20

Used in case of stream communication to indicate that order of messages is not important

2.4.2.24 #define RELIABLE 0x100

The app expects a reliable connection (e.g. TCP). As of now it is redundant since the connection creation takes care of it. Later on this may be useful if creating a connection were to be made optional

2.4.2.25 #define UNRELIABLE 0x200

The library need not worry about reliable delivery of data (e.g. UDP). As of now it is redundant since the connection creation takes care of it. Later on this may be useful if creating a connection were to be made optional

2.4.2.26 #define SIMPLE_STREAM 0x80

This is to indicate that the rate at which data is streamed is not important Currently this is the only option as far as streaming is concerned. But the later version of code may introduce teh concept of rate of streaming data

2.4.2.27 #define RT_STREAM 0x400

This flag means that the stream is expected to possess some real time guarantees. By real time guarantees it means that the stream should be sent at a certain rate. Whether each packet should be received or not, will be the choice left to the application and has to be specified by ORing another flag value

Todo

This functionality has not been implemented yet. So this flag is considered equivalent to the SIMPLE_-STREAM flag

2.4.2.28 #define STREAM_LIB_ALLOC 0x1

Let the library allocate suitable buffers

2.4.2.29 #define STREAM_APP_ALLOC 0x2

Application passes an array of buffers

2.4.2.30 #define STREAM_SIMPLE_ALLOC 0x3

Application passes one buffer to be split in stream - For sender side

2.4.2.31 #define NO_TIMEOUT 0

Timeout value in case there should be no timeout

2.4.2.32 #define ALL_GROUPS 0x1

Specify all groups to be considered for an operation

2.4.2.33 #define ANY_GROUP 0x2

Any group can be used

2.4.2.34 #define GROUP_PPE 0x4

A group of only PPEs

2.4.2.35 #define GROUP_SPE 0x8

A group of only SPEs. This does implicitly imply that any communication will have to go through the PPE responsible for the SPE since SPEs cannot communicate directly

2.4.2.36 #define GROUP_MIXED 0x10

A group comprising of SPEs and PPEs

2.4.2.37 #define OPERATION_ADDSPE 0x1

Add an SPE to the group

2.4.2.38 #define OPERATION_ADDPPE 0x2

Add a PPE to an existing group

2.4.2.39 #define OPERATION_DELSPE 0x4

Remove an SPE from an existing group

2.4.2.40 #define OPERATION_DELPPE 0x8

Remove a PPE from an existing group

2.4.2.41 #define HUGE_PAGE_THRESHOLD (1024 ∗ 1024)

Threshold to begin allocating huge pages for requested number of bytes if possible

2.4.2.42 #define RANK_MY_SPE 0x1

Option to indicate application is seeking rank of its first spe in the platform

2.4.2.43 #define RANK_MY_PPE 0x2

Option to indicate application is seeking rank of its first ppe in the platform

2.4.2.44 #define RANK_MY_SIM 0x3

Option to indicate application is seeking rank of its simulator instance in the platform

2.5 globals.h File Reference

Global constants and definitions required by the entire library.

Defines

- #define **SUCCESS** 0
- #define **TRUE** 1
- #define **FALSE** 0
- #define **LARGE_INT** 2147483647
- #define **DEBUG_LEVEL1** 1
- #define **DEBUG_LEVEL2** 2
- #define **DEBUG_LEVEL3** 3
- #define **DEBUG_LEVEL3** 4
- #define **print_l1**(str) printf((str));
- #define **print_l2**(str) printf((str));
- #define **print_l3**(str) printf((str));
- #define **print_l4**(str)

Typedefs

- typedef unsigned int **UINT**
- typedef int **INT**
- typedef INT **CONNECTION_ID**
- typedef INT **ELEMENT_ID**
- typedef INT **GROUP_ID**
- typedef INT **PLATFORM_ID**
- typedef char **CHAR**
- typedef char **BOOL**
- typedef double **TIME_T**

2.5.1 Detailed Description

Global constants and definitions required by the entire library.

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.6 group.h File Reference

Function declarations for Group functions.

Data Structures

- struct **g_info_t**

  Group information structure.

Defines

- #define **GROUP_TABLE_SIZE** 11

Typedefs

- typedef **g_info_t group_info_t**

  Group information structure.

Functions

- **Q_NEW_HEAD** (group_q, group)
- **H_NEW_TABLE** (group_table, group_q, GROUP_TABLE_SIZE)
- GROUP_ID **create_spe_group (ELEMENT_ID ∗spes, UINT num_spes)**

  Create a group of SPEs.

- **GROUP_ID create_ppe_group (ELEMENT_ID ∗ppes, UINT num_ppes)**

  Create a group of PPEs.

- **GROUP_ID create_group (ELEMENT_ID ∗ppes, UINT num_ppes, ELEMENT_ID ∗spes, UINT num_spes)**

  Create a group of SPEs and PPEs.

- **INT modify_group (GROUP_ID grp_id, INT oper_type, UINT num_ppes, UINT num_spes)**

  Modify an existing group.

- **INT destroy_group (GROUP_ID grp_id)**

  Destroy a group.

- **INT get_group_size (GROUP_ID grp_id, INT ∗num_spe, INT ∗num_ppe)**

  Get a count of elements in a group.

- **INT get_group_info (GROUP_ID grp_id, group_info_t grp_info)**

  Get all information about a group.

2.6.1 Detailed Description

Function declarations for Group functions.

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.6.2 Typedef Documentation

2.6.2.1 typedef struct g_info_t group_info_t

Group information structure.

Todo

Add functions to access/modify members of this structure instead of exposing it to the app threads

2.6.3 Function Documentation

2.6.3.1 GROUP_ID create_spe_group (ELEMENT_ID $*$ *spes*, UINT *num_spes*)

Create a group of SPEs.

The application programmer can specify the element ids of SPEs that can become a member of this group. If some or all of these SPEs are remote, the corresponding PPEs will be considered responsible for making sure that communication with these SPEs reaches them

The function chooses the most optimal set of SPEs to form the requested group in case the first argument is null.

Parameters:

spes List of SPE IDs

num_spes Number of SPEs desired in the group

Returns:

ID of the group created, Error if unsuccessful

2.6.3.2 GROUP_ID create_ppe_group (ELEMENT_ID $*$ *ppes*, UINT *num_ppes*)

Create a group of PPEs.

The application programmer can specify the element ids of the PPEs that can become a member of this group.

The function chooses the most optimal set of PPEs to form the requested group in case the first argument is null.

Parameters:

ppes List of PPE IDs

num_ppes Number of SPEs desired in the group

Returns:

ID of the group created, Error if unsuccessful

2.6.3.3 GROUP_ID create_group (ELEMENT_ID ∗ *ppes*, UINT *num_ppes*, ELEMENT_ID ∗ *spes*, UINT *num_spes*)

Create a group of SPEs and PPEs.

The application programmer can specify the element ids of the PPEs that can become a member of this group.

The function chooses the most optimal set of PPEs to form the requested group in case the first argument is null.

The application programmer can specify the element ids of SPEs that can become a member of this group. If some or all of these SPEs are remote, the corresponding PPEs will be considered responsible for making sure that communication with these SPEs reaches them

The function chooses the most optimal set of SPEs to form the requested group in case the third argument is null.

A value of 0 in either numbers would return an error

Parameters:

ppes List of PPE IDs
num_ppes Number of PPEs desired in the group
spes List of SPE IDs
num_spes Number of SPEs desired in the group

Returns:

ID of the group created, Error if unsuccessful

2.6.3.4 INT modify_group (GROUP_ID *grp_id*, INT *oper_type*, UINT *num_ppes*, UINT *num_spes*)

Modify an existing group.

The function adds or deletes SPEs and PPEs to change the requested group.

Todo

In future, the user can be given more control on specifying the desired elements in the group.

A value of 0 in either numbers is allowed here and the function will accordingly modify the group. This can change a total SPE or PPE group to a mixed group

Parameters:

grp_id Group to be modified
oper_type Type of operation to be performed
num_ppes Number of PPEs to be added/deleted to/from the group
num_spes Number of SPEs to be added/deleted to/from the group

Returns:

SUCCESS if successful, Error if unsuccessful

2.6.3.5 INT destroy_group (GROUP_ID *grp_id*)

Destroy a group.

Releases resources held by the group structure. This function blocks if any element in the group has pending communication and waits until they are all closed.

A value of ALL_GROUPS will destroy all the groups created by this application at any point in time.

Parameters:

grp_id Id of group to be destroyed

Returns:

SUCCESS if successful, Error if unsuccessful

2.6.3.6 INT get_group_size (GROUP_ID *grp_id*, INT ∗ *num_spe*, INT ∗ *num_ppe*)

Get a count of elements in a group.

Parameters:

grp_id Id of group for counting elements

num_spe Number of SPEs in the group

num_ppe Number of PPEs in the group

Returns:

SUCCESS if successful, Error if unsuccessful

2.6.3.7 INT get_group_info (GROUP_ID *grp_id*, group_info_t *grp_info*)

Get all information about a group.

Parameters:

grp_id Id of group to find

grp_info Pointer to structure where group information will be stored

Returns:

SUCCESS if successful, Error if unsuccessful

2.7 mcs_err.h File Reference

Error numbers used in the library.

Defines

- #define **ERR_MCS -1**
- **#define ERR_NOMEM -10**
- **#define ERR_DEFINE -11**
- **#define ERR_CONNECT -12**
- **#define ERR_INVALIDINTERCONNECT -14**
- **#define ERR_BIND -15**
- **#define ERR_SEND -18**
- **#define ERR_LISTEN -16**
- **#define ERR_RECEIVE -19**
- **#define ERR_LIBRARY -20**
- **#define ERR_CONNECTTIMEOUT -21**
- **#define ERR_CONNALIVE -22**
- **#define ERR_CONNINUSE -23**
- **#define ERR_TIMEDOUT -24**
- **#define ERR_CONNREFUSED -25**
- **#define ERR_HOSTUNREACH -26**
- **#define ERR_INVALIDFLAG -27**
- **#define ERR_INVALIDARG -28**
- **#define ERR_MEMBERS_UNAVAIL -29**
- **#define ERR_INVALID_GRPOPERATION -30**
- **#define ERR_INVALID_GROUP -31**
- **#define ERR_CONFIG -32**
- **#define ERR_PLATFORM -40**

2.7.1 Detailed Description

Error numbers used in the library.

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.7.2 Define Documentation

2.7.2.1 #define ERR_MCS -1

general error return

2.7.2.2 #define ERR_NOMEM -10

Whenever memory allocation returns null

2.7.2.3 #define ERR_DEFINE -11

error while defining/creating some structure

2.7.2.4 #define ERR_CONNECT -12

error while initiating a connection

2.7.2.5 #define ERR_INVALIDINTERCONNECT -14

when the interconnect type is not supported or incorrectly specified

2.7.2.6 #define ERR_BIND -15

Error binding the port of sending/receiving to the address of the sim instance

2.7.2.7 #define ERR_SEND -18

Error while sending data

2.7.2.8 #define ERR_LISTEN -16

Error listening on the port of receiving

2.7.2.9 #define ERR_RECEIVE -19

Error while waiting for some data

2.7.2.10 #define ERR_LIBRARY -20

Thread specific error value

2.7.2.11 #define ERR_CONNECTTIMEOUT -21

Connection timed out

2.7.2.12 #define ERR_CONNALIVE -22

If app tries to close IN_PROGRESS conn

2.7.2.13 #define ERR_CONNINUSE -23

If app tries to use a non-idle connection

2.7.2.14 #define ERR_TIMEDOUT -24

Connection request or data send/receive timed out

2.7.2.15 #define ERR_CONNREFUSED -25

The receiver refused to accept the connect request

2.7.2.16 #define ERR_HOSTUNREACH -26

The host being tried is not on the network

2.7.2.17 #define ERR_INVALIDFLAG -27

Invalid flag value specified

2.7.2.18 #define ERR_INVALIDARG -28

Data structure and function call related error numbers

2.7.2.19 #define ERR_MEMBERS_UNAVAIL -29

Error returned when the number elements desired in a group exceeds the number of elements present in the system

2.7.2.20 #define ERR_INVALID_GRPOPERATION -30

Error returned when modify_group specifies wrong number of parameters

2.7.2.21 #define ERR_INVALID_GROUP -31

Error returned for a non-existing group usage

2.7.2.22 #define ERR_CONFIG -32

Error returned due to file read error while retrieving config/elem info

2.7.2.23 #define ERR_PLATFORM -40

Platform errors

2.8 mcs_network.h File Reference

Header declaring available network functions.

Defines

- #define **DEFAULT_TIMEOUT** 2

Functions

- CONNECTION_ID **mcs_connect_receiver (ELEMENT_ID receiver, INT recv_port, INT conn_type)**

  Create connection for data accepting thread.

- **CONNECTION_ID mcs_connect_sender (ELEMENT_ID receiver, INT recv_port, ELEMENT_ID sender, INT send_port, INT conn_type, INT timeout)**

  Create connection for data sending thread.

- **INT mcs_connect_sender_group (GROUP_ID receivers, INT recv_port, ELEMENT_ID sender, INT base_send_port, INT conn_type, INT timeout)**

  Create connection for thread sending data to multiple receivers.

- **INT mcs_send (CONNECTION_ID conn_id, void ∗buff, UINT buff_size, INT data_type, INT timeout, UINT flags)**

  Send data once as a block to receiver element.

- **void ∗ mcs_create_simple_stream_info (UINT buff_size, UINT split_size, void ∗buff)**

  Create the stream information structure for simple stream send.

- **INT mcs_stream_send (CONNECTION_ID conn_id, void ∗info, INT data_type, INT timeout, UINT flags)**

  Send data in a stream to receiver element.

- **INT mcs_group_send (GROUP_ID grp_id, void ∗buff, UINT buff_size, INT data_type, INT timeout, UINT flags)**

  Sends packets between this source and multiple destinations.

- **INT mcs_group_stream_send (GROUP_ID grp_id, void ∗buff, UINT buff_size, UINT split_-size, INT data_type, INT timeout, UINT flags)**

  Establishes a stream between this source and multiple destinations.

- **INT mcs_recv (CONNECTION_ID conn_id, void ∗buff, UINT buff_size, INT timeout, UINT flags)**

  Receive data packet from sender.

- **void ∗ mcs_create_lib_stream_info (UINT alignment, void ∗share_info, UINT total_size, UINT split_size)**

  Create the stream information structure for library based allocation.

- **void ∗ mcs_create_app_stream_info (UINT chunk_size, UINT num_chunks, void ∗∗buffs)**

  Create the stream information structure for app based allocation.

- **INT mcs_stream_recv (CONNECTION_ID conn_id, void ∗info, INT timeout, UINT flags)**

  Receive a stream of data from sender.

- **INT mcs_group_recv (GROUP_ID grp_id, void ∗∗buff, UINT buff_size, INT timeout, UINT flags)**

  Receive data from multiple senders.

- **INT mcs_group_stream_recv (GROUP_ID grp_id, void ∗∗buff, UINT buff_size, INT timeout, UINT flags)**

  Receive a stream of data from multiple senders.

- **INT mcs_track_conn (CONNECTION_ID conn_id, conn_info_t ∗conn_info)**

  Function to query the status of a connection.

- **INT mcs_track_group_conn (GROUP_ID group_id, group_info_t ∗group_info)**

  Function to query the status of a group of connections.

- **INT mcs_wait (CONNECTION_ID conn_id)**

  Function to make an app wait explicitly for a communication to get over.

- **INT mcs_group_wait (GROUP_ID group_id)**

  Function to make an app wait explicitly for group communication.

- **INT mcs_close (CONNECTION_ID conn_id)**

  Function to close a connection.

- **INT mcs_group_close (GROUP_ID group_id)**

  Function to close connections for group members.

- **conn_info_t ∗ mcs_query_data (CONNECTION_ID c_id)**

2.8.1 Detailed Description

Header declaring available network functions.

These declarations are for the networking API available to the parallel applications

The functions declared in this file are the ones that the applications are expected to use for any form of communication between the elements on a Cell board whether they are on the same board or not. The body of these functions consists of making a decision whether the communication needs to be passed to a remote function or on the local fast path copy. If the communication is to be remote, there is an option of choosing function calls for the various interconnects here.

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.8.2 Function Documentation

2.8.2.1 CONNECTION_ID mcs_connect_receiver (ELEMENT_ID *receiver*, INT *recv_port*, INT *conn_type*)

Create connection for data accepting thread.

Function to create a connection information structure at the receiving end. This helps keep track of the various communication channels that exists and the connection id returned by the function can be used by the application to query status and other information. This also helps in avoiding the requirement of establishing communication channel multiple times if data has to be received multiple times.

Currently its required that the application thread call this function before it can start receiving data. In future, it can be made optional for block send/recv.

It also takes care of other requirements depending on the lower layer network protocol For e.g. It carries out calls to socket(), bind() and listen() in case of TCP/IP communication or socket() and bind() for UDP

Actions in case of certain values of element IDs: LOCAL_SPE or LOCAL_PPE = Either a fast path will be established or error will be returned in case the element cannot be used anymore

Todo

Make it wait for the SPE to be free if possible ANY_SPE or ANY_PPE = Library will choose the closest and easiest Specific ID = Will try to connect and return error if unsuccessful

The function doesn't explicitly check conn_type for errors and therefore it is the responsibility of the application programmer to make sure that it carries a correct value since the function will simply AND this value with the available constants to make decisions

Parameters:

receiver	Element id of the receiving element on some Cell processor
recv_port	The port at which the element process/thread is listening
conn_type	Type of connection (reliable/unreliable etc)

Returns:

Connection ID

2.8.2.2 CONNECTION_ID mcs_connect_sender (ELEMENT_ID *receiver*, INT *recv_port*, ELEMENT_ID *sender*, INT *send_port*, INT *conn_type*, INT *timeout*)

Create connection for data sending thread.

Function to create a connection information structure at the sender's end. This helps keep track of the various communication channels that exists and the connection id returned by the function can be used by the application to query status and other information. This also helps in avoiding the requirement of establishing communication channel multiple times if data has to be received sent times.

Currently its required that the application thread call this function before it can start sending data. In future, it can be made optional for block send/recv.

It also takes care of other requirements depending on the lower layer network protocol For e.g. It carries out calls to socket(), bind() and connect() in case of TCP/IP communication or socket() and bind() for UDP

Actions in case of certain values of element IDs: LOCAL_SPE or LOCAL_PPE = Either a fast path will be established or error will be returned in case the element cannot be used anymore

Todo

Make it wait for the SPE to be free if possible?? ANY_SPE or ANY_PPE = Library will choose the closest and easiest Specific ID = Will try to connect and return error if unsuccessful
REMOTE_SPE or REMOTE_PPE = Same action as above but try all possible options

The function doesn't explicitly check conn_type for errors and therefore it is the responsibility of the application programmer to make sure that it carries a correct value since the function will simply AND this value with the available constants to make decisions

Parameters:

receiver Element id of the receiving element on some Cell processor

recv_port The port at which the element process/thread is listening

sender Element id of the sender element

send_port Port to be used for sending data

conn_type Type of connection (reliable/unreliable etc).

timeout Timeout for connection in case of certain connections

Returns:

Connection ID to be used by functions for sending and receiving data Error code if unsuccessful

2.8.2.3 INT mcs_connect_sender_group (GROUP_ID *receivers*, INT *recv_port*, ELEMENT_ID *sender*, INT *base_send_port*, INT *conn_type*, INT *timeout*)

Create connection for thread sending data to multiple receivers.

Function to create connection information structures at the sender's end for multiple receivers. This helps keep track of the various communication channels that exists. The group id can be used by the application to query status and other information. This group id is returned when a group is created

Currently its required that the application thread call this function before it can start sending data. In future, it can be made optional for block send/recv.

It also takes care of other requirements depending on the lower layer network protocol For e.g. It carries out calls to socket(), bind() and connect() in case of TCP/IP communication or socket() and bind() for UDP

The function doesn't explicitly check conn_type for errors and therefore it is the responsibility of the application programmer to make sure that it carries a correct value since the function will simply AND this value with the available constants to make decisions

Parameters:

receivers Group id created using create_group call

recv_port The port at which the elements' processes/threads are listening

sender Element id of the sender element

base_send_port Port to be used for sending data from first element in the group. The rest of them use port numbers incrementally from the base port

conn_type Type of connection (reliable/unreliable etc)

timeout Timeout for connection in case of certain connections

Returns:

number of successful connections, Error code if unsuccessful

2.8.2.4 INT mcs_send (CONNECTION_ID *conn_id*, void ∗ *buff*, UINT *buff_size*, INT *data_type*, INT *timeout*, UINT *flags*)

Send data once as a block to receiver element.

This function sends data as a block between two elements in Cell The location of the element does not matter. It can transparently send data to some PPE or SPE which may not be on the same Cell board

The sending process is expected to have called **mcs_connect_sender**() (p. 27) with a non-stream connection type

Todo

This may be made optional in future

The server or receiver is expected to be listening already.

The data type value is for the use of the application. The library does not take care of anything in that respect. It could be used in future to write apps which transfer data in any format

Parameters:

conn_id Id of connection already established

buff The buffer pointing to data

buff_size Size of buffer in number of bytes irrespective of data type

data_type Datatype of elements forming the buffer

timeout Max time to wait until the data gets sent. Helps to avoid infinite waiting in case of blocking connection

flags Bitwise indication of whether send is blocking/non-blocking whether sequence of delivery is important or not etc Value like SPLIT_SEND and related are ignored

Returns:

Number of bytes sent. Error value otherwise.

2.8.2.5 void∗ mcs_create_simple_stream_info (UINT *buff_size*, UINT *split_size*, void ∗ *buff*)

Create the stream information structure for simple stream send.

When sending data as a stream, the app has a choice of allocating an array of buffers of a certain split_size or allocating a single buffer whose chunks can be sent given the chunk_size

This function creates information structure telling the library that the application has allocated as much memory it expects to need while sending data.

Parameters:

split_size Size of each split for the buffer

buff_size Size of the allocated buffer

buff Address of buffer to be split and sent

Returns:

Address of buffer to be passed for stream recv, NULL if error

2.8.2.6 INT mcs_stream_send (CONNECTION_ID *conn_id*, void ∗ *info*, INT *data_type*, INT *timeout*, UINT *flags*)

Send data in a stream to receiver element.

This function sends data as a stream between two elements in Cell The location of the element does not matter. It can transparently send data to some PPE or SPE which may not be on the same Cell board The application needs to specify the entire buffer that it wants to send and also the size of chunks that need to be sent.

The sending process is expected to have called **mcs_connect_sender()** (p. 27)

The server or receiver is expected to be listening already.

The flag value is assumed to indicate split here. Error will be returned if the split size will not split the buffer correctly

"The application code is also expected to have created the stream information by calling **mcs_create_-simple_stream_info()** (p. 29). If this is not done, the behavior of the function will be undefined"

Todo

May introduce a rate factor later in order to enable apps to specify something like send this buffer at a rate of nbytes/sec
Timeout parameter is yet to be used

Parameters:

conn_id Id of connection already established

info The buffer carrying information about the stream to be sent

data_type Datatype of elements forming the buffer

timeout Max time to wait until the data gets sent. Helps to avoid infinite waiting in case of blocking connection

flags Flags indicating things like blocking/non-blocking send etc

Returns:

Number of bytes sent. Error value otherwise.

2.8.2.7 INT mcs_group_send (GROUP_ID *grp_id*, void ∗ *buff*, UINT *buff_size*, INT *data_type*, INT *timeout*, UINT *flags*)

Sends packets between this source and multiple destinations.

This function can send the same buffer to all the elements in the group

The sending process is expected to have called **mcs_connect_sender_group()** (p. 28) with a non-stream connection type

Todo

This may be made optional in future

The function blocks until data has been queued up to be sent to all elements as of now even if the flags say NON_BLOCKING communication.

Parameters:

grp_id Id of group

buff The buffer pointing to start of data

buff_size Size of buffer in number of bytes irrespective of data type or the split size

data_type Datatype of elements forming the buffer

timeout Max time to wait until the data gets sent. Helps to avoid infinite waiting in case of blocking connection

flags Flags indicating things like blocking/non-blocking send etc

Returns:

SUCCESS if successful. Error value otherwise.

2.8.2.8 INT mcs_group_stream_send (GROUP_ID *grp_id*, void $*$ *buff*, UINT *buff_size*, UINT *split_size*, INT *data_type*, INT *timeout*, UINT *flags*)

Establishes a stream between this source and multiple destinations.

This function can send the same buffer to all the elements in the group Or it can send parts of the buffer to each one in order of the element IDs provided the buff_size $*$ num_elements in group = total size of buffer and flags & GROUP_SEND_PARTS > 0. The buff_size is expected to take care of the data type.

Error will be returned if the split size will not split the buffer correctly

The function blocks until data has been queued up to be sent to all elements as of now even if the flags say NON_BLOCKING communication.

Todo

May introduce a rate factor later in order to enable apps to specify something like send this buffer at a rate of nbytes/sec

Parameters:

grp_id Id of group

buff The buffer pointing to start of data

buff_size Size of buffer in number of bytes irrespective of data type or the split size

split_size Size of chunks for breaking buff into and sending at a time

data_type Datatype of elements forming the buffer

timeout Max time to wait until the data gets sent. Helps to avoid infinite waiting in case of blocking connection

flags Flags indicating things like blocking/non-blocking send etc

Returns:

SUCCESS if successful. Error value otherwise.

2.8.2.9 INT mcs_recv (CONNECTION_ID *conn_id*, void ∗ *buff*, UINT *buff_size*, INT *timeout*, UINT *flags*)

Receive data packet from sender.

This function receives data between two elements in Cell. The location of the element does not matter. In case the target is an SPE, the corresponding PPE calls this function and then signals the SPE which can then DMA data into wherever the application expects it.

The receiving process is expected to have called **mcs_connect_receiver()** (p. 27).

Todo

This may be made optional in future E.g. This function then calls recvfrom() in case of UDP connections

After the network specific headers are removed from the message, the received buffer will first carry the app-specific header which is metadata to help the receiving application to find some useful information about the received message like message sequence number (msg_tag), data type etc (see **globals.h** (p. 17) for the app_header structure)

The buffer is assumed to have been allocated by the application and buff_size carries the number of bytes allocated

Parameters:

conn_id Id of connection already established

buff The buffer where data is to be stored

buff_size Size of buffer in number of bytes irrespective of data type

timeout Max time to wait until the data is received. Helps to avoid infinite waiting in case of blocking connection

flags Flags indicating things like blocking/non-blocking receive etc

Returns:

Number of bytes received. Error value otherwise.

2.8.2.10 void∗ mcs_create_lib_stream_info (UINT *alignment*, void ∗ *share_info*, UINT *total_size*, UINT *split_size*)

Create the stream information structure for library based allocation.

When receiving data as a stream, the app has a choice of allocating an array of buffers of a certain split_size or passing the relevant information needed to allocate a buffer (like alignment info, size etc) to the library which can then use the metadata received from the sender to allocate the right amount of memory using the allocation functions defined in the library

This function creates information structure telling the library to allocate the required buffers as indicated by the received metadata from the sender

The application program can use predefined unit values (from **defines.h** (p. 11)) or pass a value of its own. The unit in this function will essentially signify the size of each chunk that will be used to receive a message

Parameters:

alignment Alignment requirements if any

share_info Address of sharing information

total_size Total size depending on size of buffer to be sent/received

split_size Size of each chunk that will be received

Returns:

Address of buffer to be passed for stream recv, NULL if error

2.8.2.11 void∗ mcs_create_app_stream_info (UINT *chunk_size*, UINT *num_chunks*, void ∗∗ *buffs*)

Create the stream information structure for app based allocation.

When receiving data as a stream, the app has a choice of allocating an array of buffers of a certain split_size or passing the relevant information needed to allocate a buffer (like alignment info, size etc) to the library which can then use the metadata received from the sender to allocate the right amount of memory using the allocation functions defined in the library

This function creates information structure telling the library that the application has allocated as much memory it expects to need while receiving data. The library code does not interfere in this case and received data is stored in the supplied buffers (in order of receiving or in the send sequence order depending on flag values used for communication)

The size of chunks is expected to be uniform for all received data chunks

Parameters:

chunk_size size of each chunk

num_chunks number of chunks allocated

buffs Array of buffers

Returns:

Address of buffer to be passed for stream recv, NULL if error

2.8.2.12 INT mcs_stream_recv (CONNECTION_ID *conn_id*, void ∗ *info*, INT *timeout*, UINT *flags*)

Receive a stream of data from sender.

This function receives data as a stream between two elements in Cell The location of the element does not matter.

The receiving process is expected to have called **mcs_connect_receiver()** (p. 27) to prepare an element for receiving data. In case the target is an SPE, the corresponding PPE calls this function and then signals the SPE which can then DMA data into wherever the application expects it. E.g. This function calls accept() in case of TCP connections

The receive function can find out about the split size of the buffer from the header attached to the message by the sender. Every receive is always preceded by the receive of a header which carries metadata about the the data being sent over. Alternately the application can fix the number of bytes to receive and how to receive them

The application should call one of the **mcs_create_lib_stream_info()** (p. 32) or the **mcs_create_app_-stream_info()** (p. 33) functions for getting the information structure which will then be used to decipher the placement of received stream data by the library

The status information like number of bytes received etc can be queried using the connection id.

Parameters:

conn_id Id of connection already established

info The buffer carrying information about storing the stream

timeout Max time to wait until the data is received. Helps to avoid infinite waiting in case of blocking connection

flags Flags indicating things like blocking/non-blocking receive etc

Returns:

Number of bytes received. Error value otherwise.

2.8.2.13 INT mcs_group_recv (GROUP_ID *grp_id*, void ** *buff*, UINT *buff_size*, INT *timeout*, UINT *flags*)

Receive data from multiple senders.

The receiving process is expected to have called **mcs_connect_receiver()** (p. 27).

Todo

This may be made optional in future E.g. This function then calls recvfrom() in case of UDP connections

After the network specific headers are removed from the message, the received buffer will first carry the app-specific header which is metadata to help the receiving application to find some useful information about the received message like message sequence number (msg_tag), data type etc (see **globals.h** (p. 17) for the app_header structure)

The function blocks until receive requests have been queued up for all elements as of now even if the flags say NON_BLOCKING communication.

Parameters:

grp_id Id of groups from which data is to be received

buff The array of buffers where data is to be stored

buff_size Size of each buffer in number of bytes irrespective of data type VVV This could be an array of sizes in future

timeout Max time to wait until the data is received. Helps to avoid infinite waiting in case of blocking connection

flags This parameter is ignored for the time being but kept for future use

Returns:

SUCCESS on success. Error value otherwise.

2.8.2.14 INT mcs_group_stream_recv (GROUP_ID *grp_id*, void ** *buff*, UINT *buff_size*, INT *timeout*, UINT *flags*)

Receive a stream of data from multiple senders.

The receiving process is expected to have called mcs_connect_receiver_group(). E.g. This function then calls accept() for example in case of TCP connections

The behavior of this function is similar to mcs_stream_recv except that it performs the receive from multiple senders

After the network specific headers are removed from the message, the received data will first carry the app-specific header which is metadata to help the receiving application to find some useful information about the received message like message sequence number (msg_tag), data type etc (see **globals.h** (p. 17) for the app_header structure)

The function blocks until receive requests have been queued up for all elements as of now even if the flags say NON_BLOCKING communication.

Parameters:

grp_id Id of groups from which data is to be received

buff The array of buffers where data is to be stored

buff_size Size of each buffer in number of bytes irrespective of data type

Todo

This could be an array of sizes in future

Parameters:

timeout Max time to wait until the data is received. Helps to avoid infinite waiting in case of blocking connection

flags This parameter is ignored for the time being but kept for future use

Returns:

SUCCESS on success. Error value otherwise.

2.8.2.15 INT mcs_track_conn (CONNECTION_ID *conn_id*, conn_info_t ∗ *conn_info*)

Function to query the status of a connection.

This function will return information that could be relevant to the application about a connection like its status, time of sending data, number of bytes sent or received etc.

Parameters:

conn_id Id for querying a specific connection

conn_info Pointer to structure which will be filled with relevant information The structure is defined in **connection.h** (p. 7)

Todo

Add functions to access and set various members of struct

Returns:

SUCCESS if successful, error value otherwise

2.8.2.16 INT mcs_track_group_conn (GROUP_ID *group_id*, group_info_t ∗ *group_info*)

Function to query the status of a group of connections.

This function will return information that could be relevant to the application about a group of connections like its status, time of sending data, number of bytes sent or received etc.

Parameters:

group_id Id for querying a specific connection

group_info Pointer to structure which will be filled with relevant information The structure is defined in **group.h** (p. 18)

Todo

Add functions to access and set various members of struct

Returns:

SUCCESS if successful, error value otherwise

2.8.2.17 INT mcs_wait (CONNECTION_ID *conn_id*)

Function to make an app wait explicitly for a communication to get over.

This function applies only to the non-blocking type of communication. The app can call this function to explicitly wait until some non-blocking communication completes. Can be thought of something like a barrier.

Parameters:

conn_id Id of connection on which the app should wait

Returns:

SUCCESS if the data send/recv completed, Error value otherwise

2.8.2.18 INT mcs_group_wait (GROUP_ID *group_id*)

Function to make an app wait explicitly for group communication.

This function applies only to the non-blocking type of communication. The app can call this function to explicitly wait until the non-blocking communication with all members of the given group completes. Can be thought of something like a barrier.

Parameters:

group_id Id of group on which the app should wait

Returns:

SUCCESS if the data send/recv completed, Error value otherwise

2.8.2.19 INT mcs_close (CONNECTION_ID *conn_id*)

Function to close a connection.

This function takes care of closing down a connection and making sure the data structures involved are freed. If there is some pending communication related to this connection, the function blocks

Parameters:

conn_id Id of connection to be closed

Returns:

SUCCESS if connection closed, Error value otherwise

2.8.2.20 INT mcs_group_close (GROUP_ID *group_id*)

Function to close connections for group members.

This function takes care of closing down connections and making sure the data structures involved are freed. If there is some pending communication related to this connection, the function blocks

Parameters:

group_id Group id for the group whose connections are to be closed

Returns:

SUCCESS if all connections closed, Error value otherwise

2.9 mcs_stream.h File Reference

Header declaring structures for handling stream send/recv.

Data Structures

- struct **s_lib_info_t**

  The following variables will be needed for calling the library allocation functions in case the application chooses the option of memory allocation by the library.

- **struct s_app_info_t**

  The following fields are needed in case app allocates an array of buffers each of a constant chunk size.

- **struct s_stream_t**

  This structure can be used by the sender to either allocate a simple buffer for sending data or to use the app_stream_info_t structure to emulate scatter gather kind of effect. In simpler terms, this can be used for 1D data and the app_stream_info_t can be used for 2D data if it has to be sent row by row.

- **struct s_info_t**

  The structure which will be used by the application to pass the stream buffer information. The function mcs_create_stream_info() can be used to create this structure by passing the relevant parameters.

Typedefs

- typedef **s_lib_info_t lib_stream_info_t**

  The following variables will be needed for calling the library allocation functions in case the application chooses the option of memory allocation by the library.

- **typedef s_app_info_t app_stream_info_t**

  The following fields are needed in case app allocates an array of buffers each of a constant chunk size.

- **typedef s_stream_t simple_stream_info_t**

  This structure can be used by the sender to either allocate a simple buffer for sending data or to use the app_stream_info_t structure to emulate scatter gather kind of effect. In simpler terms, this can be used for 1D data and the app_stream_info_t can be used for 2D data if it has to be sent row by row.

- **typedef s_info_t stream_info_t**

  The structure which will be used by the application to pass the stream buffer information. The function mcs_create_stream_info() can be used to create this structure by passing the relevant parameters.

Functions

- UINT **get_stream_total_len (stream_info_t ∗info)**

  Get the total length of stream.

- **UINT get_stream_split_len (stream_info_t ∗info)**

  Get the length per part of stream.

2.9.1 Detailed Description

Header declaring structures for handling stream send/recv.

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.9.2 Typedef Documentation

2.9.2.1 typedef struct s_stream_t simple_stream_info_t

This structure can be used by the sender to either allocate a simple buffer for sending data or to use the app_stream_info_t structure to emulate scatter gather kind of effect. In simpler terms, this can be used for 1D data and the app_stream_info_t can be used for 2D data if it has to be sent row by row.

Todo

Currently the receive functionality does not include arranging 1D data in 1D format as on the sender side. But this can be added later

2.9.3 Function Documentation

2.9.3.1 UINT get_stream_total_len (stream_info_t ∗ *info*)

Get the total length of stream.

Use the stream data structure to retrieve the total length of the stream

Parameters:

info Pointer to stream information

Returns:

Length of stream, 0 otherwise

2.9.3.2 UINT get_stream_split_len (stream_info_t ∗ *info*)

Get the length per part of stream.

Use the stream data structure to retrieve individual part length of the stream especially in case of a scatter-gather kind of stream

Parameters:

info Pointer to stream information

Returns:

Length of stream part, 0 otherwise

2.10 mem_mgr.h File Reference

Header declaring available memory management functions.

Functions

- void ∗ **mcs_alloc (size_t size, void ∗share_info)**
  Allocate a memory area as desired by the application thread.
- **void ∗ mcs_alloc_align (size_t size, INT alignment, void ∗share_info)**
  Allocate an aligned memory area as desired by the application.
- **void ∗ mcs_platform_alloc_align (PLATFORM_ID plid, size_t chunk_size, INT alignment)**
  Allocate memory keeping in mind the platform abstraction.
- **INT mcs_dealloc (void ∗area)**
  Deallocate a memory region.
- **INT mcs_dealloc_align (void ∗area)**
  Deallocates an aligned memory region.

2.10.1 Detailed Description

Header declaring available memory management functions.

These declarations are for the memory APIs available to the parallel applications

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.10.2 Function Documentation

2.10.2.1 void∗ mcs_alloc (size_t *size*, void ∗ *share_info*)

Allocate a memory area as desired by the application thread.

This function allocates num_bytes using large pages by default for buffers of size greater than HUGE_-PAGE_THRESHOLD.

This function calls malloc without worrying about the alignment issues otherwise.

Parameters:

size The number of bytes to be allocated.

share_info The information about elements sharing and the permissions given to them. Currently ignored.

Returns:

Void pointer to the buffer allocated if successful, else NULL

2.10.2.2 void∗ mcs_alloc_align (size_t *size*, INT *alignment*, void ∗ *share_info*)

Allocate an aligned memory area as desired by the application.

This function makes sure that memory is aligned. The expected use of this function is in case for memory allocation for SPE's use.

This function returns an error if the alignment is not a power of 2

Parameters:

size The number of bytes to be allocated.

alignment The unit of alignment like 4B aligned etc

share_info The information about elements sharing and the permissions given to them. Currently ignored.

Returns:

Void pointer to the buffer allocated if successful, else NULL

2.10.2.3 void∗ mcs_platform_alloc_align (PLATFORM_ID *plid*, size_t *chunk_size*, INT *alignment*)

Allocate memory keeping in mind the platform abstraction.

Parameters:

plid ID of associated platform

chunk_size Size of data going to each spe in Bytes

alignment Desired alignment for buffers

Returns:

Address of contiguous buffer which can then be divided among the different spes

2.10.2.4 INT mcs_dealloc (void ∗ *area*)

Deallocate a memory region.

This function will require that the passed memory region be allocated by making a call to **mcs_alloc()** (p. 40) in future. As of now it simply makes a call to free() on the buffer passed

Parameters:

area Pointer to area that is to be freed

Returns:

SUCCESS when successful, error otherwise

2.10.2.5 INT mcs_dealloc_align (void ∗ *area*)

Deallocates an aligned memory region.

This function will require that the passed memory region be allocated by making a call to **mcs_alloc_-align()** (p. 41) in future.

Parameters:

area Pointer to area that is to be freed

Returns:

SUCCESS when successful, error otherwise

2.11 platform.h File Reference

Function declarations for platform view of accelerator pool.

Data Structures

- struct **pthread_data**

  ppu spu thread data

- **struct context_data_t**

  Data for creating spe threads. Could be used for other accelerators.

- **struct acc_info_t**

  Data for specific accelerator.

Defines

- #define **MAX_PLATFORMS** 5
- #define **SPE_COUNT_ALL_SPES** 10
- #define **SPE_COUNT_MY_SPES** 11
- #define **SPE_COUNT_ALL_PPES** 12
- #define **SPE_COUNT_MY_PPES** 13
- #define **NUM_ELEM_TYPES** 2
- #define **PLT_MASTER_PORT** 9001
- #define **PLT_SLAVE_PORT** 9040

Typedefs

- typedef void ∗ **program_handle_t**
- typedef void ∗ **context_ptr_t**
- typedef **pthread_data pthread_data_t**

  ppu spu thread data

Functions

- INT **mcs_platform_init** ()

  Initialize platform data structures.

- **PLATFORM_ID mcs_platform_create (acc_info_t ∗acc_list, INT num_elems)**

  Create data structures needed for a platform.

- **INT mcs_platform_destroy (PLATFORM_ID pid)**

  Destroy all state from the platform including spe contexts.

- **INT mcs_platform_cpu_info_get (PLATFORM_ID plat_id, UINT info_requested, INT cpu_-node)**

Query basic CPU properties and resources.

- **INT mcs_platform_context_create (PLATFORM_ID plid, pthread_data_t ∗datas, UINT flags, void ∗other_params)**

  Create context for all elements in the platform.

- **INT mcs_platform_program_load (PLATFORM_ID plid, program_handle_t ∗acc_exec)**

  Load the executable.

- **INT mcs_platform_thread_create (PLATFORM_ID plid, pthread_attr_t ∗attr, void ∗(∗start_-routine)(void ∗), void ∗thread_pkg)**

  Create the accelerator threads.

- **INT mcs_platform_in_mbox_send (PLATFORM_ID plid, UINT ∗∗mbox_data, INT count, UINT behavior) INT mcs_platform_out_mbox_recv(PLATFORM_ID plid**

  Send mbox messages to all spes in the platformReceive mbox messages from all spes in the platform.

- **INT mcs_platform_acc_wait (PLATFORM_ID plid, void ∗∗value_ptr)**

  Wait for accelerator threads to finish executing.

- **INT mcs_platform_context_destroy (PLATFORM_ID plid)**

  Destroy any associated context from platform.

- **BOOL isValidPlatform (PLATFORM_ID plid)**

  Check if the given platform id is valid.

- **INT mcs_platform_get_rank (PLATFORM_ID plid, UINT info_requested)**

  Get rank of required element in the calling simulator instance.

Variables

- INT UINT ∗∗ **mbox_data**
- INT UINT INT **count**

2.11.1 Detailed Description

Function declarations for platform view of accelerator pool.

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu) Unknown`

2.11.2 Typedef Documentation

2.11.2.1 typedef struct pthread_data pthread_data_t

ppu spu thread data

This structure contains the information needed by a ppu program to access an spu using pthreads

Could let application define this structure too so long as the basic fields remain common or have a common interface that the library can use to fill in pthread and speid info.

2.11.3 Function Documentation

2.11.3.1 INT mcs_platform_init ()

Initialize platform data structures.

THIS FUNCTION MUST BE CALLED BEFORE USING ANY OF THE PLATFORM CALLS

Responsible for initializing platform structures

Returns:

SUCCESS on successful init, error otherwise

2.11.3.2 PLATFORM_ID mcs_platform_create (acc_info_t ∗ *acc_list*, INT *num_elems*)

Create data structures needed for a platform.

For now, a cell specific description: This method contacts the other simulators if required and prepares for creating context on as many spes as requested After call to this function, the programmer can just use the platform id to refer to this platform and it will take care of the elements to be involved. The number of spes is based on the usable spe_count collected at **mcs_init()** (p. 51) stage from all slaves by the master.

Todo

Introduce some kind of a check to make sure those number of of spes are actually present and if the context cannot be created for any reason, have some fallback.

Parameters:

acc_list List of accelerators and/or main processors to be involved

num_acc Number of different accs - This number should make sense

Returns:

Platform id if successful, error otherwise

2.11.3.3 INT mcs_platform_destroy (PLATFORM_ID *pid*)

Destroy all state from the platform including spe contexts.

This function is responsible for the cleanup of all platform data structures and must be called before repetitive use of create.

Parameters:

pid Id of platform to be destroyed

Returns:

SUCCESS if successful, error otherwise

2.11.3.4 INT mcs_platform_cpu_info_get (PLATFORM_ID *plat_id*, UINT *info_requested*, INT *cpu_node*)

Query basic CPU properties and resources.

Similar to the function provided by libSPE with additional flags The possible values for info_requested are defined in **defines.h** (p. 11)

Parameters:

plat_id Id of the platform being queried

info_requested Specifies the type of information requested

cpu_node Specifies the node for which the information is requested

Returns:

Value of the parameter requested

2.11.3.5 INT mcs_platform_context_create (PLATFORM_ID *plid*, pthread_data_t ∗ *datas*, UINT *flags*, void ∗ *other_params*)

Create context for all elements in the platform.

Performance vs. simplicity issue: Could have created a hash of spe_context_ptr_t vs actual location info But choosing to introduce a new data structure which will have the ptr as a part of it

Stores the context where it belongs Uses the platform id to indicate what spes reside on the machine

Todo

The interpretation of all flag options and gang are not here yet

Parameters:

plid Id of the platform spes belong to

datas Points to the user allocated data structure

flags The flags required by spe_context_create in this case

other_params Other params as may be needed for creating executable on the accelerators. Its the gang ptr in case of spes

Returns:

SUCCESS when successful, error otherwise

2.11.3.6 INT mcs_platform_program_load (PLATFORM_ID *plid*, program_handle_t ∗ *acc_exec*)

Load the executable.

This function is responsible for loading the executables that need to run on the accelerators, spes in this case. The program handle is converted to that for spes in this implementation

Parameters:

plid Id of the platform

acc_exec Pointer to the accelerator executable (currently casted to void ∗)

Returns:

SUCCESS when successful, error otherwise

2.11.3.7 INT mcs_platform_thread_create (PLATFORM_ID *plid*, pthread_attr_t ∗ *attr*, void ∗(∗)(void ∗) *start_routine*, void ∗ *thread_pkg*)

Create the accelerator threads.

This is the function that actually loops for all local spes to be used in the platform, creates their context and calls the thread create function. The other functions like context_create etc declared above, just store the parameters required to create an spe context and store required information in the platform struct. Using just one function to do all this saves looping multiple times for related tasks. The reason to create separate functions is mainly to maintain its similarity with libspe2 API.

This function has a pthread_attr_t argument right now. But this can be abstracted away if the thread_pkg structure is defined

Parameters:

plid Platform id

attr As for now pthread_attr

start_routine The thread function

thread_pkg Optional thread package to use (a struct). Default = pthreads

Returns:

number of spe threads created, error otherwise

2.11.3.8 INT mcs_platform_in_mbox_send (PLATFORM_ID *plid*, UINT ∗∗ *mbox_data*, INT *count*, UINT *behavior*)

Send mbox messages to all spes in the platformReceive mbox messages from all spes in the platform.

This is totally tied to the cell platform but it could very well be replaced with a wrapper around the more generic mcs_recv for fast path.

Parameters:

plid Id of associated platform

mbox_data Pointer to array of values to be read per spe

count Number of messages expected per spe

Returns:

Total number of messages read

2.11.3.9 INT mcs_platform_acc_wait (PLATFORM_ID *plid*, void ∗∗ *value_ptr*)

Wait for accelerator threads to finish executing.

This function is like a barrier across threads of execution running on the accelerators, in this case spes

Parameters:

plid Associated platform id
value_ptr This is typically for return status of the threads

Returns:

SUCCESS when successful, error otherwise

2.11.3.10 INT mcs_platform_context_destroy (PLATFORM_ID *plid*)

Destroy any associated context from platform.

This again is a concept pretty tied to the way SPEs are programmed but can be extended to general destruction of the context defining accelerators

Parameters:

plid Associated platform id

Returns:

SUCCESS when successful, error otherwise

2.11.3.11 BOOL isValidPlatform (PLATFORM_ID *plid*) `[inline]`

Check if the given platform id is valid.

Parameters:

plid Queried platform id

Returns:

TRUE if platform id is valid, FALSE otherwise

2.11.3.12 INT mcs_platform_get_rank (PLATFORM_ID *plid*, UINT *info_requested*)

Get rank of required element in the calling simulator instance.

Return information like the rank of PPE or SPE or overall simulator rank among the corresponding elements forming the overall platform

Parameters:

plid Associated platform id
info_requested Particular element for which rank requested

Returns:

Desired rank, error otherwise

2.12 ppe_spe.h File Reference

Declares structures and functions for PPU-SPU interaction.

Data Structures

- union **addr64**

  This union helps clarify calling parameters between the PPE and the SPE.

Functions

- INT **recv_data (void ∗buff, UINT buff_size, INT mode)**

  SPE function to receive data from PPE.

- **INT send_data (void ∗buff, UINT buff_size, INT mode)**

  SPE function to receive data from PPE.

2.12.1 Detailed Description

Declares structures and functions for PPU-SPU interaction.

The functions are here as prototypes but have not been implemented yet. But these are intended for the local ppe-spe interactions

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.12.2 Function Documentation

2.12.2.1 INT recv_data (void ∗ *buff*, UINT *buff_size*, INT *mode*)

SPE function to receive data from PPE.

This function will implement the DMA operations to pull data based on the metadata supplied in the control block passed to the SPE while creating an SPE thread. It takes care of the sync issues

This function can then be called from within the app_func() which is the application specific functionality expected from the SPE.

Parameters:

buff The buffer where data is to be loaded in local store

buff_size The size of local store area allocated for the buff

mode DMA buffering mode to be used

Returns:

Number of bytes read, error otherwise

2.12.2.2 INT send_data (void * *buff*, UINT *buff_size*, INT *mode*)

SPE function to receive data from PPE.

This function will implement the DMA operations to send data

This function can then be called from within the app_func() which is the application specific functionality expected from the SPE.

Parameters:

buff The buffer from where data is to be sent to main memory

buff_size The size of local store area allocated for the buff

mode DMA buffering mode to be used

Returns:

Number of bytes sent, error otherwise

2.13 startup.h File Reference

The initialization requirements are declared in this file.

Functions

- INT **mcs_init (BOOL need_platform, BOOL need_slaves)**
  Function to initialize the Multi Cell library.

2.13.1 Detailed Description

The initialization requirements are declared in this file.

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.13.2 Function Documentation

2.13.2.1 INT mcs_init (BOOL *need_platform*, BOOL *need_slaves*)

Function to initialize the Multi Cell library.

This function will take care of creating thread pools and other structures related to the working of the library

This function will make a call to the query function declared below. So the apps are not required to query config parameters after calling init.

Calling init more than once will return an error

THIS FUNCTION MUST BE CALLED BEFORE USING ANY OF THE LIBRARY APIs

Parameters:

need_platform	If platform calls will be used in which case init them

Returns:

SUCCESS if all structures successfuly initialized, error otherwise

2.14 timing.h File Reference

Timing related declarations.

Functions

- TIME_T **mcs_get_sim_time ()**
  Retrieve the current time from simulator instance.
- **TIME_T mcs_get_host_time ()**
  Retrieve the current time from the host OS.
- **TIME_T mcs_get_app_time ()**
  Retrieve the time for which the application has been running.
- **INT mcs_get_group_time (GROUP_ID grp_id, TIME_T ∗mean, TIME_T ∗std_dev)**
  Retrieve the app time from group elements.
- **INT mcs_sync_group_time (GROUP_ID grp_id, TIME_T ∗mean, TIME_T ∗std_dev)**
  Synchronize the app time for group elements.
- **void mcs_set_app_alarm (TIME_T for_time)**
  Set an alarm for given time.
- **void mcs_set_sim_alarm (TIME_T for_time)**
  Set an alarm for given time.
- **TIME_T mcs_measure_rtt (UINT src_sim_id, UINT dst_sim_id)**
  Measure the round trip time between two simulators.

2.14.1 Detailed Description

Timing related declarations.

Author:

Vishakha Gupta (`vishakha@cc.gatech.edu`)

2.14.2 Function Documentation

2.14.2.1 TIME_T mcs_get_sim_time ()

Retrieve the current time from simulator instance.

Returns:

Time representation. NULL on error

2.14.2.2 TIME_T mcs_get_host_time ()

Retrieve the current time from the host OS.

Returns:

Time representation. NULL on error

2.14.2.3 TIME_T mcs_get_app_time ()

Retrieve the time for which the application has been running.

This is the time from the beginning of the application after it makes a call to initialize the library. So this can be viewed as the virtual time

Returns:

Time representation. NULL on error

2.14.2.4 INT mcs_get_group_time (GROUP_ID *grp_id*, TIME_T ∗ *mean*, TIME_T ∗ *std_dev*)

Retrieve the app time from group elements.

This function retrieves timing information from all elements in the group, adjusts the network delay approximately and returns the mean and standard deviation in the timing in the group. It considers the application virtual time.

This function can be used in advanced distributed kind of applications to get a common notion of time on the app instances running on different Cell instances for running distributed algorithms

Parameters:

grp_id Id of group under consideration

mean Pointer to mean value of the app/virtual time in group

std_dev Pointer to standard deviation in the time values

Returns:

SUCCESS if successful. Error return value on error

2.14.2.5 INT mcs_sync_group_time (GROUP_ID *grp_id*, TIME_T ∗ *mean*, TIME_T ∗ *std_dev*)

Synchronize the app time for group elements.

This function sets the timing information for all elements in the group, taking into account the network delay approximately. It considers the application virtual time.

This function can be used in advanced distributed kind of applications to get a common notion of time on the app instances running on different Cell instances for running distributed algorithms

Parameters:

grp_id Id of group under consideration

mean Mean value of the app/virtual time in group

std_dev Standard deviation in the time values

Returns:

SUCCESS if successful. Error return value on error

2.14.2.6 void mcs_set_app_alarm (TIME_T *for_time*)

Set an alarm for given time.

This alarm uses time from the beginning of the application to specify the time for which the calling process/thread should be put to sleep

VVV the ideal version would be something like a signal so that the app can continue running and is then sent an event when the alarm expires

This could be useful for some backup sort of app for example

Parameters:

for_time Time for which alarm is to be set

Returns:

Void

2.14.2.7 void mcs_set_sim_alarm (TIME_T *for_time*)

Set an alarm for given time.

This alarm uses time from the simulator to specify the time for which the calling process/thread should be put to sleep

VVV the ideal version would be something like a signal so that the app can continue running and is then sent an event when the alarm expires

This could be useful for some backup sort of app for example or a timed barrier indicating the app shouldn't wait anymore for some task

Parameters:

for_time Time for which alarm is to be set

Returns:

Void

2.14.2.8 TIME_T mcs_measure_rtt (UINT *src_sim_id*, UINT *dst_sim_id*)

Measure the round trip time between two simulators.

Useful when decision needs to be made regarding choosing the components in say a platform

Parameters:

src_sim_id Id of the source simulator

dst_sim_id Id of the destination simulator

Returns:

The measured round trip time

Chapter 3

Todo

3.1 Todo List

Class c_info_t (p. **??**) Add functions to access and modify members of this structure instead of allowing direct access to the application

Class g_info_t (p. **??**) Add functions to access/modify members of this structure instead of exposing it to the app threads

Class s_stream_t (p. **??**) Currently the receive functionality does not include arranging 1D data in 1D format as on the sender side. But this can be added later

Global RT_STREAM (p. **14**) This functionality has not been implemented yet. So this flag is considered equivalent to the SIMPLE_STREAM flag

Global modify_group (p. **20**) In future, the user can be given more control on specifying the desired elements in the group.

Global mcs_connect_receiver (p. **27**) Timeout has not been used for any of these functions as of now. That is on the task list for next level implementation

REMOTE_SPE or REMOTE_PPE = Same action as above but try all possible options

Global mcs_connect_receiver (p. **27**) Implement a generic seg fault handler to take care of any buffer errors that may result from application code

Global mcs_connect_receiver (p. **27**) There is quite some missing functionality and untested components: 1. The stream send receive need to be tested 2. There is no code for local send/receive (mailbox mostly) or group 3. The switches like ANY, LOCAL etc have not been taken care of 4. Latency functions are missing. For the most part, comments indicate what is missing at what point Most common functions have been implemented.

Global mcs_connect_receiver (p. **27**) Make it wait for the SPE to be free if possible ANY_SPE or ANY_PPE = Library will choose the closest and easiest Specific ID = Will try to connect and return error if unsuccessful

Global mcs_connect_sender (p. **27**) Make it wait for the SPE to be free if possible?? ANY_SPE or ANY_PPE = Library will choose the closest and easiest Specific ID = Will try to connect and return error if unsuccessful

REMOTE_SPE or REMOTE_PPE = Same action as above but try all possible options

Global mcs_send (p. **29**) This may be made optional in future

Global mcs_stream_send (p. **30**) May introduce a rate factor later in order to enable apps to specify something like send this buffer at a rate of nbytes/sec

Timeout parameter is yet to be used

Global mcs_group_send (p. **30**) This may be made optional in future

Global mcs_group_stream_send (p. **31**) May introduce a rate factor later in order to enable apps to specify something like send this buffer at a rate of nbytes/sec

Global mcs_recv (p. **32**) This may be made optional in future E.g. This function then calls recvfrom() in case of UDP connections

Global mcs_group_recv (p. **34**) This may be made optional in future E.g. This function then calls recvfrom() in case of UDP connections

Global mcs_group_stream_recv (p. **34**) This could be an array of sizes in future

Global mcs_track_conn (p. **35**) Add functions to access and set various members of struct

Global mcs_track_group_conn (p. **36**) Add functions to access and set various members of struct

Global mcs_platform_create (p. **45**) Introduce some kind of a check to make sure those number of of spes are actually present and if the context cannot be created for any reason, have some fallback.

Global mcs_platform_context_create (p. **46**) The interpretation of all flag options and gang are not here yet