pCMALib: a Parallel FORTRAN 90 Library for the Evolution Strategy with Covariance Matrix Adaptation

Christian L. Müller
Institute for Theoretical Computer Science and Swiss Institute of Bioinformatics, ETH Zurich
Universitätsstrasse 6
Zurich, Switzerland
christian.mueller@inf.ethz.ch

Benedikt Baumgartner
Fakultät für Informatik, Technische Universität München
Boltzmannstrasse 3
Garching near Munich, Germany
bb@tum.de

Georg Ofenbeck
Institute for Theoretical Computer Science and Swiss Institute of Bioinformatics, ETH Zurich
Universitätsstrasse 6
Zurich, Switzerland
ofgeorg@student.ethz.ch

Birte Schrader
Institute for Theoretical Computer Science and Swiss Institute of Bioinformatics, ETH Zurich
Universitätsstrasse 6
Zurich, Switzerland
birtes@ethz.ch

Ivo F. Sbalzarini
Institute for Theoretical Computer Science and Swiss Institute of Bioinformatics, ETH Zurich
Universitätsstrasse 6
Zurich, Switzerland
ivos@ethz.ch

ABSTRACT

We present pCMALib, a parallel software library that implements the Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). The library is written in Fortran 90/95 and uses the Message Passing Interface (MPI) for efficient parallelization on shared and distributed memory machines. It allows single CMA-ES optimization runs, embarrassingly parallel CMA-ES runs, and coupled parallel CMA-ES runs using a cooperative island model. As one instance of an island model CMA-ES, the recently presented Particle Swarm CMA-ES (PS-CMA-ES) is included using collaborative concepts from Swarm Intelligence for the migration model. Special attention has been given to an efficient design of the MPI communication protocol, a modular software architecture, and a user-friendly programming interface. The library includes a Matlab interface and is supplemented with an efficient Fortran implementation of the official CEC 2005 set of 25 real-valued benchmark functions. This is the first freely available Fortran implementation of this standard benchmark test suite. We present test runs and parallel scaling benchmarks on Linux clusters and multi-core desktop computers, showing good parallel efficiencies and superior computational performance compared to the reference implementation.

Categories and Subject Descriptors

D.1.3 [**Software**]: Programming techniques—*Parallel programming*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Software libraries*

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

General Terms

Algorithms

Keywords

CMA-ES, evolution strategies, software library, parallel island model

1. INTRODUCTION

Over the past two decades, the parallelization of Evolutionary Algorithms (EA) has received growing attention in computer science and engineering. EA's are well suited for parallel implementation since they evolve a population of candidate solutions. The two main parallelization paradigms for EA's are the coarsely grained (a)synchronous cooperative island model and the master/slave model [8]. While libraries such as PGAPack [22] and libSRES [20] follow the master/slave approach, examples for libraries that include both approaches are the ParadisEO framework [8] and MALLBA [3]. The pCMALib presented here uses a synchronous cooperative island model. We refer to Ref. [2] for a summary of state-of-the-art techniques in the field of parallel metaheuristics.

Among the various EA, Evolution Strategies (ES) [27] have been a successful optimization paradigm for non-convex, real-valued functions. A particularly prominent example is the Evolution Strategy with Covariance Matrix Adaptation (CMA-ES) [17, 15]. In the CEC 2005 competition on real-parameter optimization [32], CMA-ES with a wide initial sample distribution and iteratively increasing population size (IPOP-CMA-ES) [5] achieved the best performance of all tested algorithms [13]. CMA-ES has been successfully applied to numerous problems in science and engineering, including computational fluid dynamics [30], automotive electronics [9], biology [25], hydrology [7], and optics. On *multi-funnel* functions, where local optima cannot be interpreted as perturbations to an underlying convex (unimodal) topology, the performance of CMA-ES can be improved [15, 23], e.g. by considering several concurrent runs that are allowed to exchange information. This is realized in the Particle Swarm CMA-ES (PS-CMA-ES) [26].

So far, several sequential implementations of CMA-ES are available in Matlab, Octave, C, C++, Java, and Python (see Ref. [12] for a complete list). Moreover, a multi-objective variant of CMA-ES (MO-CMA-ES) is included in the Shark C++ library [18]. CMA-ES has, however, a comparatively high computational cost, since an Eigen-decomposition has to be computed in each iteration. This has motivated the development of several *approximate* CMA variants with reduced computational cost, such as the Elitist CMA [19], L-CMA [21], and sep-CMA [28]. Since the evaluations of the cost function for all candidate solutions within an individual CMA-ES population are independent, they can also be distributed over multiple processors and evaluated in an embarrassingly parallel way. This is, e.g., implemented in pCMA [16]. We believe that the performance can be further increased by parallelizing over entire CMA instances rather than single function evaluations. Such parallel island models for CMA-ES have been successfully applied to the inference of gene regulatory networks [31], to benchmark problems in global optimization [26], and in the design of a fault-tolerant optimization tool [24]. To the best of our knowledge, however, no portable library implementation is currently freely available.

We present a portable, efficient, and scalable parallel library for CMA-ES, parallel CMA-ES, and PS-CMA-ES using a synchronous parallel island model. The library is implemented in Fortran 90/95 and uses the Linear Algebra Package (LAPACK), the Basic Linear Algebra Subprograms (BLAS) [4], and the Message Passing Interface (MPI). This ensures computational efficiency and portability. When When using a parallel island model CMA-ES, the library is able to dynamically exclude terminated CMA-ES instances from communication, which reduces the communication overhead and provides the opportunity of implementing alternative mechanisms of fault tolerance. Besides considerations of computational efficiency, we chose the Fortran language because it is widely used in many fields of application, including solid mechanics, fluid dynamics, hydrology, and, above all, computational chemistry, where pCMALib might serve as a valuable tool for the exploration of high-dimensional potential energy surfaces of clusters and biomolecules [34]. Our library provides an easy-to-use programming interface, good parallel scaling on multi-core and multi-processor machines, as well as Matlab bindings.

This paper is organized as follows. The next section describes the implemented (PS-)CMA-ES algorithms and the test function suite. In Sec. 3, the general software design of the library is outlined and several implementation details explained. In Sec. 4, we assess the performance of the library on multi-core and cluster computers, and Sec. 5 discusses the results and concludes this work. We focus on benchmarks of computational efficiency and parallel scalability. The success performance of (PS-)CMA-ES on optimization benchmarks is published elsewhere [5, 26].

2. IMPLEMENTED ALGORITHMS

Before describing the pCMALib library, we review the implemented algorithms and describe the Fortran 90/95 implementation of the CEC 2005 benchmark suite.

2.1 The CMA Evolution Strategy

We consider the CMA Evolution Strategy with weighted intermediate recombination, step size adaptation, and a combination of rank-μ update and rank-one update [16, 15, 14]. At each iteration of the algorithm, the members of the new population are sampled from a multivariate normal distribution $\mathcal{N}$ with mean $\mathbf{m} \in \mathbb{R}^n$ and covariance $\mathbf{C} \in \mathbb{R}^{n\times n}$. The sampling radius is controlled by the overall standard deviation (step size) σ. Let $\mathbf{x}_k^{(g)}$ the k^{th} individual at generation g. The new individuals at generation $g+1$ are sampled as:

$$\mathbf{x}_k^{(g+1)} \sim \mathbf{m}^{(g)} + \sigma^{(g)}\mathcal{N}\Big(\mathbf{0}, \mathbf{C}^{(g)}\Big) \quad k = 1, \ldots, \lambda\,. \tag{1}$$

The λ sampled points are ranked in order of ascending fitness, and the μ best are selected. The mean of the sampling distribution given in Eq. 1 is updated using *weighted intermediate recombination* of the selected points:

$$\mathbf{m}^{(g+1)} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}^{(g+1)}\,, \tag{2}$$

with

$$\sum_{i=1}^{\mu} w_i = 1, \quad w_1 \geq w_2 \geq \ldots \geq w_\mu > 0\,, \tag{3}$$

where the w_i are positive weights, and $\mathbf{x}_{i:\lambda}^{(g+1)}$ denotes the i^{th} ranked individual of the λ sampling points $\mathbf{x}_k^{(g+1)}$. While different weighting schemes are available for CMA-ES, the super-linear weight decrease $w_i = \log(\frac{\lambda-1}{2}+1) - \log(i)$ is the standard setting [17] . The covariance matrix is then adapted as described in various publications and reports [17, 15, 23, 14]. The computational complexity of the full covariance matrix update is $\mathbf{O}(n^3)$ due to a necessary Eigen-decomposition in each generation.

The behavior of CMA-ES is mainly controlled by two parameters: the initial step size σ and the population size λ. In standard CMA-ES, the population size is chosen as $\lambda = 4 + \lfloor 3 \ln n \rfloor$ [17].

Two important variants of CMA-ES, both available in pCMALib, use different standard settings for optimization in a bounded subset $[A, B]^n$: The local restart CMA-ES (LR-CMA-ES) [6] restarts a converged CMA-ES from a random position within the subset $[A, B]^n$ with a purely local initial step size of $10^{-2}(B - A)/2$. The CMA-ES with iteratively increasing population size (IPOP-CMA-ES) [5] uses a large initial step size of $(B - A)/2$, and the population size λ is doubled in each sequential restart until the maximum number of function evaluations is reached.

2.2 Particle Swarm CMA-ES

The Particle Swarm CMA-ES [26] was introduced in order to improve the performance of CMA-ES on multi-funnel fitness landscapes. Inspired by ideas from Particle Swarm Optimization (PSO), the algorithm evolves multiple CMA-ES instances *in parallel*. Each instance is considered a swarm particle, which exchanges promising solutions with **all** other swarm particles (CMA-ES instances) every I_{c} generations. Both the swarm size S (total number of parallel CMA-ES instances) and the communication interval I_{c} are new strategy parameters. The global swarm information is used both in the covariance matrix adaptation of each CMA-ES and in the placement of the population means.

The covariance matrix of each CMA-ES instance is adapted such that it is more likely to sample good candidates, based on the current global best position $\mathbf{p}_{\text{g,best}} \in \mathbb{R}^n$ in the swarm. This is achieved by mixing the standard CMA covariance matrix with a PSO covariance matrix that is influenced by global information:

$$\mathbf{C}^{(g+1)} = c_{\text{p}} \cdot \mathbf{C}_{\text{CMA}}^{(g+1)} + (1 - c_{\text{p}}) \cdot \mathbf{C}_{\text{PSO}}^{(g+1)}\,, \tag{4}$$

where the mixing weight $c_{\text{p}} \in [0, 1]$ is a new strategy parameter and $\mathbf{C}_{\text{CMA}}^{(g+1)}$ follows the original adaptation rule [14]. $\mathbf{C}_{\text{PSO}}^{(g+1)}$ is a rotated version of $\mathbf{C}_{\text{CMA}}^{(g)}$, such that the principal eigenvector $\mathbf{b}_{\text{main}}$ of $\mathbf{C}_{\text{CMA}}^{(g)}$ is aligned with the vector $\mathbf{p}_{\text{g}} = \mathbf{p}_{\text{g,best}} - \mathbf{m}^{(g)}$ that points from the current mean $\mathbf{m}^{(g)}$ toward the global best position $\mathbf{p}_{\text{g,best}}$.

$\mathbf{C}_{\text{CMA}}^{(g)}$ can be decomposed as $\mathbf{C}_{\text{CMA}}^{(g)} = \mathbf{B}\mathbf{D}^2\mathbf{B}^T$, such that the rotated covariance matrix can be constructed by rotating the eigenvectors (columns of $\mathbf{B}$), yielding the orthogonal matrix $\mathbf{B}_{\text{rot}}^{(g)} = \mathbf{R}\mathbf{B} \in \mathbb{R}^{n\times n}$ of the rotated eigenvectors. $\mathbf{C}_{\text{PSO}}^{(g+1)}$ is then given by:

$$\mathbf{C}_{\text{PSO}}^{(g+1)} = \mathbf{B}_{\text{rot}}^{(g)} \cdot \left(\mathbf{D}^{(g)}\right)^2 \cdot \left(\mathbf{B}_{\text{rot}}^{(g)}\right)^T . \quad (5)$$

The rotation matrix $\mathbf{R} \in \mathbb{R}^{n\times n}$ is uniquely and efficiently computed using *Givens Rotations* [29, 10, 26]. For each swarm member, the rotation algorithm requires $2n\,(n-1)+1$ matrix multiplications at each PSO update in order to construct $\mathbf{R}$. Hence, the computational cost increases quadratically with the number of dimensions and linearly with the swarm size S.

PS-CMA-ES also introduces a mechanism that allows individual CMA-ES instances to escape local minima. This is achieved by biasing the mean value in direction of the global best solution. After the recombination step of each CMA-ES generation, the updated mean value for the next generation $g+1$ is biased as:

$$\mathbf{m}^{(g+1)} \leftarrow \mathbf{m}^{(g+1)} + \mathbf{bias}\,. \quad (6)$$

Based on the step size σ, PS-CMA-ES distinguishes 3 exploration scenarios for individual CMA-ES instances and applies different biasing rules. We refer to Ref. [26] for a detailed description of these rules and the standard settings of all PS-CMA-ES-specific strategy parameters.

Note that PS-CMA-ES can be considered a generalization of CMA-ES. When no communication occurs between CMA-ES instances, PS-CMA-ES is equivalent to S parallel standard CMA-ES runs. In the limit case $S = 1$, PS-CMA-ES is equivalent to a single standard CMA-ES. Although the design of PS-CMA-ES is based on ideas from PSO, the algorithm is an instance of a synchronous parallel island model in which the PSO updates amount to migration events between sub-populations (CMA-ES instances). In contrast to the more common ring or torus migration (communication) topologies, PS-CMA-ES uses a complete net topology.

2.3 Benchmark function test suite

Evolutionary optimization is almost always concerned with high-dimensional, non-convex, highly multimodal, or noisy objective functions, where other optimization paradigms such as gradient-based methods, linear programming, or quadratic programming are not applicable. Since rigorously proven run-time bounds for evolutionary search methods are often not available on these types of problems, carefully designed computer experiments have to be performed in order to benchmark the effectiveness and efficiency of a new method. In order for different methods to be comparable to one another, they should all be benchmarked on a common standard set of test problems. One key attempt to provide such a standard for performance evaluation and analysis of search heuristics is the CEC 2005 test suite [32]. The test suite includes a set of 25 test functions from a variety of classes such as uni-/multimodal, (non-)separable, noisy, (a-)symmetric, multi-funnel, and scalable. Over the past years, this test suite has been extensively used and analyzed by various researchers. Along with the test functions, the CEC 2005 suite specifies a detailed protocol how to evaluate a given search heuristic: the problem dimensions range from $n = 10 \ldots 50$, the number of allowed function evaluations is restricted, measures for success performance are defined, and examples of how to present the results in tables and figures are provided. Moreover, the test suite defines measures to assess the computational cost of a search heuristic. We refer to the original publication for the full description of the test suite [32]. So far, the CEC 2005 test suite has been made available in Matlab, C, and Java. As part of pCMALib, we introduce a stand-alone Fortran 90/95 module of the test suite. This module has been thoroughly tested and is in exact agreement with all previous implementations. The benchmark suite is an indispensable tool for testing new variants of (parallel island) CMA-ES.

3. THE FORTRAN CMA-ES LIBRARY

In this section we describe the software architecture and the MPI communication protocol used in pCMALib.

3.1 Sequential CMA-ES in Fortran

Due to the Eigen-decomposition of the covariance matrix, CMA-ES is a computationally expensive optimization heuristic, which benefits from an efficient implementation. Our library is implemented according to the Fortran 90/95 standard [1]. The code has been designed to be structured and easy to read. Clearly defined interfaces are used in order to facilitate the addition of new modules such as SI-operations or local optimizers. Our Fortran implementation of the standard CMA-ES follows N. Hansen's Matlab implementation (version 2.54) [12]. We use the same naming conventions, strategy parameter data structures, and program flow. A user who is familiar with the Matlab code can, thus, easily use the present implementation whenever computational speed is a concern.

We have verified correctness of our sequential CMA-ES implementation by direct comparison to the original Matlab routine. In a first test, we used an identical sequence of pseudo-random numbers to generate samples from a multivariate Gaussian in both the Matlab and the Fortran implementations, and we compared the optimization paths (starting from the same initial position) on the sphere function in various dimensions. The paths from the two implementations were identical within machine precision. A second test consisted of an iterated optimization of functions $F1$ and $F9$ of the CEC 2005 test suite using the protocol prescribed. Both implementations showed the same success statistics within the statistical error. A detailed analysis of the computational costs is given in Sec. 4.

3.2 Parallel (PS-)CMA-ES with MPI

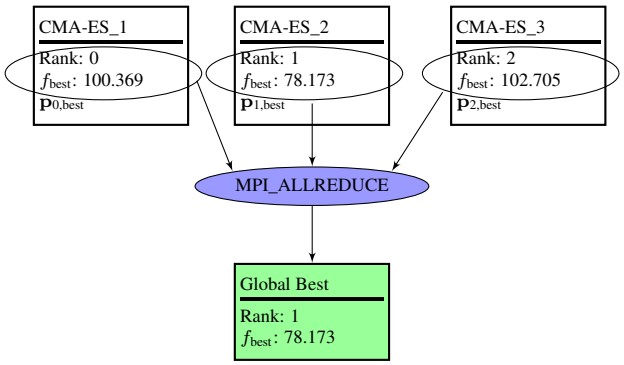
We provide parallel Fortran implementations of both the standard CMA-ES and the PS-CMA-ES. Both use an MIMD control structure, implemented using an SPMD programming paradigm, and a message passing communication model for best portability and efficiency. The architecture used in the present work parallelizes over entire CMA-ES instances, rather than over individual function evaluations, by using a (cooperative) parallel island model. Each CMA-ES instance runs in a separate process with separate memory address space. The processes are distributed over multiple processors using MPI [33]. The hardware-independence of MPI ensures portability of the library to distributed, shared, and hybrid memory machines [11] as well as loosely-coupled compute grids. On shared-memory machines, most MPI implementations substitute network communication by memory copy operations.

The rationale behind the chosen parallelization scheme is twofold. First, the application of evolutionary optimizers to any specific problem is always repeated multiple times because convergence to the global optimum is not guaranteed. A parallelization over complete CMA-ES instances simplifies the distribution of multiple optimization runs on computer grids and clusters. Second, and more importantly, it enables individual CMA-ES instances to communicate with each other and to migrate information about their current status. This can be used to improve the performance of CMA-ES runs while they are running. PS-CMA-ES as outlined in Sec. 2.2 is an example of such an extension. There, individual CMA-ES in-

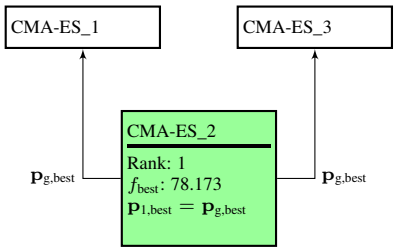
stances share their knowledge about the current GLOBAL_BEST solution at fixed generation intervals. This is implemented as explained in the next subsection. Since the GLOBAL_BEST communication is followed by an $n-$dimensional matrix rotation, the PS-CMA-ES algorithm is computationally demanding when multiple CMA-ES instances run on the same processor core and the dimension of the problem exceeds $n \approx 100$. A parallel implementation can substantially reduce the cost of PS-CMA-ES and leverage the computational resources offered by modern multi-core platforms. Due to the modular design of pCMALib, other parallel island models can be easily integrated in the software by adapting the subroutine responsible for migration.

3.2.1 Communication scheme in PS-CMA-ES

Each MPI process is identified by a unique number, called its *rank*. In order for each CMA-ES instance to inform the other swarm members (sub-populations) about its current best candidate solution, we use the communication scheme illustrated in Fig. 1. Each MPI process uses the 1-dimensional array F_BEST to store the current best fitness value and the process rank (Fig. 1(a), ellipses). The MPI collective communication *MPI_ALLREDUCE* is then used to find the global best function value within all F_BEST arrays. Since the array also contains the process rank, the corresponding CMA-ES instance is known. In a second step, this process broadcasts the position of the global best solution to the other swarm members (Fig. 1(b)). This broadcast is, however, only performed if the current global best value has improved.



(a) MPI Allreduce



(b) MPI Broadcast

Figure 1: MPI Communication of the global best solution

3.2.2 Excluding processes from communication

Since each process is a complete CMA-ES run, they can independently converge or meet other stopping criteria. In order to reduce the communication overhead in such cases, terminated CMA-ES instances are dynamically excluded from subsequent communications. This is based on the MPI concepts of process groups and communicators:

DEFINITION 3.1. *A **group** is an ordered set of processes. Each process in a group is associated with a unique integer rank. Rank values start at zero and go to N-1, where N is the number of processes in the group.*

DEFINITION 3.2. *A **communicator** consists of a group of processes and a globally unique ID (context). Processes within the same communicator can engage in collective communication operations. All MPI operations must specify a communicator.*

At the end of each CMA-ES generation, we check whether one or more processes have met a stopping criterion and are about to terminate. These processes are then excluded from execution as follows (Fig. 2):

1. The ranks of all terminating processes are determined.
2. *MPI_COMM_GROUP* and *MPI_GROUP_EXCL* are used to build a new process group that excludes these ranks.
3. A new communicator is created that contains only the group of running processes (*MPI_COMM_CREATE*).
4. The new rank of each process in the new communicator is determined.
5. All processes that are not in the new communicator are terminated.

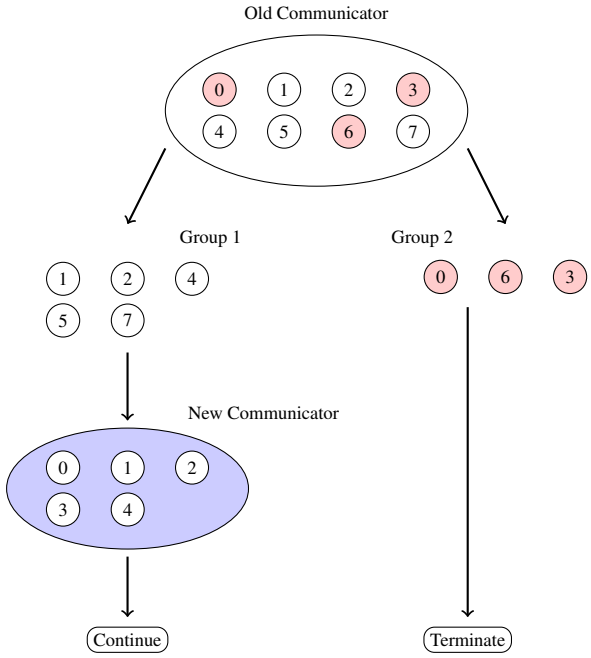


Figure 2: Dynamically excluding processes from communication

3.3 Technical details

The current implementation of pCMALib has been successfully tested on Windows XP SP2 (Intel Fortran 10.1/OpenMPI 1.3), MacOS X 10.4 (Intel Fortran 9.1 and 10.1/OpenMPI 1.2.6) and 10.5 (gfortran 4.4.0/OpenMPI 1.3), Gentoo Linux 2.6.25 (Intel Fortran 10.1/OpenMPI 1.2.8), Ubuntu Linux 8.10 (gfortran 4.3.2/OpenMPI 1.3) and OpenSolaris 2008.11 (Sun Ceres Fortran 95 8.3/OpenMPI

1.3) on both 32 bit and 64 bit architectures. Since all matrix operations in CMA-ES and PS-CMA-ES are local to a single processor core, the present library uses LAPACK and BLAS [4] for efficiently computing them. These libraries, along with an MPI library, must be available on the system in order to compile and use pCMALib.

pCMALib also interfaces with Matlab in two ways: First, it is able to store all output data in a structured binary Matlab file format (.mat), which allows analyzing and post-processing the data in Matlab. Second, it can evaluate and optimize fitness functions implemented in Matlab. This is done by opening and holding an inter-process connection to a Matlab engine. A function template is provided, which can be easily adapted to execute any Matlab command.

4. BENCHMARKS

We benchmark the computational performance and parallel efficiency of pCMALib on multi-core and distributed memory computers using computationally cheap explicit test functions. Notice that this provides a stringent assessment of the parallel scalability of the optimization algorithms themselves as their communication overhead is not masked by expensive function evaluations.

4.1 Multi-core shared memory

We first test the on-chip performance of the library on an Apple MacPRO with 2 dual-core 3 GHz Intel Xeon processors, a 4 MB L2-cache per processor, and 8×1 GB of RAM. The library was compiled with the Intel Fortran compiler version 9.1 and optimization level O3, and linked against OpenMPI version 1.2.6.

We follow the CEC 2005 test suite protocol to assess the computational efficiency of our implementation. Three time measures are defined in the protocol: T_0 is the CPU time for 1 000 000 standard mathematical operations, T_1 is the time needed to evaluate function $F3$ – a shifted, rotated, highly conditioned elliptic function – 200 000 times in dimensions $n = 10, 30, 50$, and $\hat{T}_2$ is the mean time over five executions of the complete algorithm with 200 000 evaluations of function $F3$ each. The computational cost of the algorithm is quantified by the ratio $(\hat{T}_2 - T_1)/T_0$.

We benchmark our implementations of the standard CMA-ES (on a single core), the parallel CMA-ES, and the parallel PS-CMA-ES. The standard CMA-ES is run with the standard strategy parameter settings [14] on a single core. The parallel CMA-ES benchmark uses 4 independent CMA-ES instances on the available 4 cores of the computer, without any communication between the instances. PS-CMA-ES is run with a swarm size of $S = 4$ on the 4 cores of the computer and the communication interval is set to the standard value of $I_c = 200$ [26]. The system configuration and CPU time measurements are summarized in Tab. 1, following the CEC 2005 test suite requirements [32]. For $n = 10$ and 30 we observe that the $\hat{T}_2$ of the three methods are comparable. For $n = 50$, the computational cost of PS-CMA-ES dominates due to the complexity of the $n-$dimensional matrix rotations. For comparison, we cite measurements of $\hat{T}_2$ for LR-CMA-ES and IPOP-CMA-ES determined by Auger and Hansen [5, 6] using MATLAB 7.0.1 on Red Hat Linux 2.4 running on a 3 GHz Intel Pentium 4 processor with 1 GB RAM. For $n = 10, 30, 50$, LR-CMA-ES took $\hat{T}_2$ = 51s, 45s, 68s, and IPOP-CMA-ES $\hat{T}_2$ = 17s, 24s, 56s, respectively.

4.2 Distributed memory

We assess the parallel efficiency of our implementations of CMA-ES and PS-CMA-ES on a distributed-memory computer cluster on the constrained random fitness landscape $F_{\text{rand}}(\mathbf{x}) = Y$, where $\mathbf{x}$ is defined in the bounded subset $[-100, 100]^n \in \mathbb{R}^n$. For any $\mathbf{x}$, Y is drawn from the uniform distribution $\mathcal{U}(-100, 100)$. Each

System	Mac OS X 10.4.11
CPU	2× Dual-Core Intel Xeon 3.00GHz
RAM	1GB
Language	Fortran 90

CMA-ES	T_0	T_1	$\hat{T}_2$	$(\hat{T}_2 - T_1)/T_0$
$n = 10$	9.53e-2	3.02e-1	2.71e+0	2.53e+1
$n = 30$		2.26e+0	1.13e+1	9.49e+1
$n = 50$		6.49e+0	3.04e+1	2.51e+2
Parallel CMA-ES				
$n = 10$		3.02e-1	3.96e+0	3.84e+1
$n = 30$		2.26e+0	1.39e+1	1.22e+2
$n = 50$		6.49e+0	3.53e+1	3.02e+2
PS-CMA-ES				
$n = 10$		3.02e-1	3.87e+0	3.75e+1
$n = 30$		2.26e+0	1.55e+1	1.39e+2
$n = 50$		6.49e+0	5.04e+1	4.61e+2

Table 1: System configuration and measured CPU times in seconds for standard CMA-ES, parallel CMA-ES, and parallel PS-CMA-ES. T_0 and T_1 characterize the computer platform and are independent of the algorithm used.

algorithm evaluates the fitness function 500 000 times (corresponding to drawing 500 000 uniformly distributed random numbers) on $N_{\text{proc}} = 1, \ldots, 64$ processor cores. The number of CMA-ES instances – or the swarm size in PS-CMA-ES – is always chosen equal to N_{proc} in order to avoid cache and memory congestion effects. Distributing a problem of fixed size onto an increasing number of processors measures the *strong* scaling of the algorithms, where the workload per processor decreases and the communication overhead increases. The random landscape F_{rand} ensures several properties that are indispensable for an unbiased assessment of the parallel scaling. First, the computational cost of evaluating the objective function is independent of the search dimension and the specific optimization path. Second, the random landscape guarantees that all CMA-ES instances experience the same search space. We perform three benchmarks with varying values of the strategy parameter I_c in order to disentangle the influence of the covariance matrix Eigen-decomposition and the MPI communication in PS-CMA-ES. The first set-up considers the standard parallel CMA-ES without swarm communication, i.e., $I_c = \infty$. The second benchmark evaluates the performance of the standard PS-CMA-ES with $I_c = 200$. Since I_c is in units of generations, and increasing S (N_{proc}) also increases the number of function evaluations per generation, the number of MPI communications performed in total during the fixed 500 000 function evaluations decreases. Therefore, the third set-up considers PS-CMA-ES with a constant number of MPI communication steps, independent of the swarm size S. This is achieved by setting $I_c = 200/S$. All three benchmarks are conducted in $n = 10, 30, 50, 100$ dimensions, but the figures for $n = 50$ are not shown since they are qualitatively similar to $n = 30$. The Fortran library is compiled with the Intel Fortran compiler version 10.1 and optimization level O3, and linked against OpenMPI version 1.2.8. The tests are performed on a Gentoo 2.6.25 Linux cluster consisting of 12 compute nodes. Each node contains 2 Intel Xeon 2.8 GHz quad-core processors (8 cores per node) with 2 GB of RAM per core. The nodes are connected by a dedicated Gigabit Ethernet network, entirely reserved for MPI communication (there is a second, identical network for system communication). TORQUE and Maui are used as resource manager and queuing system, respectively. In order to assess the influence of intra- vs. inter-node MPI communication, the scheduler is instructed to assign 8 MPI processes per node. Each benchmark is repeated $r = 1, \ldots, R$ times. For each repetition r, we measure the elapsed wall-clock time $t_{i,r}$ on each processor core

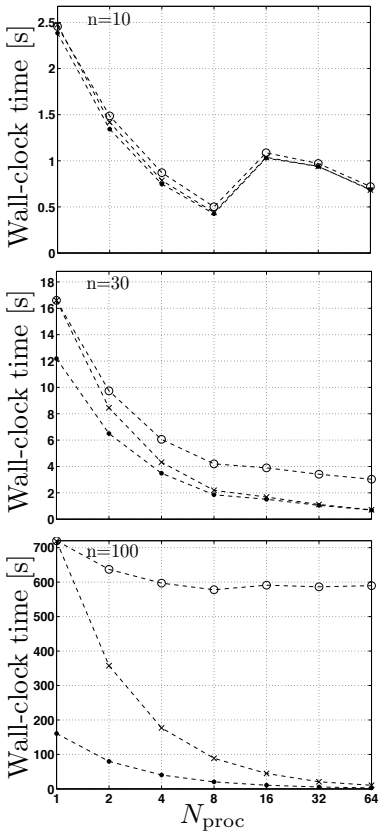


Figure 3: Overall run time $t(N_{\mathrm{proc}})$ in seconds for the parallel CMA-ES (•), and PS-CMA-ES with constant (○) and decreasing (×) number of MPI communications on the random landscape test problem in $n = 10, 30, 100$ dimensions. The number of processor cores N_{proc} is varied from 1 to 64. Each point is averaged from $R = 5$ runs. Error bars are comparable to symbol size and therefore not shown.

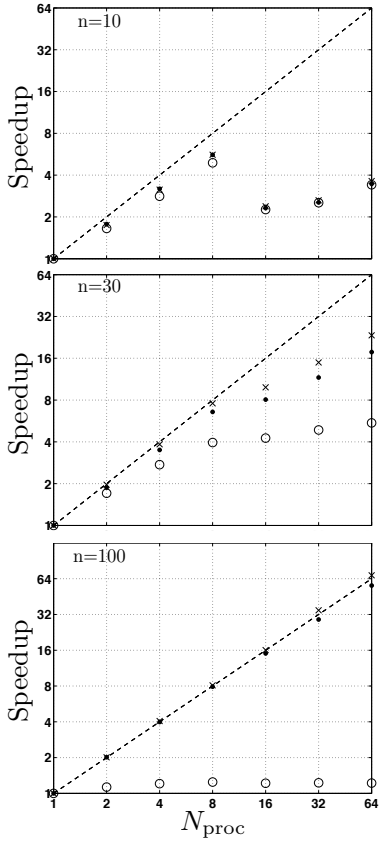


Figure 4: Parallel speedup s of the parallel CMA-ES (•), and PS-CMA-ES with constant (○) and decreasing (×) number of MPI communications on the random landscape test problem in $n = 10, 30, 100$ dimensions. The number of processor cores N_{proc} is varied from 1 to 64. Each point is averaged from $R = 5$ runs. Error bars are comparable to symbol size and therefore not shown.

$i = 1, \ldots, N_{\mathrm{proc}}$. The overall run time $t(N_{\mathrm{proc}})$ of the algorithm on N_{proc} processors is given by the maximum time over all processes, averaged over the R independent runs:

$$t(N_{\mathrm{proc}}) = \operatorname*{mean}_{r} \max_{i=1,\ldots,N_{\mathrm{proc}}} t_{i,r} \,. \tag{7}$$

From this, the parallel speedup s and efficiency e are defined as:

$$s(N_{\mathrm{proc}}) = \frac{t(1)}{t(N_{\mathrm{proc}})} \,, \quad e(N_{\mathrm{proc}}) = \frac{s(N_{\mathrm{proc}})}{N_{\mathrm{proc}}} \,. \tag{8}$$

The measured maximum wall-clock times for all 3 benchmarks are reported in Fig. 3, the speedups in Fig. 4, and the parallel efficiencies in Fig. 5.

In $n = 10$ dimensions, there are no noticeable differences between the three different test set-ups. Up to $N_{\mathrm{proc}} = 8$, i.e. on a single node, the wall-clock time decreases from 2.5s to below 0.5s. The speedup increases up to 6 and the efficiency decreases to 0.6–0.7. This should be compared to 37s for 50 000 function evaluations on $N_{\mathrm{proc}} = 4$ using the existing Matlab implementation [12]. The Fortran library thus is about 460 times faster than the Matlab implementation. When using two compute nodes (16 processes) and communicating over the network, the wall-clock time increases again, and speedup and efficiency drop considerably. This is expected as the network latency becomes the limiting factor for such a small test problem. The situation changes in higher dimensions. For $n = 30$, the wall-clock time of parallel CMA-ES decreases from 12s on a single core to below 1s on 64 cores. The two PS-CMA-ES tests need around 17s on a single core due to the additional construction of the rotation matrix. The PS-CMA-ES with constant number of MPI communications shows a similar scaling as the parallel CMA-ES, with an offset of about 4–5 seconds, corresponding to the constant communication overhead. The PS-CMA-ES with decreasing number of communications approaches the behavior of the standard parallel CMA-ES since, with increasing N_{proc}, the MPI communication overhead and the 30-dimensional rotations become negligible compared to the computational cost of CMA-ES. This is also reflected in the parallel speedup and efficiency. The standard PS-CMA-ES with $I_c = 200$ achieves the best efficiency (due to a higher computational cost on a single core), closely followed by the parallel CMA-ES. The existing Matlab implementation needed 75s for 50 000 function evaluations on $N_{\mathrm{proc}} = 4$, thus about 200 times longer. The same qualitative behaviour is observed in $n = 50$ (figures not shown), but, due to the higher computational cost, the parallel efficiency increases further. The computational costs for the basic CMA-ES operations and the matrix rotations now dominate, and the communication overhead becomes less apparent. On a single core, parallel CMA-ES needs

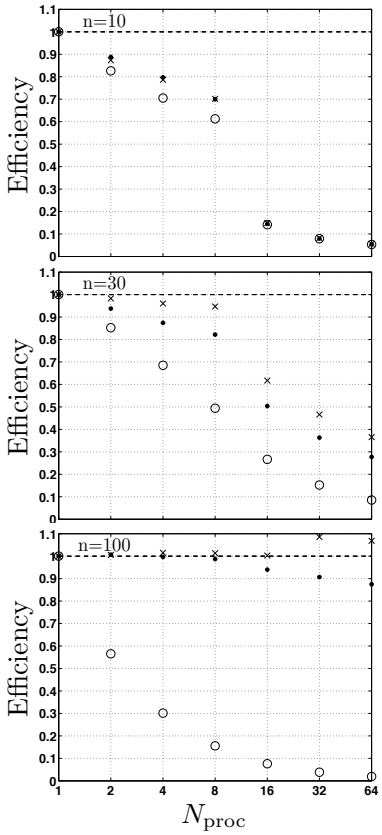


Figure 5: Parallel efficency e of the parallel CMA-ES (•), and PS-CMA-ES with constant (○) and decreasing (×) number of MPI communications on the random landscape test problem in $n = 10, 30, 100$ dimensions. The number of processor cores N_{proc} is varied from 1 to 64. Each point is averaged from $R = 5$ runs. Error bars are comparable to symbol size and therefore not shown.

40s and the two PS-CMA-ES variants around 68s. While the wall-clock time of the standard PS-CMA-ES rapidly approaches the one of CMA-ES for increasing N_{proc}, the PS-CMA-ES with a constant number of MPI communications shows an offset of around 25s due to the communication overhead and the 50-dimensional matrix rotation. The speedups of the parallel CMA-ES and the standard PS-CMA-ES for $N_{\mathrm{proc}} = 64$ are 40 and 50, respectively, corresponding to parallel efficiencies of 0.55 and 0.75. For comparison, the Matlab implementation required 187s for 50 000 function evaluations and hence was about 150 times slower than the Fortran library. For $n = 100$, the parallel scaling further improves. The efficiency for standard CMA-ES is 0.87 on 64 cores, while standard PS-CMA-ES achieves a super-linear efficiency of 1.07 due to the decreasing number of MPI communications. Using a constant number of MPI communications, the matrix rotations of the PSO update dominate, and almost no speedup is observed.

5. CONCLUSIONS AND OUTLOOK

We have presented pCMALib, a computationally efficient, scalable, and portable software library that implements (IPOP-/LR-)CMA-ES, parallel CMA-ES, and PS-CMA-ES, the latter as an example of a parallel island CMA-ES. The library also includes the first Fortran implementation of the CEC 2005 standard test suite. It is written in Fortran 90/95 and uses LAPACK, BLAS, and MPI for efficient linear algebra operations and inter-process communication. It features a run-time Matlab interface for easy visualization and interoperability, and employs a dynamic process grouping scheme to reduce the communication overhead of the PS-CMA-ES algorithm. pCMALib's parallelization paradigm is based on the synchronous cooperative island model. Parallelizing over entire CMA-ES instances enables migration of information every I_c generations, which can improve search performance using, e.g., concepts from swarm intelligence.

We have validated the correctness of the presented implementation by extensive comparison to an existing and well-tested reference implementation in Matlab, and we benchmarked the computational performance on multi-core and distributed memory parallel computers. Compared to the Matlab implementation, the presented library is several hundred times faster. Even on a small, 10-dimensional test problem, where the computational cost of evaluating the objective function is negligible, the library demonstrated excellent parallel efficiencies (strong scaling) of 0.6–0.7 on up to 8 cores of the same compute node. For larger problems, the efficiency further improves and is close to 1.0 in 100 dimensions. Also, the parallel efficiency is higher for weak scalings (where the problem size grows proportionally to the number of processors used) or when the computational cost of evaluating the objective function is larger. In such cases, the wall-clock time on a single processor core rapidly becomes prohibitive and the present library could provide an efficient and easily accessible tool to reduce the time-to-solution by exploiting the parallelism of modern computer platforms.

Ongoing and future work considers mixed multi-threading/ multi-processing paradigms in order to combine the present parallel island scheme with the classic master/slave protocol. Within each parallel process (corresponding to a CMA-ES instance), multiple threads can be used to perform several function evaluations in parallel. Such hybrid mechanisms could, e.g., be implemented using OpenMP or MPI-2, which supports one-sided communications and fork/join parallelism. We also plan to implement additional communication (migration) schemes and extend the library to asynchronous parallel island models. Moreover, pCMALib will be extended by further optimization algorithms and additional interfaces and language bindings, for example to python.

The library and its documentation will be made freely available as open-source software on www.mosaic.ethz.ch.

6. ACKNOWLEDGMENTS

We thank Jo A. Helmuth for valuable discussions and Steven Armstrong for technical support. Birte Schrader was funded by a grant from the Swiss National Science Foundation.

7. REFERENCES

[1] J. C. Adams, W. S. Brainerd, J. T. Martin, B. T. Smith, and J. L. Wagener. *Fortran 90 Handbook – Complete ANSI/ISO Reference*. Intertext Publications, McGraw-Hill Book Company, 1992.

[2] E. Alba. *Parallel Metaheuristics: A New Class of Algorithms*. Wiley-Interscience, 2005.

[3] E. Alba, G. Luque, J. Garcia-Nieto, G. Ordonez, and G. Leguizamon. MALLBA: a software library to design efficient optimisation algorithms. *International Journal of Innovative Computing and Applications*, 1(1):74–85, 2007.

[4] E. Anderson. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, 1992.

[5] A. Auger and N. Hansen. A Restart CMA Evolution Strategy with Increasing Population Size. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, volume 2, pages 1769–1776, 2005.

[6] A. Auger and N. Hansen. Performance Evaluation of an Advanced Local Search Evolutionary Algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, volume 2, pages 1777–1784, 2005.

[7] P. Bayer, C. Bürger, and M. Finkel. Computationally efficient stochastic optimization using multiple realizations. *Advances in Water Resources*, 31(2):399 – 417, 2008.

[8] S. Cahon, N. Melab, and E. G. Talbi. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

[9] R. Dölling, H. Mielenz, and C. L. Müller. Efficient simulation of automotive multi-physical systems by optimization with evolutionary algorithms. In *Proceedings of the International Conference on Applied Simulation and Modeling (ASM)*. IASTED, Acta Press, 2007.

[10] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.

[11] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, München; Wien, 2nd edition, 1999.

[12] N. Hansen. The CMA Evolution Strategy. http://www.lri.fr/ hansen/cmaesintro.html.

[13] N. Hansen. Compilation of Results on the 2005 CEC Benchmark Function Set. Technical report, Computational Laboratory (CoLab), Institute of Computational Science, ETH Zurich, 2006.

[14] N. Hansen. The CMA Evolution Strategy: A Tutorial. http://www.lri.fr/ hansen/cmatutorial.pdf, 2007.

[15] N. Hansen and S. Kern. Evaluating the CMA Evolution Strategy on Multimodal Test Functions. In *Lecture Notes in Computer Science*, volume 3242 of *Parallel Problem Solving from Nature - PPSN VIII*, pages 282–291, Berlin, Heidelberg, 2004. Springer.

[16] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[17] N. Hansen and A. Ostermeier. Completely Derandomized Self-Adaption in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[18] C. Igel, V. Heidrich-Meisner, and T. Glasmachers. Shark. *Journal of Machine Learning Research*, 9:993–996, 2008.

[19] C. Igel, T. Suttorp, and N. Hansen. A computational efficient covariance matrix update and a (1+1)-CMA for evolution strategies. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 453–460, Seattle, Washington, USA, 2006. ACM.

[20] X. Ji and Y. Xu. libSRES: a C library for stochastic ranking evolution strategy for parameter estimation. *Bioinformatics*, 22(1):124–126, 2006.

[21] J. N. Knight and M. Lunacek. Reducing the space-time complexity of the CMA-ES. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 658–665, London, England, 2007. ACM.

[22] D. Levine. User's Guide to the PGAPack Parallel Genetic Algorithm Library. Technical Report ANL-95/18, Mathematics and Computer Science Div., Argonne National Laboratory., Argonne, Illinois., June 23, 1995.

[23] M. Lunacek and D. Whitley. The Dispersion Metric and the CMA Evolution Strategy. In *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 477–484, Seattle, Washington, USA, 2006. ACM Press.

[24] K. Maier, U. Hönig, and W. Schiffmann. A fault-tolerant parallel optimization tool based on an evolutionary strategy. In T. F. Gonzalez, editor, *Proceeedings of the Parallel and Distributed Computing and Systems conference*. IASTED, Acta Press, November 2008.

[25] C. G. Moles, P. Mendes, and J. R. Banga. Parameter Estimation in Biochemical Pathways: A Comparison of Global Optimization Methods. *Genome Research*, 13(11):2467–2474, 2003.

[26] C. L. Müller, B. Baumgartner, and I. F. Sbalzarini. Particle Swarm CMA Evolution Strategy for the Optimization of Multi-Funnel Landscapes. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, Trondheim, Norway, 2009.

[27] I. Rechenberg. *Evolutionsstrategie; Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart–Bad Cannstatt, 1973.

[28] R. Ros and N. Hansen. A Simple Modification in CMA-ES Achieving Linear Time and Space Complexity. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature*, pages 296–305, Berlin, Heidelberg, 2008. Springer-Verlag.

[29] G. Rudolph. On Correlated Mutations in Evolution Strategies. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)*, pages 105–114, Amsterdam, 1992. Elsevier.

[30] I. F. Sbalzarini, S. D. Müller, P. D. Koumoutsakos, and G.-H. Cottet. Evolution strategies for computational and experimental fluid dynamic applications. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1064–1071, San Francisco, CA, USA, 2001. Morgan Kaufmann, San Francisco.

[31] C. Spieth, F. Streichert, N. Speer, and A. Zell. Utilizing an Island Model for EA to Preserve Solution Diversity for Inferring Gene Regulatory Networks. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2004)*, pages 146–151, 2004.

[32] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Technical report, Nanyang Technological University, Singapore, May 2005.

[33] University of Tennessee, Knoxville, Tennessee. *MPI: A Message-Passing Interface Standard*, 1995.

[34] D. Wales and H. Scheraga. Review: Chemistry – Global optimization of clusters, crystals, and biomolecules. *Science*, 285(5432):1368–1372, AUG 27 1999.