Parallel Extremal Optimization with Guided State Changes Applied to Load Balancing

Ivanoe De Falco[1], Eryk Laskowski[2(✉)], Richard Olejnik[3], Umberto Scafuri[1], Ernesto Tarantino[1], and Marek Tudruj[2,4]

[1] Institute of High Performance Computing and Networking, CNR, Naples, Italy
{ivanoe.defalco,umberto.scafuri,ernesto.tarantino}@na.icar.cnr.it
[2] Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
{laskowsk,tudruj}@ipipan.waw.pl
[3] Computer Science Laboratory, University of Science and Technology of Lille, Villeneuve-d'Ascq, France
richard.olejnik@lifl.fr
[4] Polish-Japanese Institute of Information Technology, Warsaw, Poland

Abstract. The paper concerns parallel methods for Extremal Optimization (EO) applied for processor load balancing for distributed programs. In these methods the EO approach is used which is parallelized and extended by a guided search of next solution state. EO detects the best strategy of tasks migration leading to a reduction in program execution time. We assume a parallel improvement of the EO algorithm with guided state changes which provides a parallel search for a solution based on two step stochastic selection during the solution improvement based on two fitness functions. The load balancing improvements based on EO aim at better convergence of the algorithm and better quality of program execution in terms of the execution time. The proposed load balancing algorithm is evaluated by experiments with simulated parallelized load balancing of distributed program graphs.

Keywords: Distributed program design · Extremal optimization · Load balancing · Parallel computing

1 Introduction

Many papers exist in literature dealing with dynamic load balancing in parallel and distributed systems [1,2]. However, they do not profit from Extremal Optimization (EO) [3] which is a technique following the approach of self-organized criticality [4]. The algorithms presented in this paper are parallelized versions of the EO–based load balancing algorithms [5,6], in which EO has been used for load balancing of processors in execution of distributed programs. In the previous paper, we have modified the EO algorithm to replace the fully random processor selection by the stochastic selection in which the probability used in the selection mechanism is guided by some knowledge of the problem (the EO-GS algorithm). The guidance is based on a formula which examines how a migrated

A.M. Mora and G. Squillero (Eds.): EvoApplications 2015, LNCS 9028, pp. 79–90, 2015.
DOI: 10.1007/978-3-319-16549-3_7

task fits a given processor in terms of the global computational balance in the system and the processor communication load. The algorithm was evaluated by simulation experiments in the DEVS (Discrete Event Simulation) model [7]. The experiments have assessed the new algorithm against different parameters of the application program graphs and the load balancing algorithm. The application speedups have been positively verified against those obtained with the standard EO-based version and in genetic algorithms.

An interesting aspect of Extremal Optimization is parallelization of the involved approach. Parallelization of Extremal Optimization can be considered in two ways. The first way is to intensify the actions aiming at a possibly stronger improvement of the current EO solution, frequently with the introduction of a population-based representation or a multipoint strategy during solution improvement. It can be done using a really parallel system or a sequential system in which the components of an EO solution are identified based on the multipoint selection and improved in a possibly concurrent way. The second way consists in using a population-based approach for solutions with parallel component improvement. Both approaches have accumulated already some bibliography. In [8] the authors propose an extended EO model in which a single EO solution is replaced by a set of EO solutions which are processed using the general EO strategy. These solutions are subject to selection and mutation to provide a set of solution vectors to be next processed in parallel.

In [9–11] the authors propose a rich set of EO improvements used for optimization of problems in molecular biology. The MEO (Modified EO) concept consists in random generation of a neighbour solution in component improvement but the best solution is selected among multiple thus formed new solutions in respect to the fitness function used in the component selection. The PMEO is a Population-based Modified EO in which a combination of a population-based approach to solution generation with the MEO approach to the selection of the best solution for further improvements. The generated solutions copy a substructure of the solution which behaves well in the solution improvement. The third approach identified in the papers is the Distributed Modified EO (DMEO) which is a combination of the PMEO approach and the distributed genetic algorithms methodology. The DMEO is based on distribution of a population of solutions into islands. The islands evolve using the PMEO method. There are transfers of best solutions between the islands with back transfers of the replaced ones. Each island improves a set of solutions to find a best island solution which are next compared to find a globally best solution.

In this paper we propose an EO-based parallel approach for solving a processor load balancing problem in execution of programs represented as layered graphs of tasks. In this approach, we first identify a load imbalance in the functioning of the executive system. Then, we apply a parallel EO-GS algorithm to select tasks which are to be migrated among processors to improve the general balance of processor loads. The EO-GS algorithm is performed in the background of the application program execution to execute a given number of EO cycles which find a number of best logical migrations of tasks. When the EO iterations

are over, the physical migrations worked out by the EO take place. In the parallel EO-GS, a method similar to PMEO is applied but with an additional fitness function which is a base for the stochastic selection of the best solution state in the neighbourhood of the one chosen for improvement.

The paper is organized as follows. In Sect. 2 the Extremal Optimization principles including the guided state changes are presented. Section 3 presents the way of using EO for processor load balancing. Section 4 describes the proposed parallel version of the algorithm. Section 5 presents the experimental results which evaluate the proposed approach.

2 Extremal Optimization with Guided State Changes

In Extremal Optimization we use iterative updates of a single solution S built of a number of components s_i, which are variables of the problem. For each component, a local fitness value ϕ_i is evaluated to select the worst variable s_w in the solution. In a generic EO, S is modified at each iteration step, by randomly updating the worst variable. As a result, a solution S' is created which belongs to the neighbourhood $Neigh(S, s_w)$ of S. For S' the global fitness function $\Phi(S)$ is evaluated which assess the quality of S'. The new solution S' replaces S if its global fitness is better than that of S. We can avoid staying in a local optimum in such EO, by using a probabilistic version τ–EO, [3]. It is based on a user-defined parameter τ, used in stochastic selection of the updated component. In a minimization problem solved by τ–EO, the solution components are first assigned ranks k, $1 \leq k \leq n$, where n is the number of the components, consistently with the increasing order of their local fitness values. It is done by a permutation π of the component labels i such that: $\phi_\pi(1) \leq \phi_\pi(2) \leq \ldots \phi_\pi(n)$. The worst component s_i is of rank 1, while the best one is of rank n. Then, the component selection probability over the ranks k is defined as follows: $p_k \sim k^{-\tau}$, for a given value of the parameter τ. At each iteration, a component rank k is selected in the current solution S according to p_k. Next, the respective component s_j with $j = \pi(k)$ randomly changes its state and S moves to a neighboring solution, $S' \in Neigh(S, s_j)$, unconditionally. The parameters of the τ–EO are: the total number of iterations $\mathcal{N}_{\text{iter}}$ and the probabilistic selection parameter τ.

τ–EO with guided state changes (EO–GS) has been proposed to improve the convergence speed of EO optimization. For this, some knowledge of the problem properties is used for next solution selection in consecutive EO iterations with the help of an additional local target function ω_s. This function is evaluated for all neighbour solutions existing in $Neigh(S, s_{\pi(k)})$ for the selected rank k. Then, the neighbour solutions are sorted and assigned GS-ranks g with the use of the function ω_s. The new state $S' \in Neigh(S, s_{\pi(k)})$ is selected in a stochastic way using the exponential distribution with the selection probability $p \sim \text{Exp}(g, \lambda) = \lambda e^{-\lambda g}$. Due to this, better neighbour solutions are more probable to be selected. The bias to better neighbours is controlled by the λ parameter. The general scheme of the EO–GS is shown as Algorithm 1.

Algorithm 1. EO algorithm with Guided State Changes (EO–GS)

initialize configuration S at will
$S_{\text{best}} \leftarrow S$
while total number of iterations $\mathcal{N}_{\text{iter}}$ not reached **do**
  evaluate ϕ_i for each variable s_i of the current solution S
  rank the variables s_i based on their local fitness ϕ_i
  choose the rank k according to $k^{-\tau}$ so that the variable s_j with $j = \pi(k)$ is selected
  evaluate ω_s for each neighbour $S_v \in Neigh(S, s_j)$, generated by s_j change of the current solution S
  rank neighbours $S_v \in Neigh(S, s_j)$ based on the target function ω_s
  choose $S' \in Neigh(S, s_j)$ according to the exponential distribution
  accept $S \leftarrow S'$ unconditionally
  if $\Phi(S) < \Phi(S_{\text{best}})$ **then**
    $S_{\text{best}} \leftarrow S$
  end if
end while
return S_{best} and $\Phi(S_{\text{best}})$

3 EO-Based Load Balancing Foundations

In this section we will recall basic theoretical foundations for the proposed EO-based load balancing. The proposed load balancing method is meant for a clusters of multicore processors interconnected by a message passing network. Load balancing actions for a program are controlled at the level of indivisible tasks which are process threads. We assume that the load balancing algorithms dynamically control assignment of program tasks t_k, $k \in \{1 \dots |T|\}$ to processors (computing nodes) n, $n \in [0, |N| - 1]$, where T and N are the sets of all the tasks and the computing nodes, respectively. The goal is the minimal total program execution time, achieved by task migration between processors. The load balancing method is based on a series of steps in which detection and correction of processor load imbalance is done, Fig. 2. The imbalance detection relies on some run-time infrastructure which observes the state of the executive computer system and the execution states of application programs. Processors (computing nodes) periodically report their current loads to the load balancing control which monitors the current system load imbalance. When load imbalance is discovered, processor load correction actions are launched. For them an EO-based algorithm is executed, which identifies the tasks which need migration and the migration target processor nodes. Following this, the required physical task migrations are performed with the return to the load imbalance detection.

To evaluate the load of the system two indicators are used. The first is the computing power of a node n: $Ind_{power}(n)$, which is the sum of potential computing powers of all the active cores on the node. The second is the percentage of the CPU power available for application threads on the node n: $Time^{\%}_{\text{CPU}}(n)$, periodically estimated on computing nodes. The percentage of the CPU power available for a single thread is computed as a quotient of the time during which the CPU was allocated to a probe thread against the time interval

of the measurement. $Time^{\%}_{\mathrm{CPU}}(n)$ value is the sum of the percentage of CPU power available for the number of probe threads equal to the number of cores on the node.

System load imbalance LI is a boolean defined based on the difference of the CPU availability between the currently most heavily and the least heavily loaded computing nodes:

$$LI = \max_{n \in N}(Time^{\%}_{\mathrm{CPU}}(n)) - \min_{n \in N}(Time^{\%}_{\mathrm{CPU}}(n)) \geq \alpha$$

The load imbalance equal true requires a load correction. The value of α is set using an experimental approach (during experiments we set it between 25 % and 75 %).

An application is characterized by two programmer-supplied parameters, based on the volume of computations and communications tasks: $\mathrm{COM}(t_s, t_d)$ is the communication metrics between tasks t_s and t_d, $\mathrm{WP}(t)$ is the load weight metrics introduced by a task t. $\mathrm{COM}(t_s, t_d)$ and $\mathrm{WP}(t)$ metrics can provide exact values, e.g. for well-defined tasks sizes and inter-task communication in regular parallel applications, or only some predictions e.g. when the execution time depends on the processed data.

A task mapping solution S is represented by a vector $\mu = (\mu_1, \ldots, \mu_{|T|})$ of $|T|$ integers ranging in the interval $[0, |N| - 1]$. $\mu_i = j$ means that the solution S under consideration maps the i–th task t_i onto the computing node j.

The global fitness function $\Phi(S)$ is defined as follows.

$$\begin{aligned} \Phi(S) = attrExtTotal(S) * \Delta_1 + migration(S) * \Delta_2 \\ + imbalance(S) * [1 - (\Delta_1 + \Delta_2)] \end{aligned} \tag{1}$$

where $1 > \Delta_1 \geq 0$, $1 > \Delta_2 \geq 0$ and $\Delta_1 + \Delta_2 < 1$ hold.

The function $attrExtTotal(S)$ represents the impact of the total external communication between tasks on the quality of a given mapping S. By "external" we mean the communication between tasks placed on different nodes. This function is normalized in the range $[0, 1]$. In executive systems with homogeneous communication links it is a quotient of an absolute value of the total external communication volume and the total communication volume of all communications (when all tasks are placed on the same node $attrExtTotal(S) = 0$, when tasks are placed in the way that all communication is external $attrExtTotal(S) = 1$); in heterogeneous executive systems equivalent measures of the communication time are used:

$$attrExtTotal(S) = totalExt(S)/\overline{\mathrm{COM}}$$

where: $\overline{\mathrm{COM}} = \sum_{s,d \in T} \mathrm{COM}(s, d)$ and $totalExt(S) = \sum_{s,d \in T : \mu_s \neq \mu_d} \mathrm{COM}(s, d)$.

The function $migration(S)$ is a migration costs metrics. The value of this function is in the range $[0, 1]$, i.e. it is equal to 0 when there is no migration, when all tasks have to be migrated $migration(S) = 1$, otherwise $0 \leq migration(S) \leq 1$:

$$migration(S) = |\{t \in T : \mu_t^S \neq \mu_t^{S*}\}|/|T|$$

where: S is the currently considered solution and $S*$ is the previous solution (or the initial solution in the algorithm).

The function $imbalance(S)$ represents the numerical load imbalance metrics in the solution S. It is equal to 1 when in S there exists at least one unloaded (empty) computing node, otherwise it is equal to the normalized average absolute load deviation of tasks in S, determined in the definition below:

$$imbalance(S) = \begin{cases} 1 & \text{exists at least one unloaded node} \\ D^*(S)/2 * N * \overline{\text{WP}} & \text{otherwise} \end{cases}$$

where: $D^*(S) = \sum_{n\in[0,N-1]} |\text{NWP}(S,n)/Ind_{power}(n) - \overline{\text{WP}}|$, $\overline{\text{WP}} = \sum_{t\in T} \text{WP}(t)/\sum_{n\in[0,N-1]} Ind_{power}(n)$, $\text{NWP}(S,n) = \sum_{t\in T:\mu_t=n} \text{WP}(t)$.

In the applied EO the local fitness function of a task $\phi(t)$ is designed in such a way that it forces moving tasks away from overloaded nodes, at the same time preserving low external (inter-node) communication. The γ parameter ($0 < \gamma < 1$) allows tuning the weight of load metrics.

$$\phi(t) = \gamma * load(\mu_t) + (1-\gamma) * rank(t) \tag{2}$$

The function $load(n)$ indicates how much the load of node n, which executes t, exceeds the average load of all nodes. It is normalized versus the heaviest load among all the nodes. The $rank(t)$ function governs the selection of best candidates for migration. The chance for migration have tasks, which show low communication with their current node (attraction) and low load deviation from the average load:

$$rank(t) = 1 - (\beta * attr(t) + (1-\beta) * ldev(t))$$

where: β is a real number between 0 and 1 – a parameter indicating the importance of the weight of attraction metrics. The attraction of the task t to its executive computing node $attr(t)$ is defined as the amount of communication between task t and other tasks on the same node, normalized versus the maximal communication inside the node. The load deviation compared to the average load $ldev(t)$ is defined as the absolute value of the difference between the load metrics of the task t and the minimum load on the node, normalized versus the highest such difference for all tasks on the node.

We use the EO-GS algorithm to perform task and target node selection for migration. Target node selection is based on additional "biased" stochastic approach, to favour some solutions over others. In our case, the valid solution state neighbourhood includes the use of all system nodes. Therefore, at each update of rank k, all nodes $n \in N$ are sorted using the $\omega(n1, n2)$ function, $n1, n2 \in N$, with the assignment of GS-ranks g to them. Then, one of the nodes is selected using the exponential distribution $\text{Exp}(g,\lambda) = \lambda e^{-\lambda g}$.

We propose the following definition of $\omega(n1, n2)$ for the sorting algorithm based on a pairwise ordering of the computing nodes $n1, n2$ as targets for migration of task j in the load balancing algorithm. It takes into account the normalized load deviation of the nodes $n1, n2$ and the attraction of the task j to the each of these nodes.

$$\omega(n1,n2) = \begin{cases} sgn(relload(n1) - relload(n2)) & \text{when } relload(n1) \neq relload(n2) \\ sgn(attrext^{\%}(j,n2) - attrext^{\%}(j,n1)) & \text{otherwise.} \end{cases}$$

where:

$$relload(n) = \frac{loaddev(n) - \min_{m\in[0,N-1]} loaddev(m)}{\max_{m\in[0,N-1]} loaddev(m) - \min_{m\in[0,N-1]} loaddev(m)}$$

$$loaddev(n) = \frac{\mathrm{NWP}(S,n)}{Ind_{power}(n)} - \overline{\mathrm{WP}}$$

$$attrext^{\%}(j,n) = \frac{attrext(j,n)}{\max_{e\in N}(attrext(j,e))}$$

$$attrext(j,n) = \sum_{e\in T(n)} (\mathrm{COM}(e,j) + \mathrm{COM}(j,e))$$

and $T(n) = \{t \in T : \mu_t = n\}$ — the set of threads, placed on computing node n.

4 Parallel Extremal Optimization Applied to Load Balancing

The general scheme of the parallel version of the EO algorithm applied in this paper to load balancing of distributed programs is presented in Fig. 1. This algorithm is a parallelized version of the Algorithm 1 in Sect. 2. In this scheme, after the setting of an initial solution S, the local fitness function $\phi(s_j)$ values are evaluated for all the components of S and the ranking of the components

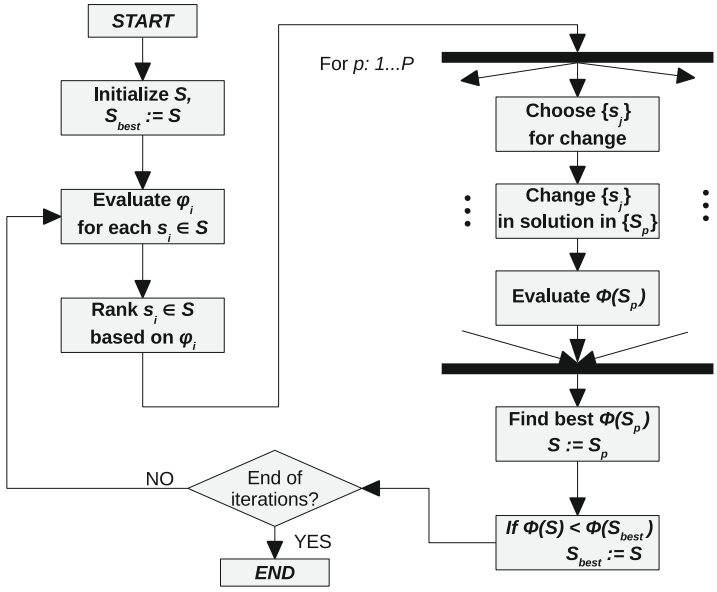

Fig. 1. The general scheme of the parallel version of the EO algorithm.

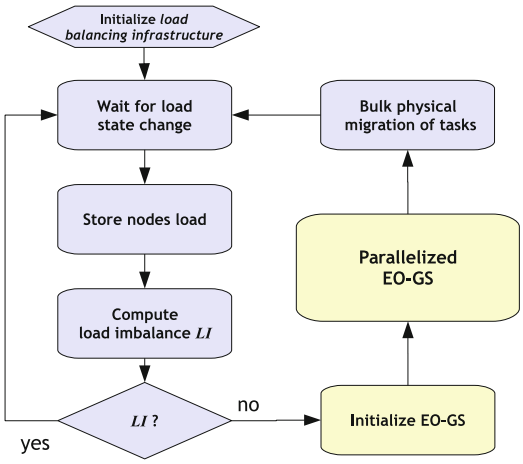

Fig. 2. The general scheme of load balancing based on parallel EO with guided state changes.

S based on $\phi(s_j)$ is constructed. Next, a parallel EO part of the algorithm starts. The population-type parallel EO algorithm is applied, which replaces the single improvement of the EO solution by a parallel search for improvements performed on a population of P EO solutions S_p. They are generated by a parallel selection of the solution components and, next, an improvement of the worst components stochastically selected based on the local fitness function with the highest probability of the worst components. During selection of components for solution improvement a single point or multipoint selection of components in the same basic solution can be performed. The set of components is improved using a parallelized random selection or a parallelized approach of guided state changes explained in Sect. 3. For each improved solution, the global fitness function $\Phi(S)$ is evaluated. The improved solution with the global fitness better than the current best value is selected as the base for next parallel EO iterations. The local and global fitness function definitions are explained in Sect. 3.

The general scheme of the load balancing algorithm based on the parallel EO-GS is shown in Fig. 2. The scheme is composed of two co-operating iterative parts represented in the left and right parts of Fig. 2. Both parts can be executed in a parallel way. In the left part, we first identify a sufficiently big load imbalance in the functioning of the executive system following the model explained in Sect. 3. When a sufficiently big processor load imbalance is noticed in program execution, then we apply a parallel version of EO-GS to select tasks which are to be migrated among processors to improve the general balance of processor loads (the right hand side of Fig. 2). The parallel EO-GS algorithm performs a given number of parallel iterations. EO-GS finds a number of best "logical" migrations of tasks, which are registered to be performed when the EO-GS iterations are over. Then, the physical migrations worked out by the EO-GS take

place. The parallel EO-GS and the load imbalance detection are executed in the background of the application program. The online overhead introduced in the application execution time is due to task migration. The overhead is strongly dependent on whether only data or task codes are migrated.

5 Experimental Results

The experiments were performed using simulated execution of application programs in a distributed system. Applications were run in a simulated distributed memory cluster of 32 multi–core processors. Communication was based on message-passing. The DEVS-based system simulator [7] and parallel EO algorithms were run in a cluster of Intel i7-based, 8–core workstations.

During the experiments we used a set of 5 synthetic exemplary programs, which were randomly generated. The exemplary programs were composed of layers of parallel tasks. Tasks from the same layer could communicate. At the boundaries between layers there was a global exchange of data.The number of tasks in an application varied from 272 to 576. The communication/computation ratio for applications was in the range $[0.10, 0.20]$. The first exemplary program is an irregular application in which the execution time of tasks depends on the processed data. Thus, it exhibits unpredictable execution time of tasks and the communication scheme and load imbalance can occur in computing nodes. The next four programs are regular applications that have fixed tasks' execution times. In regular applications load imbalance can appear due to a non–optimized task placement of tasks or runtime conditions change.

During experiments, we have compared parallel EO and EO–GS to classic (sequential) EO and EO–GS which use the same local and global fitness functions. The following parameters for load balancing control were used: $\alpha = 0.5, \beta = 0.5, \gamma = 0.75, \Delta_1 = 0.13, \Delta_2 = 0.17, \tau = 1.5$, and for EO–GS $\lambda = 0.5$. Other settings of control parameters are not discussed here since we have presented them in [5,6]. Each experiment was repeated 5 times, for each run 4 different methods of initial task placements (random, round-robin, METIS, packed) were tested. Thus, 20 runs were executed in total for each experiment to produce an averaged result. Experiments were repeated for the number of iterations of EO and EO–GS set to 30, 60, 120 and 250.

In the first experiment, we tested the performance of standard EO parallelized by concurrent random mutation. Figure 3(a) and 3(b) show the average improvements over classic EO for the number of parallel candidate solutions in the range [1, 8] (columns) and the different iteration count (rows). The reference was the speedup for the lowest number of iterations (30) and non-parallelized, classic EO. The speedup improvement is in the range $[0\,\%, 5\,\%]$, where better speedup is obtained when the number of iterations or parallel candidate solutions is higher. It means that we can substitute the number of iterations with widening the search area. The change of migrations number for different numbers of parallel candidate solutions and the iteration count, Fig. 3(b), reveals that a smaller number of migrations is obtained for parallelized version of EO.

In the second experiment, we investigated the performance of EO–GS parallelized with concurrent mutation guided by the search of the best solution state change as explained in Sect. 3, Fig. 4(a) and (b). Similarly as in the first experiment, the reference was speedup for 30 iterations and non-parallelized, classic EO. The speedup improvement was in the range $[10\,\%, 12\,\%]$, substantially higher than that for the parallelized standard EO. On the other side, the speedup improvement less depends on the number of iterations of EO and the number of parallel candidate solutions. It confirms that EO–GS is able to find load balance solutions of high quality, thus eliminating the need for further intensive search of the solution space.

We analyzed also the impact of the regularity of applications on the obtained results. For irregular applications the speedup strongly depends on load balancing. Figure 5 shows the speedup improvement of irregular applications for parallelized EO and parallelized EO with Guided Search (EO–GS). The speedup improvement depends on the number of candidates and is noticeably better than the average values for all kinds of applications. For irregular applications it is more profitable to extend the search area by parallelizing EO than to increase the number of iterations of EO. For regular applications speedup was not sensitive to the number of candidate solutions.

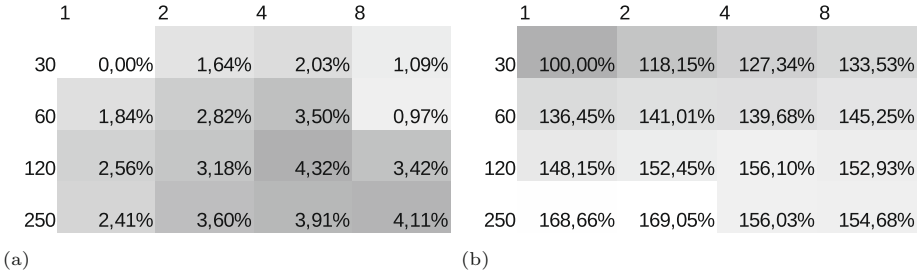

Fig. 3. Average speedup improvement (a) and migrations number change (b) of parallelized EO against classic EO for different numbers of parallel candidate solutions (columns) and the iteration counts (rows).

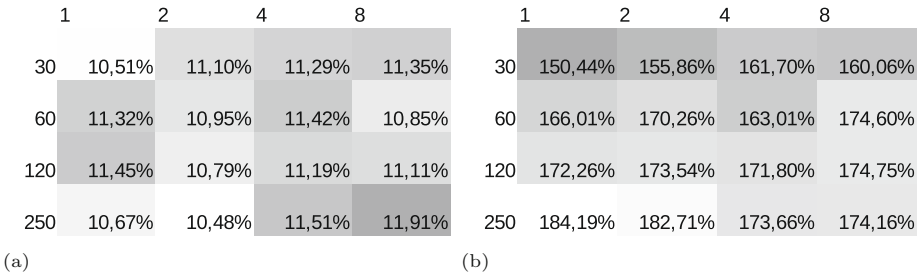

Fig. 4. Average speedup improvement (a) and migrations number change (b) of parallelized EO with Guided Search (EO–GS) against classic EO for different numbers of parallel candidate solutions (columns) and the iteration counts (rows).

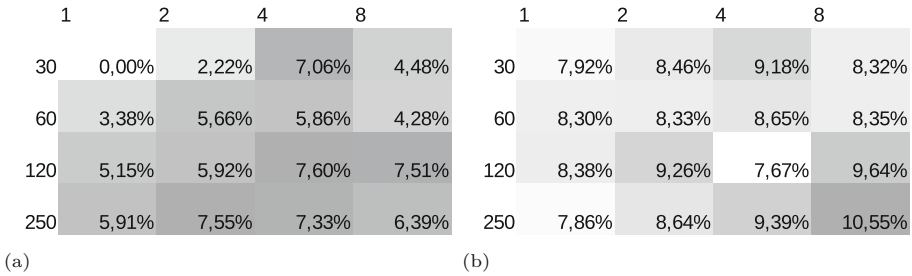

Fig. 5. Average speedup improvement of irregular applications for parallelized EO (a) and parallelized EO–GS (b) against classic EO for different numbers of parallel candidate solutions (columns) and the iteration counts (rows).

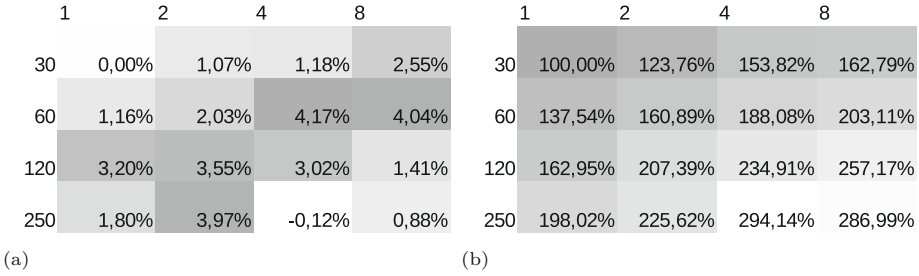

Fig. 6. Average speedup improvement (a) and migrations number change (b) in irregular application for EO parallelized according to island model against classic EO for different numbers of parallel candidate solutions (columns) and the iteration counts (rows).

In the last experiment we tested the EO parallelized using a "population model" but without solutions exchange, Fig. 6(a) and (b). In general, this kind of parallelization of EO did not provide satisfying speedup improvement. Only for irregular applications we got speedup improvement. The limiting factor was here the inadequate solution exchange between parallel EO instances.

The general conclusion coming from our experiments is that the extension of the solution search space in EO through parallel mutations of several candidate solutions even in its basic form provides satisfactory results. We were able to obtain better (or at least the same) quality of load balancing without increasing the number of iterations of EO but using computing power of multicore CPUs. Additional profit of using many parallel state changes in EO is a reduction in the number of task migrations needed to balance the system.

6 Conclusions

The paper has presented a parallel algorithm for dynamic processor load balancing in execution of distributed programs. The algorithm is based on internal

use of an improved parallel EO method – EO-GS. The purpose of the parallel EO-GS algorithm was to determine candidates for task migrations in the overall load balancing procedure. The improvement consisted in the use of a guidance of an additional state quality function which corresponded to a better use of the knowledge of the problem. The experiments with simulated load balancing following the proposed algorithm have shown that the support by this guidance was successful. Application of the parallel EO-GS enabled obtaining better quality of load balancing compared to other tested methods.

References

1. Khan, R.Z., Ali, J.: Classification of task partitioning and load balancing strategies in distributed parallel computing systems. Int. J. Comput. Appl. **60**(17), 48–53 (2012)
2. Mishra, M., Agarwal, S., Mishra, P., Singh, S.: Comparative analysis of various evolutionary techniques of load balancing: a review. Int. J. Comput. Appl. **63**(15), 8–13 (2013)
3. Boettcher, S., Percus, A.G.: Extremal optimization: methods derived from coevolution. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), pp. 825–832. Morgan Kaufmann, San Francisco (1999)
4. Sneppen, K., et al.: Evolution as a self-organized critical phenomenon. Proc. Natl. Acad. Sci. **92**, 5209–5213 (1995)
5. De Falco, I., Laskowski, E., Olejnik, R., Scafuri, U., Tarantino, E., Tudruj, M.: Load balancing in distributed applications based on extremal optimization. In: Esparcia-Alcázar, A.I. (ed.) EvoApplications 2013. LNCS, vol. 7835, pp. 52–61. Springer, Heidelberg (2013)
6. De Falco, I., Laskowski, E., Olejnik, R., Scafuri, U., Tarantino, E., Tudruj, M.: Improving extremal optimization in load balancing by local search. In: Esparcia-Alcázar, A.I., Mora, A.M. (eds.) EvoApplications 2014. LNCS, vol. 8602, pp. 51–62. Springer, Heidelberg (2014)
7. Zeigler, B.: Hierarchical, modular discrete-event modelling in an object-oriented environment. Simulation **49**(5), 219–230 (1987)
8. Randall, M., Lewis, A.: An extended extremal optimisation model for parallel architectures. In: 2nd IEEE International Conference on e-Science and Grid Computing, e-Science 2006, p. 114 (2006)
9. Tamura, K., Kitakami, H., Nakada, A.: Reducing crossovers in reconciliation graphs with extremal optimization (in japanese). Trans. Inf. Process. Soc. Japan **49**(4(TOM 20)), 105–116 (2008)
10. Tamura, K., Kitakami, H., Nakada, A.: Distributed extremal optimization using island model for reducing crossovers in reconciliation graph. In: Proceedings of the International MultiConference of Engineers and Computer Scientists 2013, Hong-Kong, March 2013, pp. 1–6 (2013)
11. Tamura, K., Kitakami, H., Nakada, A.: Distributed modified extremal optimization using Island model for reducing crossovers in reconciliation graph. Eng. Lett. **21**(2), EL_21_2_05 (2013)