COS786: Parallel and Distributed Computing

MPI: Process Creation and Management
Introduction

- MPI is designed primarily for inter-process communication.
- Nevertheless, process management is a useful ability with application in, amongst others:
  - Task farms
  - Serial applications with parallel modules
  - Runtime assessment of process requirements
MPI Goals

- Overall, to maintain portability.
- Specifically, the process model must apply to the majority of runtime environments.
- Must not take over OS responsibilities.
- Must provide an interface to OS services.
- Must guarantee communication determinism.
- Must not contain features that compromise performance.
MPI Goals

• The goals are achieved in two ways:
  • MPI does not manage the runtime environment
  • MPI maintains a consistent concept of a communicator:
    – Communicators are not changed once created.
    – Communicators are created by means of deterministic, collective operations.
Dynamic Process Model Capabilities

- Processes may be created and terminated after an MPI application has started.
- Communication between existing processes and new processes can be established.
- Communication between unrelated MPI applications can be established.
Process Creation

- Runtime environments vary greatly.
- Therefore, MPI makes no assumptions about the runtime environment.
- No OS services are exposed. For example, MPI cannot:
  - Find out which processes are running
  - Kill arbitrary processes
  - Query the environment for capabilities.
Process Creation

- Complex interaction with the runtime environment is done through a specific interface.
- To keep MPI tidy and to prevent its functionality from being compromised by environment-specific interfaces, MPI provides the `info` argument in many operations.
- Of course, using `info` implies a loss of portability.
Process Creation

- MPI does not require a virtual machine model to be present. This implies that:
  - There is no global view of an MPI application and the OS.
  - Processes spawned by one task may not be visible to another.
  - Additional hosts added to the runtime environment may not be detected.
  - Tasks spawned by processes may not be automatically distributed over all processes.
Process Creation

- Interaction between MPI and the runtime environment is limited to the following:
  - Operations that spawn new processes,
  - Use of the info argument to specify where and how to start a process,
  - Interrogation of attribute MPI_UNIVERSE_SIZE to determine the size of the runtime environment.
Process Creation

- MPI_COMM_SPAWN(command, argv, maxprocs, info, root, comm, intercomm, array_of_errcodes)
Process Creation

• (IN)command
  • A string containing the name of the program to be spawned.
  • MPI does not specify how the program is found or how the working directory is determined.
  • The implementation determines what is most appropriate for the environment.
Process Creation

• (IN)argv
  • Array of strings that are passed as arguments to the program, if supported by the OS.
  • argv[0] is the first argument and not the program name, as is convention in some environments.
  • argv must be NULL-terminated so that its size can be determined.
  • Passing MPI_ARGV_NULL as argv results in main's argc being 1 and main's argv[0] containing the program name.
Process Creation

• (IN)maxprocs
  • The number of processes to spawn. An error of class MPI_ERR_SPAWN is raised if the number cannot be achieved.
  • The info argument may contain a set of numbers of acceptable processes.
  • If less than maxprocs processes were created, reasons are given in the array_of_errcodes parameter.
Process Creation

- (IN)info
  - Handle to a container of type MPI_Info in C and MPI::Info in C++.
  - Contains (key, value) pairs that describe additional information on how a process should be started.
  - May be MPI_INFO_NULL when not required and when maximum portability is desired.
Process Creation

- Reserved keys for **info**:
  - *host*: hostname; format determined by implementation.
  - *arch*: architecture; valid range and meanings determine by implementation.
  - *wdir*: working directory of spawned processes.
  - *path*: where to look for the binary.
  - *file*: where to find additional information.
  - *soft*: alternatives allowed for maxprocs.
Process Creation

- (IN)root
  - rank of the process in the local group (determined by `comm`) that examines all arguments preceding `root`.
  - All other processes ignore the arguments that precede `root`.
Process Creation

- (IN)comm
  - Intracommunicator for the local group of spawning processes.
Process Creation

- (OUT)intercomm
  - Intercommunicator for the remote group of newly spawned processes.
Process Creation

• (OUT)array_of_errcodes
  
  Is filled with the result of the spawning of each process, up to maxprocs.

  If all processes are created then each element will contain MPI_SUCCESS.

  If \( m < \) maxprocs processes are created then \( m \) elements will contain MPI_SUCCESS and the rest will contain an implementation-specific error, of class MPI_ERR_SPAWN.

• MPI does not specify which element corresponds with which process. \texttt{info} may be used for this.
Process Creation

- In C, one may pass MPI_ERRCODES_IGNORE if one is not interested in the results.
- In C++, the argument may be omitted.
Process Creation

• Important considerations:
  • It is possible to start up an entire application by starting one process and then spawning all the other processes. This is discouraged due to performance degradation that may result.
  • Communication may be established before the child processes call MPI_INIT. The return of the spawn operation does not imply that all children are initialized but the resulting intercommunicator may be used.
Process Creation

- If the program that is spawned does not call MPI_INIT but forks another process that does, the result is undefined.
- If the program that is supplied to a spawn operation is a shell script, the result is undefined.
Process Creation (Multiple Binaries)

- MPI_COMM_SPAWN is sufficient for most cases.
- Where multiple processes should be started with different binaries, or the same binary with different arguments, MPI_COMM_SPAWN_MULTIPLE is necessary.
Process Creation (Multiple Binaries)


Spawns processes from multiple binaries, or the same binary with different arguments, establishes communication with them and places them in the same MPI_WORLD_COMM.
Process Creation (Multiple Binaries)

- (IN)count
  - Number of commands.
  - Significant to MPI only at root.
  - Determines the length of array_of_argv so a non-root process may fail if root is non-positive in the non-root process.
Process Creation (Multiple Binaries)

- (IN)array_of_commands
  - Array of strings that represent the names of binaries to execute.
  - Significant only at root.
Process Creation (Multiple Binaries)

- (IN)array_of_argv
  - Array of strings that represent the arguments for the binaries to be executed.
  - Significant only at root.
  - If some commands take no arguments then place a NULL in the first element of the corresponding sub-array.
  - Use MPI_ARGV_NULL to pass no arguments.
  - Placing MPI_ARGV_NULL instead of a sub-array is undefined.
Process Creation (Multiple Binaries)

- (IN)array_of_maxprocs
  - Array of integers that represents the maximum number of processes to start for each command.
  - Significant only at root.
Process Creation (Multiple Binaries)

- (IN)array_of_info
  - Array of handles to info structures that contain additional, environment-specific information for each command.
  - Significant only at root.
Process Creation (Multiple Binaries)

• (IN)root
  • The rank of the process in which all previous arguments are significant.
Process Creation (Multiple Binaries)

- (IN)comm
  - Handle to the local intracommunicator that represents the spawning processes.
Process Creation (Multiple Binaries)

- (OUT)intercomm
  - Intercommunicator between the newly spawned group of processes and the local group.
Process Creation (Multiple Binaries)

- (OUT)array_of_errcodes
  - Array of error codes – one per process.
Process Creation (Multiple Binaries)

- Important considerations:
  - MPI_COMM_SPAWN creates a new MPI_WORLD_COMM each time it is called.
  - MPI_COMM_SPAWN_MULTIPLE assigns all of the child processes to the same MPI_WORLD_COMM.
  - Use MPI_COMM_SPAWN_MULTIPLE whenever possible to reduce overhead.
  - In certain MPI implementations, communication between processes created separately is slower than that between processes created simultaneously.