A Study of the Parallelization of the Multi-Objective Metaheuristic MOEA/D

Antonio J. Nebro and Juan J. Durillo

Department of Computer Science, University of Málaga (Spain)
{antonio,durillo}@lcc.uma.es

Abstract. MOEA/D is a multi-objective metaheuristic which has shown a remarkable performance when solving hard optimization problems. In this paper, we propose a thread-based parallel version of MOEA/D designed to be executed on modern multi-core processors. Our interest is to study the potential benefits of the parallel approach in terms of speed-ups and the quality of the obtained Pareto front approximations when solving a benchmark composed of nine problems. The obtained results on two different multi-core based machines indicate that notable time reductions can be achieved. We have also found out that, with a few exceptions, there are not significant differences in terms of solution quality among the sequential MOEA/D and the parallel versions of it when using up to eight threads.

Key words: Multi-Objective Optimization, Metaheuristics, Parallelism, Multi-core processors

1 Introduction

The formulation of many optimization problems faced in engineering and other disciplines of knowledge involves more than one objective function. Generally speaking, multi-objective optimization is not restricted to find a single solution to a given multi-objective optimization problem (MOP), but rather it searches for a set of solutions called nondominated solutions. Each one in this set is said to be a Pareto optimum, and when they are plotted in the objective space they are collectively known as the Pareto front. As in single-objective optimization, MOPs may present features such as epistasis, deceptiveness, or NP-hard complexity [15], making them difficult to solve.

In recent years, metaheuristics [2, 8] have become popular techniques for solving MOPs; in particular, evolutionary algorithms (EAs) have been widely used, giving rise to a broad variety of algorithms, such as NSGA-II [5], SPEA2 [18], PAES [10], and many others [3, 4]. Most of these techniques are based on the concept of Pareto dominance and the use of some kind of density estimator. If we consider NSGA-II and SPEA2, two very popular algorithms in the field, the first one uses solution ranking and an estimator based on the so-called crowding distance, while the second one applies the concept of strength and a density estimator based on the distance to the k-nearest neighborhood. These ideas, particularly those applied in NSGA-II, have been used in a number of multi-objective metaheuristics, e.g., GDE3 [11], AbYSS [13], MOCell [14], etc.

In this context, MOEA/D [16] is a recent proposal differing from these popular algorithms. Instead of Pareto dominance, MOEA/D is based on decomposition, in such a way that a number of single-objective subproblems are optimized simultaneously, each of them is an aggregation of all the objectives of the MOP. Diversity is promoted by establishing neighborhood relationships among the subproblems, which are defined based on the distances among their aggregation coefficient vectors. MOEA/D and, particularly its variant using differential evolution [12], have shown a remarkable performance when solving MOPs with complicated Pareto sets [12, 17].

In this paper, we study the benefits of using modern multi-core CPUs in a parallel version of MOEA/D. Nowadays, these kinds of processors are becoming very common, bringing parallel power to PCs and laptops. As a consequence, users can speed-up their applications if they are properly parallelized. Our interest here is to design a parallel MOEA/D algorithm to determine if we can profit from running it on multi-core processors. The benefits can be related mainly to reduce the computing time; however, since the behavior of the algorithm is modified somehow in the parallel version, it is interesting to analyze whether or not the parallel algorithm improves the search capabilities of the sequential MOEA/D.

The contributions of the paper can be summarized in the following:

- A thread-based parallel version of MOEA/D has been designed and implemented.
- The algorithm has been run on two computers having modern multi-core processors, having both machines a total of eight concurrent execution flows.
- Experiments have been conducted to determine the advantages of using the parallel approach on the target machines.

The rest of the paper is organized as follows. Section 2 describes MOEA/D and pMOEA/D, the parallel version of the former. Next section gives the details of the experimentation methodology applied in our study. Section 4 and 5 include the analysis of the results of the experiments carried out. Finally, we conclude the paper and give some lines of future work in Section 6.

2 Description of MOEA/D and pMOEA/D

As commented in the previous section, MOEA/D uses a decomposition approach to transform the original MOP into a set of single-objective optimization subproblems. A pseudocode of the technique is included in Algorithm 1, which describes the variant MOEA/D-DE presented in [12].

MOEA/D requires the following components:

- A population P of N points (individuals) $x^1, \ldots, x^N$, where x^i is the current solution to the ith subproblem.

Algorithm 1 Pseudocode of MOEA/D.

```
1: // Initialization
2: $\lambda$ ← initializeWeightVectors()
3: $B$ ← initializeNeighborhood()
4: $P$ ← initializePopulation()
5: $z$ ← initializeIdealPoint()
6:
7: // Main loop
8: while not terminationCondition() do
9:    for (i = 0 ; i < populationSize; i++) do
10:       if (rand(0, 1) < $\delta$) then
11:          MP ← $B(i)$ // MP = mating pool
12:       else
13:          MP ← $P$
14:       end if
15:       parents ← selection(MP, i) // DE selection
16:       child ← recombination(parents) // DE operator
17:       child ← mutation(child, $P_m$) // Polynomial mutation
18:       evaluateFitness(child)
19:       idealPointUpdate(child, $z$)
20:       solutionsUpdate(child, MP, $n_r$)
21:    end for
22: end while
23: return $P$
```

– A set of $\lambda^1, \ldots, \lambda^N$ weight vectors. For a m-objective optimization problem, $\lambda = (\lambda_1, \ldots, \lambda_m)$ is a weight vector, i.e., $\lambda_i \geq 0$ for all $i = 1, \ldots, m$ and $\sum_{i=1}^{m} \lambda_i = 1$.
– A vector $z^* = (z_1^*, \ldots, z_m^*)$ used as reference point.
– For each $i = 1, ..., N$, a neighbourhood $B(i) = \{i_1, \ldots, i_T\}$, where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest weight vectors to λ^i.

These components are initialized in Lines 2-5 in Algorithm 1; after that, the main loop of the algorithm executes until a termination condition is met (Line 8). In each iteration, for each individual i (Line 9), the mating pool is selected between $B(i)$ and P according to a probability δ (lines 10-14); then, two parents are selected (line 15) which, along with i, are used to obtain a new individual by applying the recombination and mutation operators (Lines 16-17). The new individual is evaluated (Line 18) and the ideal point z^* is updated (Line 19). The last step (Line 20) consists in updating the population with the obtained individual according to a parameter n_r, which is used as a bound to limit the number of replaced solutions. For further details, please refer to [12].

The recombination DE operator used in line 16 is detailed next. Given an individual $r_1 = i$ and two randomly selected parents r_2 and r_3 (line 15), a new child u is generated as follows:

$$u_k = \begin{cases} x_k^{r_1} + F \times (x_k^{r_2} - x_k^{r_3}) & \text{with probability } CR, \\ x_k^{r_1}, & \text{with probability } 1 - CR \end{cases} \tag{1}$$

where $k = 1, \ldots, n$, being n the number of decision variables of the problem, and F and CR are two control parameters.

Algorithm 2 Pseudocode of pMOEA/D.

```
 1: // Initialization
 2: $\lambda \leftarrow$ initializeWeightVectors()
 3: $B \leftarrow$ initializeNeighborhood()
 4: $P \leftarrow$ initializePopulation()
 5: $z \leftarrow$ initializeIdealPoint()
 6:
 7: // Main (parallel) loop
 8: for (j = 0 ; j<numberOfThreads; j++) in parallel do
 9:     first, last $\leftarrow$ calculateIndices(j, numberOfThreads, populationSize)
10:     while not terminationCondition() do
11:         for (i = first ; i < last; i++) do
12:             if (rand(0, 1) $< \delta$) then
13:                 MP $\leftarrow B(i)$ // MP = mating pool
14:             else
15:                 MP $\leftarrow P$
16:             end if
17:             syncronized
18:                 parents $\leftarrow$ selection(MP, i) // DE selection
19:                 child $\leftarrow$ recombination(parents) // DE operator
20:             end syncronized
21:             child $\leftarrow$ mutation(child, $P_m$) // Polynomial mutation
22:             evaluateFitness(child)
23:             syncronized idealPointUpdate(child, $z$)
24:             syncronized solutionsUpdate(child, MP, $n_r$)
25:         end for
26:     end while
27: end for
28: return P
```

The idea behind our parallel version of MOEA/D is to run it on computers equipped with multi-core processors. The approach we have taken is to distribute the loop starting in Line 9 among a number of concurrent threads, according to an iterative parallelism paradigm [1]. This way, each thread could work on a portion of the population in parallel. As multi-threading is based on a parallel shared memory paradigm, the main issue is to detect the critical sections in the code in order to avoid race conditions. The pseudocode of the resulting parallel MOEA/D, named pMOEA/D, is detailed in Algorithm 2.

pMOEA/D, after the initialization step, starts a number of concurrent threads (Line 8). The next step is to calculate the indices defining the portion of the population that will be assigned to each thread (Line 9), as depicted in Fig. 1. After that, we find the same main loop as in Algorithm 1 (Line 10) with the following differences: a) in each iteration, a portion of the population is used (Line 11); and b) some pieces of the code must be synchronized to be executed in mutual exclusion. The key point is that the population is spread out among the execution threads, but the neighborhoods overlap. This means that some elements of a thread subpopulation may be modified by other thread when this updates the population, so the sentences between Lines 17-20 are intended to ensure that the selected parents are kept unchanged while they are recombined to yield a child individual. Updating the ideal point and the population are obvious procedures to be also synchronized (Lines 23 and 24).

Once we have designed pMOEA/D, what remains it to study its behavior when executed on a multi-core processor. The interest points are to know whether

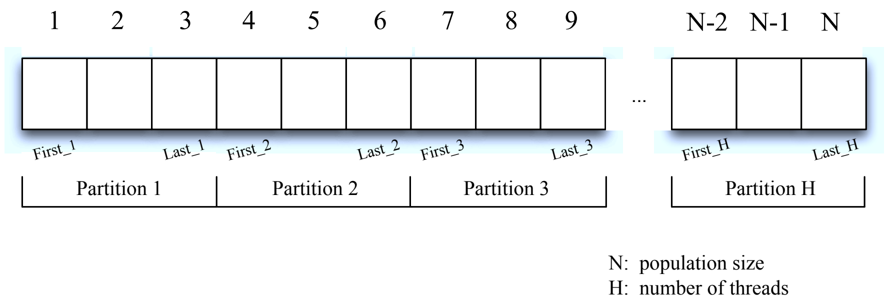

Fig. 1. Partitioning scheme used to assign portions of the population to each thread.

its search capabilities changes in respect to the sequential MOEAD or not, as well as the speed-ups that can be obtained. We explore these issues in the following sections.

3 Experimentation Methodology

We have implemented MOEA/D and pMOEA/D in Java using the jMetal framework [7]. The target computers, referred as Machine 1 and Machine 2 in the following, have the features detailed next:

– Machine 1: One Intel® Core^{TM} i7-920 processor (2.66 GHz), which features four cores and a total of eight processing threads with Intel® Hyper-Threading technology. The operating system is Linux (OpenSuse 11.1), kernel version 2.6.27.7-9 x86_64, and the JDK version is 1.6.0_15 64-Bit.
– Machine 2: Two Intel®Xeon® E5405 quad-core processors (2.00GHz). The operating system is Linux (Ubuntu 8.04.3 LTS), kernel version 2.6.24-24-generic #1 SMP; the JDK version is 1.6.0_14-b08.

First, we have carried out a set of experiments aimed at determining if pMOEA/D brings some kind of advantage against MOEA/D in terms of the quality of the produced Pareto front approximations. Our approach has been to solve a benchmark of MOPs with both algorithms. Each experiment has been repeated 100 times and the unary additive epsilon ($I^1_{\epsilon+}$) [9] quality indicator has been applied to assess the search performance of the algorithms. $I^1_{\epsilon+}$ is a convergence indicator, which is defined as follows: given an approximation set of a problem, A, the $I^1_{\epsilon+}$ indicator is the minimum factor ϵ that can be added to each point of the optimal Pareto front in such a way that the resulting front is dominated by A.

The parameter settings of both the sequential and parallel algorithms are the following:

– Population size: 600.
– Stopping condition: 150,000 function evaluations.
– Weight vectors: we have used the values provided in [17].
– Control parameters in DE: $CR = 1.0$, $F = 0.5$.
– Polynomial mutation operator: $p_m = 1/n$ (n is the number of decision variables), distribution index = 20.
– Rest of parameters: $T = 60$, $n_r = 6$, $\delta = 0.9$.

We have configured pMOEA/D with 1, 2, 4, 8, 16, and 32 threads. As benchmark problems, we have chosen the nine MOPs defined in [12], which have been named as LZ09_Fx, where $x = 1, \ldots, 9$. All of them have two objectives to optimize but LZ09_F6, which is a tri-objective MOP.

The second set of experiments are intended to know about the time reductions that can be achieved when running pMOEA/D on the target machines. As performance metrics, we have used the speed-up S_p and the efficiency E_p, being p the number of processors. They are defined by the following formulae:

$$S_p = \frac{W}{W_p} \tag{2}$$

$$E_p = \frac{S_p}{p} \tag{3}$$

where W and W_p are the execution times of the sequential and the parallel algorithm with p processors, respectively. If $S_p = p$, the speed-up is linear and the efficiency $E_p = 1$.

4 Analysis of the Search Capabilities of pMOEA/D

We start by analyzing the results obtained with Machine 1. Table 1 includes the value of the $I^1_{\epsilon+}$ indicator of the computed fronts. In this table, M$_{seq}$ refers to the sequential MOEA/D, and M$_x$ represents pMOEA/D with x threads. To ease the analysis of the results in the table, some cells have a grey colored background. Specifically, we have used two grey levels: a darker one for highlighting the best (lowest) value, and a lighter one for pointing out the second best value of the indicator. Thus, attending to the table, we observe that the best indicator values are distributed among M$_1$, M$_4$, and M$_8$. Particularly, M$_1$, pMOEA/D with one thread, seems to be the most outstanding algorithm in our comparison because it has computed the best fronts according to the $I^1_{\epsilon+}$ in three out of the nine benchmark problems, and it has obtained the second best values of the indicator in other three cases. According to the table, using pMOEA/D with more than eight threads do not give any benefit.

Now, we go one step forward and we analyze if these results are statistically confident. To come with this issue, we have applied the Wilcoxon rank sum test, a non-parametric statistical hypothesis test, which allow us to make pairwise comparisons between algorithms to know about the significance of the obtained data [6]. A confidence level of 95% (i.e., significance level of 5% or p-value under

0.05) have been used in all the cases, which means that the differences are unlikely to have occurred by chance with a probability of 95%. The obtained results are shown in Table 2. In each cell, the nine considered MOPs are represented with a symbol. Three different symbols are used: "–" indicates that there is not statistical significance between the algorithms, "▲" means that the algorithm in the row has yielded better results than the algorithm in the column with confidence, and "▽" is used when the algorithm in the column is statistically better than the algorithm in the row.

Table 2 allows us to observe at a glance that, with a few exceptions, there are not significant differences among MOEA/D and the versions of pMOEA/D with up to eight threads. Meanwhile, with 16 and 32 threads there is statistical confidence in practically all the experiments. Summarizing, the experiments carried out have shown that there is no significant differences between the quality of the obtained fronts by using up to 8 threads when using Machine 1. On the other hand, when a higher number of threads are used, the computed fronts are significantly worse in many situations.

We focus now in the results obtained when using Machine 2. Table 3 includes the values of the $I^1_{\epsilon+}$; the nomenclature used is the same as for Table 1. With this computer, the best results regarding to the $I^1_{\epsilon+}$ are distributed between M_{seq}, M_2, M_8, and M_{16}. Here, we highlight two main facts: on the one hand, the M_{seq} algorithm has obtained the best or second best values in the majority of the cases; and, on the other hand, pMOEA/D has obtained the best result in problem LZ09_F1 by using 16 threads.

Again, we apply a Wilcoxon test in order to determine whether the results are confident or not. Table 4 includes the results of the test. We observe that, as happened in Machine 1, there are not statistical differences between the results of MOEA/D and pMOEA/D up to 8 threads in most of the problems. Also with Machine 2, when using 16 and 32 threads, the number of cases where confidence has been found gets higher.

More detailed information can be obtained if we display the results using boxplots, which constitutes a useful way of depicting groups of numerical data. The boxplots of the values obtained with Machine 2 are included in Fig. 2 (the boxplots resulting from the data obtained with Machine 1 are similar). Thus, attending to the figure, we see that in practically all the problems the distribution of the $I^1_{\epsilon+}$ values are located in the same region; this points out that the results computed by all the algorithms are of similar quality in many cases.

Comparing the results obtained in both machines, they are not identical, but rather similar, allowing us to conclude that parallelizing MOEA/D with the proposed thread-based scheme rarely improves the search capabilities of the sequential algorithm; however, the differences are not statistically significant up to eight threads. This is a somehow expected result, because the parallelization scheme we have applied, in case of a fair thread scheduling, tends to a behaviour of pMOEA/D similar to the sequential algorithm. What remains to see are the computing time performance benefits of using pMOEA/D on the target multicore computers. We deal with this issue in the next section.

Table 1. Machine 1. Median and interquartile range of the $I^1_{\epsilon+}$ indicator. Cells with dark and light background indicate the best and second best values, respectively.

	M$_{seq}$	M$_1$	M$_2$	M$_4$	M$_8$	M$_{16}$	M$_{32}$
LZ09_F1	$2.31e-03_{2.9e-04}$	$2.28e-03_{3.1e-04}$	$2.32e-03_{2.8e-04}$	$2.27e-03_{3.5e-04}$	$2.29e-03_{2.9e-04}$	$2.30e-03_{3.1e-04}$	$2.42e-03_{4.0e-04}$
LZ09_F2	$7.98e-02_{2.2e-01}$	$2.18e-01_{2.4e-01}$	$2.05e-01_{2.2e-01}$	$1.99e-01_{2.1e-01}$	$6.80e-02_{2.2e-01}$	$2.29e-01_{2.5e-01}$	$2.23e-01_{2.9e-01}$
LZ09_F3	$6.18e-02_{2.2e-01}$	$5.41e-02_{2.1e-01}$	$6.21e-02_{2.2e-01}$	$5.48e-02_{8.7e-02}$	$5.53e-02_{2.2e-01}$	$5.92e-02_{2.1e-01}$	$7.86e-02_{2.3e-01}$
LZ09_F4	$5.55e-02_{2.1e-02}$	$4.98e-02_{2.0e-02}$	$5.05e-02_{1.8e-02}$	$4.99e-02_{2.4e-02}$	$4.92e-02_{1.4e-02}$	$5.79e-02_{2.0e-02}$	$6.04e-02_{2.2e-02}$
LZ09_F5	$7.78e-02_{2.7e-02}$	$7.83e-02_{2.8e-02}$	$7.86e-02_{3.1e-02}$	$7.79e-02_{1.8e-02}$	$7.63e-02_{1.9e-02}$	$8.01e-02_{3.1e-02}$	$8.17e-02_{3.6e-02}$
LZ09_F6	$1.27e-01_{2.8e-02}$	$1.21e-01_{2.7e-02}$	$1.24e-01_{2.4e-02}$	$1.19e-01_{2.1e-02}$	$1.25e-01_{3.2e-02}$	$1.22e-01_{2.6e-02}$	$1.21e-01_{3.5e-02}$
LZ09_F7	$3.25e-02_{3.4e-02}$	$3.04e-02_{2.3e-02}$	$3.03e-02_{3.9e-02}$	$2.72e-02_{2.4e-02}$	$3.18e-02_{2.4e-02}$	$3.41e-02_{3.5e-02}$	$3.34e-02_{2.2e-02}$
LZ09_F8	$1.99e-01_{8.3e-02}$	$1.85e-01_{8.5e-02}$	$2.02e-01_{9.4e-02}$	$2.09e-01_{9.1e-02}$	$2.02e-01_{8.4e-02}$	$2.22e-01_{1.2e-01}$	$2.31e-01_{1.3e-01}$
LZ09_F9	$4.14e-02_{8.8e-02}$	$3.75e-02_{7.1e-02}$	$3.75e-02_{5.1e-02}$	$3.84e-02_{6.4e-02}$	$5.71e-02_{3.6e-01}$	$6.25e-02_{3.5e-01}$	$7.01e-02_{3.3e-01}$

Table 2. Machine 1. Results after applying the Wilcoxon rank sum test to the $I^1_{\epsilon+}$ values. Each cell contain a symbol per each of the benchmark problems (LZ09_F1 .. LZ09_F9).

	M$_1$	M$_2$	M$_4$	M$_8$	M$_{16}$	M$_{32}$
M$_{seq}$	– – – – – – – – –	– – – – – – – – –	– – – – – – – – –	– – – – – – – – –	– ▲ – ▲ – – – ▲ ▲	▲ ▲ ▲ ▲ ▲ – – ▲ ▲
M$_1$		– – – – – – – – –	– – – – – – – ▲ –	– – – – – – – – ▲	– – – ▲ – – ▲ ▲ ▲	▲ ▲ ▲ ▲ ▲ – ▲ ▲ ▲
M$_2$			– – – – – – – – –	– – – – – – – – ▲	– ▲ – ▲ – – – – ▲	– – ▲ ▲ – – – ▲ ▲
M$_4$				– – – – – – – – –	– ▲ – ▲ – – ▲ – ▲	▲ ▲ ▲ ▲ ▲ – ▲ – ▲
M$_8$					– ▲ – ▲ ▲ – – – –	▲ ▲ ▲ ▲ ▲ – – ▲ ▲
M$_{16}$						▲ – – – – – – – –

Table 3. Machine 2. Median and interquartile range of the $I^1_{\epsilon+}$ indicator. Cells with dark and light background indicate the best and second best values, respectively.

	M$_{seq}$	M$_1$	M$_2$	M$_4$	M$_8$	M$_{16}$	M$_{32}$
LZ09_F1	$2.27e-03_{2.7e-04}$	$2.30e-03_{3.1e-04}$	$2.30e-03_{3.2e-04}$	$2.29e-03_{3.2e-04}$	$2.31e-03_{2.7e-04}$	$2.26e-03_{2.7e-04}$	$2.33e-03_{3.2e-04}$
LZ09_F2	$6.73e-02_{2.1e-01}$	$7.33e-02_{2.1e-01}$	$1.85e-01_{2.1e-01}$	$1.40e-01_{2.3e-01}$	$2.11e-01_{2.5e-01}$	$2.10e-01_{2.3e-01}$	$2.20e-01_{2.2e-01}$
LZ09_F3	$6.03e-02_{2.2e-01}$	$5.57e-02_{1.4e-01}$	$5.55e-02_{2.1e-01}$	$7.11e-02_{2.3e-01}$	$5.15e-02_{9.7e-02}$	$6.43e-02_{2.2e-01}$	$6.83e-02_{2.2e-01}$
LZ09_F4	$4.89e-02_{1.6e-02}$	$5.10e-02_{1.9e-02}$	$5.21e-02_{1.8e-02}$	$5.05e-02_{1.7e-02}$	$5.07e-02_{2.2e-02}$	$5.24e-02_{1.6e-02}$	$5.62e-02_{2.1e-02}$
LZ09_F5	$7.42e-02_{2.4e-02}$	$7.72e-02_{2.5e-02}$	$7.36e-02_{2.1e-02}$	$7.71e-02_{2.0e-02}$	$7.69e-02_{2.6e-02}$	$8.36e-02_{4.3e-02}$	$7.98e-02_{2.7e-02}$
LZ09_F6	$1.25e-01_{2.4e-02}$	$1.26e-01_{2.4e-02}$	$1.25e-01_{2.9e-02}$	$1.23e-01_{2.1e-02}$	$1.21e-01_{2.5e-02}$	$1.25e-01_{2.4e-02}$	$1.24e-01_{2.3e-02}$
LZ09_F7	$2.96e-02_{2.5e-02}$	$3.34e-02_{3.2e-02}$	$2.99e-02_{3.6e-02}$	$3.20e-02_{3.2e-02}$	$3.38e-02_{2.9e-02}$	$3.22e-02_{3.5e-02}$	$3.59e-02_{2.3e-02}$
LZ09_F8	$1.98e-01_{8.2e-02}$	$2.02e-01_{1.2e-01}$	$2.10e-01_{9.6e-02}$	$2.15e-01_{7.9e-02}$	$1.92e-01_{8.5e-02}$	$2.14e-01_{9.3e-02}$	$2.22e-01_{1.2e-01}$
LZ09_F9	$3.35e-02_{6.9e-02}$	$3.91e-02_{4.8e-02}$	$3.18e-02_{5.9e-02}$	$3.87e-02_{6.3e-02}$	$4.59e-02_{1.9e-01}$	$5.86e-02_{5.9e-02}$	$6.44e-02_{9.6e-02}$

Table 4. Machine 2. Results after applying the Wilcoxon rank sum test to the $I^1_{\epsilon+}$ values . Each cell contain a symbol per each of the benchmark problems (LZ09_F1 .. LZ09_F9).

	M$_1$	M$_2$	M$_4$	M$_8$	M$_{16}$	M$_{32}$
M$_{seq}$	– – – – – – – – –	– – – – – – – ▲ –	– – – – – – – ▲ –	– ▲ – – – – – – –	– ▲ – – ▲ – – ▲ –	▲ ▲ – ▲ – – ▲ ▲ ▲
M$_1$		– – – – – – – – –	– – ▲ – – – – – –	– – – – – – – – –	– ▲ – – ▲ – – – ▲	– ▲ – ▲ – – – – ▲
M$_2$			– – ▲ – – – – – –	– – – – – – – ▽ –	– ▲ – – ▲ – – – ▲	– ▲ – ▲ ▲ – – – ▲
M$_4$				– – ▽ – – – – ▽ –	– – – – ▲ – – – –	– – – ▲ – – – – ▲
M$_8$					– – – – ▲ – – ▲ –	– – ▲ ▲ – – – ▲ –
M$_{16}$						– – – ▲ – – – – –

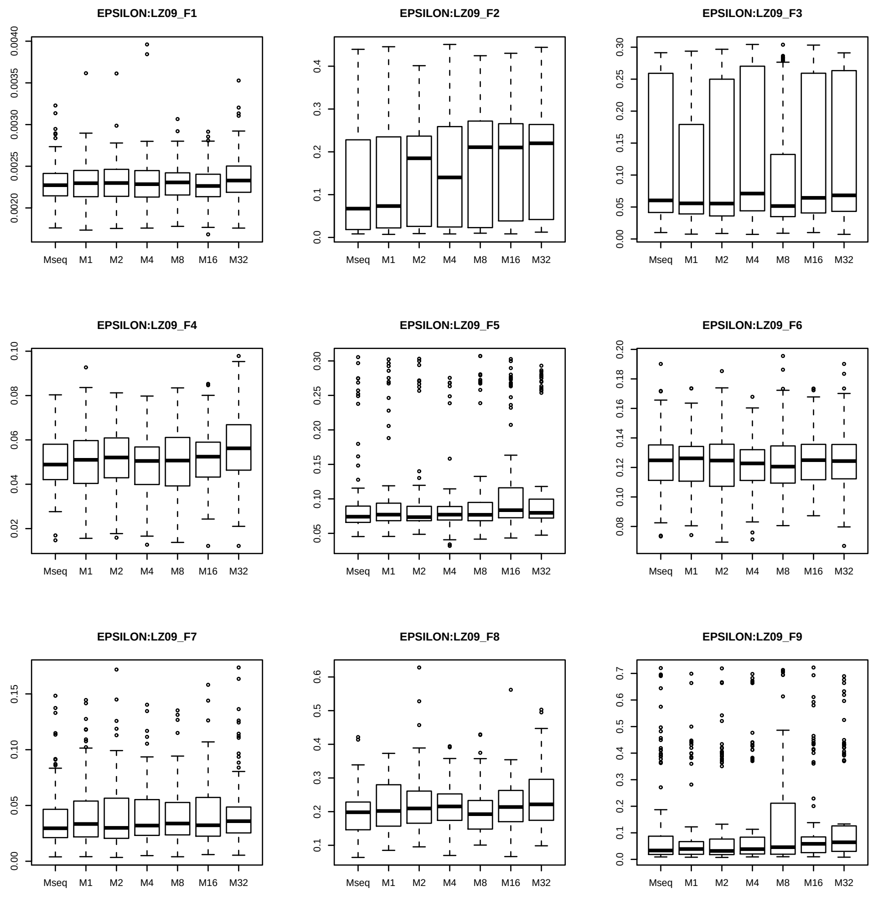

Fig. 2. Boxplots of the results of the $I^1_{\epsilon+}$ indicator obtained with Machine 2.

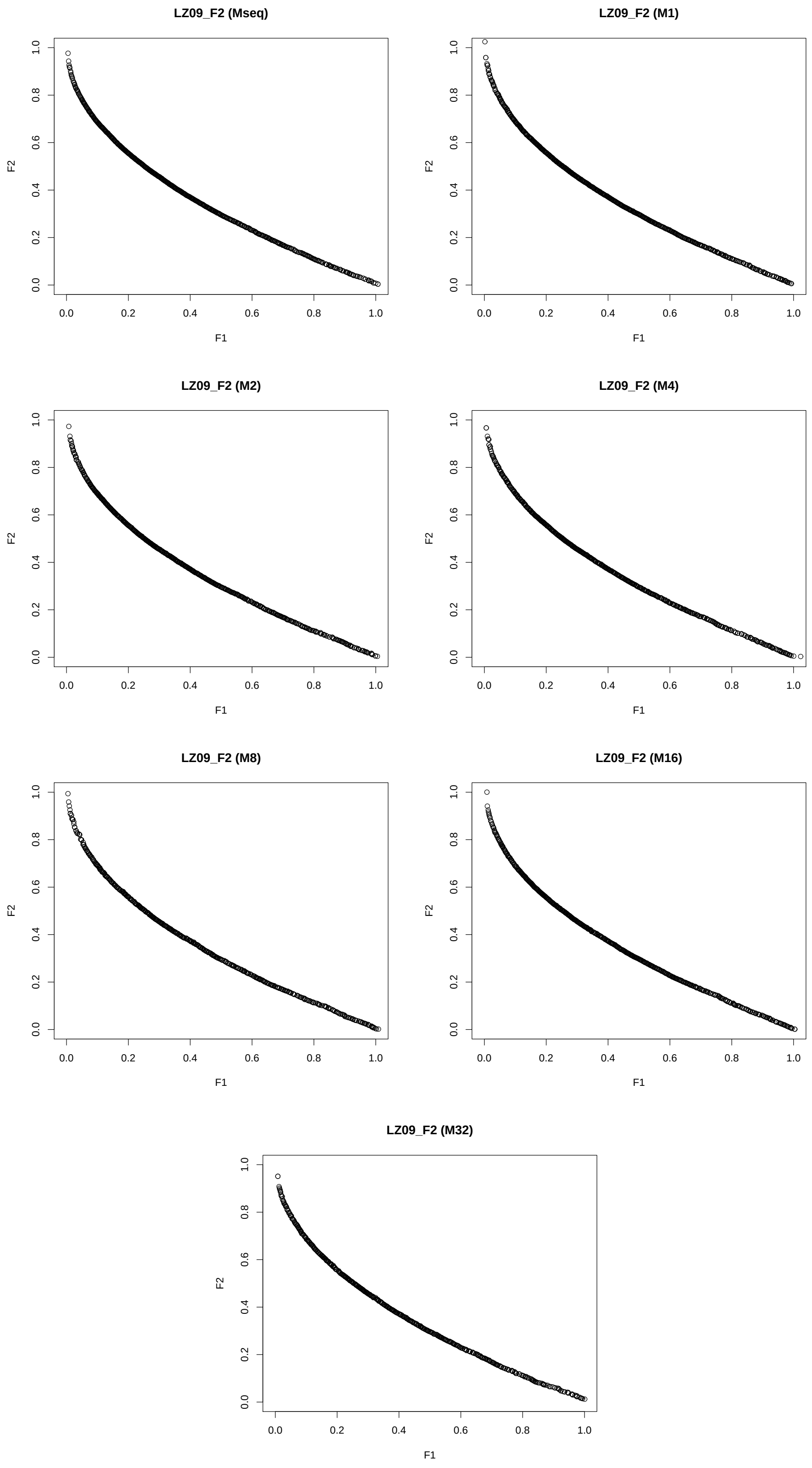

Fig. 3. Examples of Pareto fronts computed for problem LZ09_F4 with the different algorithms using Machine 2.

Finally, to illustrate the kind of Pareto front approximations yielded by the different versions of the algorithm, we include a number of examples of fronts computed for problem LZ09_F2 in Machine 2 in Fig. 3. Specifically, these are the best solution sets obtained by each algorithm regarding to the $I^1_{\epsilon+}$ indicator. These fronts are quite similar between them, although we can appreciate some slight differences.

5 Parallel Performance

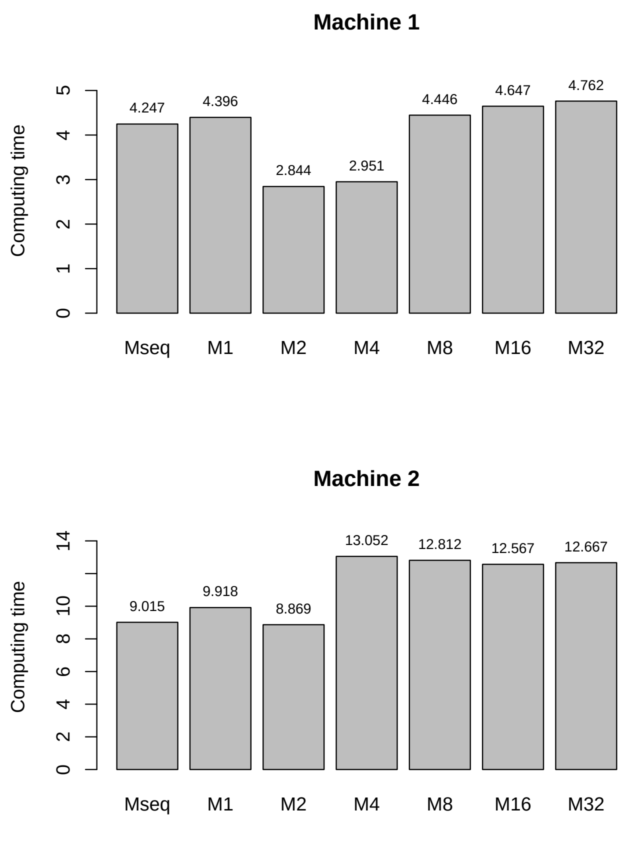

Fig. 4. Computing times (in secs) required by MOEA/D, M$_{seq}$, and pMOEA/D, M$_x$ ($x = 1, 2, 4, 8, 16, 32$) to solve problem LZ09_F1 on Machine 1 (top) and Machine 2 (bottom).

We have carried out a number of experiments to evaluate the parallel performance of pMOEA/D. The first one consist in measuring the computing time of the algorithms in both, Machine 1 and Machine 2, when solving LZ09_F1. Intuitively, as MOEA/D requires few seconds to be run in the used configuration in both computers, we should not expect large speed-ups due to the fact that the synchronization points in the parallel code impose a penalty that may not be amortized having into account such short computing times.

Fig. 4 shows the time required by the algorithms when they are run in Machine 1 (top) and Machine 2 (bottom). Starting with the former, we can observe that the sequential MOEA/D needs 4.247 secs, while the 4.396 secs of M_1 implies that there is an overhead of about 0.150 secs to run the same algorithm with one thread. The performance of the MOEA/D is improved when using pMOEA/D with two and four threads, and there are not any benefits of using eight or more threads. It is also worth noting that configuring pMOEA/D up to 32 threads only introduce an overhead of a few hundreds of milliseconds, which can be considered as acceptable.

Paying attention to the running times in Machine 2, we see that MOEA/D needs 9.015 secs, and pMOEA/D configured with only one thread requires 9.918 secs; hence, the penalty imposed by using one thread is about one sec. In this machine, only M_2 improves the performance of the sequential version: 8.869 secs vs 9.015. Furthermore, the overhead of using four or more threads is about 33% of the time of running the sequential version.

The poor speed-ups achieved in both machines were expected, as commented before, due to the short computing time of the algorithms, which is directly related to the low cost of evaluating the functions of the problem. We have to think, however, that parallelizing an algorithm which is executed in a few seconds usually does not make much sense in practice. To draw a more reasonable scenario, we have modified problem LZ09_F1 including a useless loop with the purpose of increasing the time needed to evaluate it.

After a number of preliminary experiments, we have configured the loop in such a way that the time to run MOEA/D in both computers is about 1,000 secs. This is a more realistic situation, where the time reductions of using the target multi-core CPUs could be tangible. Next, we analyze the speed-up and efficiency of the algorithms in the two machines in this context.

Fig. 5 shows the speed-up (left) and the efficiency (right) of the algorithms when they are running in Machine 1. Now, the speed-up yields more interesting figures: 1.84, 2.88, and 5.36 with two, four, and eight threads, respectively. Attending to the efficiency, it is higher than 65% in the worst cases. The speed-up and efficiency in Machine 2 are show in Fig. 6; here, both performance metrics are better than in Machine 1 due to that we obtain a quasi-linear speed-up: 1.99, 3.98, and 7.74 with 2, 4, and 8 threads respectively. As a consequence, the efficiency presents a shape practically constant and close to the 100%.

These results indicate that the speed-ups that can be achieved with the proposed parallel scheme are related to the target hardware. This can be due to that the eight processing units of the i7 CPU are not full cores, as it happens

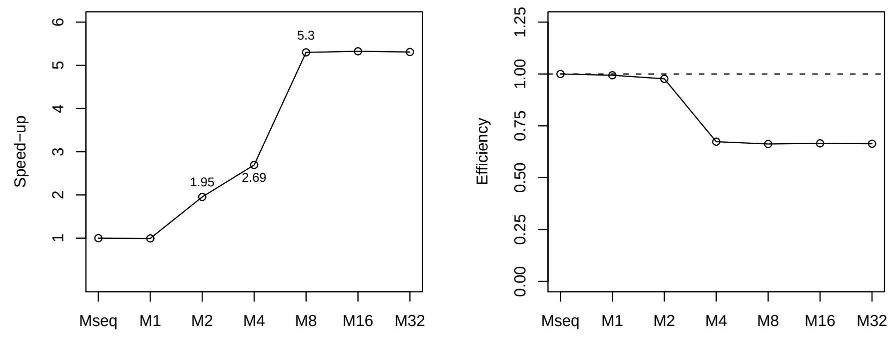

Fig. 5. Machine 1. Speed-ups and efficiency of MOEA/D (Mseq) and pMOEA/D (M_x) when solving a modified version of problem LZ09_F1 to increase its evaluation time.

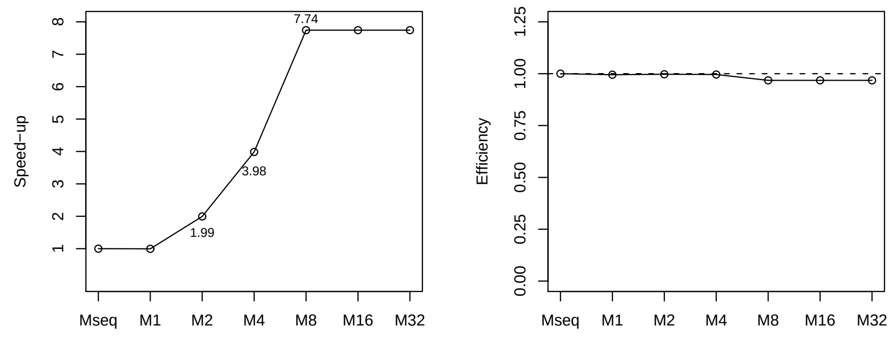

Fig. 6. Machine 2. Speed-ups and efficiency of MOEA/D (Mseq) and pMOEA/D (M_x) when solving a modified version of problem LZ09_F1 to increase its evaluation time.

with the Xeon processors, leading to an unbalance in the execution of the threads of parallel algorithm.

6 Conclusions and Further Work

In this paper we have designed a parallel version of MOEA/D for multi-core processors and we have studied its performance by using 1, 2, 4, 8, 16, and 32 threads. Particularly, we have paid attention not only to benefits in the speed-up but also to differences in the search capabilities of the algorithms.

To come with these issues, we have used two computers having different multi-core processors and we have considered two kind of experiments. The first one was composed of nine complex problems aimed at evaluating the behaviour of the algorithms. The obtained results in this experiment have pointed out that there are not significant differences between the fronts computed by MOEA/D and pMOEA/D regarding to the $I^1_{\epsilon+}$ quality indicator in most of the evaluated problems when using up to eight threads. Furthermore, in some cases, we have observed that by using eight threads it is possible to outperform the results obtained by using a lower number of them.

The second experiment was intended to evaluate the parallel performance of pMOEA/D. In this case, we have seen that, depending on the architecture where the experiments are running in, it is possible to obtain quasi-lineal speed-ups, and consequently efficiency values close to 100%.

Future work will verify those facts by executing a larger number of experiments and by applying the parallel MOEA/D to real world problems. Other parallel schemes are also matter of future work, specifically, those approaches which suit well in grid environments.

Acknowledgments. This work has been partially funded by the "Consejería de Innovación, Ciencia y Empresa", Junta de Andalucía, under contract P07-TIC-03044 (the DIRICOM project), `http://diricom.lcc.uma.es`, and the Spanish Ministry of Science and Innovation, under contract TIN2008-06491-C04-01 (the M* project), `http://mstar.lcc.uma.es`. Juan J. Durillo is supported by grant AP-2006-03349 from the Spanish government.

References

1. G.R. Andrews. *Multithreaded, Paralle, and Distributed Programming.* Addison-Wesley, 2000.
2. C. Blum and A. Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
3. C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems.* Springer, New York, second edition, September 2007. ISBN 978-0-387-33254-3.
4. K. Deb. *Multi-objective optimization using evolutionary algorithms.* John Wiley & Sons, 2001.

5. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.
6. J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.
7. J.J. Durillo, A.J. Nebro, F. Luna, B. Dorronsoro, and E. Alba. jMetal: a java framework for developing multi-objective optimization metaheuristics. Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, 2006.
8. F. W. Glover and G. A. Kochenberger. *Handbook of Metaheuristics.* Kluwer, 2003.
9. J. Knowles, L. Thiele, and E. Zitzler. A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers. Technical Report 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, 2006.
10. J. D. Knowles and D. W. Corne. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation*, 8(2):149–172, 2000.
11. S. Kukkonen and J. Lampinen. GDE3: The third evolution step of generalized differential evolution. In *IEEE Congress on Evolutionary Computation (CEC'2005)*, pages 443 – 450, 2005.
12. H. Li and Q. Zhang. Multiobjective optimization problems with complicated pareto sets, MOEA/D and NSGA-II. *IEEE Transactions on Evolutionary Computation*, 2(12):284–302, April 2009.
13. A. J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. J. Durillo, and A. Beham. AbYSS: Adapting Scatter Search to Multiobjective Optimization. *IEEE Transactions on Evolutionary Computation*, 12(4), August 2008.
14. A.J. Nebro, J.J. Durillo, F. Luna, B. Dorronsoro, and E. Alba. Mocell: A cellular genetic algorithm for multiobjective optimization. *Internatinal Journal of Intelligent Systems*, 24(7):726–746, 2009.
15. T. Weise, M. Zapf, R. Chiong, and A. J. Nebro. Why is optimization difficult? In Raymond Chiong, editor, *Nature-Inspired Algorithms for Optimisation*, volume 193/2009 of *Studies in Computational Intelligence*, pages 1–50. Springer, 2009.
16. Q. Zhang and H. Li. Moea/d: A multi-objective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 1(6):712–731, December 2007.
17. Q. Zhang, W. Liu, and H. Li. The performance of a new version of moea/d on cec09 unconstrained mop test instances. Technical Report CES-491, School of CS & EE, University of Essex, 2009.
18. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. In K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, and T. Fogarty, editors, *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, pages 95–100, Athens, Greece, 2002.