PARALLEL PROGRAMMING...

Parallel Programming: Overview

SESSION 2/3

GOAL

Programming Interface for parallel computing

MPI (Message Passing Interface)

OpenMP (Open Multi-Processing)

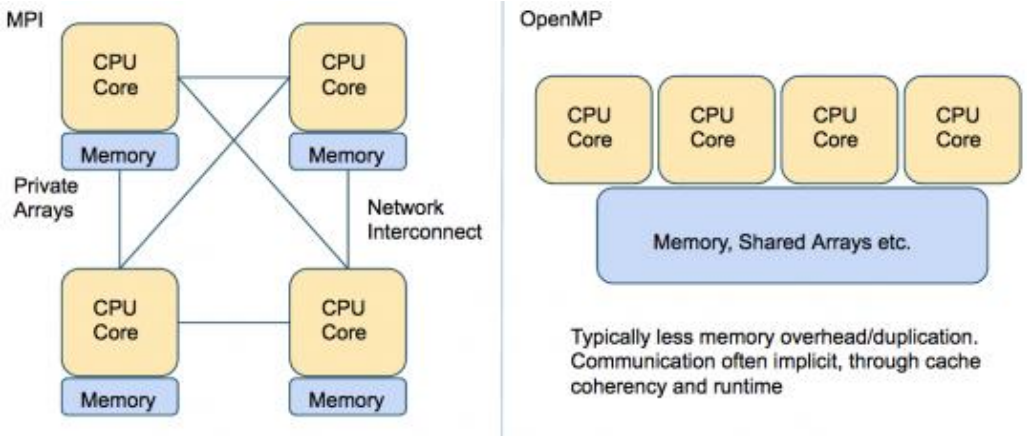
Programming interface for parallel computing

병렬 컴퓨팅을 위한 프로그래밍 인터페이스

Programming interface…

Remember

.



.

MPI (Message Passing Interface) is a multi-process model whose mode of communication between the processes is **explicit.**

==> communication management is the responsibility of the user.

.

OpenMP (Open Multi-Processing) is a multitasking model whose mode of communication between tasks is **implicit**

==> communications is the responsibility of the compiler.

Hybrid OpenMP-MPI

Hybrid MPI and OpenMP

Hybrid application programs using **MPI + OpenMP** are now common place on large HPC systems.

There are basically two main motivations:

1. Reduced memory footprint, both in the application and in the MPI library (eg communication buffers).
2. Improved performance, especially at high core counts where pure MPI scalability runs out.

Hybrid MPI and OpenMP

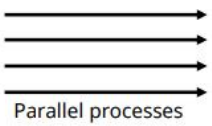
Parallel programming models

Parallel execution is based on threads or processes (or both) which run at the same time on different CPU cores

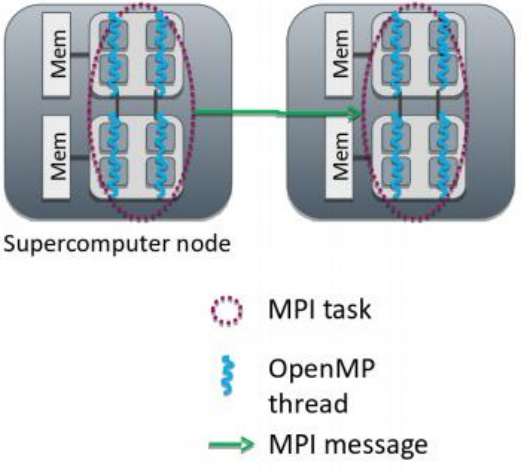
Processes
- Interaction is based on exchanging messages between processes
- MPI (Message passing interface)

Threads
- Interaction is based on shared memory, i.e. each thread can access directly other threads data
- OpenMP

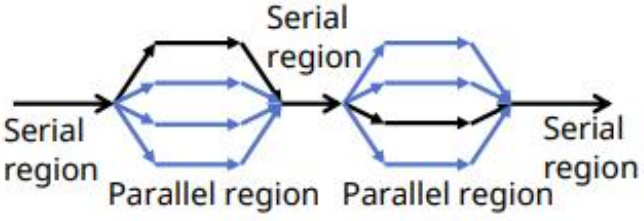
Hybrid MPI and OpenMP



1: MPI: Processes
- Independent execution units
- Have their own memory space
- MPI launches N processes at application startup
- Works over multiple nodes



2: OpenMP: Threads
- Threads share memory space
- Threads are created and destroyed (parallel regions)
- Limited to a single node



3: Hybrid programming: Launch threads (OpenMP) within processes (MPI)

- Shared memory programming inside a node, message passing between nodes
- Optimum MPI task per node ratio depends on the application and should always be experimented.

Hybrid Programming

In hybrid programming each process can have **multiple threads executing simultaneously**
All threads within a process share all MPI objects Communicators, requests, etc.

MPI defines 4 levels of thread safety:

- **MPI_THREAD_SINGLE** : One thread exists in program
- **MPI_THREAD_FUNNELED** : Multithreaded but only the master thread can make MPI calls Master is one that calls MPI_Init_thread()
- **MPI_THREAD_SERIALIZED:** Multithreaded, but only one thread can make MPI calls at a time
- **MPI_THREAD_MULTIPLE:** Multithreaded and any thread can make MPI calls at any time. Use MPI_Init_thread instead of MPI_Init if more than single thread

Hybrid Programming

Potential advantages of the hybrid approach

- Fewer MPI processes for a given amount of cores
  - Improved load balance
  - All-to-all communication bottlenecks alleviated
- Decreased memory consumption if an implementation uses replicated data
- Additional parallelization levels may be available
- Possibility for dedicating threads for different tasks
  - e.g. dedicated communication thread or parallel I/O
- Dynamic parallelization patterns often easier to implement with OpenMP

Disadvantages of hybridization

- Increased overhead from thread creation/destruction
- More complicated programming
  - Code readability and maintainability issues
- Thread support in MPI and other libraries needs to be considered

Hybrid Programming: Example

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

int main(int argc, char *argv[]) {
    int my_id, omp_rank;
    int provided, required=MPI_THREAD_FUNNELED;

    MPI_Init_thread(&argc, &argv, required,
                    &provided);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
#pragma omp parallel private(omp_rank)
{
    omp_rank = omp_get_thread_num();
    printf("I'm thread %d in process %d\n",
           omp_rank, my_id);
}
    MPI_Finalize();
}
```

```
$ mpicc -fopenmp hybrid-hello.c -o hybrid-hello
$ srun  --ntasks=2 --cpus-per-task=4
  ./hybrid-hello

I'm thread 0 in process 0
I'm thread 0 in process 1
I'm thread 2 in process 1
I'm thread 3 in process 1
I'm thread 1 in process 1
I'm thread 3 in process 0
I'm thread 1 in process 0
I'm thread 2 in process 0
```

Hybrid Programming Example

- **Numerical integration**

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi$$

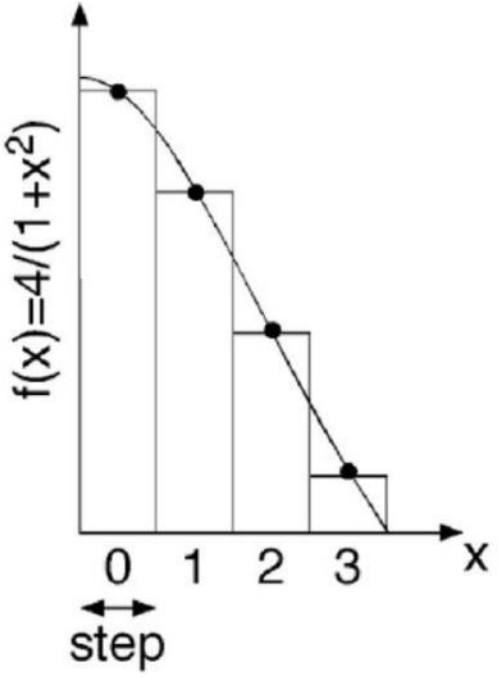
- **Discretization:**

$\Delta = 1/N$: `step = 1/NBIN`

$x_i = (i+0.5)\Delta$ $(i = 0,\ldots,N-1)$

$$\sum_{i=0}^{N-1} \frac{4}{1+x_i^2}\Delta \cong \pi$$

```
#include <stdio.h>
#define NBIN 100000
void main() {
  int i; double step,x,sum=0.0,pi;
  step = 1.0/NBIN;
  for (i=0; i<NBIN; i++) {
    x = (i+0.5)*step;
    sum += 4.0/(1.0+x*x);}
  pi = sum*step;
  printf("PI = %f\n",pi);
}
```




```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>              /* MPI header file */
#include <omp.h>              /* OpenMP header file */
#define NUM_STEPS 100000000
#define MAX_THREADS 4

int main(int argc, char *argv[]) {
   int nprocs, myid;
   int tid, nthreads, nbin;
   double start_time, end_time;
   double pi, Psum=0.0, sum[MAX_THREADS]={0.0};
   double step = 1.0/(double) NUM_STEPS;

   /* initialize for MPI */
  MPI_Init(&argc, &argv);       /* starts MPI */
   /* get number of processes */
   MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
   /* get this process's number (ranges from 0 to nprocs - 1) */
   MPI_Comm_rank(MPI_COMM_WORLD, &myid);

  nbin= NUM_STEPS/nprocs;
```

Hybrid Programming Example

```
#pragma omp parallel private(tid)
{
      int i;
      double x;
      nthreads=omp_get_num_threads();
      tid=omp_get_thread_num();
      for (i=nbin*myid+tid; i < nbin*(myid+1); i+= nthreads) { /* changed*/
         x = (i+0.5)*step;
         sum[tid] += 4.0/(1.0+x*x);
      }
}
 for(tid=0; tid<nthreads; tid++)   /*sum by each mpi process*/
      Psum += sum[tid]*step;

MPI_Reduce(&Psum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);/* added */

   if (myid == 0) {
      printf("parallel program results with %d processes:\n", nprocs);
      printf("pi = %g  (%17.15f)\n",pi, pi);
   }
   MPI_Finalize();

   return 0;
}
```

Results

- MPI

  MPI uses 8 processes:
  pi = 3.14159 (3.141592653589828)

- OpenMP

  OpenMP uses 8 threads:
  pi = 3.14159 (3.141592653589882)

- Hybrid

  mpi process 0 uses 4 threads
  mpi process 1 uses 4 threads
  mpi process 1 sum is 1.287 (1.287002217586605)
  mpi process 0 sum is 1.85459 (1.854590436003132)
  Total MPI processes are 2
  pi = 3.14159 (3.141592653589738)

Thank you for your attention !