Flexible Particle Swarm Optimization Tasks for Reconfigurable Processor Arrays

Stefan Janson and Martin Middendorf
Parallel Computing and Complex Systems Group
Department of Computer Science
University of Leipzig
Augustusplatz 10/11, D-04109 Leipzig, Germany
{janson, middendorf}@informatik.uni-leipzig.de.

***Abstract*— Multi task parallel processor arrays are a common machine architecture in which, typically, the tasks running in parallel occupy disjoint subarrays of the machine. On dynamically and partially reconfigurable processor arrays the tasks can be changed during run time. This is useful for online scenarios when the relative importance of tasks might change and therefore the assignment of computational resources to the tasks should be changed. Examples are optimization tasks in an online scenario in which the results of some tasks are needed earlier than expected at initialization. For such tasks the size of their subarrays must be increased because they need more computational resources to speed up. In this paper we design flexible Particle Swarm Optimization (PSO) algorithms for 2-dimensional reconfigurable processor arrays where the algorithms can change their size and have a good optimization behaviour. Since PSO is an iterative, individual-based optimization algorithm that relies upon interactions of neighbouring particles suitable for fine-grained parallel architectures. We propose a dynamic 2-dimensional hierarchical ordering of the particles within a tasks subarray so that the best particles are concentrated in the center. This gives the best particles the strongest influence on the swarm. A further advantage is that size reductions of the tasks can easily be done by cutting off the outer parts of the swarm which contain mainly the less good particles. It is experimentally shown that the proposed algorithms perform better than standard PSO algorithms under conditions with varying supply of computing resources that are available for the tasks. Moreover, also for conditions with constant supply of processing resources and no need for size changes the proposed algorithms perform well.**

I. Introduction

Dynamically and partially reconfigurable processor arrays offer much flexibility for the execution of tasks. For example, the number of computational resources that are assigned to a task can be changed during run time. This can be useful when the relative importance of some tasks changes in online scenarios. For example, the results of some optimization tasks might be needed earlier than expected or control tasks should react to a failure that is indicated by sensor information from the controlled unit. Then the assignment of computational resources to the tasks should be changed. Typically, on reconfigurable processor arrays each task occupies a rectangular subarray and the subarrays of different tasks are disjoint. When a task needs more computational resources, the size of its subarray has to be increased and the subarrays of other tasks might need to be reduced accordingly. Hence, efficient task execution on such reconfigurable processor arrays can profit significantly from flexible tasks which are adaptable in size depending on the needs of other tasks.

In this paper we design flexible Particle Swarm Optimization (PSO) algorithms for 2-dimensional processor arrays that can cope with variable amounts of available computing resources. Thus, we consider the case that the size of the corresponding tasks can be changed externally giving the scheduling algorithm that assigns the tasks to the machine more flexibility to achieve an efficient task layout and better utilization of the computational resources. In particular, we are interested in algorithms that have a good optimization behaviour even when several size reductions and size increases have to be done during run time. The types of size reduction operations that we consider in this paper are removing a row, a column, or a corner of a task.

Size reduction optimization tasks have been studied before in [1]. Contrary to the present study, in [1] the tasks autonomously reduced their size. The idea was to use the convergence state of the considered Ant Colony Optimization algorithms for size reduction. The more an algorithm has converged to a small region of the search space, the more decisions in the solution construction process can be fixed and the less computational resources the algorithm needs. It was shown in [1] that the size reductions can improve the optimization behaviour when tasks are allowed to run repeatedly. Note that this can be profitable because, due to the probabilistic nature of the algorithms, the outcome of different runs might be different.

The PSO metaheuristic for function optimization was introduced by Kennedy and Eberhart in [2]. A PSO algorithm iteratively explores a multidimensional search space with a swarm of individuals, that are referred to as particles. Each particle ”flies” through the search space according to its velocity vector. In every iteration of the algorithm its velocity vector is adjusted so that prior personal successful positions and the best position found by particles within a specific neighbourhood act as attractors. A hierarchical version of PSO has been introduced in [3]. It has been shown that the hierarchical ordering of the particles with respect to their current fitness is advantageous for the optimization behaviour of the algorithm. The hierarchical PSO variant is equally well suited for different types of optimization functions.

In this paper we use a similar ordering among the particles within a task so that good particles have a higher influence on the rest of the swarm. But an additional aspect is to move good particles away from the task boundaries in order to make a possible size reduction of the task easier and to hinder that good particles are removed from the swarm.

II. PSO

A short overview of the PSO metaheurstic for function optimization (see also [2]) is given in this section. PSO is based on the search behaviour of a swarm of m particles in a multidimensional search space. It is an iterative algorithm where in each iteration the velocities and positions of the particles are updated. For each particle i its velocity vector v_i is updated according to (1) where inertia weight $w > 0$ controls the influence of the previous velocity. The current position vector of the particle is denoted by x_i. Parameter $c_1 > 0$ controls the impact of the personal best position so far, stored in vector y_i, i.e. the position where the particle found the smallest function value so far — assuming that the objective function has to be minimized. Parameter c_2 determines the impact of the best position that has been found so far by any of the particles in the respective neighbourhood $\hat{y}_i$. Usually c_1 and c_2 are set to the same value. Random values r_1 and r_2 are drawn with uniform probability from $[0, 1]$.

After velocity update the particles move with their new velocity to their new positions (2). Then for each particle i the objective function f is evaluated at its new position. If $f(x_i(t+1)) < f(y_i)$ the personal best position y_i is updated accordingly, i.e. y_i is set to $x_i(t+1)$.

$$v_i(t+1) = w \cdot v_i(t) + c_1 \cdot r_1 \cdot (y_i - x_i) + c_2 \cdot r_2 \cdot (\hat{y}_i - x_i) \quad (1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2)$$

Several variations of this basic PSO scheme have been proposed. Commonly used variations are to restrict the velocity of a particle by a maximal value v_{max} or to linearly decrease w over time [4]. This is done to adjust the swarm's behaviour from exploration of the entire search space to exploitation of promising regions. In the original PSO algorithm the swarm is guided by the current global best particle, i.e. $\hat{y}_i$ in (1) is the best solution found so far by the swarm. In [5] other neighbourhood topologies, varying the degree of interconnections between the particles, are introduced. A hierarchical version of PSO has been introduced in [3], in which the position of a particle in the hierarchy depends on the quality of its so far best found position. Better particles move up the hierarchy which gives them a higher (indirect) influence on the swarm because the movement of a particle depends on its predecessor in the hierarchy.

In this paper we consider PSO using the local best neighbourhood, called PSO-lbest in the following, because it only relies upon local interaction and is therefore much more suitable to be executed on reconfigurable processor arrays than standard PSO with a global neighbourhood. A particle uses the local neighbourhood best position to update its velocity. The local neighbourhood of PSO-lbest is defined by a 1-dimensional array topology through the particle's index, so that particle i is neighboured to particles $(i+1) \bmod m$ and $(i-1) \bmod m$, where m is the total number of particles.

III. MESH PSO

The fine grained 2-dimensional mesh architecture of the reconfigurable processor arrays that are considered in this paper implies an intuitive mapping of PSO onto this architecture where each processing element (PE) receives one particle. We assume a von Neumann neighbourhood (compare [6]) between all PEs of the array, i.e., each PE is connected to it four neighbours (if they exist).

Before we describe the newly proposed PSO, we consider how to implement the PSO-lbest on the 2-dimensional array. It is implemented by lining up neighboured particles in a snake-like fashion in the task. All the PEs within one row are neighbours and the rows are interconnected alternately at each end, PE $P_{2i,c}$ is neighboured to $P_{2i+1,c}$ and $P_{2i+1,1}$ to $P_{2i+2,1}$ for rows of length c. For the PSO-lbest algorithm the assignment of particle i to PE P_i is never changed during the execution of the task.

In the newly proposed PSO algorithm — called Mesh PSO or M-PSO — the von Neumann neighbourhood topology is utilized within the swarm. An adjacent particle is considered as a potential neighbourhood best particle for the velocity update.

A detailed description of algorithm M-PSO is given in Algorithm 1. The dimension of the search space is dim. The neighbourhood N of a particle are the particles in the neighboured PEs and the communication directions are given by N(orth), S(outh), E(ast) and W(est). A send operation to several of the neighbours will be followed by a recv on the opposite site, both declaring the communication ports that will be used. First, a particle checks if any neighbour wants to have its personal best position y_i and all of these are stored in set A that defines the communication ports used. The d-th component of vector y_i is then sent, along with the computation of the new velocity, so there is no need to store the neighbourhood best position $\hat{y}_i$. Note that every communication channel is used in one direction only, since two particles cannot be the neighbourhood best of each other.

In the following subsection we introduce variants of M-PSO where a hierarchy between the particles is introduced and particles are swapped between processors.

A. Particle labels and swaps

We introduce a labelling of the particles in order to describe the introduction of a hierarchy between the particles. Particles receive labels relative to the position in the task area, with the central particles receiving higher labels. In order to determine the label for each PE, the minimal distance to the task boundaries is calculated. Therefore, all the labels are cleared and then each PE that lies on one of the borders gets assigned the label 1. Consecutively, each PE with a neighbour that

{evaluate current position}
$f(x_i)$ =evaluate(x_i)
if $f(x_i) < f(y_i)$ **then**
  $y_i = x_i$
  $f(y_i) = f(x_i)$
end if
{determine best particle in neighbourhood N}
$lbest = i$ {set self as lbest}
for all $j \in N$ **do**
  if $f(y_j) < f(y_i)$ **then**
    $lbest = j$
  end if
end for
{request best position from lbest}
par
  send(lbest, REQ_Y) {request y from $lbest$}
  A = recv(NSEW, REQ_Y) {particles with i as lbest}
end par
{update velocity}
for $d = 0$ **to** $(dim - 1)$ **do**
  send(A, y_{id})
  $v_{id} = w \cdot v_{id} + c_1 \cdot r_1 \cdot (y_{id} - x_{id}) + c_2 \cdot r_2 \cdot (\text{recv}(lbest) - x_{id})$
end for
{move with current velocity}
for $d = 0$ **to** $(dim - 1)$ **do**
  $x_{id} = x_{id} + v_{id}$
end for

Algorithm 1: The M-PSO algorithm for PE P_i holding particle i

already has a label l receives label $l+1$. This is done until each PE has a label after $\max\{row, col\}$ steps, where row and col are the number of rows and columns of the task, respectively.

During the optimization process, we want to move particles with better personal best values into the center of the swarm to increase their influence upon the swarm, i.e. to decrease the distance to any other particle, and to protect it from being removed by a task size reduction. If the PE holding the neighbourhood best particle $\hat{p}$ has a smaller label than the one holding particle p, a swap operation is initiated. Thus, the label is relevant for determining whether a swap between two adjacent PEs is performed. The labels are updated every time the task is changed, i.e. PEs are added or removed. The swap operations are performed after the particles have evaluated their current positions. In Algorithm 2 a description of the swap procedure is given. In even (odd) numbered iterations the swaps are performed in N, S (respectively E, W) direction.

The swaps are performed every k iterations. Actually, the swaps are performed at iterations ck and $ck + 1$ ($c \in \{1, 2, \ldots\}$), horizontally in even and vertically in odd iterations. The mesh PSO algorithm with swap frequency k is denoted M-PSO-k.

In a variant of the M-PSO-k algorithm (EM-PSO-k) the swap procedure is extended to give special respect to PEs

if $lbest \in$ {N,S} **then**
  if $label_{lbest} < label_i$ **then**
    $s = lbest$
    send(lbest, REQ_SWAP) {request swap with $lbest$}
  end if
end if
s = recv(NS, REQ_SWAP) {one particle requests swap}
if s is set **then**
  swap(s)
end if

Algorithm 2: Swap procedure for even numbered iteration for PE P_i (for odd numbered iterations replace N,S with E,W)

that are in the corners of a certain level (all PEs with the same label). In M-PSO-k the particles in the top/bottom left/right corners cannot get swapped towards the center of the task. They can only influence their neighbours towards their personal best position. To eliminate this possible shortcoming, we permit some swaps to the neighbouring PEs with the same label, see Fig. 1 (left). A PE is considered to be located on such a corner if it has a horizontal and a vertical neighbour of the same label and the PE does not hold the maximum label in the task. The maximum label can be determined in $O(\max\{row, col\})$.

We ensure that a particle can only be the swap candidate of exactly one other particle. Otherwise, the choice of which particle to swap with would become arbitrary. This would occur if there are three adjacent PEs with labels $l, (l-1), l$ and the central particle is the local best for its two neighbours with label l. In the rectangular arrangement this cannot happen because for every PE that is considered for a swap the direction towards the center of the task is unique. But if a corner of the task gets removed, the PEs near the missing corner (Fig. 1 (right), the marked PEs) can be swapped with their horizontal or vertical neighbours. This situation also occurs in the EM-PSO-k algorithm. Therefore, a possible swap is executed alternately horizontally and vertically in odd and even iterations.

If there are several adjacent swaps to be done — particle i is to be swapped with j and j with h — only the outmost swap is performed.

B. Details of the swap operation

Within a swap operation the data representing the two particles involved in the swap have to be exchanged. The required data consists of the current and personal best position, x_i and y_i, their function values $f(x_i)$ and $f(y_i)$ and the current velocity v_i. The PEs remain inactive during the swaps, i.e. they do not continue executing the PSO algorithm and also the contained particle will not be considered by neighbouring particles. The duration of such a swap depends on the available interconnections between the PEs and whether these links are uni- or bidirectional. In order to determine the duration of a swap, we compared the amount of data with the data that is transferred during a regular iteration of the PSO algorithm.

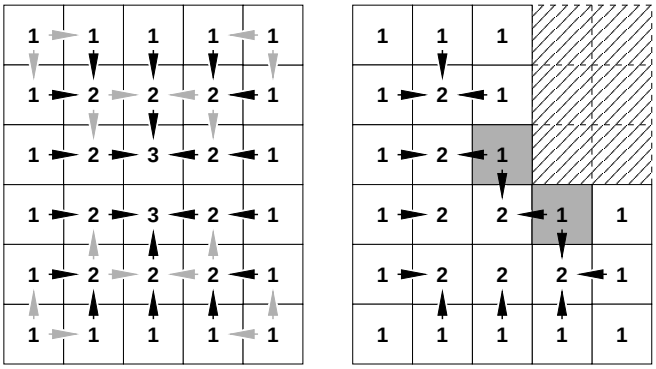

Fig. 1. Left: PE labels and possible swap directions (black arrows) for M-PSO-k, additional grey arrows for EM-PSO-k. Right: M-PSO-k task after the removal of 6 PEs, the marked PEs now have two potential neighbours for a swap.

This is four times the personal best value $f(y_i)$ to all the neighbours and once the personal best position y_i of dimension n. For a swap we have to move positions x_i, y_i and velocity v_i all of dimension n and a function value $f(y_i)$ for each particle. Considering that in a regular algorithm iteration there is also computation to be done, whereas the swap is only concerned with data movement, we estimated the duration for a swap with 5 regular algorithm iterations.

IV. Experiments

In all our experiments the PSO algorithms use parameter values $w = 0.729$ and $c_1 = c_2 = 1.494$ as recommended in [7]. The initial swarm size is $m = 30$, that may change during the algorithm execution when size changes are required. For swarm size 30 the [E]M-PSO-k tasks are of size 6×5 PEs. Each run has been repeated 100 times and average results are shown. The initial positions and velocities of the particles are chosen randomly with uniform distribution from the range $[X_{min}; X_{max}]$. The values X_{min} and X_{max} depend on the objective function (see Tab. II). During a run of an algorithm, the position and velocity of a particle were not restricted to the initialization intervals but a maximum velocity $v_{max} = X_{max}$ was applied for every component d of the velocity vector v_i.

We used a set of common test functions, given in Table I. Table II shows the values used for the dimension of these functions and the range of the corresponding initial position and velocity of the particles. The first two functions (Sphere and Rosenbrock) are unimodal functions (i.e. they have a single local optimum that is also the global optimum) and the remaining four functions are multimodal (i.e. they have several local optima). Every test run was over 10000 iterations.

First, we tested the influence of the swap frequency k of algorithms M-PSO-k and EM-PSO-k on the optimization results for $k \in \{1, 5, 10, 15, \ldots, 50\}$. We also tested algorithm M-PSO that does not perform any swaps. The different algorithm variants were tested on all the given benchmark functions and the average best solution at each iteration was ranked from 1 (best) to 12 (worst) for M-PSO-k and EM-PSO-k (each with 11 different values for k) together with M-PSO. The obtained ranks for the different test functions were then averaged. In this set of experiments no cost was assigned to the swap operations (i.e. we assumed the swap operation does not cost extra time).

In order to illustrate the effect of the different swap strategies, we also counted the number of iterations that each PE $P_{i,j}$ contained the global best particle for M-PSO, M-PSO-1 and EM-PSO-1.

In the second set of experiments we simulated an environment in which the tasks have only a variable amount of resources available. Available PEs are added and removed beyond control of the algorithm. Also, we assigned a cost to performing a swap, by inactivating the two PEs involved in a swap for 5 iterations. In our experiments we considered a setup in which the task sizes are being changed frequently. Every 100 iterations the topmost (or bottommost) row of PEs is removed from the task and after 300 iterations the three removed rows are added again for another 100 iterations. This process is done alternately for top and bottom rows, as shown in Fig. 2. Note that every PE will be removed at some time from the task during this whole process.

TABLE I
Test functions

Sphere: $F_{Sph}(\vec{x}) = \sum_{i=1}^{n} x_i^2$
Rosenbrock: $F_{Ros}(\vec{x}) = \sum_{i=1}^{n-1} \left(100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right)$
Rastrigin: $F_{Ras}(\vec{x}) = \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$
Griewank: $F_{Gri}(\vec{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
Schaffer's f6: $F_{Sch}(\vec{x}) = 0.5 - \frac{\left(\sin\sqrt{x_1^2+x_2^2}\right)^2 - 0.5}{\left(1+0.001(x_1^2+x_2^2)\right)^2}$
Ackley: $F_{Ack}(\vec{x}) = -20 \cdot \exp\left(-0.2\sqrt{1/n \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(1/n \cdot \sum_{i=1}^{n} \cos(2\pi \cdot x_i)\right) + 20 + e$

TABLE II
Parameters for the test functions

Name	Dim	Initial Range
Sphere	30	$[-100; 100]^n$
Rosenbrock	30	$[-30; 30]^n$
Rastrigin	30	$[-5.12; 5.12]^n$
Griewank	30	$[-600; 600]^n$
Schaffer's f6	2	$[-100; 100]^2$
Ackley	30	$[-32; 32]^n$

A. Significance

For evaluating the significance of the test results we used the Wilcoxon Rank Sum Test to pairwise compare the results for the different algorithms. The Wilcoxon test is a distribution free two-sample test. The results of the 100 test runs for

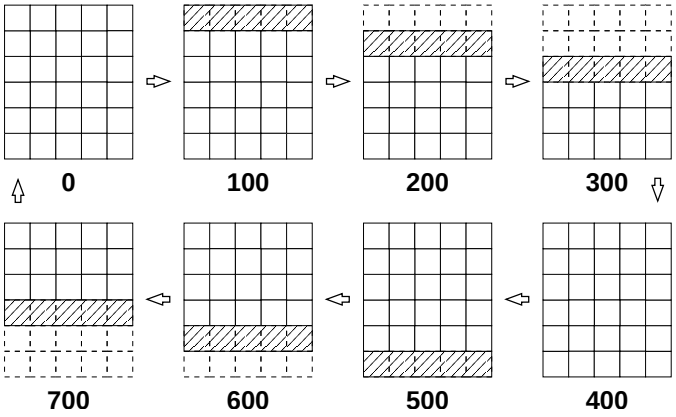

Fig. 2. The task area is changed every 100 iterations; the shaded row is to be removed

two algorithms form two independent samples. For the results values X and Y of the two algorithms their distributions, F_X and F_Y, are compared using the null-hypothesis H_0 : $F_X = F_Y$ and the one-sided alternative $H_1 : F_X < F_Y$. Only if the probability of the null-hypothesis $P(H_0)$ is at most 0.01 it is rejected and the alternative hypothesis is accepted. In the results section the significance comparison among a set of k algorithms is displayed using a $k \times k$ matrix $A = [a_{ij}]_{i,j\in[1:k]}$, where 'X' at position $a_{i,j}$ denotes that algorithm i is significantly better, i.e. provides smaller values, than algorithm j (where i and j are the position in the corresponding result table). For example, the leftmost X in the second row of Table III indicates that algorithm M-PSO is significantly better than PSO-lbest. An entry of '-' at position $a_{i,j}$ indicates that algorithm i is not significantly better than algorithm j. The entries on the main diagonal $a_{i,i}$ are left empty. We call such a matrix a significance matrix (s-matrix).

V. Results

The influence of different swap frequencies k for M-PSO-k and EM-PSO-k are displayed in Fig. 3. Both of the M-PSO-k and EM-PSO-k set of algorithms are compared to the M-PSO algorithm (swap frequency "-" in the figures) that does not perform any swaps at all. The results show that for M-PSO and EM-PSO it is not advantageous to perform swaps in every iteration. The [E]M-PSO-1 algorithms both ranked rather poor compared to algorithms with higher k. Also the M-PSO algorithm was, apart from the beginning ($t = 100$), among the worst ranking algorithms. Increasing the swap frequency up to a certain degree improved the overall ranking. For EM-PSO a value of 20 for k performed best at iterations $t = 5000$ and $t = 10000$. At $t = 5000$ M-PSO-20 also was ranked best, but at step $t = 10000$ higher k values up to 40 ranked slightly better.

The effect of the swapping operations that are used in [E]M-PSO-1 on the location of the particles with global best solution can be seen in Fig. 4. Result for M-PSO are also shown. We counted for each PE $P_{i,j}$ the number of occasions in 10000

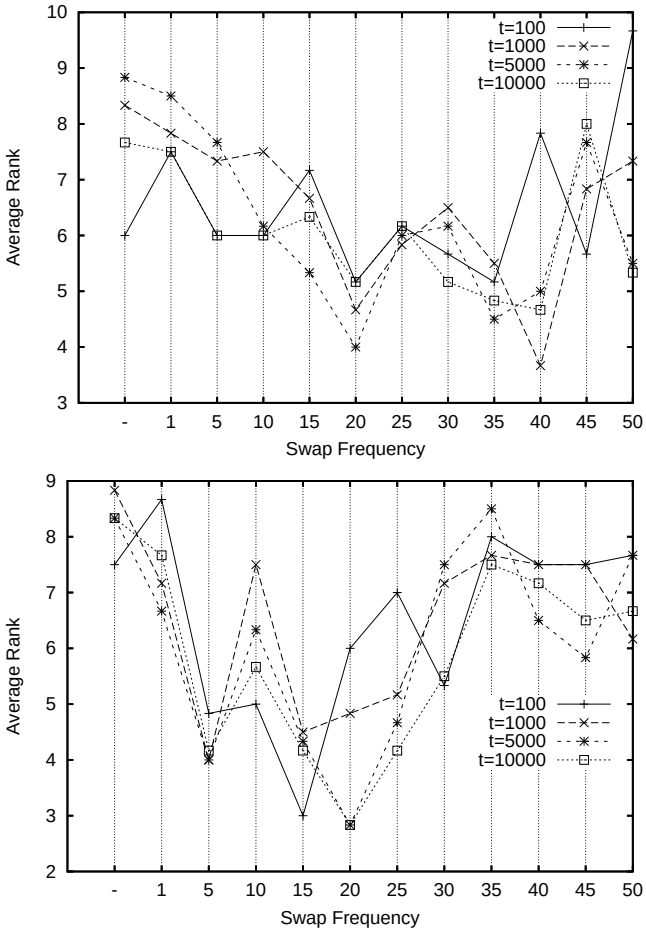

Fig. 3. Average Ranks for M-PSO-k (top) and EM-PSO-k (bottom), "-" is for M-PSO

iterations that the global best particle was positioned in this PE. The average of 100 runs for the Rastrigin function is displayed for M-PSO, M-PSO-1 and EM-PSO-1. For the M-PSO algorithm that does not perform any swaps between PEs the global best particle is very evenly distributed among the PEs. The M-PSO-1 algorithm clearly shifts the best particle towards the center of the task. The two PEs $P_{3,3}$ and $P_{4,3}$ with label 3 (compare Fig. 1) contain the best particle more often than other PEs. The PEs on the corners of a certain level, from which good particles cannot be swapped away, also have higher than average number of times the global optimum. This is changed in the EM-PSO algorithm, in which the PEs $P_{3,3}$ and $P_{4,3}$ clearly dominate and contain the global best particle most of the time.

In the following, the results of the experiments in which size changes were required are presented. We compared the algorithms PSO-lbest and M-PSO, which do not perform any swaps, to the algorithms [E]M-PSO-1 and [E]M-PSO-20. The average solution quality for each of the test functions is displayed in Fig. 5 – Fig. 10. In Table III the average solution quality at iteration $t = 10000$, the average number of occasions that the global best particle was removed, the average number

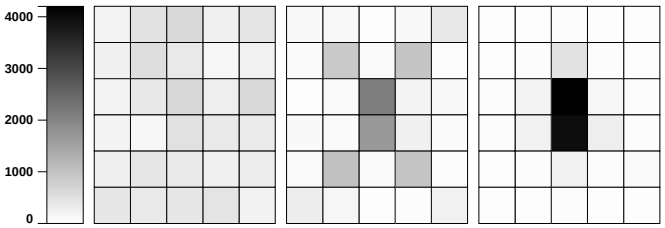

Fig. 4. Number of iterations that a PE contained the gbest particle for the Rastrigin function measured over 10000 iterations; M-PSO, M-PSO-1 and EM-PSO-1 (from left to right)

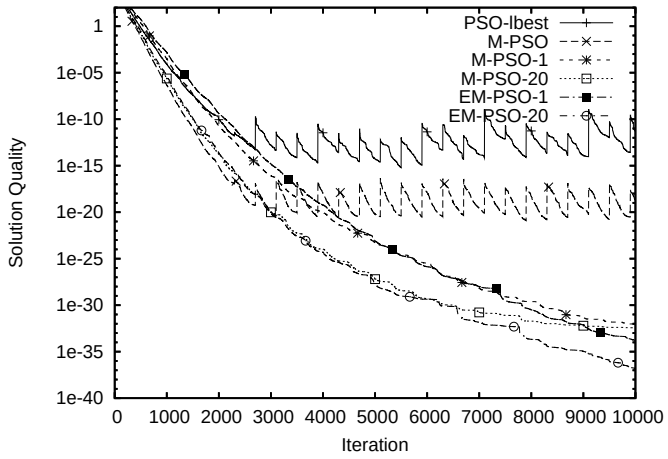

Fig. 5. Sphere — Average Solution Quality of PSO-lbest, M-PSO and [E]M-PSO-k variants.

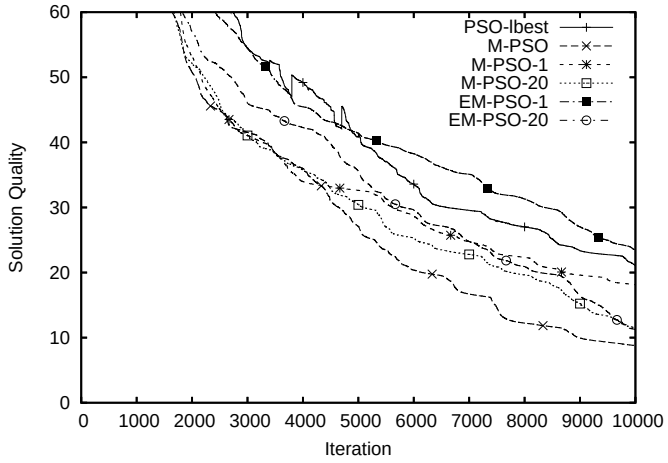

Fig. 6. Rosenbrock — Average Solution Quality of PSO-lbest, M-PSO and [E]M-PSO-k variants.

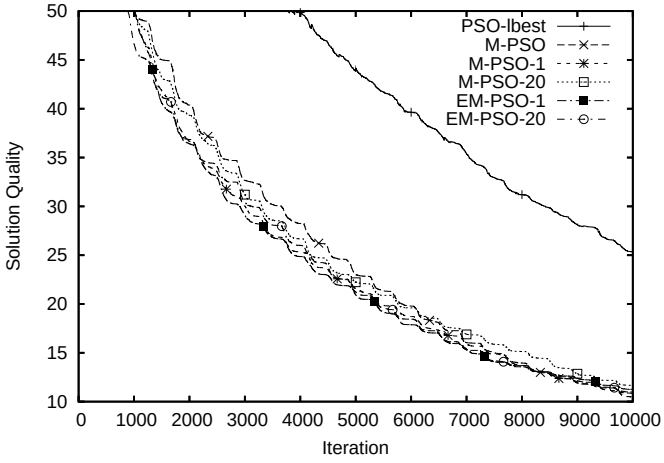

Fig. 7. Rastrigin — Average Solution Quality of PSO-lbest, M-PSO and [E]M-PSO-k variants.

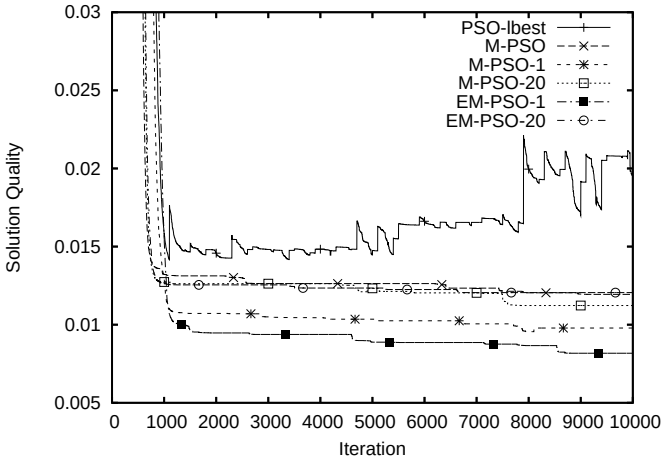

Fig. 8. Griewank — Average Solution Quality of PSO-lbest, M-PSO and [E]M-PSO-k variants.

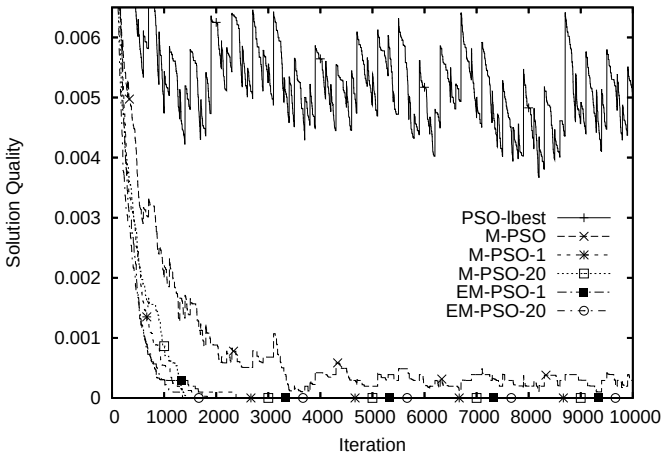

Fig. 9. Schaffer — Average Solution Quality of PSO-lbest, M-PSO and [E]M-PSO-k variants.

TABLE III

AVERAGE SOLUTION QUALITY AT ITERATION 10000 (AVG), THE AVERAGE NUMBER OF OCCASIONS THAT THE GBEST PARTICLE WAS REMOVED (REM), THE AVERAGE NUMBER OF SWAPS (SWAPS), AND THE SIGNIFICANCE MATRIX (S-MATRIX) FOR THE FINAL SOLUTION QUALITY.

Algorithm	**Avg**	**Rem**	**Swaps**	**S-Matrix**
	Ackley			
PSO-lbest	3.33e-07	32.02	-	- - - - -
M-PSO	9.74e-11	38.44	-	X - - - -
M-PSO-1	1.13e-14	14.11	1375	X X - X -
M-PSO-20	1.02e-14	18.69	439	X X X - X -
EM-PSO-1	1.25e-14	4.29	2393	X X - - -
EM-PSO-20	1.05e-14	14.27	825	X X X - X
	Griewank			
PSO-lbest	0.0195	28.8	-	- - - - -
M-PSO	0.0119	33.29	-	X - - - -
M-PSO-1	0.0098	17.66	1039	X X - - -
M-PSO-20	0.0112	20.27	360	X - - - -
EM-PSO-1	0.0082	10.02	1775	X X - - -
EM-PSO-20	0.0121	15.91	679	X - - - -
	Rastrigin			
PSO-lbest	25.3506	18.01	-	- - - - -
M-PSO	10.8400	22.94	-	X - - - -
M-PSO-1	10.4339	14.26	2174	X - - - -
M-PSO-20	11.6613	18.21	694	X - - - -
EM-PSO-1	11.2534	4.3	3430	X - - - -
EM-PSO-20	10.9875	16.57	1177	X - - - -
	Rosenbrock			
PSO-lbest	21.0293	26.15	-	- - - - -
M-PSO	8.7837	19.4	-	X - X - X -
M-PSO-1	18.1445	14.79	2965	- - - - - -
M-PSO-20	11.5377	17.66	858	X - X - X -
EM-PSO-1	23.3857	4.97	4399	- - - - -
EM-PSO-20	11.2348	16.69	1439	X - X - X
	Schaffer			
PSO-lbest	0.0049	29.58	-	- - - - -
M-PSO	2.92e-04	29.46	-	X - - - -
M-PSO-1	0	16.79	418	X X - - -
M-PSO-20	0	15.95	274	X X - - -
EM-PSO-1	0	11.22	883	X X - - -
EM-PSO-20	0	12.93	555	X X - - -
	Sphere			
PSO-lbest	4.61e-12	32.16	-	- - - - -
M-PSO	1.81e-19	39.67	-	X - - - -
M-PSO-1	9.38e-33	11.83	1526	X X - - -
M-PSO-20	3.68e-33	14.93	483	X X X - -
EM-PSO-1	1.61e-34	4.32	3161	X X X X -
EM-PSO-20	1.57e-37	12.79	983	X X X X X

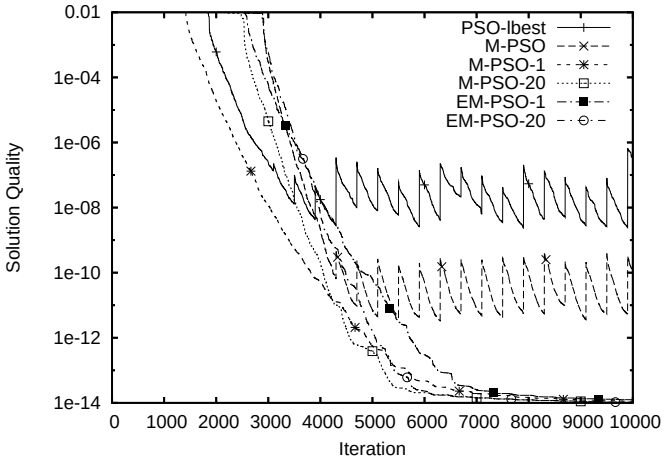

Fig. 10. Ackley — Average Solution Quality of PSO-lbest, M-PSO and [E]M-PSO-k variants.

of performed swaps and the significance matrix for the final solution quality are given.

All the [E]M-PSO[-k] algorithms performed significantly better than PSO-lbest on all the test functions, except for the Rosenbrock function, for which PSO-lbest was only the second worst algorithm. This is not only due to the lack of swaps in PSO-lbest, since the M-PSO algorithm which also does not use any swaps but uses the mesh neighbourhood achieved a significantly better result on all test functions. Hence, the results also show that the PSOs with mesh neighbourhood are well suited for the 2-dimensional processor arrays.

Note that the [E]M-PSO-k algorithms that perform swaps use less function evaluations than the PSO-lbest or the M-PSO algorithms. For example, with 1000 swaps during the task execution, 2 PEs are disabled for 5 iterations 1000 times. This results in 10000 evaluations less. This might explain why for the Rosenbrock function the M-PSO algorithms that execute few or no swaps performed significantly better than [E]M-PSO-1 which performed more than 4000 swaps (which is the highest number of swaps observed in all tests).

Since the tasks were executed with variable swarm sizes, the total number of function evaluations is smaller than the 300000 evaluations of a regular PSO algorithm of size $m = 30$ executing for 10000 iterations. The tasks consist of 4 rows of 5 PEs each on average during the reduced phase of 300 iterations and of 6 rows during the remaining 100 iterations. Therefore, on average 4.5 rows, i.e. 22.5 PEs, are active. Thus, the total number of evaluations is 25% less than for a task with constant size $m = 30$. Still, the achieved solution quality is competitive with results of PSO with $m = 30$. The frequent re-initialization of parts of the swarm turned out to be beneficial for the optimization behaviour of the algorithm.

The curves in Fig. 5 – Fig. 10 show (to a different extent) that the solution quality of algorithms PSO-lbest and M-PSO became worse about every 400 iterations (after the algorithms have been executed for about 1000 - 3000 iterations). This occurs when the inner rows 3 or 4 are removed. The PEs

within these rows are part of the algorithm for the largest number of iterations and thus they have the highest chance of containing the global best particle. Hence, removing these rows has the most detrimental effect on the solution quality.

Overall, it can be said that the algorithms [E]M-PSO-k perform better than the algorithms without swaps. It seems that it is in general advantageous not to perform swaps at every iteration (but this might depend on the optimzation function). Also, algorithm EM-PSO-k seems to be better than algorithm M-PSO-k and a possible reason is that the number of occasions when the global best particle is lost is in all cases smaller for the former algorithm. So, the last effect outweighs the disadvantage of EM-PSO-k that in all cases it did more swaps than M-PSO-k and therefore did fewer function evaluations.

VI. Conclusion

In this paper we have proposed Particle Swarm Optimization (PSO) algorithms that are suitable for reconfigurable processor arrays where the size of the corresponding tasks can be changed externally. It was argued that use of size variable tasks is important for the efficient execution of tasks on dynamic reconfigurable processor arrays. The main principles that were used in the proposed PSO algorithms were i) to use a mesh neighbourhood and ii) to perform swap operations that are based on the quality of the individuals with the aim to give the better particles a higher influence on the swarm and to concentrate them in the middle of the subarray of their task. It was shown experimentally that the proposed algorithms perform significantly better than standard PSO with the lbest neighbourhood. Moreover, it was shown that the use of swaps significantly improves the optimization behaviour in dynamic environments in which size changes occur. This holds even when the cost of swap operations leads to reduced number of function evaluations that can be performed.

Acknowledgment

This work was supported by the German Research Foundation (DFG) through the project ”Swarm Intelligence on Reconfigurable Architectures”.

References

[1] S. Janson and M. Middendorf, “A Hierarchical Particle Swarm Optimizer and its Adaptive Variant,” Accepted subject to minor revision for *IEEE Transactions on Systems, Man and Cybernetics, Part B*, preliminary version: Proc. Congress on Evolutionary Computation (CEC2003), IEEE Press, 770–776, (2003).

[2] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in *IEEE International Conference on Neural Networks (ICNN’95)*, vol. 4. Perth, Western Australia: IEEE, Nov.-Dec. 1995, pp. 1942–1947.

[3] S. Janson, D. Merkle, M. Middendorf, H. ElGindy, and H. Schmeck, “On Enforced Convergence of ACO and its Implementation on the Reconfigurable Mesh Architecture Using Size Reduction Tasks,” *The Journal of Supercomputing*, vol. 26, no. 3, pp. 221–238, Nov. 2003.

[4] Y. Shi and R. C. Eberhart, “Parameter selection in particle swarm optimization,” *Lecture Notes in Computer Science*, vol. 1447, pp. 591–600, 1998.

[5] J. Kennedy, “Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance,” in *Proceedings of the Congress on Evolutionary Computation*, P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, Eds., vol. 3. Mayflower Hotel, Washington D.C., USA: IEEE Press, 6-9 July 1999, pp. 1931–1938.

[6] J. Kennedy and R. Mendes, “Population structure and particle swarm performance,” in *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, D. B. Fogel, M. A. El-Sharkawi, X. Yao, G. Greenwood, H. Iba, P. Marrow, and M. Shackleton, Eds. IEEE Press, 2002, pp. 1671–1676.

[7] I. C. Trelea, “The particle swarm optimization algorithm: Convergence analysis and parameter selection,” *IPL: Information Processing Letters*, vol. 85, 2003.