Chapter 4.

Setting Environment Variables

This chapter describes the variables that specify the environment under which your MPI programs will run. Environment variables have predefined values. You can change some variables to achieve particular performance objectives; others are required values for standard−compliant programs.

4.1 Setting MPI Environment Variables on UNICOS and IRIX Systems

This section provides a table of MPI environment variables you can set for IRIX systems only, and a table of environment variables you can set for both UNICOS and IRIX systems. For environment variables for UNICOS/mk systems, see Section 4.2.

Table 2. MPI environment variables for IRIX systems only

Variable	Description	Default
`MPI_BUFS_PER_HOST`	Determines the number of shared message buffers (16 KB each) that MPI is to allocate for each host. These buffers are used to send long messages.	16 pages (eac is 16 KB)
`MPI_BYPASS_DEVS`	Sets the order for opening HIPPI adapters. The list of devices does not need to be space−delimited (0123 is also valid). An array node usually has at least one HIPPI adapter, the interface to the HIPPI network. The HIPPI bypass is a lower software layer that interfaces directly to this adapter. The bypass sends MPI control and data messages that are 16 Kbytes or shorter. When you know that a system has multiple HIPPI adapters, you can use the `MPI_BYPASS_ DEVS` variable to specify the adapter that a program opens first. This variable can be used to ensure that multiple MPI programs distribute their traffic across the available adapters. If you prefer not to use the HIPPI bypass, you can turn it off by setting the `MPI_BYPASS_OFF` variable. When a HIPPI adapter reaches its maximum capacity of four MPI programs, it is not available to additional MPI programs. If all HIPPI adapters are busy, MPI sends internode messages by using TCP over the adapter instead of the bypass.	0 1 2 3
`MPI_BYPASS_OFF`	Disables the HIPPI bypass.	Not enabled
`MPI_DSM_OFF`	Turns off nonuniform memory access (NUMA) optimization in the MPI library.	Not enabled
`MPI_DSM_MUSTRUN`	Specifies the CPUs on which processes are to run. For jobs running on IRIX systems, you can set the `MPI_DSM_VERBOSE` variable to request that the `mpirun` command print information about where processes are executing.	Not enabled
`MPI_DSM_PPM`	Sets the number of MPI processes that can be run on each node of an IRIX system.	2
`MPI_DSM_VERBOSE`	Instructs `mpirun` to print information about process placement for jobs running on NUMA systems.	Not enabled
`MPI_MSGS_PER_HOST`	Sets the number of message headers to allocate for MPI messages on each MPI host. Space for messages that are destined for a process on a different host is allocated as shared	128

memory on the host on which the sending processes are located. MPI locks these pages in memory. Use the `MPI_MSGS_PER_HOST` variable to allocate buffer space for interhost messages.

Caution: If you set the memory pool for interhost packets to a large value, you can cause allocation of so much locked memory that total system performance is degraded.

Table 3. MPI environment variables for UNICOS and IRIX systems

Variable	Description	Default
`MPI_ARRAY`	Sets an alternative array name to be used for communicating with Array Services when a job is being launched.	The default n set in the `arrayd.con`
`MPI_BUFS_PER_PROC`	Determines the number of private message buffers (16 KB each) that MPI is to allocate for each process. These buffers are used to send long messages.	16 pages (eac is 16 KB)
`MPI_CHECK_ARGS`	Enables checking of MPI function arguments. Segmentation faults might occur if bad arguments are passed to MPI, so this is useful for debugging purposes. Using argument checking adds several microseconds to latency.	Not enabled
`MPI_COMM_MAX`	Sets the maximum number of communicators that can be used in an MPI program. Use this variable to increase internal default limits. (May be required by standard–compliant programs.)	256
`MPI_DIR`	Sets the working directory on a host. When an `mpirun` command is issued, the Array Services daemon on the local or distributed node responds by creating a user session and starting the required MPI processes. The user ID for the session is that of the user who invokes `mpirun`, so this user must be listed in the `.rhosts` file on the responding nodes. By default, the working directory for the session is the user's `$HOME` directory on each node. You can direct all nodes to a different directory (an NFS directory that is available to all nodes, for example) by setting the `MPI_DIR` variable to a different directory.	`$HOME` on the If using `-np` (`-nt`, the defau the current directory.
`MPI_GROUP_MAX`	Sets the maximum number of groups that can be used in an MPI program. Use this variable to increase internal default limits. (May be required by standard–compliant programs.)	256
`MPI_MSGS_PER_PROC`	Sets the maximum number of buffers to be allocated from sending process space for outbound messages going to the same host. (May be required by standard–compliant programs.) MPI allocates buffer space for local messages based on the message destination. Space for messages that are destined for local processes is allocated as additional process space for the sending process.	128
`MPI_REQUEST_MAX`	Sets the maximum number of simultaneous nonblocking sends and receives that can be active at one time. Use this variable to increase internal default limits. (May be required by standard–compliant programs.)	1024
`MPI_TYPE_DEPTH`	Sets the maximum number of nesting levels for derived datatypes. (May be required by standard–compliant programs.) The `MPI_TYPE_DEPTH` variable limits the maximum depth of derived datatypes that an application can create. MPI logs error messages if the limit specified by `MPI_TYPE_DEPTH` is exceeded.	8 levels
`MPI_TYPE_MAX`	Sets the maximum number of derived data types that can be used in an MPI program. Use this variable to increase internal	1024

default limits. (May be required by standard−compliant programs.)

4.2 Setting MPI Environment Variables on UNICOS/mk Systems

This section provides a table of MPI environment variables you can set for UNICOS/mk systems.

Table 4. Environment variables for UNICOS/mk systems

Variable	Description	Default
MPI_SM_POOL	Specifies shared memory queue. When MPI is started, it allocates a pool of shared memory for use in message passing. This pool represents space used to buffer message headers and small messages while the receiving PE is doing computations or I/O. The default of 1024 bytes is the number of bytes that can be pending.	1024 bytes
MPI_SM_TRANSFER	Specifies number of queue slots. Specifies the number of slots in the shared memory queue that can be occupied by a send operation at the receiver. A slot consists of four UNICOS/mk words. By default, a single send operation can occupy 128 slots (or buffer 512 words) while the receiving PE is doing computations or I/O.	128
MPI_BUFFER_MAX	Specifies maximum buffer size. Specifies a maximum message size, in bytes, that will be buffered for MPI standard, buffered, or ready send communication modes.	No limit
MPI_BUFFER_TOTAL	Specifies total buffer memory. Specifies a limit to the amount of memory the MPI implementation can use to buffer messages for MPI standard, buffered, or ready send communication modes.	No limit

4.3 Internal Message Buffering in MPI (IRIX Systems Only)

An MPI implementation can copy data that is being sent to another process into an internal temporary buffer so that the MPI library can return from the MPI function, giving execution control back to the user. However, according to the MPI standard, you should not assume any message buffering between processes because the MPI standard does not mandate a buffering strategy. Some implementations choose to buffer user data internally, while other implementations block in the MPI routine until the data can be sent. These different buffering strategies have performance and convenience implications.

Most MPI implementations do use buffering for performance reasons and some programs depend on it. Table 5, illustrates a simple sequence of MPI operations that cannot work unless messages are buffered. If sent messages were not buffered, each process would hang in the initial `MPI_Send` call, waiting for an `MPI_Recv` call to take the message. Because most MPI implementations do buffer messages to some degree, often a program such as this will not hang. The `MPI_Send` calls return after putting the messages into buffer space, and the `MPI_Recv` calls get the messages. Nevertheless, program logic such as this is not valid by the MPI standard. The Silicon Graphics implementation of MPI for IRIX systems buffers messages of all sizes. For buffering purposes, this implementation recognizes short message lengths (64 bytes or shorter) and long message lengths (longer than 64 bytes).

Table 5. Outline of improper dependence on buffering

Process 1	Process 2
MPI_Send(2,....)	MPI_Send(1,....)
MPI_Recv(2,....)	MPI_Recv(1,....)