Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 10 (8): 664-669, 2010
ISSN 1812-5654

An Asynchronous Parallel Particle Swarm Optimization Algorithm for a Scheduling Problem

S. Hernane, Y. Hernane and M. Benyettou
Simulations and Modelling of Industrial Systems Laboratory, Department of Computer Science,
Faculty of Science, University of Science and Technology, Mohammed Boudiaf USTO,
P.O. Box 1505, El Menaouer, Oran, Algeria

Abstract: This study aimed to measure the performance of an asynchronous algorithm of Particle Swarm Optimization. Particle Swarm Optimization (PSO) is a bio-inspired algorithm founded on the cooperative behavior of agents and is known as a tool to address difficult problems in numerous and divers fields. Like evolutionary algorithms, PSO offer practical approach to solve complex problems of realistic scale and gave results at least satisfactory. In addition, the performance of production systems is related to the scheduling of work on the one hand and to the assignment of this work of the various machines of the system on the other hand. The problem is noted Np-complete. Nevertheless, it remains that solving these problems require large computational demand in terms of CPU time and memory. Also, it is possible to improve solutions quality in various manners. In this study, we apply an asynchronous parallelization strategy of PSO algorithm on a scheduling problem in hybrid Flow-Shop (FSH) systems. We use a fault-tolerant environment by exploiting the computing power of a high-performance cluster with homogeneous processors. In a master/Slave model, PSO algorithm is decomposed to several tasks that are distributed on compute slave nodes. Experimental tests are compared with those obtained by the serial algorithm. Parallel performance is evaluated and improved PSO algorithm by accelerating convergence.

Key words: Swarm intelligence, PAPSO, FSH, scheduling tasks, parallel virtual machine

INTRODUCTION

Combinatorial optimization problems are expressed by a cost function with or without constraints to be minimized or maximize on a set of definitions finished or countable. Metaheuristics bring approximate solutions to large problem instances and require an intensive scientific computation. However, their resolutions are limited by available resources capacities. Thus, suitable distributed parallel models decrease the computing time and improve the quality of obtained solutions. Also, the exploitation of several workstations is an opportunity for parallel computing but new problems are posed (resources heterogeneity, failure of nodes). These new challenges are overcome with the use of high performance clusters and grids computing. Grids computing are an emerging computing model that provides the ability to perform higher throughput computing by taking advantage of many networked computers to model a virtual computer architecture that is able to distribute process execution across a parallel infrastructure. Grids computing are often confused with cluster computing. The key difference is that a cluster is a single set of compute nodes sitting in one location, while a grid is composed of many clusters and other kinds of resources.

In this study, we analyse a parallel model performance of a bio-inspired algorithm: the parallel swarm optimization metaheuristic. The application solves a scheduling problem of tasks in a hybrid Flow Shop production system. We deploy a cluster of nodes based on resources manager (Condor) (Pruyne and Livny, 1995) coupled with Parallel Virtual Machine communication libraries (Requilé, 1995).

The scheduling problem in FSH production systems: In Flow-shop system of N stages, we consider tasks in entry that pass by the first stage, then the second, until the N th stage. The system includes an inter-stage stock and in each stage, there is only one resource (machine). Hybrid flow-shop problem is a generalization of the classical flow-shop problem by permitting multiple parallel machines available to process the tasks that have entered the stage.

Corresponding Author: S. Hernane, Simulations and Modelling of Industrial Systems Laboratory,
Department of Computer Science, Faculty of Science, University of Science and Technology,
Mohammed Boudiaf USTO, P.O. Box 1505, El Menaouer, Oran, Algeria

The hybrid flow-shop is defined by a set of N stages and each stage i (i = 1,..., N) is composed of $N_{(i)}$ parallel machines. The M tasks pass by N stages and the completion treatment dates of tasks are known as a preliminary . On each stage, a task is treated by only one machine and all machines can treat the same operations but with different performance (the machine performance can be related to the competence of the agent which uses it). Thus, the main goal is to find a scheduling of tasks in entry and an assignment of these tasks to the various machines, which optimize a performance criterion. The criterion consists in minimizing the completion time of work and is noted C_{max}. This is a combinatorial optimization problem and could not be solved by exact algorithms.

Particle swarm optimization: Like the other evolutionary algorithms, Particle Swarm Optimization (PSO) is a population based stochastic optimization technique inspired by social behavior of bird flocking or fish schooling.

Reynolds, Heppner and Grenander created simulation models on flights of birds (Li-Ping *et al*., 2005). Reynolds was intrigued by the aesthetic choreography of the birds moving while, Heppner was interested by the discovery of fundamental rules which made that the birds gathered, changed direction suddenly, dispersed then again gathered in synchronized manner. The finality of this behavior is the effective search for food allowing survival in the environment.

Sociologists confirm that a local process is at the base of this dynamic and unforeseeable behavior of individuals groups. In a research space, each individual is guided by his own experiment and benefits from the preceding experiments of the other individuals.

The PSO introduced by Kennedy and Eberhart (1995). Based on the Heppner model, they try to simulate the human social behavior (Koh *et al*., 2005). The scientists estimate that find a source of food is similar to find a solution in a common research space and assimilate the physical movements of animals to the opinion changes of individuals.

In the PSO algorithm, the birds in a flock are symbolically represented as particles. These particles can be considered as simple agents flying through a problem space. A particle's location in the multi-dimensional problem space represents one solution for the problem. When a particle moves to a new location, a different problem solution is generated (Hu *et al*., 2003).

The canonical model: Assume a D dimensional search space. The position of the ith particle in design space is a D dimensional vector $X_i = (X_{i1},..., X_{id})$. The velocity of this particle is also a D dimensional vector $V_i = (V_{i1},...,V_{id})$. The best previous position pbest encountered by the i_{-th} particle is denoted by $P_i = (P_{i1},..., P_{id})$. Then, the swarm is manipulated by the equations (Schi and Eberhart,1998):

$$V_{ID} = W \times V_{ID} + C_1 \times rand() \times (Pbest_{ID} - X_{ID}) + C_2 \times rand() \times (gbest - X_{ID}) \quad (1)$$

$$X_{ID} = X_{ID} + V_{ID} \quad (2)$$

where, rand() is a random number uniformly distributed within [0,1], C_1 is a cognitive parameter used to bias the search of a particle toward its best experience, C2 is a social parameter used to bias the search of a particle toward the best experience of the whole swarm, W is an inertia weight, used as mechanisms for the control of the velocity's magnitude

A large inertia weight facilitates global exploration by searching new spaces while a small inertia weight facilitates local search. Suitable selection of the parameter W provides a balance between the local and global exploration search abilities to reach the optimum in a minimum of iterations (Schutte *et al*., 2004).

The algorithm: In PSO algorithm, each particle is characterized by:

- Position and velocity
- The objective function cost for its current position or that acquired previously: pbest
- The knowledge of its neighbors
- The best previous and current position acquired among all the particles in the swarm gbest

The PSO is initialized with a set of random particles (solutions) and then searches for optima by updating generations (Hu *et al*., 2003).

In each step, the particle is updated and makes a compromise between three possible choices (Fig. 1):

- To follow its own way
- To return towards its best obtained position
- To move towards the best obtained position of the swarm

Each particle, updates velocity and position according pbest and gbest.

```
1 FOR EACH PARTICLE DO
        INITIALISATION
END

2 FOR EACH PARTICLE DO
        CALCUATE FITNESS VALUE;
        IF THE FITNESS VALUE IS BETTER
        THEM THE BEST FITNESS VALUE
        (pBEST)
        THEN SET CURRENT VALUE AS THE
        NEW PBEST
END

CHOOSE THE PARTICLE WITH THE BEST
FITNESS VALUE OF ALL THE PARTICLES AS
THE GBEST.

3 FOR EACH PARTICLE DO
CALCULATE PARTICLE VELOCITY
ACCORDING EQUATION (1)
UPDATE PARTICLE POSITION ACCORDING
EQUATION (2)
END
```

Fig. 1: Pseudo code of PSO algorithm

The velocity restriction constant V_{max} was also included in the algorithm. In Eq. 1, if the sum of the tree parts exceeds a constant value, then Particle's velocity is clamped to a maximum velocity V_{max} that is specified by the user. This mechanism prevents the phenomenon of swarm explosion (Li-Ping *et al*., 2005).

The pseudo code of PSO algorithm is given as follow (Koh *et al*., 2005):

An asynchronous parallelization of PSO on a high performance cluster: High-performance clusters are implemented primarily to provide increased performance by splitting a computational task across different nodes. Clusters are often managed by cluster resources which optimize the use of resources such as computational hardware, storage, network and increase robustness of long-running parallel optimization algorithms.

For our application we choose a type of resources manager known as batch queue manager interfaced with a communication libraries.

The resource management mechanism: Condor is a batch scheduler for compute-intensive jobs which provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring and resource management. Users submit their jobs, Condor places them into a queue, chooses when and where to run the jobs

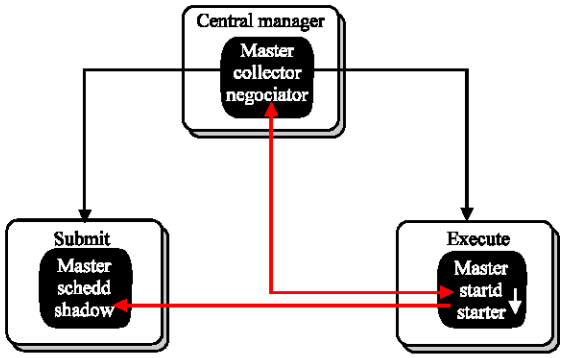

Fig. 2: Roles and daemons of condor's pool

based upon a policy, carefully monitors their progress and ultimately informs the user upon completion.

Within Condor, each node can play one or many roles simultaneously and the various roles are implemented by several daemons. It can be submit node, execute node or a central manager. The pool contains only one central manager. It is the important part in the cluster. The machine collect informations and negotiate between availables resources and requests.

As can be seen in Fig. 2, central manager is managed by master, collector and negociator daemons. Started daemon represents a given resource on execute machine. It is responsible for enforcing the policy that resource owners conFig which determines under what conditions remote jobs will be started, suspended, resumed, vacated, or killed. When the startd is ready to execute a Condor job, it spawns the starter daemon.

Starter is the entity that actually spawns the remote Condor job on a given machine.

The Schedd daemon is present on the submit machine. It represents resources requests to the Condor pool.

The Condor pool can also contain a checkpoint server. It is an optional role. This node stores the entire checkpoint files for the jobs submitted (Fig. 2).

The PVM is a viable scientific computing platform appropriate as well for multiprocessors with shared memory as with single processors with distributed memory. It is composed of a set of demons and functions libraries. Programs use PVM library routines for functions such as process initiation, message transmission and reception. Than, requests cluster's resources are managed by a master process created by Condor, PVM daemons communicate by sending and receiving messages. The main characteristics of our cluster are reported in Table 1.

Asynchronous parallel strategy of PSO algorithm: Serial algorithm follows a synchronous model by updating each

Table 1: The characteristics of the cluster

Characteristics	Description
CPU	3 Ghz
Memory	512
Cluster's size	2 to14
Network	Fast Ethernet 100 Mbits/s
OS	LINUX
Compilator	Gcc
Communication	PVM
Workload manager	Condor

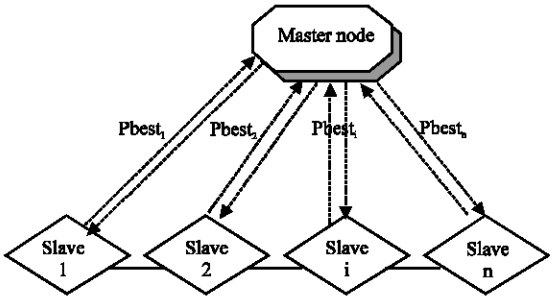

Fig. 3: Master Slave model

best individual pbest and group gbest fitness values then their associated velocities and positions. Serial PSO was already implemented to solve the FSH scheduling problem Eq. 1 nevertheless, several strategies of parallelization are possible.

The parallel algorithm should operate in such a fashion that it can be easily decomposed for parallel operation on a cluster. It is obvious to note that fitness function evaluation is done in a concurrent way. Parallel implementation follows a master/slave model (Fig. 3). The master spawned tasks. Tasks number corresponds to the swarm size. After, the master initialises and sends data to slave's nodes that achieve pbest's evaluation and transmit the result to master node. After receiving all individuals' fitness values, the master deduces gbest and updates parameters.

This is a parallel synchronous particle swarm optimization PSPSO. Step 2 of PSO algorithm is modified as follow:

```
2 do in parallel
    Compute fitness value
    If the fitness value is better than the best fitness value (pBest)
    Then set current value as the new pbest
End
```

Synchronization of nodes: The performance of a parallel program is related to execution time. By measuring how long the parallel program needs to run to solve our problem, we can directly measure its effectiveness. There are two important performance metrics: speedup and parallel effectiveness.

Speedup is the ratio of the serial program execution time T_s to the parallel execution time T_p on N processors. Parallel efficiency is the ratio between the speedup and the number N of processors. Parallel effectiveness is an indication of scalability. Ideal speedup should equal the number of processors with a maximum of parallel efficiency of 1 (100%).

Some implementations were carried out in this context, in particular (Schutte *et al*., 2004) for a global biomechanical optimization. Experimentations revealed that improved convergence rates could be obtained within a homogeneous cluster of computers without interruptions. Also, the parallel effectiveness reduced as compute nodes number increases. Indeed, it is easy to generate a bottleneck when the parallel processors reach simultaneously the shared disc of master node. Due to the synchronization requirement of the current parallel implementation, the resulting load imbalance caused by even one slow fitness evaluation was sufficient to degrade parallel performance rapidly with increasing number of nodes. This degradation produced also when master node was idle the most of time (Schutte *et al*., 2004).

This low effectiveness is overcome with asynchronous version of PSO algorithm. Parallel Asynchronous Particle Swarm Optimization (PAPSO) has proposed and implemented recently (Koh *et al*., 2005). In PAPSO, particle positions and velocities are updated continuously based on currently available data (gbest).

Thus, the master computes velocity and position's particle as soon as it received pbest from slave node. Consequently, PAPSO incorporates a dynamic load balancing scheme with a centralized task queue approach to reduce load imbalance.

For all this reasons, we chose PAPSO in our implementation. Experimental tests are used on a cluster of 14 compute slave nodes. The master node is used to coordinate the particle queue. The main characteristics of our cluster are reported in Table 1.

Experimental tests and results: Our tests treat scheduling of 20 tasks in entry on FSH of 2 stages with three machines in the first and two machines in the second. The problem is noted by FSH $(P_3,P_2)\|C_{max}$. In this case, the objective function corresponds to C_{max}. Experimental tests aim to study the impact of swarm's size and cluster's size on numerical results. Both serial and parallel algorithm are executed and compared. Ten independent experiments have been performed. The C_{max} cost is computed; Speedup and the parallel effectiveness are measured.

Our infrastructure is optimized for large sized problem instances. We apply experimental tests by varying

Table 2: Numerical results of PSO and PAPSO

Particles	C_{max} PSO	C_{max} PAPSO
14	624	614
28	539	518
42	596	528
56	515	556
70	554	493
84	543	513
98	539	-

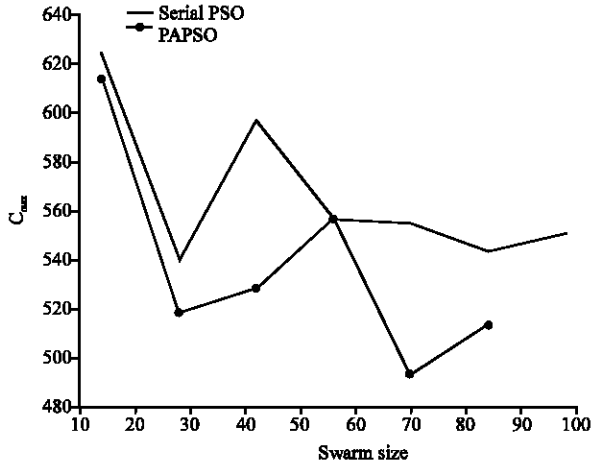

Fig. 4: Parallel distributed PSO: C_{max}

cluster's size (6, 8, 10, 12 and 14). The swarm is initialised seven times with different particle's sizes and distributed to compute nodes for evaluation of fitness values. For example, on a cluster with 10 nodes, the swarm's size vary from 10, 20, 30, 40, 50, 60 to 70 particles. With 14 nodes, the sizes of swarm vary from 14, 28, 42, 56, 70, 84 to 98 particles.

Serial execution algorithm results and parallel execution results on a cluster with 14 nodes are given by:

According to the numerical results, it appears that adding more processors improves performance of parallel PSO. The obtained results support the claim that the PAPSO algorithm outperforms the serial PSO algorithm in all cases (Table 2, Fig. 4). It is speculated that because the updating occurs immediately after each fitness evaluation, the swarm reacts more quickly to an improvement in the best-found fitness value and improved convergence is accelerated.

Concerning parallel results, we notice that an improvement is obtained when each node treats two particles (with a swarm of 28 particles). Increasing the size of the swarm produces an unavoidable performance loss in terms of convergence rate. This is when the swarm is initialised with 42 and 56 particles. In this case, each node treats 3 and 4 particles. The reason is that the more processors are used, the more number of particles each

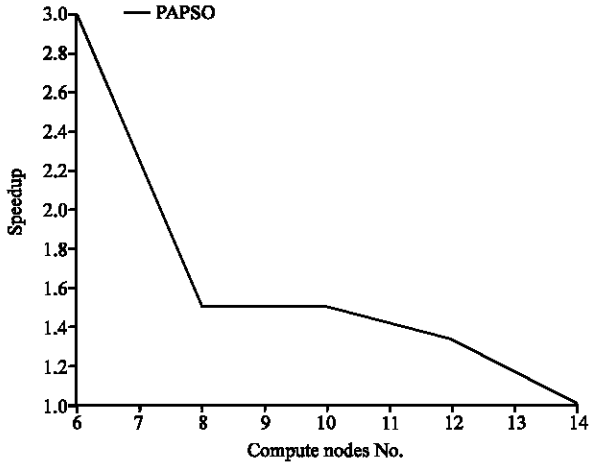

Fig. 5: PAPSO: Speedup

individual processor owns, which results in relative lower computation cost and higher communication overhead to parallel processing. Thus, the master updates velocities and positions values more slowly and solution quality decreases.

With 70 particles, convergence rate reaches the optimum. Objective function cost reaches 493. This is can be explained by increase of swarm's size. Indeed, with enhanced amount of particles, the algorithm evolves within a large research space. However, exceeding this value, the C_{max} cost increases, consequently, the swarm will react more slowly to changes of the best fitness value in the design space. This is due to communication latency between slave's nodes and the master. We observed also that our cluster limits the swarm's size to 84 particles (84 tasks) and cannot assign more than six tasks to each slave node.

In addition and in order to study parallel performance, we vary the number of compute nodes from 6 to 14.

As can be seen in Fig. 5, the speedup goes down when using more nodes. Our application requires much communication between the coordinate node and the slave's nodes. For this reason, our speedup curve presents saturation (Fig. 5). Thus, parallel effectiveness decreases when the number of nodes increases (Fig. 6). This occurs when communication time grows faster than computation time decreases.

According curves, we can conclude that the communication overhead is the main cause of the speedup saturation.

In general, our experiments present good overall performance results of our parallel asynchronous implementation in terms of optimization and solution quality. We can hope for a better improvement by increasing the size of the cluster.

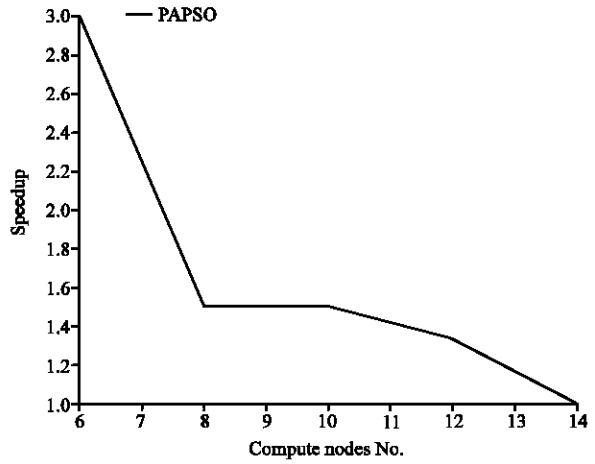

Fig. 6: PAPSO: Parallel effectiveness

CONCLUSION

This study has presented a parallel asynchronous implementation of the particle swarm optimization algorithm. The method was validated by an industrial scheduling problem. The PSO is an approach to problems whose solutions can be represented as a point in an n-dimensional solution space. Many improvements of original model PSO were proposed by adjusting parameters in the algorithm. However, Parallel optimization uses multiple computers or processors with an aim of to reduce the elapsed time and to accelerate convergence rate. Then, a good acceleration can be obtained. Both, single node and parallel implementations of the algorithm have been developed and applied on a high-performance cluster with a fault-tolerant strategy to obtain an efficient and robust parallel scheme.

Two widely used metrics have been used for the evaluation of the results. Through the obtained results, we note that decomposition of PSO algorithm in several tasks produce improvements and accelerate convergence however too many processors decreases the parallel efficiency and cluster size limits the number of tasks spawned by master node.

Lastly, we can say that distributed system technology, grid computing offers a number of potential uses and benefits for parallel and distributed algorithms and a wide range of computational problems. Nevertheless several parameters should be studied beforehand: the parallel model, the topology, the population's size as well as the infrastructure used play a dominate role in the solution quality and the parallel effectiveness. Future research includes parallelization of other another bio-inspired algorithm: ant-colony optimization.

ACKNOWLEDGMENTS

Authors Acknowledgments are intended to University of Science and Technology of Oran and in particular to Laboratory of Simulations and Modelling of Industrial Systems which has placed at our disposal the necessary material and a suitable environment for carrying out this study.

REFERENCES

Hu, X., R.C. Eberhart and Y. Shi, 2003. Engineering optimization with particle swarm. Proceedings of the IEEE Swarm Intelligence Symposium 2003, April 24-26, IEEE Computer Society, Washington, DC, USA., pp: 53-57.

Kennedy, J. and R. Eberhart, 1995. Particle swarm optimization. Proceedings of the IEEE International Conference on Neural Networks, Nov. 27-Dec. 01, Perth, WA, Australia, pp: 1942-1948.

Koh, B.I., A.D. George, R.T. Haftka and B.J. Fregly, 2005. Parallel asynchronous particle swarm optimization. Int. J. Numerical Methods Eng., 67: 578-595.

Li-Ping, Z., Y. Huan-Jun and H. Shang-Xu, 2005. Optimal choice of parameters for particle swarm optimization. J. Zhejiang Univ. Sci. A, 6: 528-534.

Pruyne, J. and M. Livny, 1995. Providing resource management services to parallel applications. Workshop on Job Scheduling Strategies for Parallel Processing. Proceedings of the International Parallel Processing Symposium (IPPS'95), April 15.

Requilé, G., 1995. PVM: Parallel virtual machine, les aspects communication. Laboratoire de Mécanique et Génie Civil-CNRS URA 1214-Université Montpellier, 1995. http://1995.jres.org/actes/appli4/1/requile.pdf.

Schi, Y. and R. Eberhart, 1998. Parameter selection in particle swarm optimization. Proceedings of the 7th Internationl Conference on Evolutionary Programming, March 25-27, Springer-Verlag London, UK., pp: 591-600.

Schutte, J., J. Reinbolt, B. Fregly and R. Haftka1, 2004. Parallel global optimization with the particle swarm algorithm. Int. J. Numerical Methods Eng., 127: 465-474.