Applied Soft Computing 13 (2013) 4481–4492

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A parallel hybrid optimization algorithm for fitting interatomic potentials

C. Voglis[a,*], P.E. Hadjidoukas[b], D.G. Papageorgiou[c], I.E. Lagaris[a]

[a] Department of Computer Science, University of Ioannina, P.O. Box 1186, GR-45110 Ioannina, Greece
[b] Professorship for Computational Science, ETH Zurich, Zurich CH-8092, Switzerland
[c] Department of Materials Science and Engineering, University of Ioannina, P.O. Box 1186, GR-45110 Ioannina, Greece

ARTICLE INFO

Article history:
Received 23 September 2012
Received in revised form 30 June 2013
Accepted 11 August 2013
Available online 5 September 2013

Keywords:
Interatomic potential
Hybrid global optimization
Particle swarm
Multidimensional search
Irregular task parallelism
Cluster programming

ABSTRACT

In this work we present the parallel implementation of a hybrid global optimization algorithm assembled specifically to tackle a class of time consuming interatomic potential fitting problems. The resulting objective function is characterized by large and varying execution times, discontinuity and lack of derivative information. The presented global optimization algorithm corresponds to an irregular, two-level execution task graph where tasks are spawned dynamically. We use the OpenMP tasking model to express the inherent parallelism of the algorithm on shared-memory systems and a runtime library which implements the execution environment for adaptive task-based parallelism on multicore clusters. We describe in detail the hybrid global optimization algorithm and various parallel implementation issues. The proposed methodology is then applied to a specific instance of the interatomic potential fitting problem for the metal titanium. Extensive numerical experiments indicate that the proposed algorithm achieves the best parallel performance. In addition, its serial implementation performs well and therefore can also be used as a general purpose optimization algorithm.

1. Introduction

Computer simulation at the atomic level is a valuable tool in several scientific disciplines such as physics, chemistry and materials science. It has been successfully used for the understanding of various phenomena, the interpretation of experimental results and the prediction of physical properties. Although first principles methods have advanced considerably in speed and accuracy during the last decades, computations based on classical interatomic potentials is still the only feasible way for molecular dynamics or Monte Carlo simulations for systems with a very large number of atoms or for long simulation times.

A variety of interatomic potentials for metals is available in the literature such as the Embedded Atom Method, the Modified Embedded Atom Method, the Finnis–Sinclair potential, and potentials based on the Second Moment Approximation of the Tight Binding theory (SMATB). For a review of the various potential models for metals and alloys see [1]. Here we describe the procedure for obtaining an interatomic potential for the metal titanium, using the SMATB formalism. Computing an interatomic potential reduces into a data fitting problem where the potential model is fitted against experimentally known physical properties. The resulting optimization problem is characterized by small dimensionality, large and varying runtime, discontinuities in the objective function and lack of derivative information.

Recently the tremendous growth of parallel computing systems and programming techniques sparked the research on parallel global optimization algorithms, specifically for molecular biology and computational chemistry problems. The main reason is that the optimization of such complex systems involves many numerical calculations and a single function evaluation may be computationally expensive. In [2] the authors calculate minimum energy clusters of atoms using a parallel build-up algorithm based on simulated annealing. A similar problem is tackled in [3], where the authors present a parallel algorithm based on local searches. In [4,5] a parallel algorithm is proposed for the three-dimensional protein conformation problem. Both algorithms apply local searches inside a multistart framework, where random points are uniformly sampled in order to explore the domain. Parallel schemes involving a Particle Swarm Optimization (PSO) [6] variant and the Nelder–Mead Simplex method [7] were presented in [8]. In that article parallelization is achieved by using two separate processes that perform function evaluations and local search applications, upon request of a main process. In [9] the authors present a hybrid between a Genetic Algorithm (GA) and PSO. The

* Corresponding author. Tel.: +30 6977053095.
E-mail addresses: voglis@cs.uoi.gr (C. Voglis), phadjido@mavt.ethz.ch (P.E. Hadjidoukas), dpapageo@cc.uoi.gr (D.G. Papageorgiou), lagaris@cs.uoi.gr (I.E. Lagaris).

1568-4946/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2013.08.007

Table 1
The physical properties (y_i) used in the fit of the interatomic potential and the corresponding calculated values. Experimental lattice constants a, c/a ratio and atomic volumes (Ω_0) are from [15], cohesive energies (E_c) are from [16] and elastic constants (C_{11}, C_{12}, C_{13}, C_{33}, C_{44}) are from [17]. The vacancy formation energy (E_{vf}) is from reference [18]. a is given in Å, c/a is dimensionless, Ω_0 is in Å^3, E_c and E_{vf} are in eV while the elastic constants are given in eV/Å^3.

	a	c/a	Ω_0	E_c	C_{11}	C_{12}	C_{13}	C_{33}	C_{44}	E_{vf}
Calc.	3.1634	1.62818	22.32	6.44	1.238	0.498	0.406	1.350	0.277	2.63
Exp.	3.1946	1.58114	22.32	6.44	1.187	0.465	0.409	1.276	0.374	2.15

hybridization is achieved by assigning the best half of the population to GA iterations and the worst half to PSO iterations. Since no local optimization algorithm is involved the parallelization is quite straightforward using a master–slave model. In the same spirit the authors in [10] present a hybridization of a tabu search and PSO that utilizes the analogy between the algorithm and the master–slave paradigm.

In this work we present a hybrid Global Optimization (GO) algorithm designed to solve interatomic potential fitting problems that follow the SMATB formalism. The algorithm under study falls into the category of *memetic algorithms* and combines a Particle Swarm Optimization (PSO) variant [11] (global component) with Multidimensional Search (MDS) [12] (local component). The efficiency of this hybridization scheme was demonstrated in [13,14]. We decided to incorporate MDS since it is suitable for discontinuous functions and a perfect candidate for parallelization. Our basic incentive is to create a powerful hybrid optimization scheme with parallelizable components so that we can handle the large runtime of the potential fitting problem at hand.

We describe in detail the parallelization capabilities and implementation details of our algorithm, using a task-based programming and runtime environment. Here, two levels of parallelism are being exploited in order to accelerate the method. The first parallelism issue is raised by the stochastic and irregular nature of task spawning. The GO algorithm is an iterative procedure but since we cannot foresee how many computational tasks may emerge at each iteration, the solution must be adaptive on the specific instance of the algorithm. The second issue comes from the fitting problem at hand. The interatomic potential involves many complex calculations and hence the execution time of the basic task varies greatly. By means of the recent OpenMP tasking model, we effectively tackle these issues on shared-memory systems. In order to extend this work on distributed memory cluster environments, we introduce a runtime system that implements a programming and execution environment for irregular and adaptive task-based parallelism on multicore clusters. This approach allows for a parallel implementation of the hybrid GO algorithm that runs efficiently on both shared and distributed memory architectures.

The rest of this paper is organized as follows: Section 2 describes in detail the fitting procedure of the interatomic potential and introduces the objective function to be minimized. Section 3 gives a description of the hybrid GO algorithm. Section 4 discusses the parallelism issues of the hybrid GO algorithm and outlines the runtime system that will support it. In Section 5 we present an experimental evaluation of the proposed scheme. Finally, we conclude with a discussion in Section 6.

2. Description of the interatomic potential

In the SMATB formalism the total potential energy E_{pot} of a system consisting of N_a interacting atoms is:

$$E_{pot} = \sum_{i=1}^{N_a} E_i$$

Table 2
Interatomic potential objective function settings.

Dimension n:	4
Search space X:	$p \in [6.00, 15.0]$ $q \in [1.20, 2.50]$ $A \in [0.05, 0.20]$ $\xi \in [1.20, 2.50]$
Best solution found in X:	$(p, q, A, \xi) = (10.6784, 2.0884, 0.0992, 1.5917)$

where the partial energies E_i are given by:

$$E_i = A \sum_{\substack{j=1 \\ j \neq i}}^{N_a} e^{-p((r_{ij}/r_0)-1)} - \xi \sqrt{\sum_{\substack{j=1 \\ j \neq i}}^{N_a} e^{-2q\left(\frac{r_{ij}}{r_0} - 1\right)}}$$

In this expression r_{ij} is the distance between atoms i and j while r_0 is set to the nearest neighbor distance ($r_0 = 3.1946$Å in the case of Titanium). There are four adjustable parameters p, q, A and ξ that need to be determined using a least squares fitting procedure.

For the fitting a set of $M = 10$ properties y_i is chosen, whose values are experimentally known and the error function that determines the agreement with the model is defined as:

$$f(p, q, A, \xi) = \sum_{i=1}^{M} w_i \left(\frac{y_i^{calc} - y_i^{exp}}{y_i^{exp}} \right)^2$$

Here, w_i are weights whose values reflect the importance of the property being reproduced. For the set of properties we have used the lattice parameter a, c/a ratio, atomic volume Ω_0, cohesive energy E_c, five elastic constants and the vacancy formation energy E_{vf}. We have opted to reproduce the atomic volume and cohesive energy as closely as possible by assigning appropriate weights. All properties y_i^{calc} are calculated using numerical methods. More specifically the lattice parameter a, and the c/a ratio are determined as the values that provide minimum energy in an HCP titanium lattice. The five elastic constants are determined by the method described in [19]. Finally the vacancy formation energy is determined by removing an atom from a perfect HCP lattice and allowing all neighboring atoms to relax. A numerical optimization procedure is then applied to the error function in order to determine the set of parameters that produce the best agreement with the experimental quantities.

The error function $f(p, q, A, \xi)$ has some important characteristics: Since all properties are numerically calculated the error function is quite noisy, has discontinuities and in addition there exist no analytic derivatives with respect to the parameters p, q, A and ξ. Hence application of numerical optimization methods that utilize analytic or numerically calculated first derivatives are not applicable. Table 1 summarizes the values of the properties used in the fit, while Table 2 provides information for the error function.

3. The hybrid global optimization algorithm

The hybrid GO algorithm explored in this paper, belongs to the class of hybrid meta-heuristics that combine population-based

optimization algorithms with local search procedures [20,21,13]. The rationale behind their development was the necessity for powerful algorithms where the global exploration capability of population-based approaches (exploration phase) would be complemented with the efficiency and accuracy of classical local optimization techniques (exploitation phase). The most common cooperation scheme keeps the main exploratory part of a population-based strategy and stochastically selects a subset of individuals on which a local search is applied.

Population-based algorithms draw their inspiration from physical systems. Based on natural selection they exhibit a remarkable capability of producing populations with fitter individuals. Genetic algorithms [22], particle swarm optimization (PSO) [6], differential evolution [23], harmony search [24], all fall into the category of population-based algorithms for global optimization.

On the other hand local search (LS) procedures are deterministic and iterative procedures that aim to converge to a local minimizer of a given objective function, using mostly local information (function values). They enjoy guaranteed convergence but provide no assurance that the best minimum will be found.

In this work, we have chosen PSO for the population-based exploratory part and Multidimensional Search (MDS) [12] for the exploitation part. The high degree of parallelization capabilities of both algorithms renders them perfect candidates for our parallel hybrid algorithm. PSO popularity has been gradually increased due to implementation simplicity that made it accessible to researchers in diverse fields, as well as due to the increased efficiency of recent variants. Today, PSO has been distinguished as one of the most promising population-based approach. Moreover it does not require smooth functions and it is robust to noise. Following the same reasoning: (a) easy implementation, (b) natural parallelization, (c) no requirements of derivative information and (d) robustness to noise, the Multidimensional Search algorithm is chosen.

Many authors in the past used the same reasoning to combine two optimization methods together [25–27]. We assume the n-dimensional continuous optimization problem:

$$\min_{x \in X \subset \mathbb{R}^n} f(x), \tag{1}$$

where the search space X is an orthogonal hyperbox in $\mathbb{R}^n$:

$$X \equiv [l_1, r_1] \times [l_2, r_2] \times \cdots \times [l_n, r_n].$$

Some of the algorithms presented in this section are described thoroughly in [13,14].

3.1. Particle swarm optimization

The PSO algorithm was introduced by Eberhart and Kennedy [6,28]. The main concept of the method includes a population, also called *swarm*, of search points, also called *particles*, searching for optimal solutions within the search space, concurrently. The particles move in the search space by assuming an adaptable position shift, called *velocity*, at each iteration.

A swarm of N particles is a set of search points:

$$S = \{x_1, x_2, \ldots, x_N\},$$

where the i-th particle is defined as:

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{in})^\top \in X, \quad i = 1, 2, \ldots, N.$$

The velocity (position shift) of x_i is denoted as:

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{in})^\top, \quad i = 1, 2, \ldots, N,$$

and its best position as:

$$p_i = (p_{i1}, p_{i2}, \ldots, p_{in})^\top \in X, \quad i = 1, 2, \ldots, N.$$

Let g_i denote the particle which attained the lowest function value so far i.e.:

$$g_i = arg \min_{j \in 1 \ldots N} f(p_j),$$

and t the algorithm's iteration counter. Then, the particle positions and velocities are updated at each iteration according to the equations [29]:

$$v_{ij}^{(t+1)} = \chi \left[v_{ij}^{(t)} + c_1 r_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + c_2 r_2 \left(p_{gj}^{(t)} - x_{ij}^{(t)} \right) \right], \tag{2}$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)}, \qquad i = 1, 2, \ldots, N,\ j = 1, 2, \ldots, n, \tag{3}$$

where χ is the *constriction coefficient*; c_1 and c_2 are positive constants called *cognitive* and *social* parameter, respectively; r_1, r_2, are random numbers drawn from a uniform distribution in the range $[0, 1]$. The best position of each particle is updated at each iteration:

$$p_{ij}^{(t+1)} = \begin{cases} x_{ij}^{(t+1)}, & \text{if } f(x_i^{(t+1)}) < f(p_i^{(t)}), \\ p_{ij}^{(t)}, & \text{otherwise.} \end{cases} \tag{4}$$

The function evaluations $f(x_i^{(t+1)})$ used for the update of best position (Eq. (4)) can be performed independently for all members of the swarm.

3.2. MDS algorithm

In early 90s, Torczon [12] introduced a novel local optimization algorithm, called Multidimensional Search (MDS), that operates on a simplex of points defined in the search space. The proposed deterministic algorithm uses a sequence of reflections, expansions and contractions of a simplex to guaranty convergence to a local minimum. It was devised to operate without any derivative information for the objective function, using function evaluations that can be executed concurrently.

At any iteration t an n-dimensional simplex

$$S^{(t)} = \{x_0^{(t)}, x_1^{(t)}, \ldots x_n^{(t)}\}, \qquad x_i^{(t)} \in \mathbb{R}^n$$

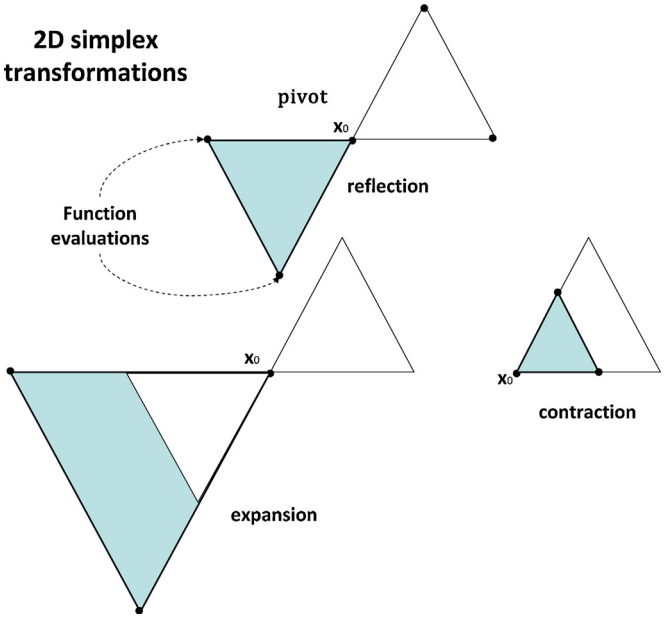


Fig. 1. MDS simplex transformations.

is maintained. The center of the simplex

$$x_c^{(t)} = \frac{1}{n}\sum_{i=0}^{n} x_i^{(t)}$$

is considered as an approximation to the minimum. The objective function is evaluated at n vertices of the simplex and the *best* vertex is defined as the one with the smallest function value. The vertices are rearranged so that

$$f(x_0^{(t)}) = \min_{i=0,\ldots,n} f(x_i^{(t)})$$

The transition to the next iteration is performed by pivoting the simplex around $x_0^{(t)}$ and attempting a reflection. The objective function is then evaluated at the n vertices of the reflected simplex. If the function value decreases, then an expansion step is attempted, in order to produce an even larger reflected simplex. If the reflection fails to produce a smaller function value (than the one of the pivot) then a contraction step that reduces the size of the simplex is attempted. The simplex transformations are illustrated in Fig. 1. The procedure is repeated until a termination criterion is satisfied. This criterion may be the maximum number of iterations, function evaluations or the detection of no further progress in one iteration. The overall MDS procedure is presented in Algorithm 1. It is clear that the function evaluations in lines 7, 12 and 22 are independent and can be performed in parallel.

Algorithm 1. MDS algorithm

Input: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; initial point $x^{(0)}$; parameters $\mu \in (1, +\infty)$, $\theta \in (0, 1)$
Output: Approximation to local minimizer: x^*

1 Create $S^{(0)} = \{x_0^{(0)}, x_1^{(0)}, \ldots x_n^{(0)}\}$ that contain $x^{(0)}$
2 $min \leftarrow \arg\min_i \{f(x_i^{(0)})\}$ and swap $x_{min}^{(0)}$ and $x_0^{(0)}$
3 **for** $t = 0, 1, \ldots$ **do**
4 Check the stopping criterion
  `// Reflection step`
5 **for** $i = 1, \ldots, n$ **do**
6 $r_i^t \leftarrow 2x_0^{(t)} - x_i^{(t)}$
7 Calculate $f(r_i^t)$
8 **end**
9 **if** $\min_i \left\{f(r_i^t), i = 1, \ldots, n\right\} < f(x_0^{(t)})$ **then**
  `// Expansion step`
10 **for** $i = 1, \ldots, n$ **do**
11 $e_i^t \leftarrow (1 - \mu)x_0^{(t)} + \mu r_i^{(t)}$
12 Calculate $f(e_i^t)$
13 **end**
14 **if** $\min_i \left\{f(e_i^t), i = 1, \ldots, n\right\} < \min_i \left\{f(r_i^t), i = 1, \ldots, n\right\}$ **then**
  `// Expansion step accepted`
15 $x_i^{(t+1)} \leftarrow e_i^t,\ i = 1, \ldots, n$
16 **else**
  `// Reflection step accepted`
17 $x_i^{(t+1)} \leftarrow r_i^t,\ i = 1, \ldots, n$
18 **end**
19 **else**
  `// Contraction step`
20 **for** $i = 1, \ldots, n$ **do**
21 $x_i^{(t+1)} \leftarrow (1 + \theta)x_0^{(t)} - \theta r_i^{(t)}$
22 Calculate $f(x_i^{(t+1)})$
23 **end**
24 **end**
25 $min \leftarrow \arg\min_i \{f(x_i^{(t+1)})\}$ and swap $x_{min}^{(t+1)}$ and $x_0^{(t+1)}$
26 **end**
27 $x^* \leftarrow \frac{1}{n}\sum_{i=0}^{n} x_i^{(t)}$

3.3. Hybrid framework

The design of the hybrid global optimization algorithm is based on [13] and provides answers to the following questions: (a) *Where* should the LS procedures be applied, (b) *When* should the LS procedures be applied, (c) *How much of the budget* should the LS procedures spend. The cooperation strategy chosen for this work states:

At each PSO iteration (when), a fixed budget (how much) LS can be applied on each best position vector, p_i, $i = 1, 2, \ldots, N$, with a prescribed fixed probability, $\rho \in (0, 1]$ (where). The overall best should always be included as a local search candidate.

Of course, many other strategies can be considered. The one above is easy to implement and resulted very good performance in [13] and in [14,30,31]. The general procedure of the hybrid algorithm discussed in our study is given in Algorithm 2.

Algorithm 2. Hybrid GO algorithm

Input: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; swarm size: N; probability for local search: ρ
Output: Best detected solution: x^*, $f(x^*)$.

// Initialization
for $i = 1, 2, \ldots, N$ **do**
  Initialize $x_i^{(0)}$ and $v_i^{(0)}$
  Set $p_i^{(0)} \leftarrow x_i^{(0)}$ // Initialize best position
  $f_i^{(0)} \leftarrow f(x_i^{(0)}$ // Evaluate particle
  $f_{p_i}^{(0)} \leftarrow f_i^{(0)}$ // Best position value
  $act_i^{(0)} \leftarrow 0$ // By default all particles perform FEs
end
// Main Iteration Loop
Set $t \leftarrow 0$
while *(termination criterion)* **do**
  // Determine which particles will apply MDS to their best position
  for $i = 1, 2, \ldots, N$ **do**
    if $rand() < \rho$ **then**
      $act_i^{(t)} \leftarrow 1$ // The i-th particle will perform MDS
    else
      $act_i^{(t)} \leftarrow 0$ // The i-th particle will perform FE
    end
  end
  // Update Best Indices
  Calculate global best index $g_i^{(t)}$
  // Update Swarm/Population
  for $i = 1, 2, \ldots, N$ **do**
    for $j = 1, 2, \ldots, n$ **do**
      $v_{ij}^{(t+1)} = \chi \left[v_{ij}^{(t)} + c_1 r_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + c_2 r_2 \left(p_{g_i j}^{(t)} - x_{ij}^{(t)} \right) \right]$
      $x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)}$
    end
  end
  // Evaluate Population or Apply Local Search
  for $i = 1, 2, \ldots, N$ **do**
    if $act_i^{(t)} = 0$ **then**
      $f_i^{(t+1)} \leftarrow f\left(x_i^{(t+1)} \right)$ // Perform FE
    else
      $[p_i^{(t+1)}, f_{p_i}^{(t+1)}, f_i^{(t+1)}] \leftarrow MDS(p_i^{(t)}, \mu, \theta)$ // Perform MDS
    end
  end
  // Update Best Positions/Individuals
  for $i = 1, 2, \ldots, N$ **do**
    if $f_i^{(t+1)} < f\left(p_i^{(t+1)} \right)$ **then**
      $p_i^{(t+1)} \leftarrow x_i^{(t+1)}$
      $f_{p_i}^{(t+1)} \leftarrow f_i^{(t+1)}$
    end
  end
  $t \leftarrow t + 1$
end
$x^* \leftarrow p_{g_i}^{(t)}$

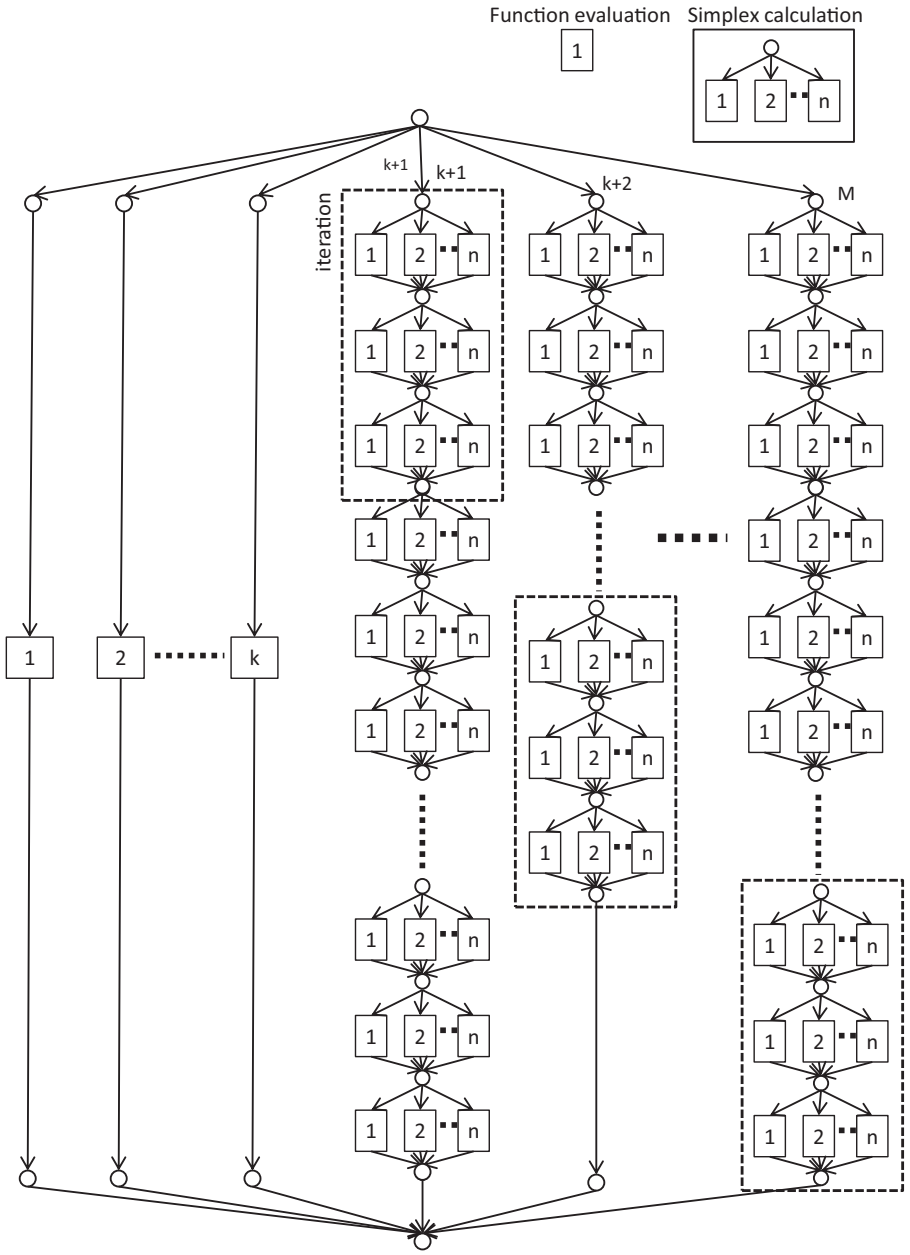


Fig. 2. Execution task graph for hybrid GO (single PSO iteration, lines 24–30 of Algorithm 2).

At the beginning of its main iteration loop, the algorithm determines for each particle whether a simple function evaluation (FE) will take place or a local search (LS) will originate. In line 17 the best position g_i is detected and in line 18 the update to the new particle position is calculated. Then in line 24 the corresponding tasks are executed. Finally the results are gathered and the new best positions are determined in line 31.

4. Parallel implementation

4.1. Parallelizing the hybrid global optimization algorithm

The stochastic nature of the GO algorithm renders it, at a first glance, a cumbersome case of effective parallelization. The basic loop in line 24 of Algorithm 2 spawns N independent tasks, each one associated with a specific particle. Some of these tasks are simple function evaluations (FE) while the rest of them are MDS local searches. The probability ρ at line 11 of the same algorithm controls which particles will execute FE (approximately $(1-\rho)\cdot N$ particles) and which MDS (approximately $\rho \cdot N$ particles). MDS calls include an iterative procedure with a second level of parallelism. At this inner level, n independent calls to the objective function are made for the calculation of the reflection simplex and subsequently n for expansion and n for contraction. The iterations of the MDS procedure are not known beforehand unless we choose a single termination criterion that is based on the maximum number of iterations or function evaluations. However in practice, termination criteria based on the proximity to the solution, are always used. In this way wasting valuable resources is avoided.

Fig. 2.

According to the above, the parallelism of the GO algorithm is highly irregular and depends on several factors: the swarm size and the probability of performing local searches, the time steps required by the local optimization for finding a minimum, the dimensionality and the execution time of the objective function. An efficient

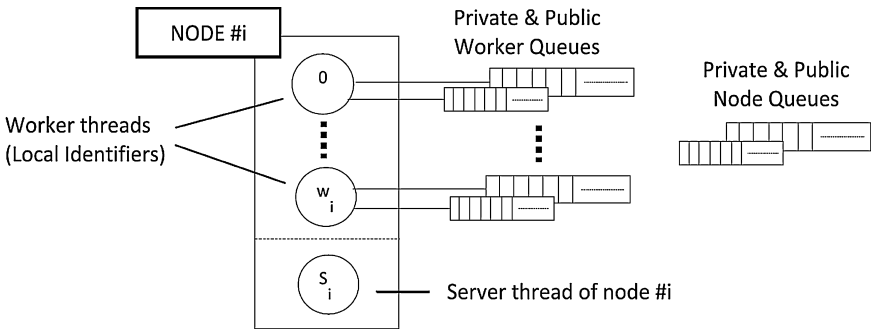


Fig. 3. Intra-node architecture of TORC.

parallel implementation of the GO algorithm requires flexible management of this dynamic and highly irregular nested parallelism. To achieve that on shared-memory platforms, we have used the recent OpenMP [32] tasking model, which extends the expressiveness of OpenMP beyond loop-level and coarse-grain parallelism. Both function evaluations and MDS calls are expressed with OpenMP tasks, with the latter dynamically spawning further function evaluation tasks. Specifically, we create only a single team of threads for all levels of parallelism. The master thread runs an implicit task, the *primary task*, that executes the main loop of the algorithm and iteratively spawns first-level tasks using the `task` construct. The rest of the threads reach the end of the parallel region and participate in their execution of these tasks. The primary task then encounters a `taskwait` construct and waits for its child tasks to complete, while the master thread participates in the their execution. Any OpenMP thread that executes an MDS task, dynamically spawns additional tasks at the innermost level of parallelism following the same fork-join approach. The underlying OpenMP runtime library is responsible for the scheduling of all these tasks across the available cores. The OpenMP tasking model allows us to instantiate the task-graph of the hybrid GO algorithm in a straightforward way and effectively exploit the multiple levels of parallelism. In addition, due to the single team of threads, our parallel application avoids the overheads and performance implications of OpenMP nested parallel regions.

4.2. TORC runtime library

Despite the advantages of the OpenMP tasking model, the above implementation is not supported on distributed memory systems. In order to exploit multicore clusters, we implemented the parallel GO algorithm with TORC [5,33], a custom task-parallel library that offers a programming and runtime environment where parallel programs can be executed unaltered on both shared and distributed memory platforms.

According to TORC, a parallel application consists of multiple MPI processes running on cluster nodes and having one or more kernel-level POSIX threads that share the process memory. Each kernel thread is a worker that continuously dispatches and executes ready-to-run tasks, submitted for execution to a set of ready queues. Due to the decoupling of tasks and execution vehicles, arbitrary nesting of tasks is inherently supported and any child task can become a parent.

In accordance with OpenMP, a parent–child relationship is supported between the tasks of the library. Furthermore, similarly to Remote Procedure Calls [34], a task results in the asynchronous execution of the user specified function and its parameters.

All data transfers in the library are performed with explicit, transparent to the user, messaging. This is achieved with a server thread in each MPI process, which is responsible for the remote queue management and the transparent and asynchronous data movement. There are private and public worker specific and node specific ready queues where tasks can be submitted for execution. When running on a single node, TORC operates as a two level threading library that seamlessly exploits intra-node task parallelism and completely avoids explicit messaging. An overview of the architecture of TORC on a single cluster node is depicted in Fig. 3.

The user can query the execution environment (number of nodes and workers) and then specify the node or worker where each task will be submitted for execution. As *task stealing* is inherently supported by TORC, the programmer has only to decide about the task distribution scheme. Upon termination or suspension of the current task, the underlying worker runs the scheduling loop: it first visits its local ready queue and tries to extract the task at the front of it. If the queue is empty, it tries to steal tasks from the rest of the intra-node ready queues. If inter-node task stealing is enabled, it will issue requests for work to remote nodes in sequential order. Task stealing is always performed from the back of the ready queues.

The parallel implementation of the hybrid GO algorithm on top of the TORC library is equivalent to the OpenMP-based one. The main application task is executed by one of the workers of the MPI process with rank 0. The first level tasks are distributed cyclically to the workers and inserted at the back of their ready queues. At the second level of parallelism, however, the spawned tasks are inserted at the front of the queue that belongs to the worker thread where the parent (MDS) task runs on. The combination of the described task distribution scheme with the stealing mechanism of the library favors the local computation of function evaluation tasks and gives priority to the remote stealing of MDS tasks.

5. Experiments – results

In this section we present experimental results designed to:

(a) analyze the objective function execution time,
(b) explore the efficiency of the hybrid algorithm in its serial implementation,
(c) measure the parallel performance of the proposed hybrid algorithm against parallel implementations of other global optimization methods,
(d) explore the scalability of the parallel implementation.

The parallel algorithm implementation was tested on two different hardware platforms:

P1 A multicore server with two 12-core AMD Opteron 6174 CPUs (2.2GHz, 512KB private L2 cache/core) and 32GB of RAM.
P2 A 12 node Sun Fire x4100 cluster interconnected with Gigabit Ethernet. Each node has 2 dual-core AMD Opteron-275 processors (2.2 GHz, 1 MB cache/core) and 4 GB of RAM.

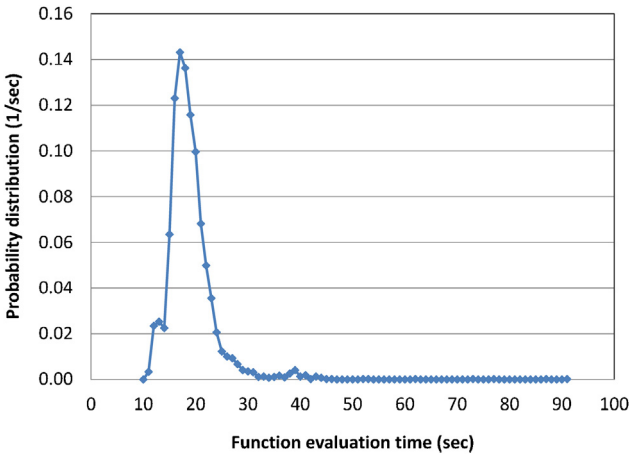


Fig. 4. Probability distribution of the function evaluation time.

The software was compiled under Linux 2.6 with GNU gcc 4.6 and MPICH2. Moreover, the objective function is written in Fortran and compiled with the corresponding GNU gfortran compiler.

5.1. Objective function time

In order to analyze the time distribution of the objective function a series of functions evaluations were performed inside search space X, at points generated by the hybrid GO algorithm. Fig. 4 depicts the probability distribution of the execution time of 5372 total function evaluations for an indicative experimental setup, executed on a single-core of the shared-memory server. The average time is 18.97 s with standard deviation 4.67, while the minimum and maximum evaluation time is 10.62 and 90.85 s, respectively. The observed variation of the function time corresponds to an additional source of irregularity in the GO algorithm and indicates the required support of adaptive load balancing from the parallel execution environment.

Table 3
Parameters for serial experiments against BBOB.

Swarm size (N)	20
Probability (ρ)	0.05
Maximum function evaluations	$500,000 \cdot n$
Maximum function evaluations per local search	2000
Termination criterion	$\lvert f_{hybrid} - f^* \rvert \leq 10^{-8}$

5.2. Serial experiments

In order to establish the efficiency of the chosen hybrid scheme, we have tested our serial implementation against the Black-Box Optimization Benchmarking (BBOB) 2009 test set [35]. Following the experimental setup instructions of [36] we performed a series of experiments using the parameters shown in Table 3.

The algorithms in BBOB 2009 include the multistart method with various local searches (BFGS, Nelder–Mead simplex, NEWUOA, Rosenbrock's method), clustering multistart (GLOBAL), single point deterministic algorithms (DIRECT, MCS), hybrid algorithms (MA-LS-Chain, DEPSO) and population based methods (GA variants, DE variants, PSO variants, CMA-ES variants, ant stigmetry). For a detailed description of the above algorithms please refer to [35]. The test set includes 24 test function categories

Table 4
Percentage of trials solved for an accuracy of 10^{-8}.

Algorithm	Description	Dimension		
		3-D	5-D	10-D
Hybrid GO	Presented hybrid algorithm (Voglis, Hadjidoukas, Papageorgiou, Lagaris)	**99.23%**	**91.41%**	**64.04%**
GLOBAL	Sampling, clustering and local search using BFGS or Nelder–Mead	44.72%	35.83%	23.06%
DIRECT	Axis-parallel search space partitioning procedure	65.83%	22.50%	9.17%
F-NEWUOA	NEWUOA algorithm using $(n^2+3n+2)/2$ points	58.33%	47.22%	33.06%
CMA-ESplus	CMA-ES restarted with Increasing POPulation size, first run with SEParable CMA	59.17%	71.11%	51.67%
G3PCX	Generalized Generation Gap with Parent Centric Crossover	47.22%	68.06%	45.56%
Random	Uniform random sampling	98.17%	91.72%	63.61%
POEMS	Prototype Optimization with Evolved Improvement Steps using hypermutations and stochastic local search	59.72%	42.22%	30.83%
BP-CMA-ES	CMA-ES restarted with budgets for small and large population size	95.83%	92.22%	86.67%
PSO_Bounds	A PSO variation	61.11%	33.33%	25.28%
ONEFIFTH	Evolutionary strategy with with 1/5th success rule for step-size adaptation	53.06%	43.33%	29.72%
BFGS	Multistart using Matlab's BFGS implementation	40.28%	36.11%	26.94%
DEPSO	PSO with Differential Evolution variations	29.17%	19.72%	13.06%
Rosenbrock	Multistart using Rosenbrock's derivative free algorithm	53.33%	41.94%	20.56%
GA	Binary coded GA	28.61%	19.44%	2.50%
DASA	Differential Ant-Stigmergy Algorithm	58.06%	44.72%	42.22%
Cauchy-EDA	EDA with isotropic Cauchy sampling distribution	59.17%	59.17%	58.61%
NEWUOA	NEW Unconstraint Optimization Algorithm builds a second order model using $2n+1$ points and with minimal Frobenius norm	58.06%	45.83%	45.00%
ALPS	Age-Layered Population Structure running a standard GA in 12 layers	85.00%	61.42%	41.28%
NELDER(2)	Nelder–Mead downhill simplex with restarted half-runs	83.33%	58.06%	32.78%
LSstep	Axis-parallel line search with the univariate STEP Select The Easiest Point	21.39%	20.83%	20.00%
NELDER	Multistart using Nelder–Mead downhill simplex algorithm	85.83%	62.78%	44.72%
PSO	Standard PSO with swarm size 40 and no restarts	64.44%	32.78%	18.33%
AMALGAM	Adapted Maximum-Likelihood Gaussian Model Iterated Density Estimation Algorithm with no-improvement stretch.	99.72%	91.94%	79.17%
EDA-PSO	Hybrid of EDA and PSO with adapted probability of applying one or the other	67.22%	48.06%	26.67%
MA-LS-Chain	Memetic Algorithm with Local Search Chaining using a steady-state GA and CMA-ES for local search with a fixed local/global search ratio	89.44%	74.44%	58.61%
BAYEDA	An EDA using Bayesian inference to learn the parameters of the continuous Gaussian distribution	10.00%	9.72%	8.89%
iAMALGAM	AMALGAM with incremental model building	98.06%	91.94%	81.39%
MCS	Multilevel Coordinate Search with additional local searches	55.28%	32.22%	20.00%
LSfminbnd	Axis-parallel line search with MATLAB fminbnd univariate search	20.83%	16.94%	13.33%

with variable dimensionality and a complete set of postprocessing scripts to generate insightful charts. We have chosen to present results as they are produced by the automatic postprocessing scripts so that the comparison would be on fair grounds. Here we show 3, 5 and 10-dimensional results since our potential fitting application is of similar dimensionality. A total of 360 test function instances for each of the three dimensionalities were used in our benchmark.

The comparison is made through the *Empirical Cumulative Distribution Function* (ECDF) of the number of function evaluations. The ECDF represents the fraction of test instances solved within the prescribed accuracy and is described in detail in [37]. In Fig. 5 we present the ECDF of the number of function evaluations needed to achieve various levels of accuracy $(10, 10^{-1}, 10^{-4}, 10^{-8})$. The results presented correspond to the 3, 5 and 10-dimensional instances of the test functions. The thick red line shows the percentage of trials that reached the global minimum with accuracy of 10^{-8} as a function of running time. The light brown lines in the background show ECDFs for the same accuracy for all algorithms benchmarked during BBOB 2009. By inspecting Fig. 5 an immediate qualitative comparison is possible. The prosed hybrid algorithm solves all 3-dimensional cases, almost 90% of the 5-dimensional and 60% of the 10-dimensional to an accuracy of 10^{-8}. Only 4 other competitive serial algorithms achieved the same percentage. The above results, establish the efficiency of our hybrid scheme regarding low dimensional optimization cases.

For a direct comparison of our hybrid to competitive algorithms from BBOB 2009, we present in Table 4 the percentage of trials that reached accuracy up to 10^{-8} using a budget of $100,000 \times n$ function evaluations. Detailed description of the algorithms can be found in [35]. Table 4 confirms that the proposed algorithm is very competitive in solving small dimensional problems.

5.3. Parallel experiments

The purpose of the parallel implementation experiments is twofold. First to compare the performance of our parallel hybrid algorithm against some well established parallel global optimization implementations. The second goal is to investigate the effect of the input parameters to the overall speedup.

Comparison to other parallel algorithms. Despite the multitude of parallel global optimization algorithms presented in the literature only a handful of real systems is actually implemented and distributed freely. Among them, we distinguish PAGMO [38] that offers an island model paradigm able to parallelize hybrids between global and local optimization algorithms, VTDIRECT95 [39] that provides a parallel implementation of a deterministic global optimization algorithm called DIRECT [40] and CMA-ES [41] which evolves a population using a multivariate normal distribution.

All algorithms were tested on platform P1. For the PAGMO algorithm we established an archipelago of 24 fully connected MPI islands, each one running a DE algorithm with population size equal to 6. This gives us a total of 144 particles. For the VTDIRECT implementation we used in total 24 processes that were split in both 8 subdomains and 24 subdomains. For the CMA-ES we implemented a simple MPI scheme that circularly assigns function evaluation tasks to processes and used swarm sizes of 24, 48 and 144. Finally in our hybrid implementation we also used a swarm size of 144, probability $\rho = 0.0$ and $\rho = 0.5$ and a maximum of 2000 function evaluations per local search. We measured the total running time and the parallel efficiency of all schemes. We present speedup and efficiency results in Table 5. An illustration of the efficiency is also given in Fig. 6. It is clear that our parallel implementation achieves the best speedup and efficiency score.

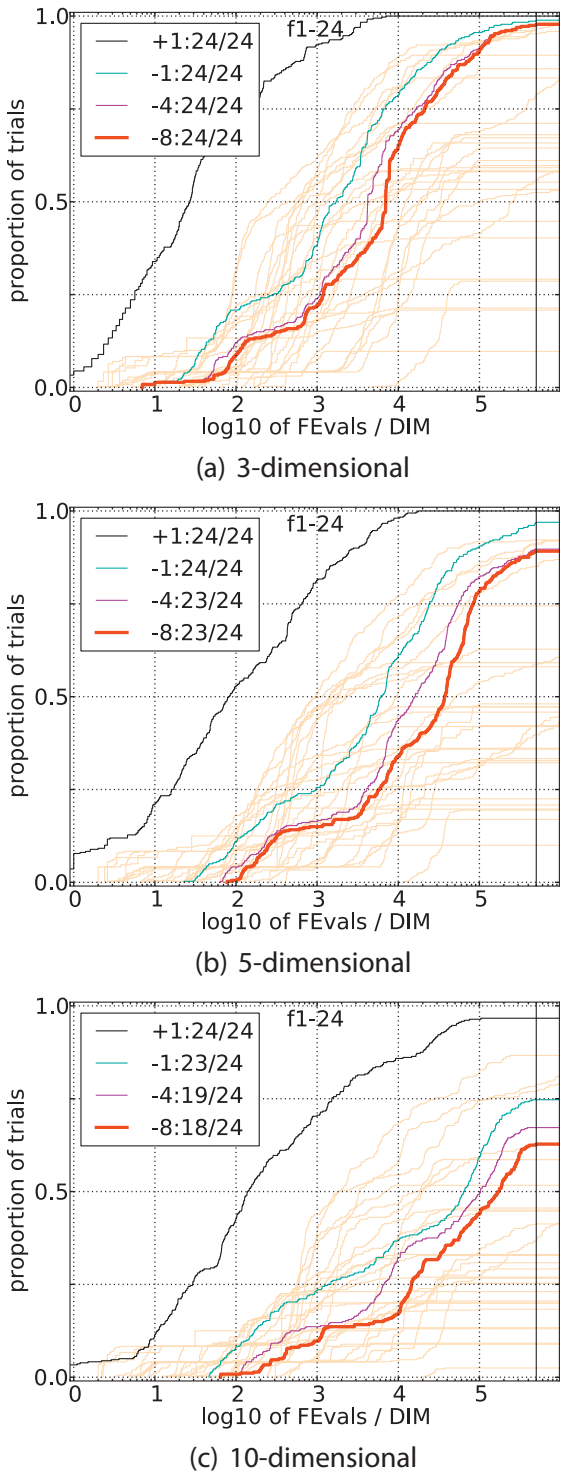


Fig. 5. Empirical cumulative distribution functions (ECDFs) of number of function evaluations (FEvals) divided by search space dimension n, to fall below $f^* + \delta f$ with $\delta f = 10^k$, where k is the first value in the legend. The thick red line represents the most difficult target value $f^* + 10^{-8}$. Light brown lines in the background show ECDFs for $\delta f = 10^{-8}$ of all algorithms benchmarked during BBOB-2009. Legends indicate the number of functions that were solved in at least one trial. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of the article.)

Speedup calculation. The second experiment studies the scalability of the hybrid GO algorithm for a fixed set of control parameters (probability $\rho = 0.5$, swarm size $N = 128$) and a specific number of total function evaluations. In all experiments the target parameters in Table 2 where approximated by an order of 10^{-4}.

Table 5
Parallel comparison table.

Algorithm	Description	Speedup	Efficiency
CMA-24	$S=24$	18.58	77%
CMA-48	$S=48$	19.53	81%
VTD-24	24 sub-domains	19.85	83%
VTD-8	8 sub-domains	16.05	67%
PAGMO-DE	24 MPI islands, fully connected	21.41	89%
PMEM-0	Swarm size 144, $p=0$	21.61	90%
PMEM-0.5	Swarm size 144, $p=0.5$	21.79	91%

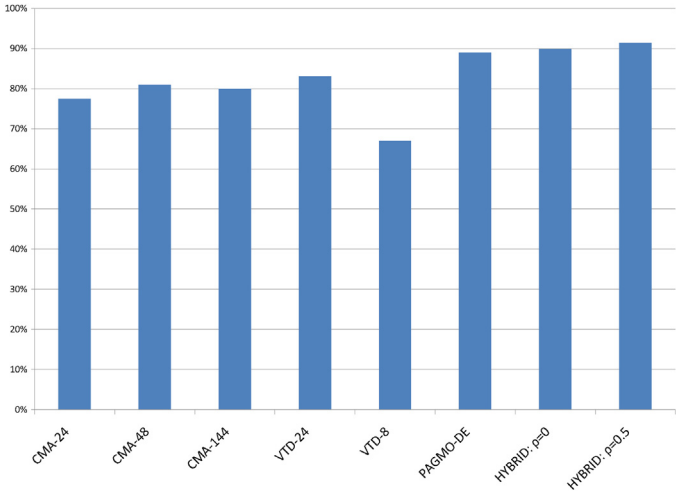


Fig. 6. Parallel efficiency comparison.

Fig. 7(a) and (b) shows the speedup for the particular test case on the shared-memory and the cluster system, respectively. We observe that the application scales well on both architectures. On the 24-core system, the maximum speedup for the OpenMP version is 21.30 (88.7% efficiency), while the TORC-based version attains comparable performance (20.69 speedup, 86.2% efficiency). Moreover, the latter version obtains a speedup of 36.12 on the 48 cores of the cluster (75.3% efficiency). The decrease in the performance with the number of cores is mostly attributed to the inherent load imbalance of the algorithm, as the computational work (tasks) cannot be distributed evenly. Furthermore, the objective function time increases due to the higher contention on the memory subsystem.

Varying swarm size N. In this experiment we study the parallel performance of the GO algorithm with respect to the swarm size. We have used a fixed number of processing elements for each platform: all the 24 cores of the shared-memory server and 8 nodes (32 cores) of the cluster. In addition, the probability of local optimization is set to 0.1 in all cases.

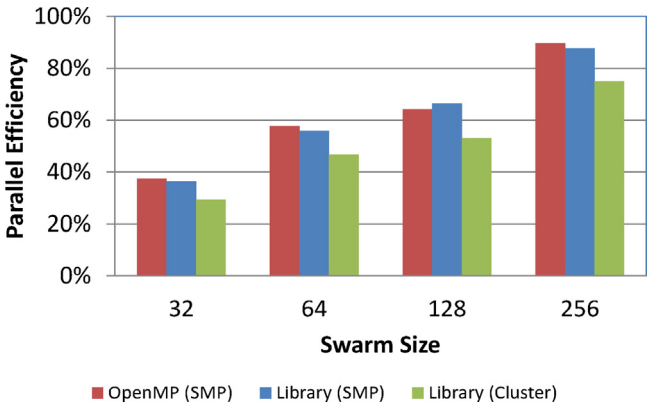


Fig. 8. Parallel efficiency with respect to swarm size.

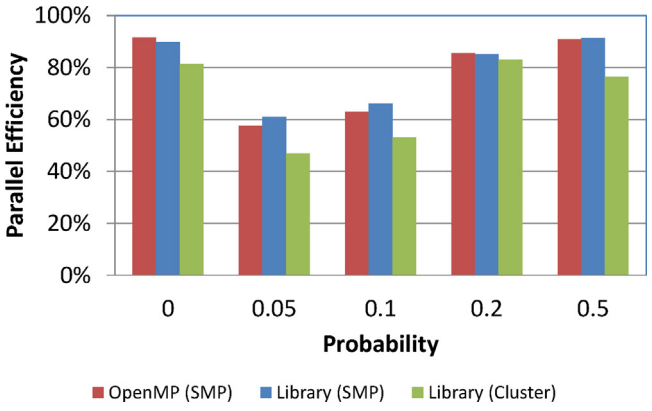
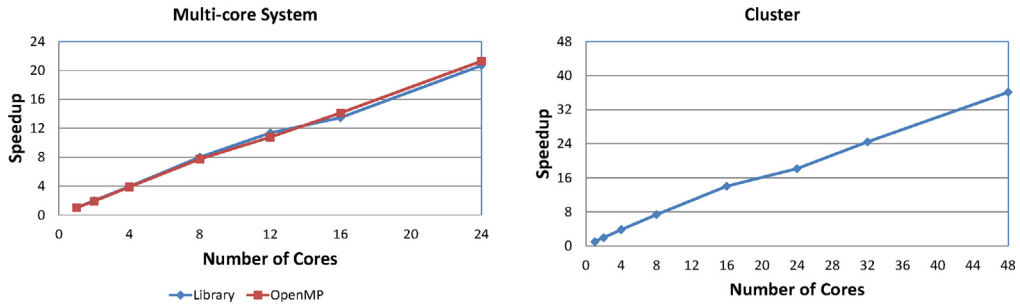


Fig. 9. Parallel efficiency with respect to probability.

In Fig. 8 we present the obtained efficiency of the parallel GO algorithm for swarm size 32, 64, 128 and 256. We observe that the parallel performance improves as the swarm size increases. This is attributed to the higher number of available tasks, which are distributed more evenly to the processors and thus better load balancing and hardware utilization is achieved. In the same figure, we also observe that the OpenMP and the library-based implementation of the GO algorithm exhibit comparable performance.

Varying probability ρ. Similarly to the previous one, our last experiment studies how the GO algorithm behaves with respect to the probability ρ. Fig. 9 shows the parallel efficiency for a swarm of 128 particles and probability ρ equal to 0, 0.05, 0.1, 0.2 and 0.5. For this particular configuration, we observe that the pure PSO approach ($\rho=0$) attains good load balancing and, thus, the highest efficiency. When local optimizations are used, the performance



(a) Shared-memory server (b) Cluster

Fig. 7. Speedup measurement.

is improved as the probability increases, because more function evaluations tasks become available for execution at each step of the algorithm.

6. Conclusions

We presented a parallel global optimization algorithm suitable for the specific class of time consuming potential fitting problems that follow the SMATB formalism. The proposed algorithm is a hybrid between PSO, for the global exploration, and the MDS algorithm that locates local minimizers. Both algorithms are parallelizable and the hybridization strategy results in a highly irregular and dynamic task graph. We introduce a runtime environment suitable to support irregular and dynamic tasks on both shared and distributed memory platforms.

We have performed extensive experiments using the serial and parallel implementations of the proposed algorithm. The parallel implementation scores high efficiency on both cluster and shared memory systems. Furthermore, it achieves the best parallel performance when compared to other parallel algorithms. Experiments using the serial implementation of the algorithm demonstrate its applicability on a wide range of general optimization problems.

Concerning future work a different parallelization scheme of MDS algorithm can be explored. In this new scheme reflection, expansion and contraction take place in a single parallel loop resulting in $3 \cdot n$ parallel function evaluation tasks. In this way we retain good efficiency in even larger parallel systems without compromising the properties of the algorithm. Furthermore, following Torczon's guidelines in [12] a full lookahead step of the MDS algorithm will increase exponentially the number of parallel tasks. On small sized problems, launching a large number of parallel function evaluations increases the parallel efficiency.

It would be also interesting to implement different hybrid schemes combining new global and local components. Note that various new derivative-free local search methods have been developed that would be excellent candidates for the local component. Extending the above, a new class of adaptive hybrid optimization schemes that are not limited to the use of a single local search but choose a proper one from a rich has been proposed and it indeed exhibits encouraging results [42].

Considering the potential fitting application one can extend the SMATB formalism to include alloys of metals. The resulting optimization problem is of higher dimensionality and complexity and hence requires longer execution times. Predicting the structural behavior of metallic alloys is an application of huge practical importance.

References

[1] J.H. Li, X.D. Dai, S.H. Liang, K.P. Tai, Y. Kong, B.X. Liu, Interatomic potentials of the binary transition metal systems and some applications in materials physics, Physics Reports 455 (1-3) (2008) 1–134.

[2] T. Coleman, D. Shalloway, Z. Wu, A parallel build-up algorithm for global energy minimizations of molecular clusters using effective energy simulated annealing, Journal of Global Optimization 4 (2) (1994) 171–185.

[3] M. Locatelli, F. Schoen, Local search based heuristics for global optimization: atomic clusters and beyond, European Journal of Operational Research 222 (1) (2012) 1–9.

[4] R.H. Byrd, E. Eskow, A. Van Der Hoek, R.B. Schnabel, K.P.B. Oldenkamp, A parallel global optimization method for solving molecular cluster and polymer conformation problems, in: Proceedings of 7th SIAM Conference on Parallel Processing for Scientific Computing, SIAM, Citeseer, 1995, pp. 72–77.

[5] C. Voglis, P.E. Hadjidoukas, V.V. Dimakopoulos, I.E. Lagaris, D.G. Papageorgiou, Task-parallel global optimization with application to protein folding., in: International Conference on High Performance Computing and Simulation (HPCS), 2011, IEEE, 2011, pp. 186–192.

[6] J. Kennedy, R.C. Eberhart, Particle swarm optimization., in: Proceedings of IEEE International Conference on Neural Networks, vol. IV, IEEE Service Center, Piscataway, NJ, 1995, pp. 1942–1948.

[7] J.A. Nelder, R. Mead, A simplex method for function minimization, The Computer Journal 7 (4) (1965) 308–313.

[8] Y. Zhang, D. Gallipoli, C.E. Augarde, Simulation-based calibration of geotechnical parameters using parallel hybrid moving boundary particle swarm optimization, Computers and Geotechnics 36 (4) (2009) 604–615.

[9] W. Deng, R. Chen, J. Gao, Y. Song, J. Xu, A novel parallel hybrid intelligence optimization algorithm for a function approximation problem, Computers and Mathematics with Applications 63 (1) (2012) 325–336.

[10] Y. Shimizu, T. Miura, M. Ikeda, A parallel computing scheme for large-scale logistics network optimization enhanced by discrete hybrid pso, Computer Aided Chemical Engineering 27 (2009) 2031–2036.

[11] K.E. Parsopoulos, M.N. Vrahatis, UPSO: A unified particle swarm optimization scheme, in: Lecture Series on Computer and Computational Sciences, vol. 1, Proceedings of the International Conference of Computational Methods in Sciences and Engineering (ICCMSE 2004), VSP International Science Publishers, Zeist, The Netherlands, 2004, pp. 868–873.

[12] J.E. Dennis Jr., V. Torczon, Direct search methods on parallel machines, SIAM Journal on Optimization. Citeseer 1 (4) (1991) 448–474.

[13] Y.G. Petalas, K.E. Parsopoulos, M.N. Vrahatis, Memetic particle swarm optimization, Annals of Operations Research 156 (1) (2007) 99–127.

[14] C. Voglis, K.E. Parsopoulos, D.G. Papageorgiou, I.E. Lagaris, M.N. Vrahatis, Mempsode: A global optimization software based on hybridization of population-based algorithms and local searches, Computer Physics Communications 183 (5) (2012) 1139–1154.

[15] C.S. Barrett, T.B. Massalski, Structure of Metals, 3rd ed., McGraw-Hill, London, 1980.

[16] C. Kittel, Introduction to Solid State Physics, John Wiley and Sons, New York, 1976.

[17] G. Simmons, H. Wang, Single Crystal Elastic Constants and Calculated Aggregate Properties, The MIT Press, 1971, pp. 269.

[18] V.O. Shestopal, Specific heat and vacancy formation in titanium at high temperatures, Soviet Physics–Solid State 7 (11) (1966) 2798–2799.

[19] L. Fast, J.M. Wills, B. Johansson, O. Eriksson, Elastic constants of hexagonal transition metals: Theory, Physical Review B 51 (24) (1995) 17431.

[20] R. Dawkins, The Selfish Gene, Oxford University Press, New York, 1976.

[21] P. Moscato, Memetic algorithms: a short introduction, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, London, 1999, pp. 219–235.

[22] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, Reading, MA, 1989.

[23] K. Price, Differential evolution: a fast and simple numerical optimizer, in: Proceedings NAFIPS'96, 1996, pp. 4–525.

[24] Z.W. Geem, J.H. Kim, et al., A new heuristic optimization algorithm: harmony search, Simulation 76 (2) (2001) 60–68.

[25] J. Olenšek, T. Tuma, J. Puhan, Á. Bürmen, A new asynchronous parallel global optimization method based on simulated annealing and differential evolution, Applied Soft Computing 11 (1) (2011) 1481–1489.

[26] W. Zhu, Nonlinear optimization with a massively parallel evolution strategy-pattern search algorithm on graphics hardware, Applied Soft Computing 11 (2) (2011) 1770–1781.

[27] M.M. Noel, A new gradient based particle swarm optimization algorithm for accurate computation of global minimum, Applied Soft Computing 12 (1) (2011) 353–359.

[28] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings Sixth Symposium on Micro Machine and Human Science, IEEE Service Center, Piscataway, NJ, 1995, pp. 39–43.

[29] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (1) (2002) 58–73.

[30] C. Voglis, G.S. Piperagkas, K.E. Parsopoulos, D.G. Papageorgiou, I.E. Lagaris, Mempsode: comparing particle swarm optimization and differential evolution within a hybrid memetic global optimization framework, in: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion, ACM, 2012, pp. 253–260.

[31] C. Voglis, G.S. Piperagkas, K.E. Parsopoulos, D.G. Papageorgiou, I.E. Lagaris, Mempsode: an empirical assessment of local search algorithm impact on a memetic algorithm using noiseless testbed, in: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference Companion, ACM, 2012, pp. 245–252.

[32] OpenMP Architecture Review Board. Openmp specifications. Available at: http://www.openmp.org

[33] P. Hadjidoukas, C. Voglis, V. Dimakopoulos, I. Lagaris, D.G. Papageorgiou, High-performance numerical optimization on multicore clusters, in: Euro-Par 2011 Parallel Processing, 2011, pp. 353–364.

[34] R. Srinivasan. RFC 1831: RPC: Remote procedure call protocol specification version 2, August 1995. Status: Proposed standard.

[35] N. Hansen, A. Auger, R. Ros, S. Finck, P. Pošík, Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009., in: Proceedings of the 12th Annual Conference Companion on Genetic and Evolutionary Computation, ACM, 2010, pp. 1689–1696.

[36] N. Hansen, A. Auger, S. Finck, R. Ros, Real-parameter black-box optimization benchmarking 2010: experimental setup, 2010 http://hal.inria.fr/inria-00462481/PDF/RR-7215.pdf

[37] H.H. Hoos, T. Stützle, Evaluating las vegas algorithms: Pitfalls and remedies., in: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., 1998, pp. 238–245.
[38] F. Biscani, D. Izzo, C.H. Yam. A global optimisation toolbox for massively parallel engineering optimisation. arXiv preprint arXiv:1004.3824, 2010.
[39] J. He, L.T. Watson, M. Sosonkina, Algorithm 897: Vtdirect95: Serial and parallel codes for the global optimization algorithm direct, ACM Transactions on Mathematical Software (TOMS) 36 (3) (2009) 1.
[40] D.R. Jones, C.D. Perttunen, B.E. Stuckman, Lipschitzian optimization without the lipschitz constant, Journal of Optimization Theory and Applications 79 (1) (1993) 157–181.
[41] N. Hansen, The cma evolution strategy: a comparing review, Towards a New Evolutionary Computation 7 (2006) 5–102.
[42] Y.-S. Ong, M.-H. Lim, N. Zhu, K.-K. Wong, Classification of adaptive memetic algorithms: a comparative study, IEEE Transactions on Systems, Man and Cybernetics 36 (1) (2006) 141–152.