Hybrid MPI

Paul Preney, OCT, M.Sc., B.Ed., B.Sc.
preney@sharcnet.ca

School of Computer Science
University of Windsor
Windsor, Ontario, Canada

July 14, 2021

Table of Contents

What is Hybrid MPI?

Hybrid MPI is combining MPI with other concurrent programming models within the same program, e.g.,

- MPI + pthreads code
- MPI + OpenMP code
- MPI + Threaded Building Blocks (TBB)
- MPI + CUDA
- MPI + OpenMP + CUDA
- etc.

What is MPI?

MPI stands for **M**essage **P**assing **I**nterface and:

- MPI enables writing **parallel** code.
- MPI **programs** are comprised of a number of MPI **processes**.
- MPI **inter-process communication** is done by **passing messages**.
- MPI on clusters is designed to take advantage of special **high-speed interconnects** between nodes.
- Each MPI process can be run on a **different** node/computer.
- Each MPI process' **address space** is **separate** from all other processes.

What Is A Thread?

A thread-of-execution is:

- is able to be executed **concurrently** relative to other threads-of-execution
- a construct that has an **independent set of values** for a single core's registers

A thread:

- can be implemented as a software construct, and/or,
  - e.g., allowing more threads than available hardware cores
- can be tied directly to hardware.
  - e.g., *n* thread(s) per hardware unit

What Is A Process?

A process:

- runs on a single node/computer
- is **scheduled and controlled** by the **operating system**
- is comprised of **one or more** threads-of-execution
  - A **serial** or **single-threaded** program has **exactly one** thread-of-execution.
  - A **parallel** or **multi-threaded** program has **more than one** thread-of-execution.
- is **allocated/obtains** access to resources which **all** of its threads-of-execution can access
  - e.g., RAM assigned to a process so its threads-of-execution can freely access it
  - NOTE: **Resource contention** must be handled using various forms of **synchronization**, e.g., barriers, fences, locks, atomics, etc.
- is **unable to access** the resources allocated to or obtained by **other processes**
  - except when permitted by special operating system calls or by hardware

What Is A Program?

A program:

- is made up of one or more interconnected processes that
- functions as a single executable entity

Examples:

- scripts,
- executable/**.exe** files,
- MPI programs running over a number of nodes,
- etc.

Table of Contents

What is Hybrid MPI?

Hybrid MPI is combining MPI with other concurrent programming models within the same program, e.g.,

- MPI + pthreads code
- MPI + OpenMP code
- MPI + Threaded Building Blocks (TBB)
- MPI + CUDA
- MPI + OpenMP + CUDA
- etc.

Why Hybrid MPI?

MPI makes it easy to communicate messages between nodes/computers.

- e.g., one does not need to be concerned with how to make use of a cluster's high-speed interconnection fabric

Hybrid MPI:

- allows one to leverage features of other frameworks + MPI
- allows one to transform existing single-node code and make it work across nodes.
  - e.g., one needs more cores for processing
  - e.g., often one cannot afford to change/rewrite existing single-node code

Why Hybrid MPI? (con't)

Without MPI:

- one would have to write code to communicate over high-speed interconnection fabrics
  - Different clusters may not use the same fabric.
  - One may also need to wholly or partially implement topologies, scatters, gathers, broadcasts, reductions, etc. when directly programming with a fabric's API.

Why Not Hybrid MPI?

Non-MPI code might perform worse than pure MPI code within a node.

- It is easy to have all threads idle except one when performing MPI communication.
- There may be issues with affinity, data placement, cache coherence, etc.
  - e.g., whole node jobs that aim to achieve maximum performance
- Overall the code is more complicated to maintain and enhance over time.
  - e.g., MPI API calls + pthreads/OpenMP/CUDA/etc. API calls

Always benchmark and review design decisions concerning your code.

Writing Hybrid MPI Code

If starting from scratch, a common approach is:

- Start by writing sequential code that solves the problem.
- Then parallelize the sequential code using threads, e.g., pthreads/OpenMP/TBB/etc.
- Then wrap MPI around such to run code across nodes.

If one already has pthreads/OpenMP/TBB/etc. code:

- Wrap MPI around such to run code across nodes.
  - NOTE: This avoids having to rewrite, test, and debug existing code.

Writing Hybrid MPI Code (con't)

A typical MPI program has this structure:

basic-mpi.c

```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
  MPI_Init(&argc, &argv);

  int rank, nprocs;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  printf("Hello from rank %d of %d.\n", rank, nprocs);

  MPI_Finalize();
}
```

Writing Hybrid MPI Code (con't)

To make this program a hybrid MPI program, change the call to `MPI_Init()` to `MPI_Init_thread()`, e.g.,

basic-hybrid-mpi.c

```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
  int mpi_init_provided = 0;
  MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &mpi_init_provided);
  if (mpi_init_provided < MPI_THREAD_MULTIPLE)
  {
    fprintf(stderr, "ERROR: Unable to establish required environment.\n");
    MPI_Abort(MPI_COMM_WORLD, 1);
  }

  /* ... */
  MPI_Finalize();
}
```

MPI_THREAD_* Values

`MPI_Init_thread()` requires one of four values:

- `MPI_THREAD_SINGLE`: Only one thread exists in the application.
  - equivalent to calling `MPI_Init()`
- `MPI_THREAD_FUNNELED`: Multithreaded, only **main** thread can make MPI calls.
  - i.e., the thread that called `MPI_Init_thread()`
  - i.e., `MPI_Is_thread_main(&some_int_var)` sets `int_var` to true (a non-zero value) when called from the main thread
- `MPI_THREAD_SERIALIZED`: Multithreaded, only **one thread at a time** can make an MPI call.
  - e.g., the programmer may need to wrap MPI calls with barriers, fences, locks, etc.
- `MPI_THREAD_MULTIPLE`: Multithreaded, MPI calls are thread-safe.

MPI_THREAD_* Values (con't)

The call to `MPI_Init_thread()` sets its fourth (i.e., last) argument to the `MPI_THREAD_*` value that the current MPI implementation supports.

basic-hybrid-mpi.c

```c
#include <stdio.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
  int mpi_init_provided = 0;
  MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &mpi_init_provided);
  if (mpi_init_provided < MPI_THREAD_MULTIPLE)
  {
    fprintf(stderr, "ERROR: Unable to establish required environment.\n");
    MPI_Abort(MPI_COMM_WORLD, 1);
  }

  /* ... */
  MPI_Finalize();
}
```

Example Slurm sbatch Job

sbatch-simple-hybrid.sh

```
#!/bin/bash
#SBATCH --ntasks=3                  # Total number of MPI ranks
#SBATCH --cpus-per-task=2           # Total number of threads per rank
#SBATCH --mem-per-cpu=1000M         # Total RAM per CPU
#SBATCH --time 1-00:00
# place other SBATCH stipulations here if needed

module load openmpi

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
srun ./a_hybrid_mpi_openmp_program.exe
```

which can be submitted with:

- `sbatch -A your-account sbatch-simple-hybrid.sh`

Example mpirun command

If running on your own computer, use `--bind-to none` with `mpirun`, e.g.,

- `OMP_NUM_THREADS=2 mpirun --bind-to none -np 3 ./mpi-omp-prog.exe`
  - i.e., 3 MPI ranks with 2 (OpenMP) threads per MPI rank

Example mpirun command (con't)

Binding Options

- Slurm's `srun` and MPI's `mpirun` allow one to bind to CPU cores, sockets, etc.
- If you are not setting such explicitly, then you may (or may not) need to also use the `--bind-to none` when using `mpirun`.
  - `mpirun`'s `--bind-to none` ensures your processes and threads are not explicitly bound to any CPU code, etc.
  - Examine `man mpirun` for `mpirun`'s options.
  - Examine `man sbatch`, `man salloc`, `man srun` for Slurm's binding options.
    - NOTE: Such options are subject to Linux' cgroup restrictions, e.g., as set by Slurm.

A Simple OpenMP+MPI program

example-mpi-omp.c

```c
#include <stdio.h>
#include <omp.h>
#include <mpi.h>

int main(int argc, char *argv[])
{
  int mpi_init_provided = 0;
  MPI_Init_thread(&argc, &argv, MPI_THREAD_MULTIPLE, &mpi_init_provided);
  if (mpi_init_provided < MPI_THREAD_MULTIPLE)
  {
    fprintf(stderr, "ERROR: Unable to establish required MPI environment.\n");
    MPI_Abort(MPI_COMM_WORLD, 1);
    return 1;
  }

```

A Simple OpenMP+MPI program (con't)

example-mpi-omp.c

```
16   int nprocs, rank;
17   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
18   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
19
20   #pragma omp parallel default(shared)
21   {
22     int num_threads = omp_get_num_threads();
23     int tid = omp_get_thread_num();
24     printf("Hello from thread %d of %d from MPI rank %d of %d\n", tid, num_threads, rank,
       ↪  nprocs);
25   }
26
27   MPI_Finalize();
28 }
```

A Simple OpenMP+MPI program (con't)

Compile with:

```
mpicc -o example-mpi-omp.exe example-mpi-omp.c -fopenmp
```

This can be run locally with:

```
OMP_NUM_THREADS=2 mpirun -np 3 --bind-to none ./example-mpi-omp.exe
```

Or use Slurm's `sbatch` or `salloc`.

- Use `srun` instead of `mpirun`.

Table of Contents

Benchmarking in MPI: MPI_Wtime()

mpi-bench.c

```c
// ...
// record start time in rank 0...
double time_point = 0.0;
if (rank == 0)
  time_point = MPI_Wtime();

// ... do something ...

// output elapsed time in rank 0...
if (rank == 0)
{
  time_point = MPI_Wtime() - time_point;
  printf("time: %lg seconds\n", time_point);
}
// ...
```