Performance Study of Parallel Programming Paradigms on a Multicore Clusters using Ant Colony Optimization for Job-flow scheduling problems

Nagaveni V # Dr. G T Raju*

*Department of Computer Science and Engineering
Bharathiar University Coimbatore, 641046, Tamilnadu, India

#Department of Computer Science and Engineering
RNS Institute of Technology, Bangalore -560061, Karnataka, India

Abstract: Parallel Programming technique is an important technique for various intensive and wide ranges of applications which uses multicore and manycore architectures. These architectures are most suitable for clusters than grids, supported by different network topologies. Message Passing and Hybrid Programming are the two techniques employed for the multicore clusters. Ant Colony Optimization(ACO) is one of the important heuristic based approach used to optimize the code in which computational speed is high compared to others. ACO is tested using Job flow scheduling problems as a benchmark and compared with both the approaches and then concluded that parallel ACO gives better performance.

Keywords: Parallel Programming, Message Passing, Hybrid Programming, ACO, Job flow Scheduling problems

I. INTRODUCTION

The research on distributed and parallel systems is one of the most important areas in computer science nowadays. In particular, the use of multicore architectures used in clusters, grids and clouds, supported by different types of networks with different characteristics and topologies, has been considered for the development of parallel algorithms and also for the execution of different intensive processes computation.

Multicore processor is formed by the integration of two or more computational cores within the same chip. These cores are simpler and slower, when combined allows enhancing the global performance of the processor with an efficient use of energy. Incorporation of this type of processors to ordinary clusters gives an architecture called as multicore cluster. These types of architectures establish communications between processing units which are heterogeneous and it can be divided into two groups: Internode and Intranode. Internode communications are done between the cores that are in different nodes and they communicate by exchanging messages through the interconnection network. Intranode communications are done between the cores that are within the same node and they communicate through the different memory levels of the node.

Parallel programming paradigms are classified into three: shared memory, message passing and hybrid programming based on the way tasks communicate and synchronize. In shared memory architectures, such as multicores, tasks communicate and synchronize by reading and writing variables in a shared address space. OpenMP is the most widely used library to program shared memory. Message passing is the most commonly chosen paradigm for distributed architectures, such as traditional clusters. MPI is the most widely used library to program under this paradigm.

Multicore clusters are hybrid architectures that combine distributed memory with shared memory. These architectures have a great interest in analyzing hybrid parallel programming paradigms that allow communications both through message passing and shared memory. The performance of two parallel algorithms designed for the same application but using different programming paradigms is compared over a multicore cluster. The application selected is a divide and conquer technique by means of Ant Colony Optimization algorithm, and it is selected as its computational complexity is $O(n^3)$.

The Ant Colony Optimization algorithm is explained, together with sequential and the parallel algorithms used. In Section 4, the experimental work carried out is described, where as in Section 5, the results obtained are presented and analyzed. Finally, in Section 6 the conclusions and future lines of work are presented.

Job flow scheduling is one of the well-known and important algorithmic paradigm suitable for execution on multi-core clusters. The JSP is known to be a strong NP-hard problem. Hence the JSP included $m$ objectives must also be an NP-hard problem. Mathematical programming approaches for solving multi-objective scheduling problem are computationally intractable for practical problems.

II. RELATED WORK

Numerous works have been done till now that analyze and compare parallel programming paradigms for multicore clusters. To mention but a few, there are different comparisons between message passing and combinations of message passing with shared memory. But all those results vary depending on the type of the problem solved, the algorithms used and the features of the hardware architectures used, which makes research in this area even more significant.
III. Sequential Ant Colony Optimization Algorithm

The ant colony optimization algorithm (SACO) is a probabilistic technique for solving computational problems which can be reduced for finding good paths through graphs. This Sequential ant colony optimization algorithm (ACO) is a member of the ant colony algorithms family, in swarm intelligence methods, and it constitutes some metaheuristic optimizations. The original idea has been diversified to solve a wider class of numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants.

A. Convergence

For some versions of the algorithm, it is possible to prove that it is convergent (i.e., it is able to find the global optimum in finite time). The first evidence of a convergence ant colony algorithm was made in 2000, the graph-based ant system algorithm, and then other algorithms like Ant colony System has been devised. Like most metaheuristics, it is very difficult to estimate the theoretical speed of convergence.

1. Example pseudo-code and formulae

procedure ACO_MetaHeuristic
while(not_termination)
    generateSolutions()
    daemonActions()
    pheromoneUpdate()
end while
end procedure

B. Edge selection

An ant is a simple computational agent in the ant colony optimization algorithm. It iteratively constructs a solution for the problem at hand. The intermediate solutions are referred to as solution states. At each iteration of the algorithm, each ant moves from a state \( x \) to state \( y \), corresponding to a more complete intermediate solution.

C. Pheromone update

When all the ants have completed a solution, the trails are updated by local pheromone update algorithm.

IV. Parallel Ant Colony Optimization (PACO):

PACO is a population-based approach based on the social behavior of ants in which we find the use of parallel computing to reduce computation time, improve solution quality or both.

Most parallel ACO implementations can be classified into two general approaches. The first one is the parallel execution of the ants construction phase in a single colony. Initiated by Bullnheimer ET AL., it aims to accelerate computations by distributing ants to computing elements. The second one, introduced by Stützle, is the execution of multiple ant colonies. In this case, entire ant colonies are attributed to processors in order to speedup computations as well as to potentially improve solution quality by introducing cooperation schemes between colonies.

Recently, a more detailed classification was proposed by Pedemonte ET AL. It shows that most existing works are based on designing parallel ACO algorithms at a relatively high level of abstraction which may be suitable for conventional parallel computers. However, as research on parallel architectures is rapidly evolving, new types of hardware have recently become available for high performance computing. Among them, we find multicore processors which provide great computing power at an affordable cost but are more difficult to program.

D. Parallelization strategies

In PACO algorithms, artificial ants cooperate while exploring the search space, searching good solutions for the problem through a communication mediated by artificial pheromone trails. The construction solution process is incremental, where a solution is built by adding solution components to an initially empty solution under construction. The main steps of the Ant System (AS) algorithm can be described as: initialization phase, ants’ solutions construction, ants’ solutions evaluation and pheromone trails updating. In Algorithm 2 a pseudo-code of AS is given.

Algorithm 2: Pseudo-code of Ant System.

```
// Initialization phase
Pheromone trails \( \tau \); Heuristic information \( \eta \);

// Iterative phase
while termination criteria not met do
    Ants’ solutions construction; Ants’ solutions evaluation;
    Pheromone trails updating;
    Return the best solution;
end while
```

After setting the parameters, the first step of the algorithm is the initialization procedure, which initializes the heuristic information and the pheromone trails. When all ants construct a complete path (feasible solution), the solutions are evaluated. Then, the pheromone trails are updated considering the quality of the candidate solutions found; also a certain level of evaporation is applied. When the iterative phase is complete, that is, when the termination criteria are met, the algorithm returns the best solution generated.

E. Hardware-oriented parallel ACO

Even though they mostly follow the parallel ants and multiple ant colonies approaches, hardware-oriented approaches are dedicated to specific and untraditional parallel architectures. Scheuermann ET AL. [21, 22] designed parallel implementations of ACO on Field Programmable Gate Arrays (FPGA).

![Fig 1. Taxonomy for parallel ACO.](image-url)
In addition to a complete survey, Pedemonte ET AL. proposed taxonomy for Parallel ACO which is illustrated in Fig. 1. Although it provides a comprehensive view of the field, its relatively high level of abstraction does not capture some important features that are crucial for obtaining efficient implementations on modern high performance computing architectures.

Fig. 2. Architectures used for parallel ACO.

This approach deals with the management of multiple colonies which use a global shared memory to exchange information. The whole algorithm executes on a single system and a single node so there is no parallelism at these levels. The colonies are executed in parallel and spawn multiple parallel ants. Therefore, colonies are associated to processes and ants to threads. At the programming level, this can be implemented either with multiple operating system processes and multiple threads or with multiple nested threads. In this implementation, we choose the latter as the available SMP node supports nested threads with a shared memory available to all processors.

Parallelizing ACO in multiple search processes is quite simple: we only need to create a parallel region at the beginning of the sequential algorithm. This way, we can create as many threads as we have colonies. To illustrate the scheme of multiple interacting colonies in a shared-memory model, the simple case of a common best global solution located in the shared memory is implemented. In a shared-memory context, there is no such thing as an explicit broadcast communication step. It is replaced by the use of the global best solution as a dedicated structure in the shared memory. However, it is now used differently and more frequently. At each information exchange step, each thread compares its local value of the best solution located in the shared memory available to all processors.

VI. IMPLEMENTATION

Tests were carried out on a cluster of multicores with four blades and two quad core Intel Xeon e5405 2.0 GHz processors each. Each blade has 10 Gb RAM memory (shared between both processors) and 2 x 6Mb L2 cache for each pair of cores. The operating system is GNU/Linux Fedora 11 (64 bits).

A. Algorithms Used

The algorithms used in this work were developed using C language (gcc compiler version 4.4.2) with the MPI (mpicccompilerversion1.4.3) library for message passing and OpenMP for thread management. The algorithms are detailed below:

- MP: this algorithm is based on the solution described in Section3.2.1, where P is the number of cores used.
- HP: this algorithm is based on the solution described in Section3.2.2, where P is the number of blades used and T is the number of cores in each blade.
B. Tests Carried Out

Based on the features of the architecture, both algorithms were tested using all the cores with different numbers of nodes: two, three and four; this means that $P = \{8, 16$ and $24\}$ for MP. In the case of HP, one process per node was used; this means that $P = \{2, 3, 4\}$ and $T = \{8\}$. Various problem sizes were used: $N = \{3000, 5000, 7000$ and $9000\}$. Each particular test was run five times, and the average execution time was calculated for each of them.

VII. RESULTS

To assess the behavior of the algorithms developed when escalating the problem and/or the architecture, the speedup and efficiency of the tests carried out are analyzed. The speedup metric is used to analyze the algorithm performance in the parallel architecture as indicated in Equation (1).

$$\text{SpeedUp} = \frac{\text{SequentialTime}}{\text{ParallelTime}} \quad (1)$$

To assess how good the speedup obtained is, the efficiency metric is calculated. Equation (2) indicates how to calculate this metric, where $p$ is the total number of cores used.

$$\text{Efficiency} = \frac{\text{SpeedUp}}{p} \quad (2)$$

Figure 3 shows the Speed increased, achieved by the algorithms MP and HP when using eight, sixteen and twenty four cores architecture for different problem sizes ($N$).

The chart of figure 4 shows that both algorithms increase their efficiency as the size of the problem increases and, on the other hand, as it is to be expected in most parallel systems, the efficiency decreases when the total number of nodes used increases. The efficiency levels obtained with both algorithms are low due to the number of communication and synchronization operations carried out and the idle time that processes and threads might have. Despite this, it can be seen that the best efficiency levels are obtained by HP. The difference in favor of HP is due to several factors:

- First, HP reduces latency and maximizes the bandwidth of the interconnection network, since, by using a single, multi-threaded process instead of multiple processes for each node, it groups all task messages corresponding to a node in a single, larger message. It also removes competition for the network at node level.
- Finally, since the distance matrix $d$ is divided in less parts and the work assigned to each of these portions is distributed dynamically among the threads in each process, HP achieves a more balanced work distribution versus the fully static strategy used by MP.

VIII. CONCLUSION AND FUTURE WORK

The performance of two parallel programming paradigms (message passing and hybrid) are compared for current cluster architectures, taking a study case of job flow scheduling problems by means of the Ant Colony Optimization algorithm. The algorithms were tested using various work and architecture sizes. The results obtained show that the hybrid parallelization better leverages the hardware features offered by the support architecture, which in turn yields a better performance.

Future lines of work include the development and optimization of hybrid solutions for other types of applications and their comparison with solutions based on message passing or shared memory or both.

REFERENCES


