©2010 International Journal of Computer Applications (0975 – 8887)
Volume 1 – No. 5

Optimized Convex Hull With Mixed (MPI and OpenMP) Programming On HPC

Sandip V.Kendre
Lecturer
SIT and Research
Center, Nashik

Dr.D.B.Kulkarni
I/C H.O.D. and Professor
Walchand college of
Engineering, Sangli
(Autonomous institute)

ABSTRACT

As a programmer, one is aspired to solve ever larger, more memory intensive problems, or simply solve problems with greater speed than possible on a sequential computer. A programmer can turn to parallel programming and parallel computers to satisfy these needs. Parallel programming methods on parallel computers gives access to greater memory and Central Processing Unit (CPU) resources which is not available on sequential computers. This paper discusses the benefits of developing 2D and 3D convex hull on mixed mode MPI, OpenMP applications on both single and clustered SMPs. In this experimentation for purpose of optimization of 3D convex hull we merged both MPI and OpenMP library which gives another mixed mode programming method to get optimized results. The job is divided into sub-jobs and are submitted to cluster of SMP nodes using MPI and these sub-jobs are computed in parallel using OpenMP threads in SMP nodes. Experiments on sequential, MPI, OpenMP and Hybrid programming models show that Hybrid programming model substantially outperforms others.

Keywords

HPC, MPI, OpenMP, SMP, threads, mixed mode programming.

1. INTRODUCTION

Today most systems in high-performance computing (HPC) features a hierarchical hardware design. Shared memory nodes with several multi-core CPUs are connected via a network infrastructure. Hybrid parallel programming must combine distributed memory parallelization on the node interconnected with shared memory parallelization inside each node. In this paper cases are pinpointed where a hybrid programming model can certainly be the superior solution because of reduced communication needs and memory consumption or enhanced load balance. Finally it gives an outlook on possible standardization goals and extensions that could make hybrid programming easier to do with performance in mind.

2. Characteristics of MPI and OpenMP.

2.1 MPI (Message Passing Interface)

The message passing programming model is a distributed memory model with explicit control parallelism. Processes are only able to read and write to their respective local memory. Data is copied across local memories by using the appropriate subroutine calls. The message passing interface (MPI) standard defines a set of functions and procedures that implements the message passing model.

Characteristics:-

1. MPI codes will run on both distributed and shared memory architectures.
2. Portable.
3. Particularly adaptable to coarse grain parallelism.
4. Each process has its own local memory.
5. Explicit messaging.

2.2OpenMP (Shared Memory Programming)

OpenMP is an industry 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 parallelized. This makes use of a fork-join model.

Characteristics:-

1. OpenMP codes will only run on shared memory machines.
2. Not portable.
3. Permits both course grain and fine grain parallelism.
4. Uses directives which help the compiler parallelize the code.
5. Each thread sees the same global memory.
6. Implicit messaging.
7. Use fork-join model for parallel computation.

3. Design of MPI + OpenMP Mixed Hybrid Model.

3.1 Introduction

The cluster-computing paradigm has come to be recognized as the most cost effective approach to large-scale numerical simulations. As the clustered SMPs (Symmetric Multiprocessing) become more prominent, it becomes more important for the applications to be portable and efficient on these systems. Message passing codes written in MPI are obviously portable and should transfer easily to clustered SMP systems. But, shared memory model such as OpenMP should offer a more efficient parallelization strategy within a node. The aim of this paper is obtain optimal performance from both the processors of all the nodes in the cluster while doing parallel processing with MPI and OpenMP. The primary motivation for considering the use of OpenMP on SMP architecture is to achieve optimum performance. In particular, if an application is using MPI for on-node communications.

3.2 Mixed Mode Programming

By utilizing a mixed mode programming model one should be able to take advantage of the benefits of both models. For example a mixed mode program may allow making 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 parallelization occurring at the top level, and OpenMP parallelization occurring below.

3.3 Implementing a mixed mode application

Figure 3.1 shows a simple mixed mode pseudo code, to demonstrate how a mixed mode code is implemented. MPI is initialized and finalized 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.

Although a large number of MPI implementations are thread-safe, but this cannot be guaranteed. To ensure the code is portable all MPI calls should be made within the sequential regions of the code. This often creates little problem as the majority of codes involve the OpenMP parallelization occurring beneath the MPI parallelization and hence the majority of MPI calls occur outside the OpenMP parallel regions. When MPI calls occur within an OpenMP parallel region the calls should be placed inside a CRITICAL, MASTER or SINGLE region, depending on the nature of the code. Care should be taken with SINGLE regions, as different threads can execute the code. Ideally, the number of threads should be set from within each MPI process using ***omp set num threads (n)*** as this is more portable than the OMP NUM THREADS environment variable.

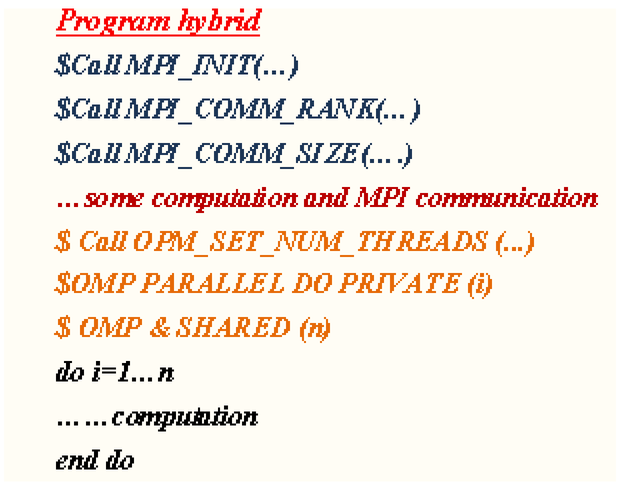
```
Program hybrid
$Call MPI_INIT(...)
$Call MPI_COMM_RANK(...)
$Call MPI_COMM_SIZE(....)
... some computation and MPI communication
$ Call OPM_SET_NUM_THREADS (...)
$OMP PARALLEL DO PRIVATE (i)
$ OMP & SHARED (n)
do i=1...n
......computation
end do
```

Figure 3.1:- Simple Mixed Mode Pseudo Hybrid code

4. Mixed mode program for Convex Hull.

The *convex hull* of a set of points in space is the surface of minimum area with convex (outward) curvature that passes through all the points in the set. In three dimensions, this set must contain at least four distinct, non-coplanar points to make a closed surface with nonzero enclosed volume. With the possible benefits of writing a mixed mode application, this section describes how convex hull code is implemented in OpenMP, MPI and as a mixed MPI/OpenMP code.

The parallel version of convex hull program is design in OpenMP and Hybrid. Mostly MPI cannot be used for parallelizing convex hull program, as it is difficult to handle recursive call in MPI. While designing parallel version with OpenMP first divide the set of point in 2 parts by drawing line between minimum x and maximum x co-ordinate. This part will be executed serially by master thread. Now with every point in upper half draw a triangle then a set of three point is obtained i.e. p1(x1, y1), p2(x2, y2), p3(x3,y3). Where p1 is the point having minimum x co-ordinate and p2 is point having maximum x co-ordinate and p3 is the point of the triangle having maximum area.

Calculate area and find out the point which is inside or outside of the triangle by following determinate formula.

$$f(X) = (p1.x * p2.y - p1.x * p3.y - p2.x * p1.y + p2.x * p3.y + p3.x * p1.y - p3.x * p2.y)$$

……………………………………………. (1)

If the value of the f(x) is negative it is considered that the point is inside the triangle otherwise the point is outside the triangle i.e. the point is considered to be on convex hull. The above process is called recursively by another thread till every input point is considered as shown in figure3.2 and results are shown in table 4.2.

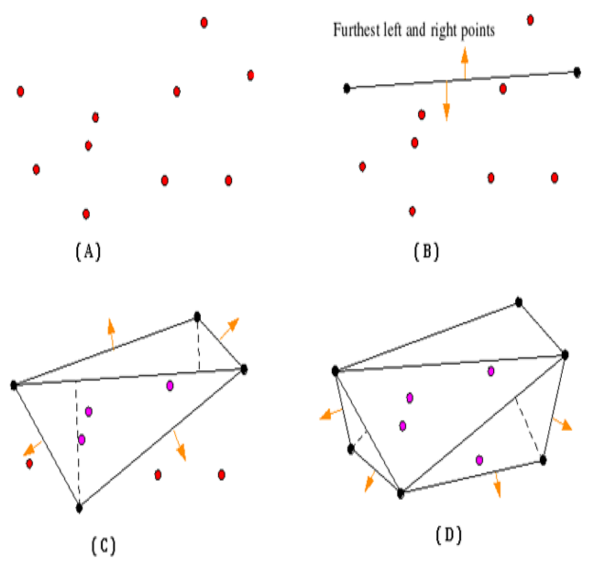

Figure 3.2: 2D convex hull solution step.

Further extend this method for 3D convex hull using 4 x 4 determinants. This case divide problem into three subtasks for parallel computation of each plane which is assigned to OpenMP threads. Continue this process till every input point is considered. Above two methods gives better results on hybrid than that of MPI and OpenMP. The algorithms for 3D convex hull are explained below.

4.1. Algorithm for Hybrid 3D Convex Hull

Same process can be used for implementation of 3D convex hull. For this instead of area calculate volume of trapezoid using 4*4 determinants and use the three independent planes for parallel computation of convex hull.

```
1.  Quick hull( Point Set S, Point Upper, Point Lower )

2.  FIND_MAX(X,Y,Z)
3.  MPI_INIT (….)
4.  Quick hull( Point Set S, Point upper )
5.  OMP_SET_NUM_THREADS(n)

6.  OMP PARALLEL DO PRIVATE (...) & $OMP
    SHARED (...)

7.  DO ....find volume (….)

            is left (...)

8.  S1= points of S on or left of plane

9.  Return    Quick hull( Point Set S, Point upper )
    end do.

10. OMP end parallel do

11. Do same for  Quick hull( Point Set S, Point Lower )

12. Do some MPI computation and communication.

13. Call  MPI_Finalise()

14. End
```

4.3 Result and performance analysis.

In this section following some cases are discussed using Total view 2.8.6.0 analysis tool. For the analysis of hybrid model, the architecture used is HP ProLiant DL786 G5 series server, which delivers leading headroom and expendability for x86 visualization and enterprise applications. With eight number of processor supported Quad-Core AMD Opteron Processor. Following are some cases on which discuss performance analysis of mixed Hybrid model.

1. Performance analysis of OpenMP and Hybrid programming for 2D convex hull programs on Multi-core system for finding outermost point from the given set of point as shown in the table 4.2 and figure4. 2.

Table 4.2:- Performance of OpenMP Vs Hybrid for solving 2D convex hull.

Number of points	Sequential Time(sec)	OpenMP Time (sec.)	Hybrid Time (sec.)
100	0.004	0.000853	0.002181
1000	0.006	0.002163	0.002793
100000	0.083	0.059732	0.032832

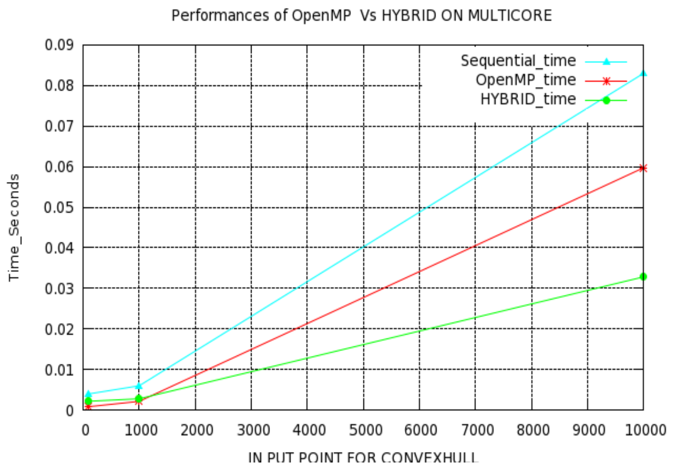

Figure 4.2:- Performance of Sequential Vs OpenMP Vs Hybrid for solving 2D convex hull.

2. Performance analysis of OpenMP with different number of threads on multi-core system as shown in the table4.3 and figure4.3.

Table 4.3:- Time of Convex hull OpenMP program with respect to number of threads.

No. Threads	OpenMP_Time
4	8.309097
8	8.233838
16	5.803572
20	5.298377
40	2.774500
50	2.743215
100	1.890797
150	1.780325
200	1.138228
250	1.137623
300	1.192194

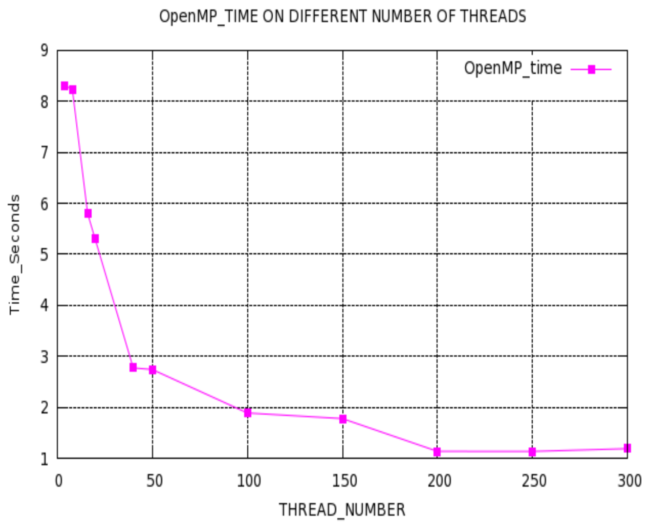

Figure 4.3:- Time of Convex hull OpenMP program with respect to number of threads.

4. Performance analysis of OpenMP and Hybrid programming for 3D convex hull programs on dual-core system for finding outermost point from the given set of point as shown in the table4.4 and figure4.4

Table 4.4:- Performance of OpenMP Vs Hybrid for solving 3D convex hull.

Number of Points	OpenMP Time (sec.)	Hybrid Time(sec.)
50	0.001331	0.002744
100	0.009259	0.010443
200	0.150393	0.145481
300	1.658472	1.047387
400	11.54087	6.922600

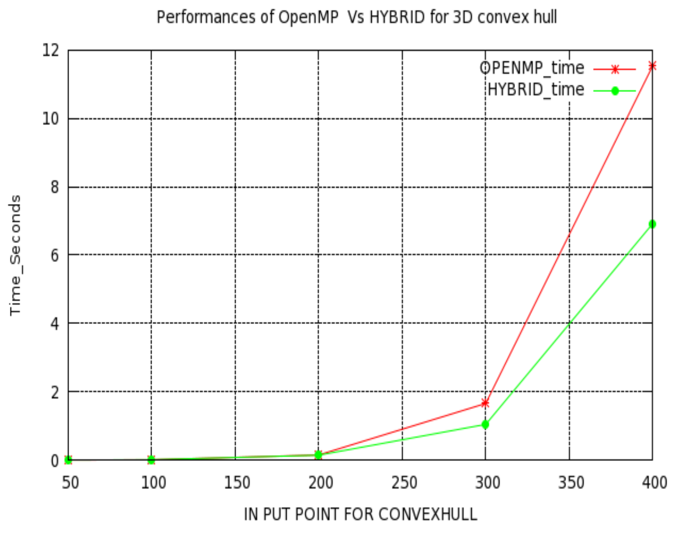

Figure 4.4:- Performance of OpenMP Vs Hybrid for solving 3D convex hull.

5. CONCLUSION

With the increasing prominence of clustered SMPs in the HPC market, the importance of writing the most efficient and portable applications for these systems will grow faster. While message passing is required between nodes, OpenMP offers an efficient, and considerably easier, parallelization strategy within an SMP node. Hence a mixed mode programming model may provide the most effective strategy for an SMP cluster.
In practice, serious consideration must be given to the nature of the codes before embarking on a mixed mode implementation. In some situations we get significant benefit obtained from a mixed mode implementation in the parallel (MPI) code if it suffers from:

1. Poor scaling with MPI processes due to i.e. load imbalance or too fine a grain problem size.
2. Memory limitations due to the use of a replicated data strategy.
3. A restriction on the number of MPI processes combinations.

Thus in addition, to 2D convex hull and 3D convex hull algorithms it is observed that the hybrid mixed mode programming model gives better performance than that of the MPI and OpenMP programming model, for the appropriate number of tasks and threads assigned to each processor.

6. REFERENCES

[1] D.S. Henty, "*Performance of hybrid message-passing and shared-memory parallelism for Discrete Element Modeling*", presented at Supercomputing, Dallas, 2000. http://www.sc2000.org/proceedings/techpapr/papers/pap154.pdf.

[2] MPI: "A *Message-Passing Interface standard",* Message Passing Interface Forum, June1995.
http://www.mpi-forum.org/

[3] OpenMP, The OpenMP ARB. http://www.OpenMP.org/

[4] "*A performance comparison of C with MPI and OpenMP*" on the Origin 2000, J. Hoeflinger, Centre for Simulation of Advanced Rockets. http://polaris.cs.uiuc.edu/ hoefling/Talks/

[5] D.K. Tafti, "*Computational power balancing*", Help for the overloaded processor. http://access.ncsa.uiuc.edu/Features/Load-Balancing/

[6] P. Lanucara and S. Rovida, "*Conjugate-Gradient algortihms: An MPI-OpenMP implementation on distributed shared memory systems",* proceeding of the 1st European Workshop on OpenMP, Lund, Sweden, 1999.