A Comparison of MPI and Hybrid MPI+OpenMP Programming Paradigms Using Multi-Core Processors

Rajkumar Sharma

Vikram University Ujjain, INDIA

Abstract

A decade ago CPUs speed could not be increased without extra ordinary cooling and consequently hit a clock speed barrier. Processors design then was switched to multi-core architecture that minimizes energy consumption. The multi-core architecture is introduced to improve computing performance by providing hardware parallelism through more CPU cores, each having restrained clock speed. This has been a break through in High Performance Computing (HPC). While more processor cores rendered effective execution results, multi-core technology inaugurated an extra layer of complexity for programming issues. To exploit each core in a multi-core environment, application software should be optimized by using multithreading. Multi-core processors can even degrade the performance for single threaded application due to reduction in clock speed. In this paper we compare performance of multithreading fine-grained and course-grained computational problems further flavored as computation-intensive and data-intensive problems by using MPI and hybrid MPI+OpenMP approach.

Keywords- Parallel computing; MPI and OpenMP programming paradigms; multicore processors

1. INTRODUCTION

Multi-core processors cluster is becoming more popular than traditional Symmetric Multi Processors (SMP) cluster. Commodity multi-core processors presents more cost-effective solution to HPC community than expensive SMP clusters. Moreover, multi-core processors deliver better processing results as cores inside same CPU die communicate with high speed interconnection whereas in cluster of SMP nodes inter-processor communication takes place through motherboard within a node, resulting a comparatively slower communication. Both scientific & business applications can be benefited from multi-core processors [4]. Execution time can be minimized by running multiple threads on multiple cores. Multiple cores are effective for data parallel applications where same code can run through multiple threads on different sets of data as well as for functionally decomposed computation intensive tasks where each task run in parallel on different cores[6]. Prior to multi-core architecture, hyper threading (HT) technique was used which may also be combined with multi-core processors. In hyper threading technique, two threads are executed on a single core arranged in time slice manner or driven by interrupt mechanism. For the operating system perception, hyper threading is treated as separate cores. However, dedicated cores in multi-core processors provide better performance than hyper threading.

A major challenge to exploit multi-core architecture, is to convert single threaded applications to multithreading codes[7]. OpenMP programming standards consist of compiler directives that define and identify parallel region of the code that can run as threads. Some programs use proprietary compiler directives to form parallelism through threads whereas OpenMP provides a higher level of abstraction to programmers and create parallelism in a fork-and-join programming model[1]. In this model program begins sequential execution as a single process or thread. When the directive for parallel region is found, the single thread becomes master thread and create several other slave thread to execute parallel tasks. At the end of parallel region all threads are synchronized & joined to produce clubbed results. In OpenMP programming paradigm, all threads use shared memory which create chances of memory contention among threads. This issue is resolved by implementing memory coherence protocol for data consistency.

MPI (Message Passing Interface) is a library that contains message passing routines. MPI is used for parallel processing based on distributed memory model such as Network of Workstations (NOW) or Cluster of Workstations (COW). Communication among nodes takes place through message passing. As each process has its own private memory, there is no chance of memory contention.

OpenMP and MPI programming models can be used within same program as hybrid MPI+OpenMP paradigm[5], suitable for architecture consisting of both shared and distributed memory such as cluster of multi-core processors. The MPI can be used to provide process level parallelism across nodes while OpenMP can be used
to implement loop level parallelism within a node by using compiler directives[8] as shown in Figure 1.

![Figure 1. MPI and Hybrid MPI+OpenMP Programming Paradigm](image)

In this paper, we compare performance of fine-grained as well as course-grained computational problems by using MPI and hybrid (MPI+OpenMP) programming paradigms. We evaluate the suitability of programming model based on type of computational problems.

2. RELATED WORK

MPI+OpenMP programming paradigm have been reported in several published work that mainly experimented on SMP cluster. Jost & Jin [1] compare MPI, OpenMP and hybrid approach by taking different number of CPUs in SMP cluster. IBM SP systems are used by Cappello & Daniel [3] to compare NAS parallel benchmarks on SMP cluster. Authors also show a study of communication and memory access patterns in the cluster. Hits rates of L1 and L2 cache are studied by Wu & Taylor [2] on multi-core cluster by using NAS parallel benchmarks SP and BT. Chen & Watson III [4] compare results between Intel and AMD processors cluster by using OpenMP directives and a locally developed threading library.

3. FINE-GRAINED AND COURSE-GRAINED PROBLEMS

Sub-calculation obtained by dividing a parallel calculation, can be carried out in parallel on different processor. A computation problem is fine-grained when sub-calculation are dependent on the results of other sub-calculation.

A higher level of synchronization among processor is required to solve such problems. In a computation problem, when each sub-calculation is independent of all other calculations, than it is a course-grained computation problem. In this study, we further flavor problem types as computation-intensive and data-intensive problems as shown in Figure 2 and compare performance of each type of problem classified as FGCI, FGDI, CGCI and CGDI on multi-core cluster.

4. EXPERIMENTAL RESULTS AND PERFORMANCE COMPARISON

Most of the authors had compared MPI and MPI+OpenMP on cluster of SMP nodes whereas in this study we evaluated our results on cluster of multi-core commodity nodes. We performed our experiment on cluster of sixteen nodes comprising of dual-core and quad-core processors. We take two problems of each category as stated above created for experimental purposes and compare execution times. The comparison is shown in Table 1 and Figure 3. We observe that hybrid MPI+OpenMP outperforms MPI approach with 10% to 18% improvement in execution time. The resources of multi-core cluster is best exploited in CGCI problems.
where we get maximum 18% performance improvement whereas in case of FGDI we get minimum performance gain as MPI communications increase for fine-grained problems and data latencies increase for data-intensive problems. However, MPI+OpenMP approach provided better execution results in all four types of problems.

TABLE I. COMPARATIVE EXECUTION TIME (MS)

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>MPI</th>
<th>Hybrid</th>
<th>% Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>FGCI 1</td>
<td>2824</td>
<td>2477</td>
<td>14%</td>
</tr>
<tr>
<td>FGCI 2</td>
<td>2650</td>
<td>2345</td>
<td>13%</td>
</tr>
<tr>
<td>FGDI 1</td>
<td>3911</td>
<td>3523</td>
<td>11%</td>
</tr>
<tr>
<td>FGDI 2</td>
<td>4057</td>
<td>3688</td>
<td>10%</td>
</tr>
<tr>
<td>CGCI 1</td>
<td>2189</td>
<td>1855</td>
<td>18%</td>
</tr>
<tr>
<td>CGCI 2</td>
<td>2142</td>
<td>1831</td>
<td>17%</td>
</tr>
<tr>
<td>CGDI 1</td>
<td>3358</td>
<td>2895</td>
<td>16%</td>
</tr>
<tr>
<td>CGDI 2</td>
<td>3153</td>
<td>2766</td>
<td>14%</td>
</tr>
</tbody>
</table>

Figure 3. Comparative Execution Time (ms)

Hybrid (MPI+OpenMP) approach reduces parallel application execution time by utilizing system resources efficiently. Figure 4 to Figure 7 show comparative chart of average CPU utilization. We observe about 10-20% improvement in CPU utilization in case of Hybrid (MPI+OpenMP) approach.

Figure 4. CPU Usage for FGCI Problem (MPI approach)

Figure 5. CPU Usage for FGCI Problem (MPI+OpenMP approach)

Figure 6. CPU Usage for CGDI Problem (MPI approach)

Figure 7. CPU Usage for CGDI Problem (MPI+OpenMP approach)

Hybrid (MPI+OpenMP) approach reduces communication overhead of MPI. Figure 8 shows comparison of number of messages passed between MPI and hybrid approach. We observe maximum reduction in number of messages passed in case of CGCI problems.
5. CONCLUSION

Commodity multi-core processors cluster is becoming more popular for High Performance Computing (HPC). The major attraction from such cluster is cost-effectiveness than an expensive cluster of SMP nodes. Most of the authors compared MPI and hybrid (MPI+OpenMP) programming paradigm on cluster of SMP nodes. In this paper, we compare performance of MPI and hybrid (MPI+OpenMP) programming paradigm on cluster of commodity multi-core nodes. We have shown that hybrid programming approach yields better performance results in most of the cases. It yields better CPU utilization. We have also investigated the suitability of cluster depending on the nature of the problem classified as fine-grained or course-grained problems combined with computation-intensive or data-intensive nature of the problems. We have shown that cluster of commodity multi-core processors is best suited for CGCI types of problems and least suitable for FGDI types of problems.

REFERENCES