OpenMP and MPI hybrid programming

Ing. Michal ÁČ

Armed forces academy of General Milan Rastislav Štefanik
Demänová 393
031 06 Liptovský Mikuláš
acmic@mosr.sk

Abstract. OpenMP and MPI are two approaches that can help us achieve increasing of computing potential. We try to mix them together into OpenMP and MPI hybrid programming model exploited to give the best performance on a single computing system.

Keywords: OpenMP, MPI, hybrid programming, clustered systems, parallel computing, computing potential, shared memory systems, hybrid programming

1. Introduction

It is all about computing performance in the world today. People are trying to find the best possible technique to accomplish their computing tasks the most fastest way . There are many ways how to achieve their goals and we would like to introduce one of the approaches. We heard a lot about HPC and HTC system, clusters computers that can work together, how they can access their sources, cpu, memory systems, bus communications and so on, but it is not as simple as it seems. OpenMP (*application program interface that supports multi-platform shared memory multiprocessing programming*) and MPI (*message passing interface - standardized and portable message-passing system*)are two approaches that can help us achieve increasing of computing potential. Each has it‘s pros and cons, so we try to mix them together into OpenMP and MPI hybrid programming model exploited to give the best performance on a single computing system.

Memory architectures (shared, distributed) are gradually becoming more prominent on the HPC market, as advances in technology have allowed larger numbers of CPUs to have access to a single memory. In addition, manufacturers are increasingly clustering these systems together to go beyond the limits of a single system. MPI codes written in MPI are obviously portable and should transfer easily to clustered systems. In theory a shared memory model such as OpenMP should offer a more efficient parallelisation strategy. Parallelisation is the key.

2. Shared memory and clustered architectures

There are a number of possible benefits of writing a mixed mode code whether the archutecture is shared memory systems or and shared clustered systems (ditributed memory systems). Lets write some notes about these architectures.

2.1 Shared memory systems (SMP)

Shared Memory architectures involve a number of processors sharing a global memory space. This will eventually constrain the scalability of the system because memory access becomes a severe bottleneck as the number of processors is increased. Figure 1 shows a schematic presentation of a shared memory system.

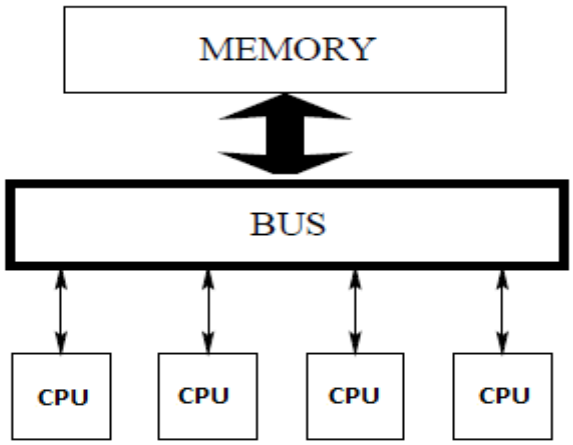

Fig. 1. Schematic presentation of a shared memory system. CPUs are communicating with shared memory over bus. All cpus are using same memory.

2.2 Clustered system

Clustered SMP systems can be described as a hybrid of shared memory and distributed memory systems. The cluster consists of a number of shared memory nodes each containing a number of processors sharing a global memory space. Nodes can communicate with each other via some form of fast interconnection network.

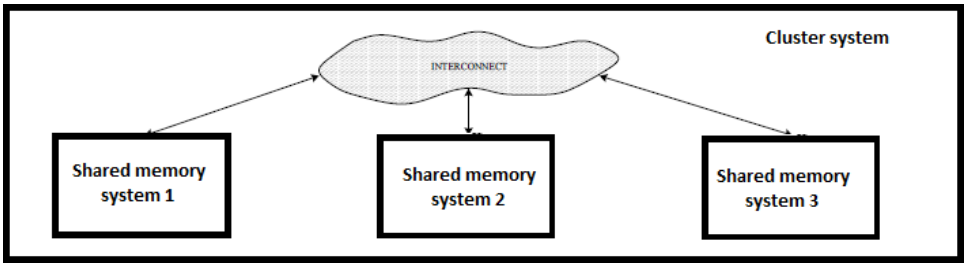

Fig. 2. Schematic presentation of a clustered memory system. Each node is shared memory system and can communicate with other via interconnect and can access others system resources.

3. Programming in MPI and OpenMP

At first we should talk a bit about each approach of programming. Both are using parallel procedures, let us what they can provide us with.

3.1 MPI

The message passing programming model is a distributed memory programming model with explicit control parallelism.

Characteristics:

1. MPI codes will run on both distributed and shared memory architectures.
2. Portable. MPI implementations have been ported to almost all systems including Unix and Windows platforms.
3. Particularly adaptable to coarse grain parallelism.
4. A large number of vendor optimised MPI libraries exist.
5. Each process has its own local memory (distribution).
6. Data are copied between local memories via messages which are sent and received via explicit (message passing) calls.

Explicit call can provide a better performance. Communications often cause synchronisation. There are also some problems that MPI can cause such as load balancing , overhead communications, difficulty to change some parts of the code.

3.2 OpenMP

OpenMP is standard for shared memory programming. Based on a combination of compiler directives, library routines and environment variables it is used to specify parallelism on shared memory machines. Directives are added to the code to tell the compiler of the presence of a region to be executed in parallel,

together with some instructions as to how the region is to be parallelised and some region executed in serial. This uses a fork-join model.

Characteristics:

1. OpenMP codes will only run on shared memory machines.
2. Fairly portable.
3. Permits both coarse grain and fine grain parallelism.
4. Uses directives which help the compiler parallelise the code.
5. Each thread sees the same global memory, but has its own private memory.
6. Implicit messaging.
7. High level of abstraction (higher than MPI).

3.3 Mixed mode programming

By utilising a mixed mode programming model we should be able to take advantage of the benefits of both models. For example a mixed mode program may allow us to make use of the explicit control data placement policies of MPI with the finer grain parallelism of OpenMP. The majority of mixed mode applications involve a hierarchical model, MPI parallelisation occurring at the top level, and OpenMP parallelisation occurring below. In Figure 3, we can see that from the start of the program, we use MPI calls for execution on three processes (P0 to P2) , and then with OpenMPI code we devide the „task“ to separate threads (0-3) on each process.

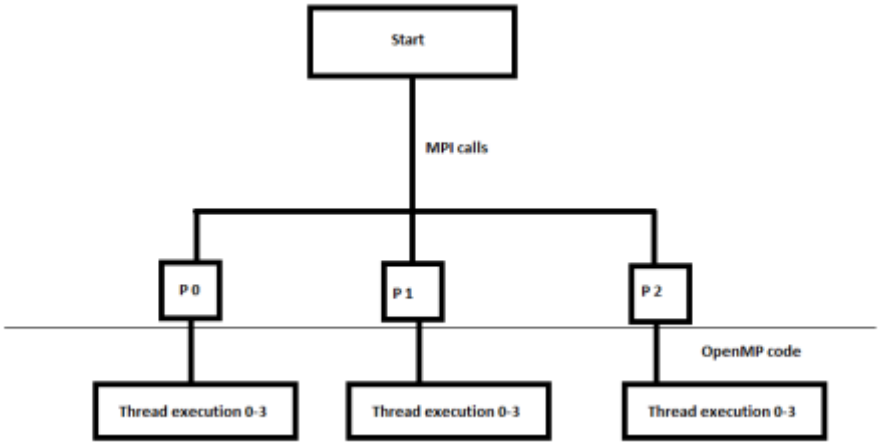

Fig. 3. Schematic presentation of a hierarchical mixed mode programming model.

4. Implementing a mixed mode application

Figure 4 shows a simple mixed mode Hello World code, to demonstrate how a mixed mode code is implemented. MPI is initialised and finalised in the usual way,

using the MPI INIT and MPI FINALIZE calls. An OpenMP PARALLEL region occurs between these calls, spawning a number of threads on each process.

```
program mixed
implicit none
include 'mpif.h'
integer ierror, rank
integer OMP_GET_THREAD_NUM, thread
call MPI_INIT (ierror)
call MPI_COMM_RANK (MPI_COMM_WORLD, rank, ierror)
!$OMP PARALLEL
write(*,*) 'hello world', rank, OMP_GET_THREAD_NUM()
!$OMP END PARALLEL
call MPI_FINALIZE (ierror)
end
```

Fig. 4. A simple mixed mode Hello World code.

Figure 5 shows an execution of the program on MPI two processes and OpneMP two threads.

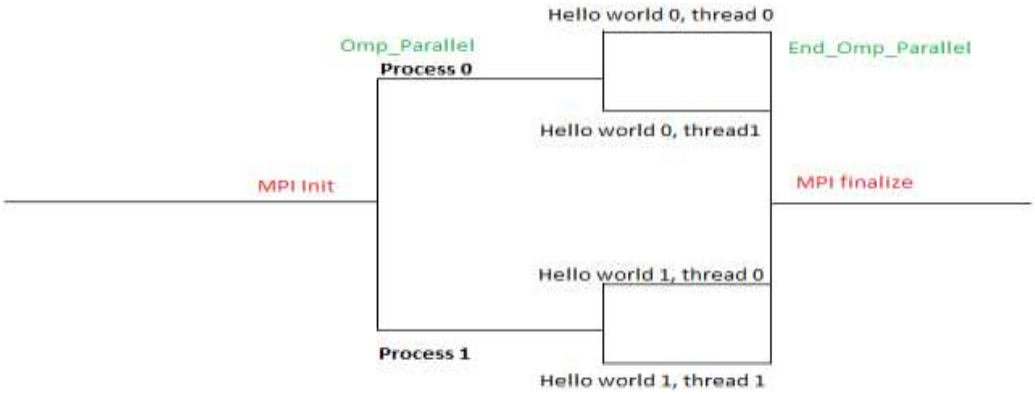

Fig. 5. The flow of execution of a simple mixed mode Hello World code.

5. Conclusions

We cannot tell which model is the ideal programming model for all codes. In some situations, however, significant benefit may be obtained from a mixed mode implementation, but not in all situtations. For example, benefit may be obtained if the parallel (MPI) code suffers from: poor scaling with MPI processes due to load imbalance or a fine grain size problem, memory limitations due to the use of a replicated data strategy, or a restriction on the number of MPI processes combinations. In addition, if the system has a poorly optimised or limited scaling MPI implementation then a mixed mode code may increase the code performance. The HPC market is increasing, the focus is on the portable implementation of the code and its optimalization. MPI in general is a communication between nodes and it is required, OpenMP offers an efficient, and often considerably easier, parallelisation strategy within nodes. Therefore mixed model offers the most efficient strategy for parallelism. Mixed code has a potential to take the best from both separate approaches and creates the best performance on the system.

However, it is clear that this style of programming will not always be the most effective mechanism for all problems. In some cases it can be even less effective. We should take a serious consideration to the codes, when we would like to implement mixed code. On the other hand, when we suffer from bad optimalization, load balancing mixed mode code may increase the code performance.

References

1. L.A. Smith, Mixed Mode MPI / OpenMP Programming, Edinburgh Parallel Computing Centre, Edinburgh, EH9 3JZ
2. Christian Terboven, Dieter an Mey, OpenMP in theReal World, Center forComputing andCommunication RWTH Aachen University, Germany