Parallel Programming in HPC

OpenMP & MPI in C++

An Introduction for Beginners

Dr A.Khalatyan
Researcher at eScience/Supercomputing/IT
Leibniz-Institut für Astrophysik Potsdam (AIP), Germany
AIP, 2025

OpenMP: Shared Memory Parallelism

- OpenMP is used for multi-threading on shared memory systems
- Threads share the same address space
- Minimal modifications needed for existing C++ code

OpenMP Example: Parallel Loop

```cpp
#include <iostream>
#include <omp.h>

int main() {
    const int N = 1000000;
    double sum = 0.0;

    #pragma omp parallel for reduction(+:sum)
    for (int i = 0; i < N; i++) {
        sum += 1.0 / (i + 1);
    }

    std::cout << "Sum: " << sum << std::endl;
    return 0;
}
```

Explanation: OpenMP Parallel Loop

- `#pragma omp parallel for` distributes loop iterations among threads
- `reduction(+:sum)` ensures correct summation across threads
- Number of threads can be controlled via `OMP_NUM_THREADS` environment variable

MPI: Distributed Memory Parallelism

- MPI (Message Passing Interface) enables communication between processes
- Processes do **not** share memory
- Used in large-scale clusters

MPI Example: Parallel Sum

```
#include <mpi.h>
#include <iostream>
#include <vector>
int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    int N = 1000000;
    double local_sum = 0.0;
    for (int i = rank; i < N; i += size)
        local_sum += 1.0 / (i + 1);

    double global_sum = 0.0;
    MPI_Reduce(&local_sum, &global_sum, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (rank == 0)
        std::cout << "Total Sum: " << global_sum << std::endl;
    MPI_Finalize();
    return 0;
}
```

Explanation: MPI Parallel Sum

- `MPI_Comm_rank` and `MPI_Comm_size` determine process rank and number of processes
- Each process computes a partial sum
- `MPI_Reduce` gathers results to process 0

OpenMP vs. MPI

Feature	OpenMP	MPI
Memory Model	Shared	Distributed
Overhead	Low	Higher (communication)
Scalability	Limited	High
Ease of Use	Easier	More complex
Best for	Multi-core CPUs	Large-scale clusters

Hybrid Parallelism: MPI + OpenMP

- Use MPI for inter-node communication
- Use OpenMP for intra-node parallelism
- Efficient for modern HPC architectures

Hybrid Example: MPI + OpenMP

```
#include <mpi.h>
#include <omp.h>
#include <iostream>
int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    #pragma omp parallel
    {
        int thread_id = omp_get_thread_num();
        #pragma omp critical
        std::cout << "Hello from thread " << thread_id << " in process " << rank << std::endl;
    }

    MPI_Finalize();
    return 0;
}
```

Summary

- OpenMP is useful for shared-memory parallelism
- MPI is needed for large-scale distributed parallelism
- Hybrid MPI+OpenMP offers the best of both worlds
- Choosing the right model depends on hardware and problem size

Questions?

Thank you!