Effective Processor Load Balancing using Multi-Objective Parallel Extremal Optimization

Ivanoe De Falco
Institute of High Performance Computing and Networking, CNR
Naples, Italy
ivanoe.defalco@icar.cnr.it

Eryk Laskowski
Institute of Computer Science, Polish Academy of Sciences
Warsaw, Poland
laskowsk@ipipan.waw.pl

Richard Olejnik
Université Lille, CNRS, Centrale Lille, UMR 9189 - CRISTAL
Lille, France
richard.olejnik@univ-lille1.fr

Umberto Scafuri
Institute of High Performance Computing and Networking, CNR
umberto.scafuri@icar.cnr.it

Ernesto Tarantino
Institute of High Performance Computing and Networking, CNR
ernesto.tarantino@icar.cnr.it

Marek Tudruj
Institute of Computer Science PAS
Polish-Japanese Academy of Information Technology
Warsaw, Poland
tudruj@ipipan.waw.pl

ABSTRACT

The paper presents how Extremal Optimization can be used in a parallel multi-objective load balancing algorithm applied in execution of distributed programs. Extremal Optimization is used to find task migration which dynamically improves processor load balance in a distributed system. In the proposed multi-objective approach we use three objectives relevant to distributed processor load balancing in execution of program tasks. They are: computational load balance of processors, the volume of inter-processor communication and task migration metrics. In the algorithms additional criteria are used which are based on some knowledge on the influence of the computational and communication loads on task execution. The proposed algorithms are assessed by simulation experiments with distributed execution of program macro data flow graphs. Two methods of finding compromise solutions based on the Pareto front were used: one based on a geometric (Euclidean) distance of solutions and the second one based on the Manhattan (taxicab geometry) distance. The influence of the distance geometry on the final solutions is discussed.

CCS CONCEPTS

• **Mathematics of computing** → **Evolutionary algorithms**; • **Applied computing** → **Multi-criterion optimization and decision-making**; • **Computing methodologies** → *Shared memory algorithms*; *Discrete-event simulation*;

KEYWORDS

extremal optimization, multi-objective optimization, processor load balancing

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208289

ACM Reference Format:
Ivanoe De Falco, Eryk Laskowski, Richard Olejnik, Umberto Scafuri, Ernesto Tarantino, and Marek Tudruj. 2018. Effective Processor Load Balancing using Multi-Objective Parallel Extremal Optimization. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205651.3208289

1 INTRODUCTION

Extremal Optimization (EO) [1, 2] is a nature inspired optimization method which has small computational complexity and low memory requirements. These features make EO a promising approach to be applied in the algorithms for processor load balancing in distributed systems. In several previous papers [4–7] we have examined how EO could be applied to processor load balancing in execution of distributed programs specified as macro data flow graphs. The discussed algorithms concerned processor load balancing using both sequential and parallel single objective EO approaches. Based on these algorithms and experimental results we have noticed that a multi-criteria load balancing approach could improve the load balancing algorithms by taking into account more complex optimization aims. It served as inspiration for a multi-criteria approach to load balancing [8], in which we considered three parameters of program execution in cluster environments: computational load of processors, inter-processor communication intensity and some metrics relating to the quality of task migrations. In our previous papers on single criterion load balancing based on EO we have assumed the EO global fitness function which was a linear combination of the three parameter kinds. Consequently, in the current paper, we have taken a multi-objective approach and defined separate optimization objectives based on some modifications of the three mentioned criteria with unchanged general load balancing optimization axes. It enabled obtaining better quality of load balancing than with the single criterion approach.

The three objectives have been included into the currently studied parallel multi criteria EO algorithm. Its general structure consists in defining parallel branches in which series of steps are executed with the use of the dynamically changing three objectives (acting as EO global fitness functions). For each objective respective

methods of the EO solution element selection for improvement were used (implied by EO local fitness functions). The algorithm delivered the Pareto front of the optimization results and also the final compromise solutions obtained by finding the solutions which minimize the distance of the Pareto solutions to an ideal point with respect to a given norm. In the case of our target of load balancing, we have examined two norms: the Euclidean distance in a three-dimensional Cartesian space and the Manhattan distance.

The focus of the current paper is primarily on developing foundations for the multi criteria EO-based load balancing methodology in general. In this aspect, the proposed program and system metrics used as the base of the load balancing algorithms enable particular heuristics which have originality features. Good reviews and classifications of classic load balancing methods are presented in [3, 10, 11]. Good reviews of load balancing methods based on evolutionary algorithms including EO are contained in [6, 7, 12, 13]. When we scan load balancing methods known in the literature or we analyze features of current parallel computing environments with load balancing, we notice that except of our works none of others includes EO as the distributed processor load balancing algorithm component. So, the approaches proposed in this and our previous papers have clear originality features, employ low computational complexity and limited use of memory space of EO in load balancing. In the proposed algorithms, a special parallel EO-GS approach (EO with a Guided Search) is applied which assures that selection of a new solution from the neighbourhood is guided by some knowledge of the problem.

The algorithms presented in the current paper are improved versions of the EO-based load balancing algorithms presented in [5–7] by using a multi-objective optimization approach. Large surveys on general methods of multi-objective optimization can be found in [17, 18]. Extensive surveys on multi-objective optimization methods combined with evolutionary algorithms in general can be found in [19, 20]. These surveys are too extensive to be discussed here. Some useful papers which support the general multi-objective optimization technology are presented in [21–26]. Multi-objective approach applied to EO has already been discussed in several papers [27–30]. They propose basic methods of multi-objective optimization and cover different technical aspects of this approach. However, they are oriented towards generalized optimization problems and do not cover specific multi-objective evolutionary algorithms applied to processor load balancing.

The separability between the local fitness values for particular solution components and the global fitness values is not straightforward in EO-based load balancing algorithms. The relevant data cannot be expressed by any simple analytical formulae and should be based on measurements done in a real system. The simplest way was to evaluate them by simulating the application behaviour in a given system, what in fact we were doing using the DEVS-based simulator [15]. The algorithms were assessed by experiments with simulated load balancing of distributed program represented as macro data flow graphs. The experimental results obtained by simulation compared the compromise solutions obtained using our multi-criteria parallel EO-GS approach with those of a classical EO-GS algorithm with the mentioned above singular local and global fitness functions. The superiority of the multi-objective approach against the mentioned above single-objective one has been demonstrated. The experimental results indicate also that, in general, both metrics, Manhattan and Euclidean, may be useful for certain types of application graphs. However, for application graphs used during the experimental research, the Manhattan distance is more effective, since it reduces the number of migrations more strongly.

The paper is organized as follows. In Section 2 the EO principles are re-called. Section 3 describes the processor load balancing approach based on the proposed multi-objective parallelized EO-GS algorithm. Section 4 presents the experimental assessment of the proposed load balancing approach.

2 EXTREMAL OPTIMIZATION ALGORITHMS

Extremal Optimization is a nature-inspired, local search optimization heuristic. It was proposed by Boettcher and Percus [1], following the Bak–Sneppen approach of self–organized dynamic criticality [14]. As an attractive optimization method for NP–hard combinatorial problem, it is also used in many other optimization domains.

A probabilistic version of EO (τ–EO) operates on a single solution S consisting of a given number of components s_i, each of which is a variable of the problem. At each algorithm iteration, a local fitness value ϕ_i is assigned to each of them. Then, for a minimization problem, the components are ranked in decreasing order of local fitness values. The worst component s_j is of rank 1, while the best one is of rank G, where G is the number of components. Then, a distribution probability over the ranks k is considered as follows: $p_k \sim k^{-\tau}$, $1 \leq k \leq G$ for a given value of the parameter τ. Finally, at each update, a rank k is selected according to p_k so that the component s_i of rank k randomly changes its state and the solution moves to a neighboring one, $S' \in Neigh(S)$, unconditionally. At the end of iteration, the global fitness $\Phi(S')$ is computed, and the new solution S' is saved if its global fitness value is better than that of the best solution found so far. The only parameters are the total number of iterations $\mathcal{N}_{\text{iter}}$ and the probabilistic selection parameter τ. For minimization problems τ–EO proceeds as in the Algorithm 1.

To foster the convergence speed of EO optimization, we have proposed a modified version of τ–EO algorithm, called Extremal

Algorithm 1 Extremal Optimization algorithm (EO)

1: initialize configuration S at will
2: $S_{\text{best}} \leftarrow S$
3: **while** total number of iterations $\mathcal{N}_{\text{iter}}$ not reached **do**
4: evaluate ϕ_i for each variable s_i of the current solution S
5: rank the variables s_i based on their fitness ϕ_i
6: choose the rank k according to the distribution probability $k^{-\tau}$ so that the variable s_j with $j = \pi(k)$ is selected
7: choose $S' \in Neigh(S)$ such that s_j must change
8: accept $S \leftarrow S'$ unconditionally
9: **if** $\Phi(S) < \Phi(S_{\text{best}})$ **then**
10: $S_{\text{best}} \leftarrow S$
11: **end if**
12: **end while**
13: **return** S_{best} and $\Phi(S_{\text{best}})$

Optimization with Guided Search (EO-GS) [5, 6]. In EO-GS, some knowledge of the problem properties is used during the next solution selection in consecutive EO iterations with the help of an additional local target function ω_s. The value of this function is evaluated for all neighbours $Neigh(S)$ of rank k. Then, the neighbour solutions are sorted and assigned GS-ranks g with the use of the function ω_s. The new state $S' \in Neigh(S)$ 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.

3 LOAD BALANCING BASED ON THE PARALLEL MULTI-OBJECTIVE EO

In the paper we propose Parallel Multi-Objective EO-GS-based load balancing algorithm for a cluster of multi-core processors interconnected by a message passing network. In our approach, the classical EO-GS method is extended in two ways. First, we use a multi-objective approach in which we apply a number of objectives supporting the load balancing problem. Second, our algorithm is a parallelized version of EO-GS, following our previous research, presented in [8]. In the following sections we describe our multi-objective approach for load balancing of distributed programs and its parallel extension.

3.1 Processor load balancing scheme

The goal of the load balancing algorithms is to dynamically control assignment of program tasks $t_k, k \in 1, \ldots, |T|$ to processors (computing nodes) $n, n \in 0, 1, \ldots, |N| - 1$, where T and N are the sets of all the tasks and the computing nodes, respectively. Load balancing actions are performed on-line to dynamically preserve the even distribution of application tasks on processors. 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. 1. The imbalance detection relies on some run-time infrastructure which observes the state of processors in the executive computer system and the execution states of application programs.

When load imbalance is discovered, processor load correction actions are launched. For them a multi-objective EO-GS algorithm is executed to identify the tasks which need migration and the processor nodes which will be migration targets. 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: $\text{power}_{\text{CPU}}(n)$, which is the sum of potential computing powers of all the active cores on the node, available for application execution. The second is the percentage of the CPU power available for application threads on the node n: $\text{time}_{\text{CPU}}(n)$, periodically estimated on computing nodes.

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

$$I = \max_{n=0,\ldots,|N|-1} (\text{time}_{\text{CPU}}(n)) - \min_{n=0,\ldots,|N|-1} (\text{time}_{\text{CPU}}(n)) \geq \alpha \quad (1)$$

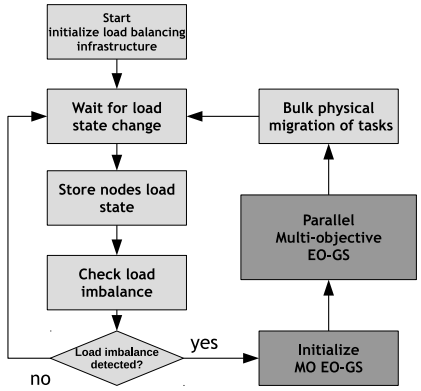

Figure 1: The general scheme of load balancing based on Multi-Objective EO with guided search.

The load imbalance equal true requires a load correction. The value of α is determined experimentally (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: $\text{com}(t_s, t_d)$ is a communication metrics between tasks t_s and t_d, $\text{wp}(t)$ is a load weight metrics introduced by a task t. $\text{com}(t_s, t_d)$ and $\text{wp}(t)$ 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. Even when the values are exact, we assume that there can some fluctuations of tasks execution or CPU power availability, so the dynamic load balancing is required.

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

3.2 Multi-objective Extremal Optimization

In our solution we solve a processor load balancing problem in execution of distributed programs with the use of a multi-objective EO-GS algorithm (MOEO-GS), shown as Algorithm 2. The proposed MOEO-GS algorithm follows the general scheme of EO-GS approach described in the section 2, with the exception that it uses a set of EO local and global fitness functions and maintains the Pareto front of non-dominated solutions.

During an iteration, the selection and solution improvement are performed using a single objective (i.e. a single EO local and a respective single global fitness functions). It is selected in a probabilistic way from the MO objectives specified for our load balancing problem (the local and global fitness functions used in our MOEO-GS algorithm are defined in the Subsection 3.4).

The Pareto front is analyzed at the end of the algorithm to deliver the S_{best} solution. To approximate the optimal solution we look for the so-called compromise solution, e.i. the solution as close as possible to the ideal point. The compromise solution S_{best} is selected from D_{S} using given a distance measure. In our implementation

Algorithm 2 Multi-objective EO with Guided Search (MOEO-GS)

initialize configuration S at will
$S_{\text{best}} \leftarrow S$
$D_{\text{S}} \leftarrow \emptyset$ {the set of non-dominated solutions (Pareto-front)}
while total number of iterations $\mathcal{N}_{\text{iter}}$ not reached **do**
  $c \leftarrow$ a criterion for evaluation in the current iteration
  evaluate $\phi_{i,c}$ for each variable s_i of the current solution S
  rank the variables s_i based on their local fitness $\phi_{i,c}$
  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 S is non-dominated **then**
    include S in D_{S}, remove dominated solutions from D_{S}
  end if
end while
select S_{best} from D_{S} using $\Phi(S)$
return S_{best} and $\Phi(S_{\text{best}})$

Algorithm 3 Parallel Multi-objective EO-GS (PMEO)

initialize S using current nodes load state
$S_{\text{best}} \leftarrow S$
while number of outer iterations $\mathcal{N}_{\text{outer}}$ not reached **do**
  distribute S_{best} on processors $p \in \{1, \ldots, P\}$
  parallel for on processors $p \in \{1, \ldots, P\}$ **do**
    execute sequential MOEO-GS on processor p with the number of iterations $\mathcal{N}_{\text{inner}}$ {the inner loop}
    $S_{\text{p}} \leftarrow$ result of MOEO-GS
  end for
  gather S_{p} from processors $p \in \{1, \ldots, P\}$
  $S_{\text{best}} \leftarrow$ find the best value of $S_{\text{p}}, p \in \{1, \ldots, P\}$
end while
return S_{best} and $\Phi(S_{\text{best}})$

of the algorithm we use alternatively two distance metrics: the Manhattan norm (taxicab geometry [9]) and the Euclidean norm.

3.3 Parallel MOEO-GS

A parallel version of the EO algorithm that has been used in the multi-objective load balancing algorithm reported in this paper uses the sequential MOEO-GS as its basic building block. The general scheme of the algorithm is presented in Alg. 3. The algorithm begins with an initialization of the EO starting "best" solution based on current loads of all computing nodes in the distributed application. Next, a parallel part of the algorithm starts, with parallel iterative execution of sequential versions of EO.

The algorithm consists of two loops: the outer main global data exchange loop and the inner parallel EO loop nested inside the outer loop. The inner loop is executed in parallel branches of the algorithm scheme. The inner loop body represents a sequential EO algorithm executed a number of times. These can be classic EO, EO-GS, or multi-objective EO which was described in the previous section.

When all the parallel inner EO loops are terminated, the solution with the best value S_{best_p} among the global fitness function gathered from all parallel branches is registered as the current best in the outer loop. Next, the algorithm enters the solution exchange phase, in which an initial starting EO solution from previous iterations is identified for the next outer loop iteration. The solution is next distributed among P parallel branches of the scheme. Then, the next parallel EO algorithm outer loop iteration starts.

If the total number of EO iterations to be executed in P parallel branches is denoted as $\mathcal{N}_{\text{iter}}$ then the number of inner loop iterations in a single parallel branch is equal to $\mathcal{N}_{\text{iter}}/P$. Parallel branches are executed on separate cores of a multicore processor we use for the algorithm.

3.4 Global and local fitness functions

In our algorithm, we use three objective functions oriented on supporting the load balancing problem: total computational load unbalance in execution of application tasks on processors, total volume of communication between tasks placed on different computing nodes and task migration number which should be possibly small in fighting imbalance of processor loads. Since MOEO-GS is a minimization algorithm, it looks for the solutions with lower values of the global fitness (or components with lower values of the local fitness, respectively).

The definitions of fitness functions use two auxiliary formulas:

$$\text{nwp}(S, n) = \sum_{t \in T : \mu_t = n} \text{wp}(t) \tag{2}$$

$$W_T = \sum_{t \in T} \text{wp}(t) / \sum_{n=0,\ldots,|N|-1} \text{power}_{\text{CPU}}(n) \tag{3}$$

where $\text{nwp}(S, n)$ is the sum of computational load of program tasks allocated to processor n in the solution S, and W_T is the average computational weight of program tasks attributed to one unit of computational power of processors.

A load imbalance normalization constant is equal to maximal numerical value of the imbalance (i.e. when all tasks are assigned to the slowest processor):

$$D_{\text{norm}} = (|N|-2) * W_T + \sum_{t \in T} \text{wp}(t) / \min_{n=0,\ldots,|N|-1} \text{power}_{\text{CPU}}(n) \tag{4}$$

The first objective concerns the reduction of the computational load imbalance among executive processors in the system during a given phase of distributed program execution i.e. defined by the current MOEO-GS solution S. The global fitness functions $\Phi(S)$ for computational load unbalance (objective **U**) is defined as follows:

$$\Phi^1_U(S) = \begin{cases} 1 & \text{exists at least one free node} \\ \text{deviation}(S)/D_{\text{norm}} & \text{otherwise} \end{cases} \tag{5}$$

where:

$$\text{deviation}(S) = \sum_{n=0,\ldots,|N|-1} |\,\text{nwp}(S, n)/\text{power}_{\text{CPU}}(n) - W_T\,| \tag{6}$$

The function $\Phi^1_U(S)$ 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 absolute load deviation of tasks from average load in S.

The local fitness function for MOEO-GS algorithm for the objective 1 is designed as follows:

$$\phi_U(t) = \gamma * \text{load}(\mu_t) + (1-\gamma) * (1 - \text{ldev}(t)) \tag{7}$$

where 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 function ldev(t) is defined as the difference between the load metrics of the task t and the average task load on the node μ_t, normalized versus the highest such value for all tasks on the node [6].

The second objective for the MOEO-GS algorithm is the global EO-GS fitness function $\Phi(S)$ for external communication (objective **C**) defined as follows:

$$\Phi_C(S) = \sum_{s,d \in T: \mu_s \neq \mu_d} \text{com}(s,d) / \sum_{s,d \in T} \text{com}(s,d) \tag{8}$$

The function $\Phi_C(S) \in [0,1]$ represents the impact of the external (i.e. inter-node) communication between tasks on the quality of a given mapping S. It is a quotient of the sum of external communication volume and the total communication volume in a program. When all tasks are placed on the same node $\Phi_C(S) = 0$, when tasks are placed so that all communication is external $\Phi_C(S) = 1$.

The local fitness function for objective 2 is designed as follows:

$$\phi_C(t) = 1 - \text{attr}(t) \tag{9}$$

where 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 attraction inside the node [6].

The third objective for the MOEO-GS algorithm is concerned with task migrations induced by the current EO-GS solution S in terms of the computational load imbalance. The global EO-GS fitness function for migration (objective **M**) corresponds to the number of migrations:

$$\Phi^1_M(S) = \text{migration}(S) \tag{10}$$

$$\text{migration}(S) = |\{t \in T : \mu^S_t \neq \mu^{S*}_t\}| / |T| \tag{11}$$

where: μ^S_t is the current node of the task t in the solution S, and μ^{S*}_t is the node of the task t in the initial solution at the start of the algorithm. The function $\Phi^1_M(S) \in [0,1]$ is a migration number metrics. It is equal to 0 when there is no migration, when all tasks have to be migrated $\Phi^1_M(S) = 1$, otherwise $0 \leq \Phi^1_M(S) \leq 1$.

The local fitness function $\phi_M(t)$ for migration objective $\Phi^1_M(S)$ is designed as follows:

$$\phi_M(t) = \begin{cases} 1 & \text{when task } t \text{ is migrated} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

The $\phi_M(t)$ local fitness function forces the migration of already migrated tasks, thus increasing the probability that finally more tasks will occupy their initial computing nodes.

An **alternative version** of the global fitness functions $\Phi_U(S)$ for the computational load unbalance objective is based on the migration quality coefficient. The function improvement(n) indicates how much the current placement of tasks on a node n improves (i.e. decreases) the load imbalance of the application, comparing the initial task placement:

$$\text{improvement}(n) = |\frac{\text{nwp}(S,n)}{\text{power}_{\text{CPU}}(n)} - W_T| - |\frac{\text{nwp}(S^*,n)}{\text{power}_{\text{CPU}}(n)} - W_T| \tag{13}$$

where S is the currently considered solution and S^* is the initial task placement at the start of the algorithm.

$$\Phi^2_U(S) = \frac{\text{totalimpr}(S) + 1}{2} \tag{14}$$

where:

$$\text{totalimpr}(S) = \frac{\sum_{n=0,\ldots,|N|-1} \text{improvement}(n)}{D_{\text{norm}}} \tag{15}$$

The function totalimpr(S) $\in [-1,1]$ indicates whether there is the improvement (when totalimpr(S) < 0) or deterioration (when > 0) in the total load balance in the system comparing the initial placement of tasks of the application. In such a way the task placement which ensures the best total processors balance improvement is a preferred outcome of the algorithm.

To summarize, the following two parallel MOEO-GS variants were designed: **MO-1** which uses $\Phi^1_U(S), \Phi_C(S), \Phi_M(S)$ global fitness functions and **MO-2** with $\Phi^2_U(S), \Phi_C(S), \Phi_M(S)$ global fitness functions, respectively. These variants use the respective local fitness functions, defined in equations 7, 9 and 12, which match the used global fitness functions, respectively.

4 EXPERIMENTAL ASSESSMENT

In this section we present experimental assessment of the algorithms. The experiments have been conducted using simulated execution of application programs in a distributed system. The applied simulator was built following DEVS discrete event system approach [15].

The simulated model of execution corresponds to parallelization based on message-passing, using the MPI library for communication. The general structure of a program resembled typical numerical programs or physical phenomena simulations. The exemplary programs were modeled as Temporal Flow Graphs, TFG, [16]. In the TFG model, program consists of a set of modules called phases, composed of parallel tasks, Fig. 2. Tasks of the same phase can communicate. At the boundaries between phases there is a global exchange of data.

During experiments we used a set of 11 synthetic exemplary programs, which were randomly generated. The number of tasks in an application varied from 64 to 1000. The communication to computation ratio C/E (the quotient of the communication time to the execution time) for applications was in the range $[0.05, 0.20]$.

During the experimental research, two parallel MOEO-GS variants were used: **MO-1** and **MO-2**. Moreover, both MOEO-GS variants can use different method of selection of a compromise solution among the set of non-dominated solutions included in the Pareto front. As we already have shown in Subsection 3.4, methods based on a geometric distance (Euclidean) and the Manhattan distance are used. They are denoted as **MO-1E**, **MO-1M** and **MO-2E**, **MO-2M**, respectively.

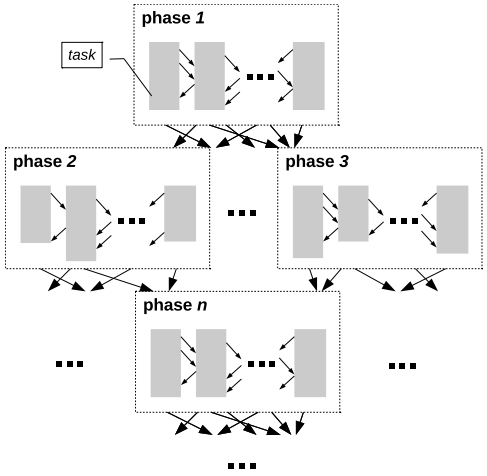

Figure 2: The general structure of exemplary applications.

4.1 Performance of the presented algorithms

To assess performance of the presented multi-objective algorithms, we used two reference single objective load balancing algorithms based on sequential EO. One (denoted as SO-C) aims in balancing exclusively computational loads of processor nodes. It is based on a classical sequential EO without guided search. The second one (denoted as SO-WS) is based on a single objective EO with the guided search and is using a global fitness function which is a weighted sum of the three aforementioned criteria (see Subsection 3.4) according to the equation:

$$\Phi_{\mathrm{WS}}(S) = \Phi_C(S)\Delta_1 + \Phi_M(S)\Delta_2 + \Phi_U^1(S)[1 - (\Delta_1 + \Delta_2)] \qquad (16)$$

where Δ_1 and Δ_2 are weights from the range (0,1).

Load-balanced execution was studied in systems containing from 2 to 32 homogeneous processors. The parameters used in MOEO-GS, the load balancing control and the weighted sum of the global fitness function of the single objective EO-WS were selected based on experiments, presented in [4, 5]. We applied such parameter values for which balanced performance between application speedup and migration count was obtained: $\alpha = 0.5$, $\tau = 1.5$, $\lambda = 0.14$, $\mathcal{N}_{\mathrm{iter}} = 500$, $\beta = 0.5$, $\Delta_1 = 0.13$, $\Delta_2 = 0.17$, $\gamma = 0.75$.

Our simulation environment allowed us to model the task migration cost. We have assumed the cost of migration of a single task to be equal to 20% of the task computational weight, as a medium migration cost between 0% and 40% considered also in further experiments. Besides the paralel speedup, we collected the migration statistics to better characterize our proposed parallel MOEO-GS algorithms. It gives the average total number of task migrations in application execution.

During the experiments, we assumed the number of iterations for EO and MOEO $\mathcal{N}_{\mathrm{iter}} = 500$ and the exchange rate of solutions between parallel branches every 25 inner iterations. We used $P = 4$ parallel branches, thus, the inner loop count is 25, and the outer count is 5 ($4 \times 5 \times 25 = 500$).

The results are averages of 10 runs. For each run 4 different methods of initial task placements (random, round-robin, METIS graph partitioning, packed /i.e. round-robin mapping of equal groups of tasks/) were used. Thus, the result for each parameter set is an average of 40 runs.

The summary of results for the entire set of used applications is presented in Fig. 3 and 4. The first one presents the parallel speedup of applications as a function of the number of the simulated computing nodes. The graph shows the average speedup for all applications with irregular communication pattern. It can be noted that regardless of the method used to select a final, compromise solution, multi-criteria algorithms generally achieve better speedup than their sequential counterparts. It can also be noted that for

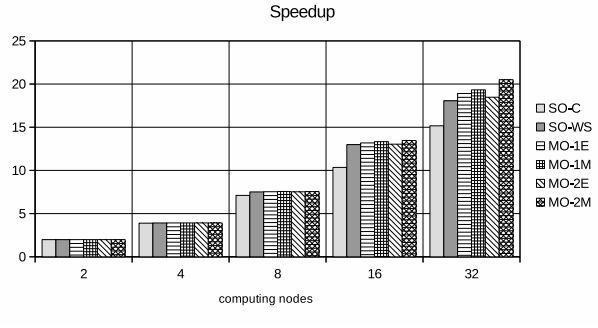

Figure 3: Application parallel speedup.

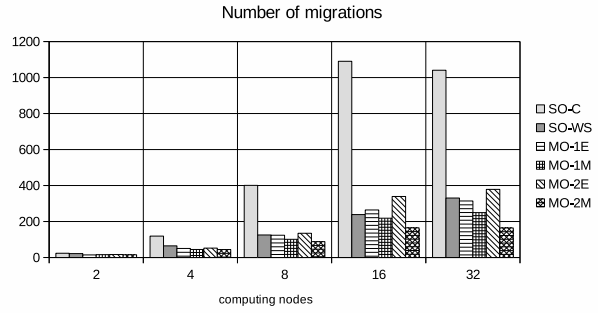

Figure 4: The number of task migrations per single application execution.

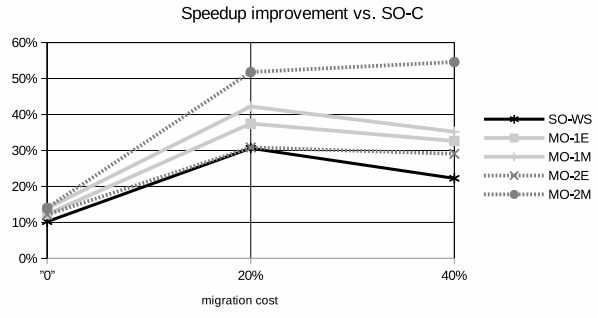

Figure 5: Application speedup improvement versus SO-C for execution on 32 computing nodes.

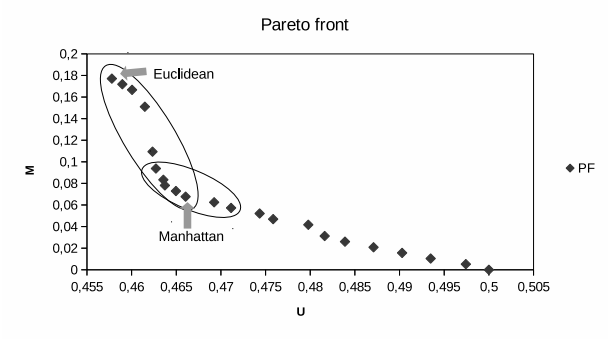

Figure 6: The Pareto front for single execution of MO-2 for P9F exemplary application run on 32 computing nodes.

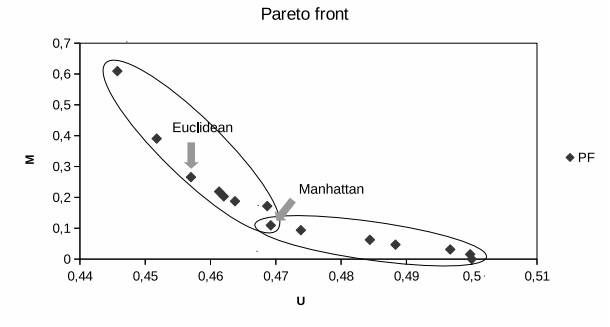

Figure 7: The Pareto front for single execution of MO-2 for P9A exemplary application run on 32 computing nodes.

variant MO-2, there is a greater variation in results depending on the method used to select a compromise solution. For the applications used during the experiments, the method based on the Manhattan distance gives better results.

A similar situation occurs for the number of migrations (Fig. 4). With the exception of MO-2E, the parallel load balancing algorithms are characterized by a smaller (sometimes much smaller) number of migrations required for application load balancing. The results for the number of migrations are correlated with the results for the speedup: the lower the number of migrations in the application, the greater the speedup. We also collected the results for a variable migration cost. The speedup âĂŃâĂŃfor the varying migration weights is shown in Fig. 5. These results confirm the hypothesis that the speedup correlates with the number of migrations. It can be noted that the algorithms that give the smallest number of migrations give much better speedup for the increasing cost of migration. This result also explains the large variation in the speedup results for the MO-2E and MO-2M algorithms. This is due to the fact that the MO-2M variant of the algorithm is able to find a good balance with fewer number of migrations, which gives big gains when the migration is expensive.

In addition to presenting the results for the parallel MOEO-GS method, the aim of the research was also to determine the impact of the method used to select a compromise solution on the quality of results. For this purpose, during the experimental research, the resulting Pareto front was recorded for each execution of the MOEO-GS algorithm. The charts of sample, representative Pareto front sets are shown in Fig. 6 and 7. They show the result for one algorithm execution, whereby the set of all non-dominated solutions was chosen for which the values of communication objective were equal to 1 (i.e., the entire application communication was external and occurred between different computing nodes). By omitting the communication criterion, it was possible to draw the Pareto front set on the plane. Fig. 6 and 7 show also the best solutions chosen when the Euclidean distance or the Manhattan distance was used. The final compromise solution found by each method is also marked using an arrow. It can be noted that the resulting Pareto front curves for applications used during the experiments are not convex.

4.2 The role of the communication criterion

During the analysis of the Pareto sets for individual executions of exemplary applications, it was observed that very often all non-dominated solutions had the value of the communication criterion equal to 1 (maximum). This means that all communication was external and occurred between different computing nodes. The analysis showed that this phenomenon results from the structure of applied exemplary applications. Due to the irregular nature of communication, there is a very little probability of finding a distribution of tasks on processors so as to optimize communication (to make it internal within one node). The bigger the application, the harder it is to find such a distribution. This is confirmed by the results of the experimental research. For the smaller P9A graph (320 tasks) about 50% of the non-dominated solutions had the value of the communication criterion less than 1. For the largest used application (P9F, 1000 tasks) only 2% of solutions had the value of the communication criterion less than 1.

This means that for graphs with an irregular communication structure, the communication criterion (**C**) plays an auxiliary role only. During the operating of the MOEG-GS algorithm, the optimization of unbalance (**U** objective) and optimization of the migration number (**M** objective) are of fundamental importance.

4.3 Experimental results summary

Exemplary Pareto front charts indicate that for the applications used in the experimental studies, Euclidean and Manhattan metrics find compromise solutions from Pareto fronts completely distant from each other. This is due to the substantial difference in the cutting curve for these measures. For the Manhattan distance, it is rhomboid, whereas for the Euclidean, it is elliptical. In the examples given, the Manhattan distance prefers solutions with fewer migrations, but with a slightly larger computing node unbalance than the Euclidean distance. The Euclidean metrics prefers solutions with the least possible computing node unbalance.

It should be noted that for given applications the Pareto curve is not convex, thus in general it is not possible to convert one distance metrics to another by changing the weights of individual optimization criteria. Some solutions can never be chosen as a compromise by Manhattan and vice versa: the Euclidean metrics will never choose some solutions as compromise, regardless of the weights used.

In general, this means that both metrics, Manhattan and Euclidean, may be useful for certain types of applications. The research shows that for application graphs with an irregular communication patterns, the Manhattan distance is more effective. It reduces the number of migrations more strongly. It is also consistent with the results for various migration weights, in which the metrics that strongly reduce the number of migrations (ie. Manhattan based MO-1M, MO-2M) give better results than Euclidean when the migration weight increases.

5 CONCLUSIONS

The paper has presented a parallel multi-objective approach applied to Extremal Optimization used in processor load balancing in execution of distributed programs. The presented algorithms are based on EO with Guided Search which improves the convergence of the entire algorithm. In the multi-objective EO approach, three objectives relevant in processor load balancing for distributed applications are simultaneously controlled: total computational load balance, total volume of external communication and the number of task migration. Different global fitness function variants for computational load balancing were designed and verified. The proposed algorithms were assessed by simulation experiments on EO-controlled execution of macro data flow graphs of distributed programs. The experiments have shown that the multi-objective approach added to the EO algorithms for load balancing has improved the quality of obtained results. The paper provides also a more detailed coverage of the internal properties of the proposed multi-objective algorithms including the analysis of the Pareto front itself.

REFERENCES

[1] S. Boettcher, A.G. Percus, Extremal optimization: methods derived from co-evolution, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99), San Francisco, Morgan Kaufmann, 1999, pp. 825-832.

[2] Y.Z. Lu, Y.W. Chen, M.R. Chen, P. Chen, G.Q. Zeng, Extremal Optimization: Fundamentals, Algorithms, and Applications, CRC Press, Chemical Industry Press, 2016, pp. 334.

[3] K. Barker, N. Chrisochoides, An evaluation of a framework for the dynamic load balancing of highly adaptive and irregular parallel applications, Proc. of the ACM/IEEE Conference on Supercomputing, Phoenix, ACM Press, 2003.

[4] I. De Falco, E. Laskowski, R. Olejnik, U. Scafuri, E. Tarantino, M. Tudruj, Load Balancing in Distributed Applications Based on Extremal Optimization, LNCS Vol. 7835, EvoCOMNET 2013, Vienna, April 2013, Springer 2013, pp. 52–61.

[5] I. De Falco, E. Laskowski, R. Olejnik, U. Scafuri, E. Tarantino, M. Tudruj, Improving Extremal Optimization in Load Balancing by Local Search, LNCS Vol. 8602, EvoCOMNET 2014, Granada, April 2014, Springer 2014, pp. 51–62.

[6] I. De Falco, E. Laskowski, R. Olejnik, U. Scafuri, E. Tarantino, M. Tudruj, Extremal Optimization applied to load balancing in execution of distributed programs, Applied Soft Computing, 30 (2015), pp. 501–513.

[7] I. De Falco, E. Laskowski, R. Olejnik, U. Scafuri, E. Tarantino, M. Tudruj, Parallel Extremal Optimization in processor load balancing for distributed applications, Applied Soft Computing, 46 (2016), pp. 187–203.

[8] Ivanoe De Falco, Eryk Laskowski, Richard Olejnik, Umberto Scafuri, Ernesto Tarantino, Marek Tudruj, Multi-objective parallel extremal optimization in processor load balancing for distributed programs, in: Peter A. N. Bosman (eds.): Genetic and Evolutionary Computation Conference (GECCO2017), Berlin, Germany, July 15-19, 2017, Companion Material Proceedings, ACM, pp. 1796–1803

[9] Taxicab geometry, `https://en.wikipedia.org/wiki/Taxicab_geometry` (accessed 20-11-2017).

[10] C. Xu, Francis C. M. Lau, Load balancing in parallel computers: Theory and Practice, Kluwer Academic Publishers, 1997.

[11] R. Z. Khan, J. Ali, Classification of task partitioning and load balancing strategies in distributed parallel computing systems, International Journal of Computer Applications 60 (17) (2012), pp. 48–53.

[12] M. Mishra, S. Agarwal, P. Mishra, S. Singh, Comparative analysis of various evolutionary techniques of load balancing: a review, International Journal of Computer Applications 63 (15) (2013), pp. 8–13.

[13] Tanvi, K. Kaur, A Study on Extremal Optimization Based Load Balancing Techniques, Indian Journal of Computer Science and Engineering, Vol. 8, No. 2, Apr-May 2017, pp. 95-101.

[14] K. Sneppen, et al., Evolution as a self–organized critical phenomenon, Proc. Natl. Acad. Sci. 92 (1995), pp. 5209–5213.

[15] B. Zeigler, Hierarchical, modular discrete–event modelling in an object-oriented environment, Simulation 49 (5) (1987), pp. 219–230.

[16] C. Roig, A. Ripoll, F. Guirado, A new task graph model for mapping message passing applications, IEEE Trans. on Parallel and Distributed Systems 18 (12) (2007), pp. 1740–1753.

[17] Y. Collette, P. Siarry, Multi-objective Optimization: Principles and Case Studies, Springer, 2004, p. 293.

[18] M. Ehrgott, Multi-criteria Optimization, Springer, 2005, p. 324.

[19] C. A. Coello Coello, Evolutionary Multi-Objective Optimization: A Historical View of the Field, IEEE Comp. Intelligence Magazine, Feb. 2006, pp. 28–36.

[20] C. A. Coello Coello, G. B. Lamont, D. A. Van Veldhuizen, Evolutionary Algorithms for Solving Multi-Objective Problems, Springer, 2007, p. 800.

[21] J. D. Knowles, D. W. Corne, Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy, Evolutionary Computation 8(2), MIT Press, 2000, pp. 149-172.

[22] P. L. Yu. A class of solutions for group decision problems. Management Science, 19(8):pp. 936-946, 1973.

[23] C. Busing, K-S. Goetzmann, J. Matuschke, Compromise Solutions in Multicriteria Combinatorial Optimization, Technical Report TU Berlin, 9-2011, pp. 27.

[24] K. Deb, M. Mohan, and S. Mishra, Evaluating the E-Domination Based Multi-Objective Evolutionary Algorithm for a Quick Computation of Pareto-Optimal Solutions, Evolutionary Computation, Vol. 13, Nb. 4, MIT 2005, pp. 501-525.

[25] K. Deb, M. Mohan, S. Mishra, Towards a quick computation of well-spread pareto-optimal solutions. Second Evolutionary Multi-Criterion Optimization Conference, EMO-03, LNCS Vol. 2632, 2003, pp. 222-236.

[26] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, Combining convergence and diversity in evolutionary multi-objective optimization, Evolutionary Computation, 10(3), 2002, pp. 263-282.

[27] M.-R. Chen, Y.-Z. Lu, A novel elitist multi-objective optimization algorithm: multi-objective extremal optimization, Shanghai Jiao Tong University.

[28] E. Ahmed, M. F. Elettreby, On multi-objective evolution model, International Journal of Modern Physics C 2004, 15 (9), pp. 1189–1195.

[29] P. Gómez-Meneses, M. Randall, A. Lewis, A Hybrid Multi-objective Extremal Optimisation Approach for Multi-objective Combinatorial Optimisation Problems, Bond University, Griffith University, Australia, 2010.

[30] R. L. Galski, F. L. de Sousa, F. M. Ramos, I. Muraoka, Spacecraft thermal design with the generalized extremal optimization algorithm, Proceedings of Inverse Problems, Design and Optimization Symposium, Rio de Janeiro, Brazil, 2004.