ParaDisEO-Based Design of Parallel and Distributed Evolutionary Algorithms

S. Cahon, N. Melab, E.-G. Talbi, Marc Schoenauer

ParaDisEO-Based Design of Parallel and Distributed Evolutionary Algorithms

S. Cahon[1], N. Melab[1], E.-G. Talbi[1], and M. Schoenauer[2]

[1] Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
59655 - Villeneuve d'Ascq Cedex
[2] Projet Fractales – INRIA Rocquencourt – 78153 Le Chesnay Cedex
{cahon, melab, talbi}@lifl.fr, marc.schoenauer@inria.fr

Abstract. ParaDisEO is a framework dedicated to the design of parallel and distributed metaheuristics including local search methods and evolutionary algorithms. This paper focuses on the latter aspect. We present the three parallel and distributed models implemented in ParaDisEO and show how these can be exploited in a user-friendly, flexible and transparent way. These models can be deployed on distributed memory machines as well as on shared memory multi-processors, taking advantage of the shared memory in the latter case. In addition, we illustrate the instantiation of the models through two applications demonstrating the efficiency and robustness of the framework.

1 Introduction

There are (at least) two broad categories of optimization problems that requires heavy computational resources: large combinatorial problems, and complex numerical engineering problems.

Large combinatorial problems are continuously evolving in terms of requirements, constraints, etc. Therefore, one needs flexible and adaptable algorithms to solve them. Furthermore, these problems are often NP-hard, characterized by a complex landscape and large instances with many decision variables.

Combinatorial optimization algorithms fall into two categories: exact methods and metaheuristics. Exact methods allow to find optimal solutions but they are often inefficient and unpractical. On the opposite, the metaheuristics aim at finding efficiently near-optimal solutions. Metaheuristics include Evolutionary Algorithms (EAs), local search methods, and the like.

Similarly, the advances of both modelization techniques and numerical simulation algorithms have increased the demand for numerical optimization algorithms. And the practitioner is faced with the following dilemma: either use some deterministic method to precisely optimize a simplified model, or use a stochastic method to approximately optimize a complex model. Evolutionary Algorithms are a good choice in the latter situation.

Unfortunately, EAs applied to real-world problems, either large combinatorial problem or complex numerical optimization applications, are known to

P. Liardet et al. (Eds.): EA 2003, LNCS 2936, pp. 216–228, 2004.

be time consuming. On the other hand, the proliferation of networks/clusters of workstations and parallel machines offers more and more facilities to the users: EAs are also known to allow efficient Parallelization/distribution and multi-threading, achieving high performance and robustness in reasonable time ... at the cost of a sometimes painful apprenticeship of parallelization techniques.

Many implementations of EAs have been proposed and some of them are available on the Web. Code developers are often tempted to reuse them to save time. However, understanding and reusing a third party code is generally a long, tedious and error prone task. Indeed, one needs to examine the internal working of the code and make several modifications to adapt it to a new problem. A better reuse approach consists in using a framework, such as ParaDisEO [4] or MALLBA [1], dedicated to the design and development of EAs. Good frameworks should be well documented and tested. But their success strongly depends on their ability to allow flexible and adaptable design. For instance, adaptation should simply require to parametrize some existing EAs and at least to write the fitness function. The same is true for the parallelization/distribution of EAs: In order to facilitate this step for those who are unfamiliar with parallel mechanisms, the frameworks must integrate the up-to-date parallelization techniques and allow their exploitation and deployment in a transparent manner.

This paper focuses on the ParaDisEO framework [1], a flexible approach for the transparent design and deployment of parallel and distributed evolutionary algorithms. ParaDisEO is basically an extension of Evolving Objects (EO) [9], an open source framework based on C++ templates allowing a flexible design of EAs. The extensions include local search methods (descent search, simulated annealing and tabu search for combinatorial optimization, gradient-based search for numerical optimization), hybridization mechanisms (coupling local search and global evolutionary search) and parallel/distributed models. This paper will concentrate on the latter topic: the different models supported by ParaDisEO will be presented, together with their flexible and transparent parameterization. The user is relieved from the burden to explicitly manage the communication and concurrency. Furthermore, the models can be efficiently deployed both on shared memory multi-processors and on distributed networks.

The reader is referred to [2] for a state of the art about parallel and distributed evolutionary algorithms, their design, many techniques of hybridization, and a full taxonomy of applications done until now. In this paper, the use of ParaDisEO will be illustrated on two test cases: the spectroscopic data mining [10] and the network design optimization [11]. The results show that they allow an efficient and robust deployment.

[1] This work is a part of a current French joint grid computing project ACI-GRID DOC-G (Challenges in Combinatorial Optimization on Grids)

The paper is organized as follows: Section 2 highlights the major requirements for the design of a useful framework of EAs and rapidly surveys existing frameworks freely available on the Web. Section 3 presents the ParaDisEO framework and details the major parallel/distributed supported models. In Section 4, we identify the main issues regarding the implementation and deployment of the models on shared memory multi-processors and parallel/distributed machines. Section 5 is dedicated to experimentation and evaluation of the models through the two applications quoted above. In Section 6, we conclude with some perspectives of this work.

2 Design Requirements and Frameworks

Using a framework can be successful only if some important user criteria are satisfied. The major of them regarding the previous problem statement are the following:

- **Ease of use**: the framework has to be relatively user-friendly. However, this obviously depends on the skill level of the programmer.
- **Flexibility/adaptability**: the integrated evolutionary algorithms can be adapted to a large variety of problems by just parameterizing or specializing them.
- **Openness**: the platform must allow the design and integration of new algorithms by reusing (parts of) existing ones.
- **Portability**: in order to satisfy a large number of users the framework must support different material architectures and their associated operating systems.
- **Performance and robustness**: as the optimization applications are often time-consuming the performance issue is crucial. Moreover, the execution of the algorithms must be robust to guarantee the reliability of the results.

Many frameworks have been realized in the combinatorial/numerical optimization area tempting to meet these design requirements. However, very often only some of them are deeply developed. We present below a non-exhaustive list of existant open source frameworks. They are in some way relatively outstanding to the best of our knowledge.

- **The MALLBA Project**[1]: The library is a set of common *software skeletons* including both metaheuristics and exact methods. Skeletons are generic classes allowing ease of use in some way and flexible design. Robustness is achieved by strong and weak hybridization mechanisms. The performance issue is addressed by providing parallel models deployable on LAN/WAN environments. Communications are based on *NetStream*, a flexible and simple OOP message passing service, upon the Message Passing Interface. Portability is ensured by the utilization of the C++ language and standards such as MPI.

- **The DREAM Project**[3]: The software infrastructure is devoted to support infohabitants evolving in an open and scalable way. It considers a virtual pool of distributed computing resources, where the different steps of an E.A. are automatically processed. It is coupled with the GUIDE, EASEA, and JEO tools. The integrated architecture allows the user to choose his/her specification interface according to his/her skill level. The main focus of DREAM is the ease-of-use and flexibility. Portability is enabled by the use of the Java language. It is drastically limited regarding the performance and robustness requirements.
- **ECJ** (`http://www.cs.umd.edu/projects/plus/ec/ecj/`): EJC is a rich Java-based portable evolutionary computation and genetic programming library. It is highly flexible as nearly all the classes are dynamically determined at runtime by a user-provided parameter file. In addition, all structures are arranged to be easily modifiable. Furthermore, parallelism and multi-threading allow efficient executions. Many other features are present including multi-objective optimization, checkpointing possibilities, etc. However, local search methods and hybridization mechanisms are not addressed.
- **JDEAL** (`http://laseeb.ist.utl.pt/sw/jdeal/`): JDEAL is another portable Java framework for E.As. Both local and parallel/distributed models are allowed to deal with the performance requirement. Other features include the paradigm freedom, facilities for the reuse and extension of existing code, and the being of both a documentation and a tutorial. Therefore, the ease-of-use and flexibility can be in some way achieved. However, as JDEAL local search methods and hybridization mechanisms are not addressed.
- **GALOPPS**[8]: The GALOPPS system is only dedicated to the design of serial or parallel genetic algorithms. A PVM extension allows the deployment of some cooperative island G.A. Moreover, some techniques of checkpointing/restarting are implemented so that a trace could be stored during the execution path. It is very limited regarding the quoted design requirements.

3 The ParaDisEO Framework

The design factors identified in the previous section are our main objectives in the design of EO/ParaDisEO. In order to meet them the following choices have been made:

- **Object-Oriented technology**: the evolutionary algorithms that are provided are software skeletons. They are implemented as templates allowing to factor out the common behaviors of evolutionary algorithms in generic classes, to be instantiated by the user. The object-oriented technology is important to meet the three first requirements.
- **Transparent parallelism and distribution**: parallelism and distribution are two important ways to achieve high performance execution. In ParaDisEO, parallelism and distribution are implemented so that the user can for instance add parallelism to his/her sequential algorithm very easily and transparently.

– **Hybridization**: the hybridization paradigm allows one to obtain robust and better solutions. Here again, the OO technology facilitates its design and implementation.

The "EO" part of ParaDisEO means Evolving Objects. Before ParaDisEO feature are presented let us give a brief description of EO.

3.1 EO

EO is a C++ open source framework downloadable from `http://eodev.sourceforge.net`. The framework is originally the result of an European joint work [9]. EO includes a paradigm-free Evolutionary Computation library (EOlib) dedicated to the flexible design of EAs through evolving objects superseding the most common dialects (Genetic Algorithms, Evolution Strategies, Evolutionary Programming and Genetic Programming). Flexibility is enabled through the use of the object-oriented technology. Templates are used to model the EA features: coding structures, transformation operators, stopping criteria, etc. These templates can be instantiated by the user according to his/her problem-dependent parameters. The OO mechanisms such as inheritance, polymorphism, ... are powerful ways to design new algorithms or evolve existing ones. Furthermore, EO integrates several services making it easier to use including visualization facilities, on-line definition of parameters, application checkpointing, etc.

Moreover, a Graphical User Interface has been developed for EO. This GUI, called *GUIDE* (*Graphic User Interface for DREAM Experiments*), and based on the specification language for EAs *EASEA* [5]. *GUIDE* was developed during the European project *DREAM* [3], and provides a friendly graphical interface allowing the user to specify his/her own EA. One panel is dedicated to the specification of the problem-specific parts of the algorithm (the genome, and the corresponding operators). Another panel is devoted to the specification of the evolution engine (the implementation of artificial Darwinism). And a third panel is entirely devoted to specifying the distribution mechanism: as it was initially conceived for the DREAM framework, this panel only knows about the island model (see section 1). But within this model, the distribution panel of GUIDE allows the user to specify a topology, a number of migrants, selection and replacement mechanisms for the migrants, ... i.e. all the parameters needed for the ParaDisEO island model.

3.2 ParaDisEO

In its original version, EO does not enable the design of local search methods such as descent search or hill-climbing, simulated annealing, tabu search or gradient-based search. Moreover, it does not offer any facility for either parallelism and distribution, or hybridization. Sticking out those limitations was the main objective in the design and development of ParaDisEO. In the following, we will focus only on parallel/distributed mechanisms.

Note, however, as far as hybridization is concerned, that the two levels (High and Low) and the two modes (Relay and Teamwork) of hybridization identified in [13] are available in ParaDisEO: High level hybridization consists of making self-contained EAs cooperate. Their internal work is not considered, thus their coupling is weak. At the contrary, the Low level addresses the functional composition of an EA. For example, a transformation operator can be replaced by a local search method. On the other hand, the Relay mode means that a set of EAs are applied in a pipelined way. The Teamwork mode allows a concurrent cooperation between EAs.

Going back to parallelism, three major parallel models are implemented in the platform: the island asynchronous cooperative model, the parallel/distributed population evaluation and the distributed evaluation of a single solution. These models will now be detailed in turn (again, detailed descriptions ot those models as well as a comprehensive survey of existing models is given in [1]).

3.3 Model 1 – Island Asynchronous Cooperative Model

Different EAs are simultaneously deployed and then cooperate to compute better and robust solutions (fig. 1). They exchange in an asynchronous way genetic stuff to diversify the search. The objective is to allow to delay the global convergence, especially when the EAs are heterogeneous regarding the variation operators. The migration of individuals follows a policy defined by few parameters: the migration decision criterion, the exchange topology, the number of emigrants, the emigrants selection policy, and the replacement/integration policy.

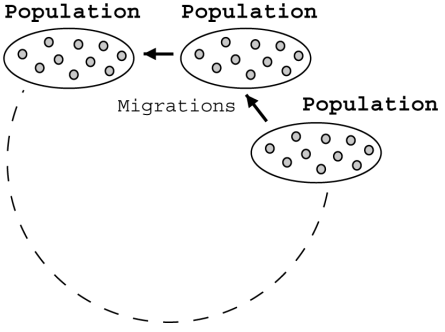

Fig. 1. The cooperative island evolutionary algorithm

– **Migration decision criterion**: Migration can be decided according to either *blind* or *intelligent* criterion. The blind can be either periodic or probabilistic. Periodic decision making consists in periodically performing the migration by each EA. The user has to set the value of the migration frequency. Probabilistic migration is performed at each generation with a user-defined probability. On the other hand, the criteria for the intelligent approaches are here driven by some quality improvement criterion. The user has to specify

a threshold for the improvement, and migration will take place every time the improvement of the best fitness in the population during two successive generations will be below that threshold.
As in the rest of EO, all the quoted migration criteria are predefined in ParaDisEO as class methods. Hence the user can easily either use them directly, or combine them as building blocks to design his/her own specific criterion.

- **Exchange topology**: The topology specifies for each island its neighbors with respect to migration, i.e. the other islands to which it will send its emigrants, and the ones from which it will receive immigrants (both lists can be different. Two well-known topologies are predefined: ring and hypercube. And of course, the user has the possibility to specify his/her own topology according to problem-dependent or machine-dependent features.
- **Number of emigrants**: This emigrants parameter is defined either as a percentage of the population size, or as a fixed number of individuals.
- **Emigrants selection policy**: The selection policy indicates in a deterministic or stochastic way how to select emigrant individuals from the population of a source EA. Different selection policies are already defined in EO, and used for instance to select the parents undergoing variation operators (e.g. roulette wheel, ranking, stochastic or deterministic tournaments, uniform sampling, ...). The flexibility of the library makes it easy to reuse these policies for emigrants selection.
- **Replacement/integration policy**: Symmetrically, the replacement policy defines in a deterministic or stochastic way how to integrate the immigrant individuals in the population. Again, the replacement strategies defined in EO are used here, including tournaments, EP-like stochastic replacement, elitist and pure random replacements.

3.4 Model 2 – Parallel/Distributed Population Evaluation

The parallelization evaluation step of an EA is required as it is in general the most time-consuming. The parallel evaluation follows the centralized model (fig. 2). The farmer applies the following operations: selection, transformation and replacement as they require a global management of the population. At each generation, it distributes the set of new solutions between different workers. These evaluate and return back the solutions and their quality values. An efficient execution is often obtained particularly when the evaluation cost of each solution is costly.

The model can be deployed either on a shared-memory multi-processor or a distributed memory machine. In the first case, the user has to indicate the number of parallel evaluators (workers). The different workers pick up individuals from a shared list-based population. In the distributed model the user must give the size work unit size (grain). The parameter represents the number of individuals to be sent to a worker at a time. At each generation an evaluator may to process one or several work units.

The model described above is the synchronous master/slave model [2]. The two main advantages of the asynchronous model over the synchronous model are first the fault tolerance of the asynchronous model, an second the robustness in case the fitness evaluation can take very different computation times (e.g. for nonlinear numerical optimization). Whereas some time-out detection can be used to address the former issue, the latter one can be partially overcome if the grain is set to very small values, as individuals will be sent out for evaluations upon request of the slaves.

The asynchronous evaluation process will be soon released in ParaDisEO, so that both breeding and evaluation step could be done concurrently. The farmer manages the evolution engine and two queues of individuals, each one with a given fixed size: individuals to be evaluated, and awaiting solutions being evaluated. The first ones wait for a free evaluating node. When the queue is full the process blocks. The second ones are assimilated into the population as soon as possible.

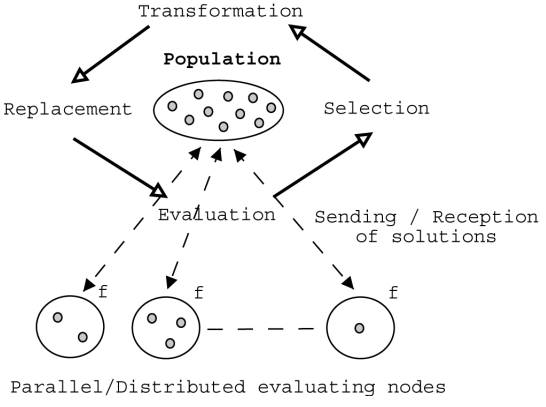

Fig. 2. The parallel/distributed evaluation of a population

3.5 Model 3 – Distributed Evaluation of a Single Solution

The quality of each solution is evaluated in a parallel centralized way. That model is particularly interesting when the evaluation function can be itself parallelized as CPU time-consuming and/or IO intensive. In that case, the function can be viewed as an aggregation of a certain number of partial functions. For instance, if the problem to be solved is multi-objective each partial function evaluates one objective. The partial functions could also be identical if for example the problem to deal with is a data mining one. The evaluation is thus data parallel and the accesses to data base are performed in parallel. Furthermore, a reduction operation is performed on the results returned by the partial functions. As a summary, for this model the user has to indicate a set of partial functions and an aggregation operator of these (fig. 3).

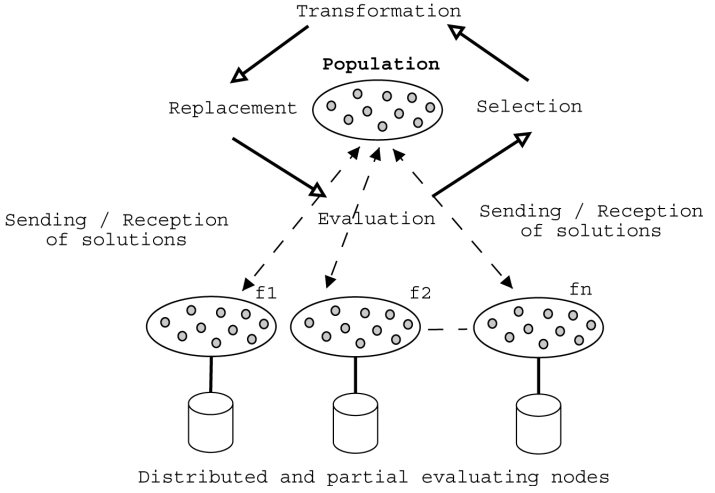

Fig. 3. The distributed evaluation of a single solution

4 Implementation and Transparent Deployment

One has to note here that the implementation and deployment of the presented parallel models is transparent for the user. Indeed, the user does not need to manage the communications and threads-based concurrency. We present below some implementation and deployment issues with regards to the underlying parallel hardware support: shared memory multi-processors and distributed memory machines.

– The migration decision maker represents the kernel of the island asynchronous cooperative model. It uses the user-defined parameters to take migration decisions, perform migrations according to the selection and replacement policies. The implemented migration algorithm is the following:

```
1. If there are any waiting immigrants integrate them into the
local population, using the replacement operator.
2. Process the received immigration requests if any. For each
request, choose some individuals according to the selection
operator, then send them as requested.
3. Check if according to the migration decision maker a
migration operation is necessary. If so participate to the
migration process.
```

The deployment of the algorithm requires to choose the technical means to be used to perform the migrations of the individuals according to the hardware support. On shared memory multi-processors the implementation is multi-threaded and the migrations are memory copy-driven. Threads access in a concurrent way the populations of the different EAs stored on the shared

memory. On distributed memory machines the mechanism is communication-driven. Migrations are performed by explicit message exchange.
- In the parallel/distributed population evaluation model the population is always stored on a single processor. However, the assignment of the individuals to the processors for evaluation depends on the hardware architecture. On shared memory multi-processors threads pick up (copy) individuals from the global list in a concurrent way. On distributed memory processors the population is divided into sub-populations according to the user-defined granularity parameter. These sub-populations are then assigned to the processors using communication messages. When an evaluator returns its results to the farmer, that evaluator will immediately be sent another sub-population – if there still are individuals to be evaluated. The performance of the model hence strongly depends on the grain size. Very fine-grained sub-populations induce a huge amount of communication, leading to a small acceleration over sequential programs. Conversely, very coarse-grained sub-populations imply a lower degree of parallelism, and thus again leads to a smaller acceleration. Some compromise has to be found between both extreme cases.
- The distributed evaluation of a single solution model is well-suited for distributed memory machines. The data of the evaluation function have to be distributed among different processors. A distribution strategy has to be defined. One has to note here that the problem is application-dependent. Up to date, in ParaDisEO it is supposed that the data is already distributed. The computation i.e. the user-specified partial functions are sent in explicit messages to the workers which apply them on their local data.

Finally, to meet the portability objective the implementation of the parallel/distributed and hybridization mechanisms is based on different standards. Indeed, the *de facto* communication libraries PVM and MPI allow a portable deployment on networks of heterogeneous workstations. Furthermore, the Posix Threads multi-programming library contributes to enable a portable execution on shared memory multi-processors.

5 Applications

ParaDisEO has been experimented with several applications including academic ones such as TSP and graph coloring, and industrial ones. In this section, we will address two real-world problems: the NIR spectroscopic data mining and the mobile telecommunication network design.

- **Near InfraRed spectroscopic data mining**: the problem consists in discovering, from a set of data samples, a predictive mathematical model for the concentration of sugar in beet. Each data sample contains a measure of the concentration of sugar in one beet sample, and a set of its absorbances to 1024 NIR wavelengths. According to the Beer Lambert law the problem is linear. The Partial Least Square (PLS) statistical method is known to be well-suited to deal with such problem. However, the number of wavelengths

in the resulting predictive model is high, decreasing the understandability of the results. Well, but the analysis of the data allow to highlight that many absorbances are correlated between them (redundancy problem) and less correlated with the concentration (irrelevance problem). In order to withdraw both irrelevant and redundant wavelengths a feature selection has to be performed. As feature selection is an NP-hard problem we use a genetic algorithm (GA) to solve it. The PLS method is used as the fitness function of the GA. More exactly, the prediction error returned by the PLS method is the fitness value of the individuals (selections of wavelengths).
The hybridization GA-PLS is CPU time consuming. The fitness evaluation is particularly costly as the PLS method handles large matrices. Parallelism is required to get results at short notice. The models 1 and 2 have been exploited in a straightforward way. The experimentation has been done on an IBM cluster of SMP. The model is composed of four distributed islands, each of them deployed on a shared memory multi-processor (10 processors) dedicated to evaluate selections of wavelengths. The database is replicated on each cluster node and shared between the processes of the same node. Thus, four populations of 200 individuals are evolving and migrations occur when no improvement has been noticed during the last ten iterations. Both selection and replacement operators are performed by using the stochastic tournament operator with a rate pressure fixed to 0.8.
The obtained results are encouraging since over 90% of wavelengths proved to be useless or harmful. Therefore, the processing time required to compute the concentration of sugar in a new sample would be divided by ten. In addition, the prediction accuracy is increased by 37% compared to the PLS without feature selection.

– **Mobile telecommunications network design**: One of the major problems in the engineering of mobile telecommunication networks is the design of the network. It consists in positioning base stations on potential sites in order to fulfill some objectives and constraints [11]. More exactly, the multi-objective problem is to find a set of sites for antennas from a set of pre-defined candidates sites, to determine the type and the number of antennas, and their configuration parameters (tilt, azimuth, power, ...).
It is a hard practical problem with many decision variables. Tackling it in a reasonable time requires to use of parallel heuristics. A parallel distributed GA is investigated to solve the problem. The three parallel models have been exploited. The islands in the model 1 evolve with heterogeneous variation operators. Migrations are performed according to a ring topology. Emigrants are randomly selected with a fixed percentage of 5% of the population. In the model 2, the offspring are distributed with a chunk fixed to one. Furthermore, the fitness evaluation of a network is not trivial as it handles large precalculated databases. Each node contributing in the deployment of the model 3 is assigned a subpart of the geographical map.

Experimentations on the hardware platform quoted above have shown that the parallel models are efficient. Indeed, they allow to obtain robust and high

quality solutions. In addition, as Figure 4 illustrates it they permit a quasi-linear parallel execution speed-up for the data mining application. For the network design application, the speed-up is quasi-linear on over 16 processors. However, the scalability is not achieved as the granularity i.e. (a subpart of the geographical map) of parallelism decreases while increasing the number of processors. The processing time required by the aggregation of the partial results widely exceeds the computation load performed by each participant processor.

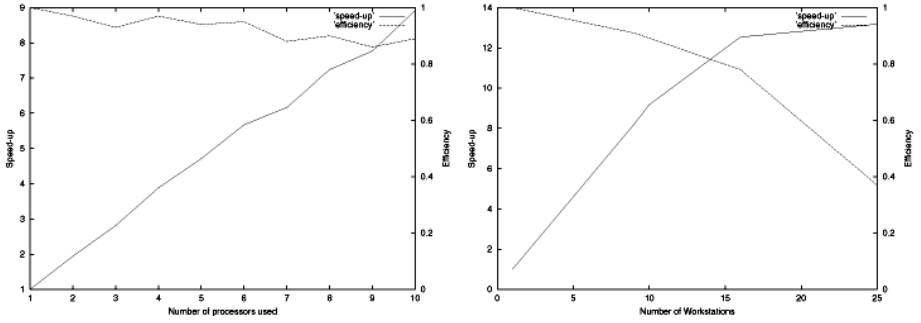

Fig. 4. Here are the measures of speed-up and efficiency obtained for the two described applications according to the respective number of processors/workstations used

6 Conclusion and Future Work

We have presented the ParaDisEO approach for the flexible and transparent design of parallel and distributed evolutionary algorithms. Flexibility and transparency are achieved through parameterizable templates. Three major complementary models have been presented. Their implementation is based on two communication libraries and the Posix threads multi-programming library. These standards allow a portable execution of the models and their deployment on as well shared memory multi-processors as distributed memory machines.

As many problems are multi-objective in practice we are currently investigating the ParaDisEO-based design of multi-objective EAs. As the framework is flexible the design is straightforward. Moreover, the models 1 and 2 do not require any design change. The model 3 could be exploited by evaluating in parallel the different objectives of the fitness function. In addition, we are addressing some issues related to the deployment of the three models in adaptive environments. As workstations are continuously leaving and joining the network, ParaDisEO must provide dynamic task stopping and restarting mechanisms. Furthermore, it has to integrate checkpointing in order to deal with the fault tolerance issue. To do that we are investigating a new version of ParaDisEO on top of the Condor-PVM [12] resource manager. Finally, to allow a deployment on grid-based environments we focus on grid-oriented algorithmics and implementation

issues. To meet that objective we are planning to use Condor-G [7]. The latter grid execution support is a coupling of Condor and Globus [6], a toolkit that allows a large scale multi-domain execution of parallel and distributed applications.

References

1. E. Alba and the MALLBA Group. MALLBA: A library of skeletons for combinatorial optimisation. In R.F.B. Monien, editor, *Proceedings of the Euro-Par*, volume 2400 of LNCS, pages 927–932. Springer-Verlag, 2002.
2. E. Alba and M. Tomassini. Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002.
3. M.G. Arenas, P. Collet, A.E. Eiben, M. Jelasity, J.J. Merelo, B. Paechter, M. Preuß, and M. Schoenauer. A framework for distributed evolutionary algorithms. In *Proceedings of PPSN VII*, september 2002.
4. S. Cahon, E-G. Talbi, and N. Melab. ParadisEO: A Framework for Parallel and Distributed Biologically Inspired Heuristics. In *Nature Inspired Distributed Computing Workshop, in IEEE IPDPS2003 (Int. Parallel and Distributed Processing Symposium)*, page 201. Nice, France, IEEE Press, April 2003.
5. P. Collet, E. Lutton, M. Schoenauer, and J. Louchet. Take it easea. In PPSN2000editors, editor, *PPSN2000*, LNCS 1917, pages 891–901, 2000. http://sourceforge.net/projects/easea.
6. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *ntl J. Supercomputer Applications*, 11(2):115–128, 1997.
7. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–246, 2002.
8. E. Goodman. An introduction to galopps – the "genetic algorithm optimized for portability and parallelism" system. Technical report, Intelligent Systems Laboratory and Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, November 1994.
9. M. Keijzer, J.J. Merelo, G. Romero, and M. Schoenauer. Evolving Objects: A General Purpose Evolutionary Computation Library. *Proc. of the* 5^{th} *Intl. Conf. on Artificial Evolution (EA'01), Le Creusot, France*, October 2001.
10. N. Melab, S. Cahon, E-G. Talbi, and L. Duponchel. Parallel genetic algorithm based wrapper feature selection for spectroscopic data mining. In *BioSP3 Workshop on Biologically Inspired Solutions to Parallel Processing Problems, in IEEE IPDPS2002 (Int. Parallel and Distributed Processing Symposium)*, page 201. Fort-Lauderdale, USA, IEEE Press, April 2002.
11. H. Meunier, El-Ghazali Talbi, and P. Reininger. A Multiobjective Genetic Algorithm for Radio Network Optimization. In *Congress on Evolutionary Computation*, volume 1, pages 317–324. IEEE Service Center, July 2000.
12. J. Pruyne and M. Livny. Interfacing condor and pvm to harness the cycles of workstation clusters. In *Future Generations of Computer Systems*. P. Sloot, to appear.
13. E-G. Talbi. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics, Kluwer Academic Publishers*, Vol.8:541–564, 2002.