A fault-tolerant parallel heuristic for assignment problems

E-G. Talbi, J-M. Geib, Z. Hafidi, D. Kebbal*

Abstract. This paper presents a new approach for parallel heuristic algorithms based on adaptive parallelism. Adaptive parallelism was used to dynamically adjust the parallelism degree of the application with respect to the system load. This approach demonstrates that high-performance computing using heterogeneous workstations combined with massively parallel machines is feasible to solve large assignment problems. The fault-tolerant algorithm allows a minimal loss of computation in case of failures. The proposed algorithm exploits the properties of this class of applications in order to reduce the complexity of the algorithm. The parallel heuristic algorithm combines different search strategies: simulated annealing and tabu search. Encouraging results have been obtained in solving the quadratic assignment problem. We have improved the best known solutions for some large real-world problems.

1 Introduction

The quadratic assignment problem (QAP) is one of the hardest among the NP-hard combinatorial optimization problems [PRW94]. The QAP arises in many applications such as VLSI module placement and process-processor mapping in parallel processing [MT91]. The most widely used metaheuristics to solve the QAP are: simulated annealing, tabu search and genetic algorithms [Ree93]. Large real-world problems cannot be solved within a reasonable amount of time. Consequently, parallel heuristics must be used.

The proliferation of powerful workstations and fast communication networks (such as ATM and Myrinet) with constantly decreasing cost/performance ratio have shown the emergence of heterogeneous workstation networks (NOWs) and homogeneous clusters of processors (COWs) as platforms for high performance computing. These parallel platforms are generally composed of an important park of machines shared by many users. Load analysis of those platforms during long periods of time showed that only a few percentage of the available power was used and a substantial amount of idle time [Nic87][TL89]. In addition, a workstation belongs to an owner who will not tolerate external applications degrading the performance of his machine. The personal character of workstations must be taken into account. The faulty nature of workstations increases considerably the failure frequency of NOWs. If the failure rate of a node is q, the failure rate of a NOW of n workstations is at least nq, considering the network failures. Therefore, the performance of the NOW decreases and users can suffer from losses of their computation, especially for long running applications such us solving the QAP.

* LIFL URA-369 CNRS / Université de Lille 1, Bât. M3 59655 Villeneuve d'Ascq Cedex FRANCE. E-mail: {talbi, geib, hafidi, kebbal}@lifl.fr

Many parallel biologically inspired metaheuristics have been proposed in the literature. In general, they don't use advanced programming tools (such as load balancing, adaptive parallelism and fault-tolerance) to efficiently use the machines. Most of them are developed for dedicated parallel homogeneous machines. Our aim is to develop a fault-tolerant parallel adaptive heuristic, which can benefit greatly from a platform having combined computing resources of massively parallel machines (MPPs) and networks of workstations (NOWs and COWs). The parallel algorithm proposed combines two search strategies: tabu search (TS) [Glo89] and simulated annealing (SA) [KGV83]. TS and SA achieved widespread success in solving practical optimization problems in different domains (resource management, process design, logistics, telecommunications, etc.). Promising results of applying TS and SA to a variety of academic optimization problems (traveling salesman, quadratic assignment, time-tabling, job-shop scheduling, etc.) are reported in the literature [GL92].

2 The heuristic algorithm

2.1 Tabu search

A combinatorial optimization problem is defined by the specification of a pair (X, f), where the search space X is a discrete set of all (feasible) solutions, and the objective function f is a mapping $f : X \longrightarrow R$. A neighborhood N is a mapping $N : X \longrightarrow P(X)$, which specifies for each $S \in X$ a subset $N(S)$ of X of neighbors of S. The most famous local search optimization method is the *descent method*. A descent method starts from an initial solution and then continually explores the neighborhood of the current solution for a better solution. If such a solution is found, it replaces the current solution. The algorithm terminates as soon as the current solution has no neighboring solution of better quality. Such a method generally stops at a local but not global minimum.

Unlike a descent method as it might make a down-hill move at each iteration of the search, *tabu search* (TS) selects the best neighbor solution even if this results in a worst solution than the current one. However, this strategy may result in cycling. So a *tabu list* is introduced to keep information about recently examined moves (short-term memory). A tabu move applied to a current solution may appear attractive because it gives, for example, a solution better than the best found so far. We would like to accept the move in spite of its status by defining *aspiration conditions*. Other advanced techniques may be used such as *intensification* to encourage the exploitation of a promising region in the search space, and *diversification* to encourage the exploration of new regions (long-term memory).

2.2 An heuristic for the QAP

The QAP can be defined as follows. Given:

- a set of n objects $O = \{O_1, O_2, ..., O_n\}$, and a set of n locations $L = \{L_1, L_2, ..., L_n\}$,
- a flow matrix C, where each element c_{ij} denotes a flow cost between the objects O_i and O_j, and a distance matrix D, where each element d_{kl} denotes a distance between location L_k and L_l,

find an object-location bijective mapping $M : O \longrightarrow L$, which minimizes the objective function $f = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij}.d_{M(i)M(j)}$

The QAP is NP-hard [PRW94]. This fact has restricted exact algorithms (such as branch and bound) to small instances ($n < 22$). An extensive survey and recent developments can be found in [PRW94]. To apply TS to the QAP, we must define the neighborhood structure of the problem and its evaluation, the short-term memory to avoid cycling and the long-term memory for the intensification/diversification phase.

Neighborhood structure and evaluation: To represent a solution of the QAP, we use a representation which is based on a permutation of n integers $s = (l_1, l_2, ..., l_n)$, where l_i denotes the location of the object O_i. We use a pair exchange move in which two objects of a permutation are swapped. We use the formulae reported in [Tai95] to efficiently compute the variation in the objective function due to a swap of two objects. The evaluation of the neighborhood can be done in $O(n^2)$ operations.

Short-term memory: The tabu list contains pairs (i, j) of objects that cannot be exchanged (recency-based restriction). The size of the tabu list varies between $\frac{n}{2}$ and $\frac{3n}{2}$. The maximum number of TS tasks is set to n and each TS task is initialized with a different tabu list size from $\frac{n}{2}$ to $\frac{3n}{2}$ with an increment of 1. The aspiration function allows a tabu move if it generates a solution better than the best found solution. The total number of iterations is limited to $1000n$.

Long-term memory and simulated annealing: We use as a long-term memory a frequency-based memory which complements the information provided by recency-based memory. A matrix $F = (f_{i,k})$ represents the long-term memory. Let S denote the sequence of all solutions generated. The value $f_{i,k}$ represents the number of solutions in S for which $s(i) = k$. This quantity identifies the number of times the object i is mapped on the location k. The different values are normalized by dividing them by the average value which is equal to $\frac{1+nb\_iterations}{n}$. If no better solution is found in $100n$ iterations, the *intensification* phase is started. The intensification phase starts from the best solution found in the current region and an empty tabu list. The use of the frequency-based memory will penalize non-improving moves by assigning a larger penalty to swaps with greater frequency values.

A simulated-annealing like process is used in the intensification phase. The relevance of our approach compared to pure simulated-annealing is that it exploites memory of TS for selecting moves. For each move $m = (i, j)$, an incentive value P_m is introduced in the acceptance probability to encourage the incorporation of good attributes. The value of P_m is initialized to $Max(f_{i,s(i)}, f_{j,s(j)})$. Therefore, the probability that a move will be accepted diminishes with small values of P_m. The *diversification* phase is started after $100n$ iterations are performed without any improvements of the restarted TS algorithm. Diversification is performed in $10n$ iterations. The search will be forced to explore new regions by penalizing the solutions often visited. The penalty value associated to a move $m = (i, j)$ is $I_m = Min(f_{i,s(i)}, f_{j,s(j)})$. A move is tabu-active if the condition $I_m > 1$ is true. Therefore, we will penalize moves by assigning a penalty to moves with greater frequency. The number of iterations is

Step 1 : Initialization
- Choose the best found solution S_0 ($S = S_0$)
- $nbiter = 0$ /* current iteration */
- $T := T_{max}$ /* Initial temperature $T_{max} = 10$ */

Step 2 : Iteration
- **Repeat**
  - nbiter := nbiter + 1
  - **For** j:=1 **To** NB_MAX **Do** /* $NB\_MAX = n$ */
    - Generate a random move m which transforms S to S'
    - $\triangle S = f(S') - f(S)$
    - **If** $(\triangle S < 0)$ or $(random(0,1) < exp(-\frac{\triangle S}{T * P_m}))$ **Then** $S := S'$
  - **End For**
  - $T := a.T$ /* annealing schedule with $a = 0.9$ */
- **Until** $T < T_{min}$ /* $T_{min} = 0.5$ */

Fig. 1. Simulated annealing with a penalized objective function for the intensification phase.

large enough to drive the search out of the current region. When the diversification phase terminates, the tabu status based on long-term memory is dropped.

3 A parallel adaptive fault-tolerant implementation

In this paper, a straightforward approach has been used to introduce adaptive parallelism in TS. It consists in multiple independent TS algorithms running in parallel. This requires no communication between the sequential tasks. The algorithms are initialized with different solutions. Different parameter settings are also used (size of the tabu list).

3.1 Adaptive parallelism

Parallel adaptive algorithms are parallel computations with a dynamically changing set of tasks. Tasks may be created or killed as a function of the load state of the parallel machine. A task is created automatically when a node becomes idle. When a node becomes busy, the task is killed. As far as we know, no work has been done on parallel adaptive heuristics.

The programming style supported is the master/workers paradigm (SPMD model). The master process generates work to be computed by the workers. Each worker process receives a job from the master, computes a result and sends it back to the master. The master/workers paradigm works well in a dynamic environment because:

- when a node becomes idle, a worker process can be started there ;
- when a node becomes busy, the master process can terminate the worker process and reschedule its work on the next available node.

The master implements a central memory through which pass all communications, and that captures the global knowledge acquired during the search. The number of workers created initially by the master is equal to the number of idle nodes in the parallel platform. Each worker implements a sequential TS task. The initial solution is generated randomly and the tabu list is empty.

The parallel adaptive TS algorithm reacts to two events:
- Transition of the load state of a node from idle to busy : If a node hosting a worker becomes loaded, the master *folds up* the application by withdrawing the worker. The concerned worker puts back all pending work to the master and dies. The pending work is composed of the current solution, the best local solution found, the short-term memory, the long-term memory and the number of iterations done without improving the best solution. The master updates the best global solution if it's worst than the best local solution received.
- Transition of the load state of a node from busy to idle : When a node becomes available, the master *unfolds* the application by starting a new worker on it. Before starting a sequential TS, the worker task gets the values of the different parameters from the master: the best global solution and an initial solution which may be an intermediate solution found by a folded TS task, which constitute a "good" initial solution. In this case, the worker receives also the state of the short-term memory, the long-term memory and the number of iterations done without improving the best solution.

The parallel run-time system to be used has to support dynamic adaptive scheduling of tasks, where the programmer is totally preserved from the complex task of managing the availability of nodes and the dynamics of the target machine. Piranha (under Linda) [GK91], CARMI/Wodi (under PVM/Condor) [PL95], and MARS [HTG96] are representative of such scheduling systems. We have used the MARS dynamic scheduling system. The MARS system is implemented on top of the UNIX operating system. We use an existing communication library: PVM[2]. The execution model is based on a preemptive multi-threaded run-time system: PM[2] [3]. The basic functionality of PM[2] is the LRPC (Lightweight Remote Procedure Call), which consists in forking a remote thread to execute a specified service.

3.2 Fault-tolerance

Solving very large optimization problems can take several hours. The aspects of fault tolerance must be taken into account. The checkpointing/roll-back mechanism is a well-known software solution for fault-tolerant programming. The approach consists of saving the application state periodically on a stable storage. When a failure occurs, the application state is restored from its last checkpoint and the computation resumes with minimal losses. All coordinated checkpoint algorithms [CL85, Pla93] presented in the literature use in general $O(n^2)$ [4] control messages and handle considerable amounts of data because they store address space of all processes of the application.

[2] Parallel Virtual Machine
[3] Parallel Multi-threaded Machine
[4] n: is the number of processes

Checkpointing: The checkpointing algorithm benefits from the properties of parallel adaptive programs to reduce the number of control messages and the size of the checkpoint file at checkpoint time [KTG97]. The approach consists of gathering partial results from all application workers using the fold service, storing them in the master address space and checkpointing only the master task.

The master plays the role of the coordinator. When a checkpointing operation is started, it broadcasts a request to all its workers in order to put back their partial results. On receiving this request, each worker suspends its execution and starts a specific thread which puts back the partial results to the master. The control thread is synchronized with the computation threads in order to define a consistent local state of the worker. After receiving partial results from all its workers, the master takes its local checkpoint.

Recovery: The recovery consists of getting back the master state from the checkpoint file. The rolled back master reenrolls in MARS run-time system normally as a new application but with its old context preserved at checkpoint time and without workers. The data partially processed is redistributed dynamically to the new workers created after requesting the system for available resources at reenrollment time. Thus, the number of workers will be function of the current system load.

The number of control messages exchanged in the checkpointing algorithm is $2n + 2$. Therefore, the complexity is within $O(n)$. Indeed, gathering all partial results, storing them into the master address space and taking only the master local checkpoint without workers can decrease the size of the checkpoint file with an order of n. Our checkpointing/rollback algorithm presents a number of advantages: transparency, portability on heterogeneous sytems and load balancing at recovery time,

4 Computational results

For our experimental study, we collected results from a platform combining a NOW and a COW. The NOW is composed of 126 workstations (PC/Linux, Sparc/Sunos, Alpha/OSF, Sparc/Solaris, ...) owned by researchers and students of our University. The COW is an Alpha-farm composed of 16 processors connected by a crossbar switched interconnection network. The parallel adaptive heuristic competes with other applications (sequential and parallel) and owners of the workstations.

4.1 Adaptability of the parallel algorithm

The performance measures we use when evaluating the adaptability of the parallel heuristic algorithm are execution time, overhead, the number of nodes allocated to the application, and the number of fold and unfold operations. The overhead is the total amount of CPU time required for scheduling operations. Table 1 summarizes the results obtained for 10 runs. The average number of nodes allocated to the application does not vary significantly, but we have a significant load variation (number of fold and unfold operations). During an average execution time of 2h25mn, 79 fold

Table 1. Experiment results obtained for 10 runs of a large problem (Sko100a) on 100 processors.

	Mean	Deviation	Min	Max
Execution time (mn)	145.75	23.75	124	182
Overhead (sec)	8.36	0.24	8.18	8.75
Number of nodes allocated	71	15.73	50	92
Number of fold operations	79	49.75	24	149
Number of unfold operations	179	45.55	120	248

operations and 179 unfold operations are performed. This corresponds to one new node every 0.8mn and one node loss every 2mn. We see also that the scheduling overhead is low comparing to the total execution time (0.09% of the total execution time).

4.2 Fault-tolerance of the parallel algorithm

Table 2 presents the running times on a network of 13 SUN4 workstations of the parallel algorithm without checkpointing, and the overhead induced by the mechanism with a 5 *mn* checkpointing period over all its checkpoints represented in the *Checkp number* column. The results show that the overhead induced by the checkpointing mechanism for the master is not important (under 1% of the running time of the application). The mean number of workers which participate in each checkpoint is

Table 2. Running times and checkpointing overhead

Without Ckpt (sec)	Checkp number	Checkp Overh (sec)	%
18821	63	127	0.67

equal to 10.96. The mean checkpoint overhead of the workers is equal to 1.56 *sec* and represents 0.32% of the total execution time, which is insignificant. Beside these enhancements, our algorithm as we have mentionned it above tries to reduce the amount of data stored in the checkpoint file by checkpointing only the master task. The storage space required is 1713 Kbytes.

4.3 QAP results

To evaluate the performance of the parallel heuristic algorithm in terms of solution quality and search time, we have used standard QAP problems of different types (QAP library): random and uniform distances and flows (Tai100a), random flows on grids (Sko100a-c, Tho150, Wil100), and real-life (Esc128, Tai150b, Tai256c). The parallel algorithm was run 10 times to obtain an average performance estimate. Table 3 shows the best known, best found, worst, average value and the standard deviation

of the solutions obtained for the chosen large instances ($n > 50$). The search cost was estimated by the wall-clock time to find the best solution, and hence account for all overheads. For random-grid problems, we found the best known solutions

Table 3. Results for large problems ($n \geq 50$) of different types.

Instance	Best known	Best found	Gap	Search time (mn)
Tai100a	21 125 314	21 193 246	0.32%	117
Sko100a	152 002	152 036	0.022%	142
Sko100b	153 890	153 914	0.015%	155
Sko100c	147 862	147 862	0%	132
Wil100	273 038	273 074	0.013%	389
Tho150	8 134 030	8 140 368	0.078%	287
Esc128	64	64	0%	230
Tai150b	499 348 972	**499 342 577**	-0.0013%	415
Tai256c	44 894 480	**44 810 866**	-0.19%	593

(for *Sko100c*) or solutions very close to best known solutions. The most difficult instance for our algorithm is the random-uniform *Tai100a*, in which we obtain a gap of 0.32% above the best known solution. For this class of instances, it is difficult to find the best known solution but simple to find "good" solutions. For the third class of instances (real-life or real-life like), the best known solutions for *Tai150b* and *Tai256c* (generation of grey patterns) have been improved. According to the results, we observe that the algorithm is well fitted to a large number of instances but its performance decreases for large uniform-random instances (Tai100a).

5 Conclusion and future work

The dynamic nature associated to the load of NOWs and COWs makes essential the adaptive scheduling of tasks composing a parallel application. The main feature of our parallel heuristic algorithm is to adjust, in an adaptive manner, the number of tasks with respect to available nodes, to fully exploit the availability of machines.

An experimental study has been carried out in solving the QAP. The parallel algorithm which combines tabu search and simulated annealing includes different tabu list sizes and intensification/diversification mechanisms. The performance results obtained for several standard instances from the QAP-library are very encouraging in terms of :

- Adaptability and fault-tolerance: the overhead introduced by scheduling ans checkpointing operations is very low, and the algorithm reacts very quickly to changes of the machines load.
- Efficiency and robustness: the parallel algorithm has always succeeded in finding the best known solutions for small problems ($n < 50$). The best known solutions of large real-life problems "charts of grey densities" (*Tai256c, ...*) and the real-life like

problem *Tai150b* have been improved. The parallel algorithm often produces best known or close to best known solutions for large random problems.

Our platform has been used to provide adaptive parallelism to other biologically inspired metaheuristics such as genetic algorithms and ant colonies.

References

[CL85] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3:63–75, Feb 1985.

[GK91] D. Gelernter and D. L. Kaminsky. Supercomputing out of recycled garbage: Preliminary experience with Piranha. In *6th ACM International Conference on Supercomputing*, Jul 1991.

[GL92] F. Glover and M. Laguna. Tabu search. In C. R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, pages 70–150. Blackwell Scientific Publications, 1992.

[Glo89] F. Glover. Tabu search - part I. *ORSA Journal of Computing*, 1(3):190–206, 1989.

[HTG96] Z. Hafidi, E. G. Talbi, and J-M. Geib. MARS: Adaptive scheduling of parallel applications in a multi-user heterogeneous environment. In *European School of Computer Science ESPPE'96: Parallel Programming Environments for High Performance Computing*, pages 119–122, Alpe d'Huez, France, Apr 1996.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[KTG97] D. Kebbal, E. G. Talbi, and J-M. Geib. A new approach for checkpointing parallel applications. In *Int. Conf. on Parallel and Distributed Processing Techniques and Applications PDPTA'97*, pages 1643–1651, Las-Vegas, USA, June 1997.

[MT91] T. Muntean and E. G. Talbi. A parallel genetic algorithm for process-processor mapping. In M. Durand and F. El-Dabagh, editors, *2nd Symposium on High Performance Computing*, pages 71–82, Montpellier, France, Oct 1991. Elsevier Science Pub., North-Holland.

[Nic87] D. A. Nichols. Using idle workstations in a shared computing environment. *ACM Operating System Review*, 21(5):5–12, Nov 1987.

[PL95] J. Pruyne and M. Livny. Parallel processing on dynamic resources with CARMI. In *Proc. of the Workshop on Job Scheduling for Parallel Processing IPPS'95, LNCS No.949*, pages 259–278. Springer Verlag, Apr 1995.

[Pla93] J.S. Plank. *Efficient Checkpointing on MIMD Architectures.* PhD thesis, Faculty of Princeton University, 1993.

[PRW94] P. M. Pardalos, F. Rendl, and H. Wolkowicz. The quadratic assignment problem: A survey and recent developments. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 16:1–42, 1994.

[Ree93] C. R. Reeves. *Modern heuristic techniques for combinatorial problems.* Black Scientific Publications, 1993.

[Tai95] E. Taillard. Comparison of iterative searches for the quadratic assignment problem. *Location Science*, 3:87–103, 1995.

[TL89] M. M. Theimer and K. A. Lantz. Finding Idle Machines in a Workstation-Based Distributed System. *IEEE Transactions on Software Engineering*, 15(11):1444–1458, November 1989.

This article was processed using the LaTeX macro package with LLNCS style