COS786: Parallel and Distributed Computing

MPI: Environment, Errors and Profiling
Environment Settings

- Certain environment settings should be checked to:
  - Increase portability amongst MPI implementations
  - Maintain resilience against MPI implementation upgrades
Environment Settings

- MPI_GET_VERSION(version, subversion)
  - (OUT)version
  - (OUT)subversion

- This method may be called before MPI_INIT.
Environment Settings

- Attributes that describe the environment are attached to MPI_COMM_WORLD on initialization.

- Useful attributes include:
  - MPI_TAG_UB
  - MPI_IO
  - MPI_WTIME_IS_GLOBAL
Environment Settings

- Tag values (MPI_TAG_UB)
  - Range in value from 0 to MPI_TAG_UB, inclusive.
  - MPI_TAG_UB is at least 32767.
  - The upper bound is implementation-dependent.
Environment Settings

• I/O Rank (MPI_IO)
  • This is the rank of the process that can provide language-standard I/O operations.
  • Not all calls may return the same value.
  • If all processes provide I/O then MPI_ANY_SOURCE is returned.
  • If no process provides I/O then MPI_PROC_NULL is returned.
Environment Settings

- Synchronized time (MPI_WTIME_IS_GLOBAL)
  - Boolean flag to indicate whether or not the clocks across hosts are synchronized.
  - Synchronization must be explicitly achieved.
  - Variation is limited to 0.5 times the duration of travel by a zero-length message from A to B to A.
  - If a message is sent from A to B, then $T_A < T_B$
  - The attribute may be assigned to communicators other than MPI_COMM_WORLD.
Environment Settings

- MPI_ATTR_GET(comm, key, value, flag)
  - (IN)comm : communicator
  - (IN)key : attribute key
  - (OUT)value : attribute value
  - (OUT)flag : true if value was obtained, false if no attributed [key] exists.

- This method is deprecated in MPI-2.
Environment Settings

- `MPI_COMM_GET_ATTR(comm, key, value, flag)`
  - (IN)comm : communicator
  - (IN)key : attribute key
  - (OUT)value : attribute value
  - (OUT)flag : true if value was obtained, false if no attributed [key] exists.
Environment Settings

- **MPI_GET_PROCESSOR_NAME(name, result_len)**
  - (OUT)name: The name of the physical host.
  - (OUT)result_len: The length of [name]
- Returns the name of the processor upon which it was called.
- [name] must provide for up to MPI_MAX_PROCESSOR_NAME characters (including the null-terminator in C/C++)
Error Handling

- An MPI implementation may choose not to handle certain errors. The implementation may even be unable to handle some errors.
- Any errors that an MPI implementation handles are implementation-dependent.
- MPI errors generate an MPI exception.
Error Handling

- An error handler is associated with a communicator.
- The error handling routine will be used for any MPI exception that occurs as a result of an operation involving that communicator.
- Operations that do not involve a communicator are associated with MPI_COMM_WORLD.
- Error handlers are local; different processes may attach different handlers to the same communicator.
Error Handling

- A new communicator inherits the error handler of its parent (or template) communicator.

- As such, it is possible to create a global error handler by associating a handler with MPI_COMM_WORLD immediately after initialization.
Error Handling

• Predefined error handlers:
  • MPI_ERRORS_ARE_FATAL
  • MPI_ERRORS_RETURN

• Different MPI implementations may provide additional predefined error handlers.

• Users may create their own error handlers.
Error Handling

- MPI_ERRORS_ARE_FATAL is associated with MPI_WORLD_COMM by default.
- If a user wishes to test individual operations and respond accordingly to errors then MPI_ERRORS_RETURN may be used.
- However, fatal errors can be too harsh and checking all return codes too fine-grained so a proper, global error handler is recommended.
Error Handling

- Once an error is detected, the state of MPI is undefined.
- MPI does not guarantee that the program may continue after an error is detected.
- The real purpose of error handlers is to provide custom error strings as well as the opportunity to shut down the program gracefully.
- However, a good quality implementation will isolate the effects of certain errors and allow the continuation of the program.
Error Handling

- MPI_ERRHANDLER_CREATE(function, errhandler)
  - (IN)function : user-defined procedure
  - (OUT)errhandler: handle to the error handler
Error Handling

- MPI_ERRHANDLER_SET(comm, errhandler)
  - (IN)comm : communicator handle
  - (IN)errhandler : error handler to associate with the communicator
- Sets the communicator's error handler at the calling process only.
Error Handling

- **MPI_ERRHANDLER_GET**(comm, errhandler)
  - (IN)comm : communicator handle
  - (OUT)errhandler : handle to the error handler associated with [comm]
Error Handling

- MPI_ERRHANDLER_FREE(errhandler)
  - (IN/OUT)errhandler : handle to the error handler to be freed
- The resulting value of [errhandler] is MPI_ERRHANDLER_NULL.
- The error handler will be deallocated once all associated communicators have also been deallocated.
Error Handling

- **MPI_ERROR_STRING(errorcode, string, resultlen)**
  - *(IN)*errorcode : the code returned by an MPI operation
  - *(OUT)*string : The corresponding text.
  - *(OUT)*resultlen : Length of [string].
- [string] must provide for up to MPI_MAX_ERROR_STRING characters of storage, including the null-terminator for C/C++. 
Error Handling

• An MPI implementation may determine any of the error codes except for MPI_SUCCESS.
• This allows the implementation to provide detailed error messages.
• So that an application may interpret the custom error codes, MPI_ERROR_CLASS translates an error code into one of a number of predefined error classes.
Error Handling

- **MPI_ERROR_CLASS(errorcode, errorclass)**
  - *(IN)*errorcode : code returned by an MPI operation
  - *(OUT)*errorclass : class associated with [errorcode]
Error Handling

• Some error classes include:
  • MPI_SUCCESS: No error.
  • MPI_ERR_RANK: Invalid rank.
  • MPI_ERR_COMM: Invalid communicator.
Process Termination

- **MPI_ABORT(comm, errorcode)**
  - (IN)comm : handle to communicator for which tasks should be aborted
  - (IN)errorcode : code to be returned to the environment of the invoking process

- The error code should be provided so that the OS may interpret the reason for the abort.

- While this operation is intended for dynamic process management, MPI does not require that only the specified processes be aborted.
Timers and Synchronization

- MPI defines a timer.
- Timers in POSIX are inconvenient and may not provide a high-resolution timing source.
Timers and Synchronization

- **DOUBLE PRECISION MPI_WTIME()**
  - Returns the number of seconds of wall-clock time that have elapsed since some time in the past.
  - The past time is guaranteed not to change during the lifetime of the process.
  - The times returned are local to the node that issued the call to MPI_WTIME.
  - There is no requirement that all nodes return the same time.
Timers and Synchronization

- **DOUBLE PRECISION MPI_WTICK()**
  - Returns the resolution of MPI_WTIME in seconds.
  - The result represents the number of seconds that elapse between each clock tick.
MPI Profiling Interface

- MPI provides an interface that allows for the creation of profiling tools.
- The interface makes no assumptions about how it is used.
- No specifications exist for what data are collected or how they are processed.
MPI Profiling Interface

- The profiling interface provides access to a third party by means of name shifted operations.
- MPI operations that begin with the prefix “MPI_” may be accessed also by operations that begin with the prefix “PMPI_”.
MPI Profiling Interface

- Example: collect information about MPI_SEND

```c
int MPI_SEND(void* buffer, 
              const int count, 
              MPI_Datatype datatype, int dest, 
              int tag, MPI_Comm comm)
{
    double start_time = MPI_WTIME();
    int result = PMPI_SEND(...);
    ReportTotalTime(MPI_WTIME() + start_time);
    return result;
}
```
MPI Profiling Interface

- User code must be able to control the profiler during runtime. For instance, to enable/disable the profiler at specific points in the computation.
- The MPI_PCONTROL operation is provided for this purpose.
MPI Profiling Interface

- MPI_PCONTROL(level, …)
  - (IN)level : the profiling level
- The level is profiler-dependent but is recommended to be: 0 (off), 1 (normal), 2 (flush).
- Ordinary MPI does not make use of this operation, which is implemented as a no-op.
- The default state, after MPI_INIT, is recommended to be wherein profiling is enabled at its normal level.
MPI Profiling Interface

- Complication: Multiple Counting
  - Parts of the MPI library may be implemented in terms of basic MPI operations.
  - Due to name shifting, a profiling function could be called from within an MPI function that was initially called by a profiling function.
  - So that such detail, where useful, is not eliminated, the MPI specification ignores this problem.
  - It is up to the profiler author to determine how to handle this situation, be it single-threaded or multi-threaded.