Anthony Christoforou

Assignment 1: An In-Depth Exploration of Parallelism via MPI

Abstract

The elementary "Hello World" MPI program serves as an introductory vehicle to comprehend parallel computing concepts. This report explicates the functionalities embedded within each MPI function call, problems faced during the library installation, and the observable outcomes when deploying the program on different numbers of processors.

Introduction

Utilizing the Message Passing Interface (MPI) library, the program establishes a parallel computing environment, identifies the characteristics of participating processes, and generates a 'Hello World' message enriched with these details. This exercise affords insights into the initialization, identification, and graceful termination of parallel processes.

Challenges Encountered

Upon attempting to integrate the MPI library into my development environment, I faced a hurdle concerning the software's path recognition. The issue necessitated specifying the full path to the library, which added an extra layer of complexity to the initial setup.

In-Depth Analysis of MPI Function Calls

`MPI_Init`: Initializing the MPI Ecosystem

```
MPI_Init(NULL, NULL);
```

- **Objective**: Sets up the MPI environment, a mandatory first step in any MPI-based application.
- **Parameters**: Conventionally, command-line arguments are inserted here; nonetheless, `NULL` values suffice for this example.
- **Impact**: Allows for the employment of additional MPI operations.

`MPI_Comm_size`: Procuring Total Process Count

```
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
```

- **Objective**: Quantifies the number of processes within a specific group—here, denoted by `MPI_COMM_WORLD`.

- **Parameters**: Utilizes `MPI_COMM_WORLD` for global representation and a pointer to an integer to hold the size.
- **Impact**: `world_size` retains the cumulative number of processes.

`MPI_Comm_rank`: Assigning Unique Process IDs

```
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
```

- **Objective**: Allocates a unique identification number, referred to as 'rank,' to each process.
- **Parameters**: Takes `MPI_COMM_WORLD` for the process group and an integer pointer to hold the rank.
- **Impact**: `world_rank` holds the identification for each process within the group.

`MPI_Get_processor_name`: Fetching Processor Name

```
char processor_name[MPI_MAX_PROCESSOR_NAME];
int name_len;
MPI_Get_processor_name(processor_name, &name_len);
```

- **Objective**: Retrieves the identifier of the CPU executing a given process.
- **Parameters**: Utilizes a character array to capture the processor's name and an integer pointer to record its length.
- **Impact**: The processor's name is consigned to `processor_name`.

`MPI_Finalize`: Graceful MPI Termination

```
MPI_Finalize();
```

- **Objective**: Appropriately shuts down the MPI environment, facilitating a clean exit.
- **Parameters**: None.
- **Impact**: Completes all MPI processes and allows for safe termination.

Execution Procedure

1. **Compilation**: Utilize `make` command for compilation, facilitated through a Makefile.
2. **Execution**: `mpirun -np 4 build/hello_world` to execute the program on 4 processors.

Output Example

```
Hello world from processorhectelaptop, rank0out of 4processors
Hello world from processorhectelaptop, rank2out of 4processors
Hello world from processorhectelaptop, rank3out of 4processors
Hello world from processorhectelaptop, rank1out of 4processors
```

Reflection and Limitations

The project aims to familiarize students with basic parallel programming using MPI. A successful execution on a 4-core system validates the program's ability to run multiple processes in parallel. However, limitations arise when the number of processors exceeds the system's available slots, as evidenced by an error message when attempting to run on 5 processors.

This constraint underscores the necessity for resource management and scalability considerations in parallel computing endeavors.

```
❯ mpirun -np 5 build/hello_world
--------------------------------------------------------------------------
There are not enough slots available in the system to satisfy the 5
slots that were requested by the application:

  build/hello_world

Either request fewer slots for your application, or make more slots
available for use.

A "slot" is the Open MPI term for an allocatable unit where we can
launch a process.  The number of slots available are defined by the
environment in which Open MPI processes are run:

  1. Hostfile, via "slots=N" clauses (N defaults to number of
     processor cores if not provided)
  2. The --host command line parameter, via a ":N" suffix on the
     hostname (N defaults to 1 if not provided)
  3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)
  4. If none of a hostfile, the --host command line parameter, or an
     RM is present, Open MPI defaults to the number of processor cores

In all the above cases, if you want Open MPI to default to the number
of hardware threads instead of the number of processor cores, use the
--use-hwthread-cpus option.

Alternatively, you can use the --oversubscribe option to ignore the
number of available slots when deciding the number of processes to
launch.
--------------------------------------------------------------------------
```