Journal of Computational Science 17 (2016) 599–608

Contents lists available at ScienceDirect

Journal of Computational Science

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

Novel parallelization of simulated annealing and Hooke & Jeeves search algorithms for multicore systems with application to complex fisheries stock assessment models

Sergio Vázquez[a], María J. Martín[a,*], Basilio B. Fraguela[a], Andrés Gómez[b], Aurelio Rodríguez[b], Bjarki Þór Elvarsson[c]

[a] Grupo de Arquitectura de Computadores, Universidade da Coruña, Spain
[b] Galicia Supercomputing Center (CESGA), Santiago de Compostela, Spain
[c] Marine Research Institute, Reykjavik, Iceland

ARTICLE INFO

Article history:
Received 29 October 2015
Received in revised form 22 April 2016
Accepted 7 July 2016
Available online 19 July 2016

Keywords:
Marine ecosystems
Gadget
Simulated annealing
Hooke & Jeeves
Multicore systems
OpenMP

ABSTRACT

Estimating parameters of a statistical fisheries assessment model typically involves a comparison of disparate datasets to a forward simulation model through a likelihood function. In all but trivial cases the estimations of these models tend to be time-consuming due to issues related to multi-modality and non-linearity. This paper develops novel parallel implementations of popular search algorithms, applicable to expensive function calls typically encountered in fisheries stock assessment. It proposes two versions of both Simulated Annealing and Hooke & Jeeves optimization algorithms with the aim of fully utilizing the processing power of common multicore systems. The proposals have been tested on a 24-core server using three different input models. Results indicate that the parallel versions are able to take advantage of available resources without sacrificing the quality of the solution.

1. Introduction

Statistical fisheries models typically involve a comparison of the output of a non-linear model of fish population dynamics with available data through a likelihood function. The choice of an optimization algorithm for these types of model can be far from trivial. In all but the simplest examples, such as stock production models [1], where the data required to contrast with the model is relatively small, the time required for adequate parameter estimation can be substantial and estimation issues, such as multi-modal likelihoods, can increase the complexity further (see [2] and references therein).

Commonly the parameter estimation procedure involves a combination of search algorithms, both global and local, in an attempt to combine the strengths of a global search with the speed of a local search algorithm. Various combinations of search algorithms have been investigated to identify an optimal search procedure for a specific task (e.g. [3,4]), but these investigations indicate that a particular combination is problem specific.

Various tools have been developed to aid in the creation of statistical stock assessment models. One such tool is Gadget (Globally applicable Area Disaggregated General Ecosystem Toolbox), which is a modeling environment designed to build models for marine ecosystems, including both the impact of the interactions between species and the impact of fisheries harvesting the species [5,6]. It is an open source program written in C++ and it is freely available from the Gadget development repository at www.github.com/hafro/gadget.

Gadget works by running an internal model based on many parameters describing the ecosystem, and then comparing the output from this model to observed measurements to obtain a goodness-of-fit likelihood score. By using one or several search algorithms it is possible to find a set of parameter values that gives the lowest likelihood score, and thus better describes the modeled ecosystem. The optimization process is the most computationally intensive part of the process as it commonly involves repeated evaluations of the computationally expensive likelihood function, as the function calls a full ecosystem simulation for comparison to data. In addition to that, multiple optimization cycles are sometimes performed to ensure that the model has converged to an

[*] Corresponding author.
E-mail addresses: sergio.vazquez@udc.es (S. Vázquez), mariam@udc.es (M.J. Martín), basilio.fraguela@udc.es (B.B. Fraguela), agomez@cesga.es (A. Gómez), aurelio@cesga.es (A. Rodríguez), bthe@hafro.is (B.Þó. Elvarsson).

http://dx.doi.org/10.1016/j.jocs.2016.07.003
1877-7503/© 2016 Elsevier B.V. All rights reserved.

optimum as well as to provide opportunities to escape from a local minimum, using heuristics such as those described by [7]. Once the parameters have been estimated, one can use the model to make predictions on the future evolution of fish stocks.

Gadget can be used to assess a single fish stock or to build complex multi-species models. It has been applied in many ecosystems such as the Icelandic continental shelf area for the cod and redfish stocks [7–9], the Bay of Biscay in Spain to predict the evolution of the anchovy stock [10], the North East Atlantic to model the porbeagle shark stock [11], or the Barents Sea to model dynamic species interactions [12]. Models developed using the Gadget framework have also been used to provide tactical advice on sustainable exploitation levels of a particular resource. Notably the southern European Hake stock, Icelandic Golden redfish, Tusk and Ling stocks, and Barent sea Greenland halibut are formally assessed by the International Council for the Exploration of the Sea (ICES) using Gadget (see [13–16] for further details).

The aim of this work is to speedup the costly optimization process of Gadget so that more optimization cycles can be performed and a more reliable model can be achieved. The methodology followed comprises three main stages: First, the code was profiled in order to identify its bottlenecks; then, sequential optimizations were applied to reduce these bottlenecks; finally, the most used optimization algorithms were parallelized.

Currently, there are three different optimization algorithms implemented in Gadget: the Simulated Annealing (SA) algorithm, based on [17] and [18], the Hooke & Jeeves (H&J) algorithm [19], and the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [20]. All of them can be used alone or combined into a single hybrid algorithm. Combining the wide-area searching characteristics of SA with the fast local convergence of H&J is, in general, an effective approach to find an optimum solution. In this work the performance of the objective function has been improved and both algorithms have been parallelized using OpenMP [21], so that the proposed solution can take advantage of today's ubiquitous multicore systems. Namely, the main contributions of this work are:

- A sequential optimization of Gadget that reaches speedups of around 3× for realistic models.
- Both deterministic reproducible and speculative parallelizations for shared-memory systems of SA that include the improvements for continuous variables proposed in [17,18]. As we will see in Section 4, these parts of the algorithm, which are not present in problems defined on discrete variables, were the most problematic by far for the speculative parallelization.
- Both deterministic reproducible and speculative parallelizations for shared-memory systems of H&J.
- A detailed evaluation using actual data from different institutions and up to 24 cores.

The rest of this paper is organized as follows. Section 2 covers related work. Section 3 describes the sequential optimizations applied. Sections 4 and 5 describe the basics of the SA and the H&J search algorithms and discuss the parallelization strategies proposed for each one of them. Section 6 describes the objective function. Section 7 presents the experimental evaluation of all the proposals on a multicore system. Finally, conclusions are discussed in Section 8.

2. Related work

A large number of solutions exist for the parallel implementation of the SA algorithm on distributed memory systems [22–26]. In [25] five different parallel versions are implemented and compared. The main difference among them is the number of communications required and the quality of the solution. The solution is better when the communication among the processes increases. However, in distributed memory systems the communication overhead plays a important role in the global performance of the parallel algorithm. Thus, solutions where communication takes place after every move are not considered a good alternative on these architectures. In shared-memory systems, however, this kind of solution is viable in terms of computational efficiency, as in this case the communications are performed through the use of shared variables.

There are also parallelization proposals for shared-memory systems [27–30] and for clusters of SMP nodes [31], this latter one being based on a combination of MPI [32] and OpenMP. Also, some researchers have explored separate distributed and shared-memory parallelization schemes for the same problem [33]. The SA algorithms considered in these works are suitable for the resolution of problems defined on discrete variables, but they lack the critical improvements proposed in [17,18] to deal with continuous variables, which are required in Gadget. Also, to the best of our knowledge, this work is the first to propose and compare reproducible and speculative parallel versions of the algorithm.

As regards the Hooke & Jeeves method, different parallel versions of similar direct search algorithms exist in the literature, see for instance [34–39]. These parallel proposals can be classified into synchronous [34–36] or asynchronous methods [37–39]. The use of asynchronous strategies can lead to an improvement in efficiency when the time needed to evaluate the objective function depends on the point to be evaluated. In the optimization problem studied in this work, the objective function evaluations finish in approximately the same amount of time for all the points and thus, synchronous solutions will be proposed for the parallel execution of the H&J method.

Paramin [40,6] is a parallel version of Gadget originally written with PVM [41], and later migrated to MPI [32]. As the parallel version proposed in this paper, Paramin is based on the distribution of the function evaluations among the different processes. However, unlike our approach, in each parallel step it updates all the parameters whose moves lead to a better likelihood value. Moves that are beneficial one by one could become counterproductive when they are applied together. Thus, this parallel version leads to a worse likelihood score in some cases, an important problem that does not happen in our version. Additionally, Paramin is meant to be executed in a distributed infrastructure, such as a cluster of computers, and it needs an installed version of the MPI library. On the other hand, our approach focuses on taking advantage of common desktop multicore systems and it only needs to add the appropriate OpenMP compile-time flag to the selected compiler, reducing the computing expertise needed to profit from it.

3. Optimization of the sequential program

The profiling of the original application revealed that it could be optimized mainly in three aspects. First, some light class member functions that were invoked very often in the application from other classes could not be inlined by the compiler because they were not expressed in the header files that describe their associated classes, but in the associated implementation files. Moving their definition to the header file allowed their effective inlining, largely reducing their weight in the runtime.

Second, we found several opportunities to avoid recalculations of values, i.e., computations that are performed several times on the same inputs, thus yielding the same result. While some of these opportunities could be exploited just by performing loop-invariant code motion, other more complex situations required a more elaborated approach such as dynamically building tables to cache the

```
Algorithm SimulatedAnnealing($\overrightarrow{p}, \overrightarrow{VM}, ns, T, nt, MaxIterations$)
    input  : - initial vector of $N$ parameters $\overrightarrow{p} = p_i$, $1 \le i \le N$
             - step lengths vector with one element per parameter $\overrightarrow{VM} = VM_i$, $1 \le i \le N$
             - frequency of $\overrightarrow{VM}$ updates $ns$
             - temperature $T$
             - frequency of temperature $T$ reductions $nt$
             - maximum number of evaluations $MaxIterations$
    output: optimized vector of $N$ parameters $\overrightarrow{p} = p_i$, $1 \le i \le N$
    $iterations = 0$
    $param_i = i$, $1 \le i \le N$                          //permutation of parameters
    $bestLikelihood =$ evaluate($\overrightarrow{p}$)
    while not terminationTest($bestLikelihood, \overrightarrow{p}$) do
        for $a = 1$ to $nt$ do
            reorder($\overrightarrow{param}$)
            for $j = 1$ to $ns$ do
                for $i = 1$ to $N$ do
                    $iterations = iterations + 1$
                    $np = param_i$                          //parameter to modify
                    $\overrightarrow{tmp\_p} = \overrightarrow{p}$
                    $tmp\_p_{np} =$ adjustParam($p_{np}, \overrightarrow{VM}$)
                    $likelihood =$ evaluate($\overrightarrow{tmp\_p}$)
                    accept($T, bestLikelihood, likelihood, \overrightarrow{p}, \overrightarrow{tmp\_p}$)
                    if $iterations = MaxIterations$ then
                        return $\overrightarrow{p}$
                    end
                end
            end
            adjustVM($\overrightarrow{VM}$)
        end
        reduceTemperature($T$)
    end
    return $\overrightarrow{p}$
```

Fig. 1. Simulated annealing algorithm.

results of these calculations and later resorting to these tables to get the associated results.

The third most important optimization applied consisted in modifying the data structure used to represent bidimensional matrices in the heaviest functions. The original code used a layout based on an array of pointers, each pointer allowing the access to a row of the matrix that had been separately allocated. These matrices were changed to be stored using a single consecutive buffer, thus improving the locality, avoiding indirections and helping the compiler to reason better on the code.

4. Parallelization of the simulated annealing algorithm

The SA algorithm used in Gadget is a global optimization method that tries to find the global optimum of a function of N parameters. At the beginning of each iteration, this search algorithm decides which parameter np is going to be modified to explore the search space. Then, the algorithm generates a trial point with a value of this parameter that is within the step length given by element np of vector $\vec{VM}$ (of length N) of the user selected starting point by applying a random move. The function is evaluated at this trial point and its result is compared to its value at the initial point. When minimizing a function, any downhill step is accepted and the process repeats from this new point. An uphill step may be also accepted to escape from local minimum. This decision is made by the Metropolis criteria. It uses a variable called Temperature (T) and the size of the uphill move in a probabilistic manner to decide the acceptance. The larger T and the size of the uphill move are, the more likely that move will be accepted. If the trial is accepted, the algorithm moves on from that point. If it is rejected, another point is chosen instead for a trial evaluation. Each ns evaluations of the N parameters the elements of $\vec{VM}$ are adjusted so that half of all function evaluations in each direction will be accepted. Thus, the modification of each element of $\vec{VM}$ depends on the number of moves accepted along that direction. Also, each nt adjustments to $\vec{VM}$ the temperature is reduced. Thus, a temperature reduction occurs every $ns \times nt$ cycles of moves along every direction (each $ns \times nt \times N$ evaluations of the function). When T declines, uphill moves are less likely to be accepted and the percentage of rejections rises. Given the scheme for the selection for $\vec{VM}$, $\vec{VM}$ falls. Thus, as T declines, $\vec{VM}$ falls and the algorithm focuses upon the most promising area for optimization. The process stops when no more improvement can be expected or when the maximum number of function evaluations is reached. Notice that some of the most important portions of this algorithm, such as those related to ns and $\vec{VM}$ are not part of the basic SA algorithm, but required for the optimized implementation of SA on functions defined on continuous variables as described in [17,18]. A pseudocode of the SA algorithm used in Gadget is shown in Fig. 1.

This algorithm was parallelized using OpenMP. The followed parallelization pattern consisted in subdividing the search process in a number of steps that are applied in sequence, parallelism being exploited within each step. Since the expensive part of a search algorithm is the evaluation of the fitness function, the parallelism is exploited by distributing the function evaluations across the available threads. The results of the evaluations are stored in shared variables so that all the values can be analyzed by the search algorithm in order to decide whether to finish the search process, and if this is not the case, how to perform the next search step.

This work aims to reduce the execution time without diminishing the quality of the solution. For this purpose, two parallel algorithms were developed: the reproducible and the speculative algorithm. The first one follows exactly the same sequence as the original serial algorithm, and thus it finishes in the same point and with the same likelihood score. The speculative version, however, is allowed to change the parameter modification sequence in order to increase the parallelism, and thus it can finish in a different point and with a different likelihood value. In this latter case, some specification variables of the sequential algorithm were adapted to prevent the algorithm from converging to a worse likelihood value.

In the parallel versions not all the function evaluations provide useful information to the search process. Thus, the number of `effective` evaluations is taken into account as ending criterion instead of the number of total evaluations.

The same parallelization strategies have been applied to the H&J algorithm, as described in Section 5.

4.1. Reproducible version

In this version each thread performs the evaluation of the function with the modification of a different parameter in parallel. For example, with four threads, in the first step, the thread 1 could perform the evaluation changing the first parameter, the thread 2 the second parameter, the thread 3 the third parameter, and the thread 4 the fourth parameter.

From these evaluations the algorithm moves to the first point (in sequential order) that is accepted (directly or applying the Metropolis criteria), dismissing the others. As a result, all the evaluations up to the first one with an accepted point are taken into account to advance in the search process, and for this reason they will be counted as effective evaluations, while all the subsequent simulations are discarded. For the previous example, if the modification to the first parameter is rejected and the modification to the second parameter is accepted, the calculations performed by threads 1 and 2 are considered, and the calculations by threads 3 and 4 are discarded, even if one of them obtains a better likelihood that the obtained by thread 2. In this case, 4 evaluations are performed in parallel but only 2 of them are recorded as effective evaluations.

Following our example, since the last evaluation considered modified the second parameter, in the next step, thread 1 will perform the evaluation modifying the third parameter, thread 2 the fourth parameter, thread 3 the fifth parameter, and thread 4 the sixth parameter. In order to obtain the same result as in the sequential algorithm, the parameters re-evaluated will take the same value as in the previous step. This required changing the way of generating the random values. Namely, while the sequential version relies on `rand`, the reproducible version uses a random function which generates its result from a seed provided by the user. This allows to re-generate previously generated random values by providing the same seed, which is recorded by our implementation for this purpose. Another change was that while the random numbers of the sequential version follow a single sequence, the parallel version uses three different seeds, giving place to three sequences of random numbers. One seed (`seedP`) is used to change the order in which the parameters are modified, another one (`seedM`) is used for the acceptance of the Metropolis criteria, and the last one (`seed`) is used for the calculation of the new value of the parameters.

4.2. Speculative version

As in the reproducible version, each thread performs the evaluation with the modification of a different parameter in parallel, but now the move with the best likelihood is selected. This point will be accepted if it obtains a better likelihood than the initial point. Otherwise, the Metropolis criteria is applied to decide its acceptance. For example, with 4 threads, in the first step thread 1 could perform the simulation changing the first parameter, thread 2 the second parameter, thread 3 the third parameter and thread 4 the fourth parameter. If thread 3 obtains the best likelihood, the modification of the third parameter will be the one accepted and the other modifications will be discarded. In the next parallel step, thread 1 will work with a change in the fifth parameter, thread 2 in the sixth parameter, thread 3 in the seventh parameter and thread 4 in the eighth parameter.

Note that the speculative version performs and considers evaluations that are never performed in the sequential version. This makes it follow search paths that are different from those of the sequential and the reproducible versions. For this reason, this algorithm can finish in a different point and with a different likelihood value.

In this version, unlike in the reproducible version, some discarded simulations provide information to the search process even though they do not modify the parameters. This forced us to adapt some of the most important variables used during the search process. These variables, which we describe providing their semantics in the sequential version, are:

$\vec{nacp}$ vector that stores for each parameter the number of times a change in its value has been accepted. It affects the calculation of the step lengths vector $\vec{VM}$, explained at the beginning of Section 4. Namely, the higher $\vec{nacp}$, the larger value will have $\vec{VM}$ and the changes performed in the parameter will be bigger.

ns scalar that provides the frequency of updates to $\vec{VM}$.

For this parallel version the above variables were changed as follows:

$\vec{nacp}$ Its value for a given parameter increases whenever (a) the change evaluated in that parameter improves the likelihood, or (b) the change in that parameter does not improve the likelihood but it happens to be the best one and it is accepted applying the Metropolis criteria. Notice that the second situation implies that none of the changes improved the likelihood.

ns In the parallel algorithm not all the parameter modifications provide useful information to the search process. Thus, to avoid to change the step length associated to each parameter ($\vec{VM}$ vector) too often, in addition to the global scalar `ns`, a vector $\vec{vns}$ has been defined with the aim of processing the step length individually. It is increased whenever (a) the change simulated in the parameter improves the likelihood, or (b) the parameter change is rejected (also applying the Metropolis criteria). This way, it is increased in all the situations except when the change would be accepted by the Metropolis criteria.Also, in order to avoid decreasing the temperature too fast, which would lead to a premature halt of the algorithm, it is necessary to take into account that the number of discarded evaluations increases with the number of threads used in the parallel algorithm. For this reason, in the parallel algorithm each temperature iteration consists of $ns \times numThreads/k$ parallel steps, that is, $ns \times numThreads/k \times numThreads$ evaluations, where ns is the scalar with the same value as in the sequential version, $numThreads$ is the number of threads used to execute the parallel version and k is a constant to be determined experimentally. We have found that $k = 2$ leads to good results in all the cases.

Finally, the counter of the number of effective evaluations increases every time a parameter change is rejected (also applying the Metropolis criteria) and once for every parallel step.

```
Algorithm HookeJeeves($\overrightarrow{p}$, $\overrightarrow{delta}$, $MaxIterations$, $rho$)
    input  : - initial vector of $N$ parameters $\overrightarrow{p} = p_i$, $1 \le i \le N$
             - modification length for each parameter $\overrightarrow{delta} = delta_i$, $1 \le i \le N$
             - maximum number of evaluations $MaxIterations$
             - reduction control variable $rho$
    output: optimized vector of $N$ parameters $\overrightarrow{p} = p_i$, $1 \le i \le N$
    $iterations = 0$
    $\overrightarrow{tmp\_p} = \overrightarrow{p}$
    $bestLikelihood$ = evaluate($\overrightarrow{p}$)
    $initialLikelihood = bestLikelihood$
    while not terminationTest($iterations$, $MaxIterations$, $bestLikelihood$, $\overrightarrow{p}$) do
        $iterations = iterations + 1$
        for $i = 1$ to $N$ do
            $tmp\_p_i = p_i + delta_i$
            $likelihood$ = evaluate($\overrightarrow{tmp\_p}$)
            if $likelihood < bestLikelihood$ then
                $bestLikelihood = likelihood$
            else
                $tmp\_p_i = p_i - delta_i$
                $likelihood$ = evaluate($\overrightarrow{tmp\_p}$)
                if $likelihood < bestLikelihood$ then
                    $bestLikelihood = likelihood$
                else
                    $tmp\_p_i = p_i$
                end
            end
        end
        adjust($\overrightarrow{p}$, $\overrightarrow{tmp\_p}$, $\overrightarrow{delta}$, $rho$, $initialLikelihood$, $bestLikelihood$)          //includes pattern moves
        $\overrightarrow{p} = \overrightarrow{tmp\_p}$
    end
    return $\overrightarrow{p}$
```

Fig. 2. Hooke & Jeeves algorithm.

5. Parallelization of the Hooke & Jeeves algorithm

Hooke and Jeeves (H&J) [19] is a pattern search method that consists in a sequence of exploratory moves from a base point, followed by pattern moves that provide the next base point to explore.

The exploratory stage performs local searches in each direction by changing a single parameter of the base point in each move. For each parameter, the algorithm considers first an increment *delta* in the positive direction. If the function value in this point is better than the old one, then the algorithm selects this new point like the new base point. Otherwise, an increase *delta* is done in the negative direction, and if the result is better than the original one, then the algorithm selects this new point as the base point. Otherwise the algorithm keeps the initial value and proceeds to evaluate changes in the next parameter. Once all the parameters have been explored, the algorithm continues with the pattern moves stage.

In the pattern stage, each one of the parameters is increased by an amount equal to the difference between the present parameter value and the previous one (its value before the exploratory stage). The aim is to move the base point towards the direction that improved the result during the previous stage. Then, the function is evaluated in this new point. If the function value is better, this point becomes the new base point for the next exploratory moves. Otherwise, the pattern moves are ignored.

The search proceeds in series of these two stages until a minimum is found or the maximum number of evaluations is reached. If after a exploratory stage the base point does not change, *delta* is reduced. The amount of the reduction is determined by a user-supplied parameter called *rho*. Taking big steps gets to the minimum more quickly, at the risk of stepping right over an excellent point. Fig. 2 summarizes this search algorithm.

Our parallel versions use an even number of threads to parallelize the exploratory stage. Namely, when our parallel versions are run using *numThreads* threads, each parallel step explores in parallel the movements in the positive and in the negative direction of *numThreads*/2 consecutive parameters. For example, if we have 4 threads, in the first parallel evaluation one thread will evaluate parameter 1 + *delta*, another one the parameter 1 – *delta*, another one the parameter 2 + *delta* and finally another one the parameter 2 – *delta*.

5.1. Reproducible version

The reproducible version follows exactly the same sequence as the original sequential algorithm. This version chooses the first move (in sequential order) that improves the likelihood as base point for the next search step. All the subsequent evaluations are disregarded and they will not be taken into account to increase the counter of the number of effective evaluations.

For the previous example using 4 threads, if the first evaluation that obtains a better likelihood corresponds to the movement of the first parameter in the negative direction (parameter 1 – *delta*), the two evaluations using the second parameter are discarded (parameter 2 + *delta* and parameter 2 – *delta*). The next step will start from parameter 2 and it will evaluate the moves in the positive and negative direction of both parameters 2 and 3.

5.2. Speculative version

This version chooses the move that gives place to the best likelihood as base point for the next search step. Since all the evaluations are taken into account to choose this value, if *n* was the last parameter considered in a parallel step, this version always starts the evaluations of the next parallel step from parameter *n* + 1.

For this version the counter of the number of effective evaluations is only increased in two situations. First, for every parameter in which none of the movements improved the likelihood, the counter is increased by two units, to account for the two movements tested. Second, among all those movements that improve the likelihood, only the best one in the parallel step is considered. If this movement is in the positive direction, the counter is increased by one unit, otherwise it is increased by two units, because negative movements are always evaluated after a positive movement has been discarded.

Finally, in both the reproducible and the speculative version, the outer loop that iterates on the parameters to perform the exploratory moves has step *numThreads*/2, since each parallel step evaluates the two possible movements for *numThreads*/2 parameters.

6. The objective function

A model developed using the Gadget framework consists of forward simulation models that provide a full simulation of the ecosystem for a given parameter value and whose outputs are then compared to observations through a weighted likelihood function. As in many fisheries applications, no single data source or type is used in the estimation process, but a disparate set of data, sources and types. Typically a Gadget model uses some index of biomass/abundance that is regressed against the modelled biomass/abundandce trends, (age-)length distributions from the various fleets and often proportion mature and/or sex ratios. The appendix to [9] gives a detailed description of the typical processes of a model and how the likelihood function is composed.

Here three different models were used to evaluate the performance of the modified search algorithms:

HAKE This model is used by the IEO (Spanish Institute of Oceanography) to assess the southern hake stock and give catch advice through ICES.

TUSK It is a single-species model of tusk (*brosme brosme*) in Icelandic waters developed by the MRI (Marine Research Institute, Iceland) which is used by ICES as the basis for catch advice.

HADDOCK It is a single-species, single-area model used to model the Icelandic haddock. It is the example model provided with Gadget. It is available for download from the Gadget website.

The three models are fairly different in characteristics. For example, HADDOCK is a toy example used for illustrative purposes and its parameter space is fairly limited. The TUSK model is an actual assessment model used to give tactical advised and is based on over thirty years of data, while the HAKE model extends the complexity by examining complicated fleet interactions and discards.

7. Experimental results

The different parallel implementations were evaluated in an HP ProLiant XL230a Gen9 server running the Scientific Linux release 6.4. It consists of two processors Intel Haswell E5-2680 at 2.5 GHZ with 12 cores per processor and 64 GB of memory. All the codes were compiled with the gnu g++ compiler version 4.9.1 and the compilation flag -O3.

Table 1 describes the parameters used in our tests of the SA algoritm. It includes the input parameters found in the description in Section 4 and the temperature reduction factor *rt*. As for HJ, all our tests used *MaxIterations* = 100, 000, *rho* = 0.5 and $\vec{delta} = (2 * (\vec{rand} \bmod 2) - 1) * rho$ where $\vec{rand}$ is the same vector of random integers in all the cases. In both algorithms we set up the parameters so that in all the experiments the algorithms always stopped after making *MaxIterations* effective evaluations and we used the same value of this parameter (100,000) for all the models. This was done in order to facilitate the comparisons among different code versions and different numbers of threads for the same model. It is also important to notice that in both search algorithms each model optimizes a different number of parameters due to its own nature, this value being between 38 (HADDOCK) and 62 (HAKE), and each parameter may have different valid ranges in each model.

Table 1
SA parameters.

Parameters	Model		
	HAKE	TUSK	HADDOCK
MaxIterations	100,000	100,000	100,000
T	1,000	3E+7	100
nt	2	2	2
ns	5	5	5
$\vec{VM}$	$\vec{1}$	$\vec{1}$	$\vec{1}$
rt	0.85	0.85	0.85

Table 2
Execution times, in seconds, consumed by each algorithm and the whole application in the original code.

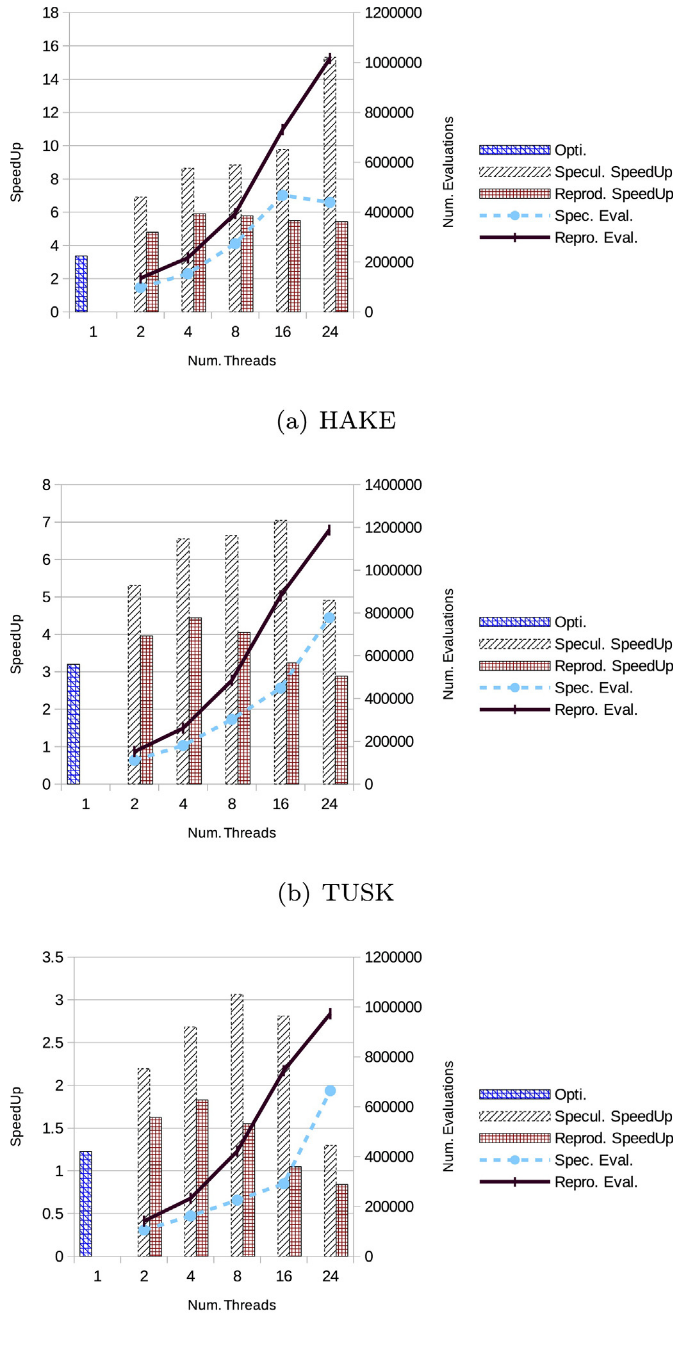
	SA	H&J	SA+H&J
HAKE	5,256	5,310	10,566
TUSK	4,705	4,752	9,457
HADDOCK	1,161	1,166	2,327

In these experiments the two algorithms are executed in sequence so that the second starts from the solution obtained by the first one. The SA algorithm is used first to move to the area of the optimum. Then, the H&J method is applied to refine the solution. This hybrid algorithm combines the power and robustness of the SA algorithm with the faster convergence of a local method as the H&J algorithm. It is a common choice as, in general, it is an effective approach to find an optimum point. The performance of the parallel algorithms was evaluated in terms of both the computational efficiency (speedup) and the quality of the solutions (likelihood score obtained). All the speedups were calculated with respect to the execution time of the original sequential application (Gadget v2.2.00, available at www.hafro.is/gadget/files/gadget2.2.00.tar.gz). These execution times are shown in Table 2, and they correspond to the time consumed by each algorithm and the whole application during the execution of the hybrid algorithm.

Table 3 contains, for each model, the number of parameters to optimize, as well as the average time required by an evaluation of its likelihood function during the SA search process and the standard deviation of the measured times. The measurements were performed after the optimizations described in Section 3 were applied. As we can see, the runtime is very related to the number of parameters to optimize, HADDOCK being much faster than the other two models. This will limit its speedup as the relative parallel overhead will be greater for this model. The standard deviation of the evaluation time is between 1.2% of the average for HADDOCK and 4.9% for TUSK. This reduced variability indicates that the benefits of an asynchronous solution for this

Table 3
Number of parameters to optimize and average and standard deviation (in ms.) of the time of a likelihood evaluation.

Model	# of parameters	Avg. time	Std. dev. time
HAKE	62	18.77	0.28
TUSK	47	17.98	0.89
HADDOCK	38	10.79	0.13

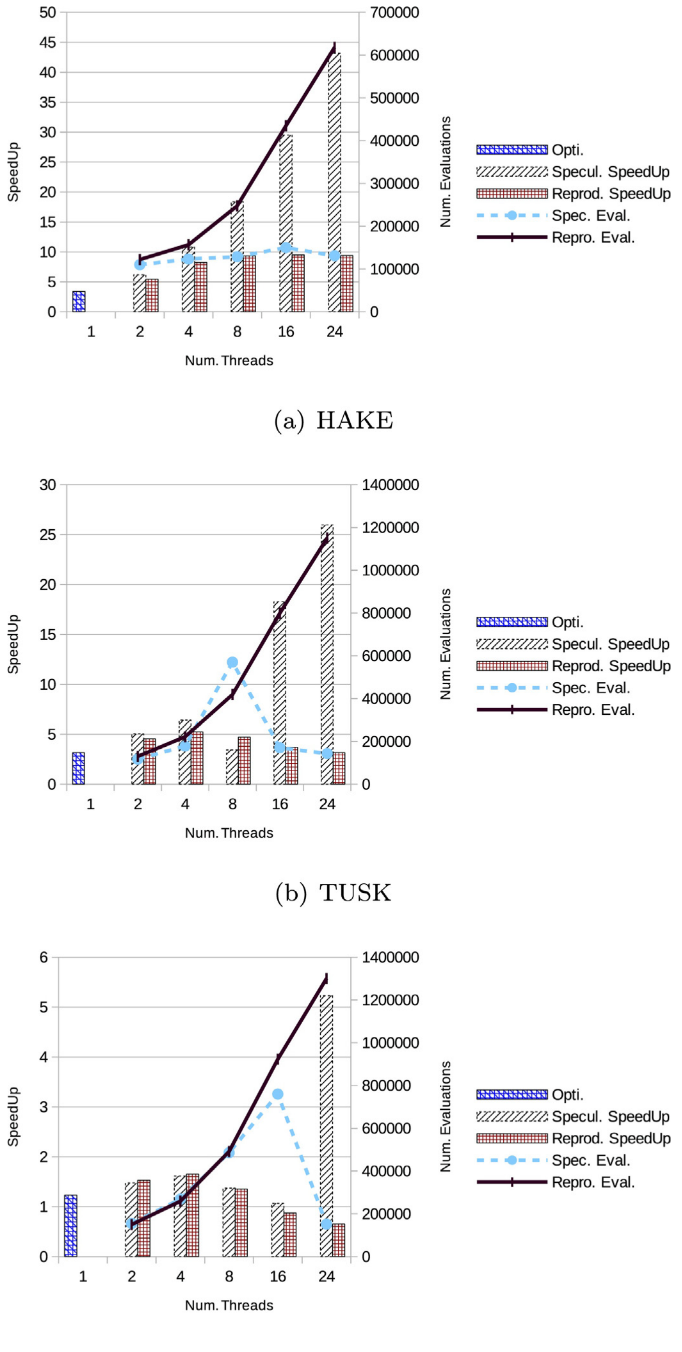


(a) HAKE

(b) TUSK

(c) HADDOCK

Fig. 3. Speedups for the SA algorithm, fill bars indicate relative speedup and the lines the number of function calls as a function of number of threads and by approach.



(a) HAKE

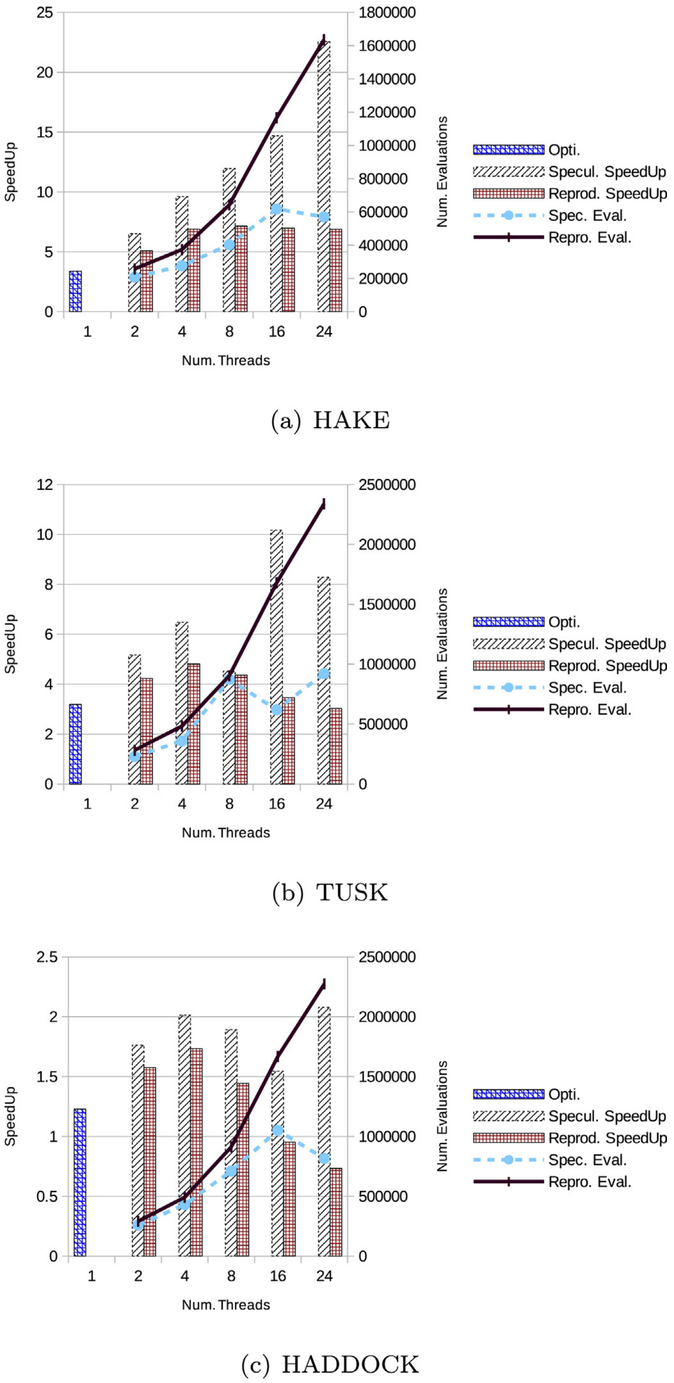
(b) TUSK

(c) HADDOCK

Fig. 4. Speedups for the H&J algorithm, fill bars indicate relative speedup and the lines the number of function calls as a function of number of threads and by approach.

problem would be very small compared to the complexity it would introduce in the application. The values are very similar for the HJ search, the conclusions being thus analogous.

Fig. 3 shows the speedups for the optimized sequential version and the two parallel versions developed, the reproducible one and the speculative one, of the SA algorithm, for the 3 model examples considered and for different number of threads. The total number of evaluations performed for the parallel algorithms is also shown in the figures. The optimization of the sequential code obtains a significant reduction in the execution time for HAKE and TUSK, achieving a speedup of 3.4 for the HAKE model. On the other hand, the reduction in HADDOCK is quite limited. As we explained in Section 6, HADDOCK is a simplified example provided with Gadget to show how the software works. In this example some elements of the model have been skipped for simplicity, which affects to the speedup obtained with the sequential optimization. In the case of the parallel versions, the number of performed evaluations increases with the number of threads, showing a larger increment for the reproducible version, as a greater number of evaluations has to be discarded in this case. For this reason, the speedup of the reproducible version does not improve beyond 4 cores. The speculative version behaves better, although the results depend significantly on the example model. The best results are for the HAKE model because in this case the starting point is closer to the optimum point (see Table 4, discussed at the end of this



(a) HAKE

(b) TUSK

(c) HADDOCK

Fig. 5. Speedups for the whole application, fill bars indicate relative speedup and the lines the number of function calls as a function of number of threads and by approach.

Table 4
Likelihood obtained using SA.

	HAKE	TUSK	HADDOCK
Initial	1,015.1691	26,128.3060	0.96417375
Sequential	1,015.1130	6,537.8195	0.85868597
2	1,015.1120	6,539.2886	0.86255539
4	1,014.8995	6,511.2774	0.85186922
8	1,015.0495	6,509.1936	0.85107118
16	1,015.0250	6,507.5369	0.85250038
24	1,015.0400	6,507.1087	0.85083064

Table 5
Likelihood obtained using H&J after SA.

	HAKE	TUSK	HADDOCK
Sequential	1,015.0478	6,511.6363	0.85396624
2	1,015.0641	6,515.1060	0.85601157
4	1,014.8474	6,507.7478	0.85074075
8	1,015.0103	6,507.2579	0.85062291
16	1,014.9763	6,507.0483	0.85109544
24	1,015.0167	6,507.0332	0.85077653

section), which increases the rate of rejects and thus, the number of effective evaluations. Note that for this example the number of total evaluations does not increase as fast as in the other models.

Similar results are obtained for the reproducible version of the H&J algorithm, as can be seen in Fig. 4. Its speculative version however can achieve noticeably larger speedups. This is related to the larger ratio of rejections that this model, run in the neighborhood of the optimal point, will tend in general to generate, compared to SA, which performs a preceding global search. Finally, Fig. 5 shows the speedups of the whole application. Note that the speedup of the reproducible version only improves up to 4 cores, whereas the speculative version obtains the best results when using 16 or 24 cores. The best results are for the HAKE model, where the execution time is reduced from 176 to 25 minutes using the reproducible version and 4 cores and to only 8 minutes using the speculative version and 24 cores.

As regards the quality of the solution, the reproducible versions obtain exactly the same likelihood score as the original sequential version. The scores obtained by means of the sequential and the speculative versions for SA and H&J run after SA (which yields thus the results of the hybrid SA+H&J search) algorithms are shown in Tables 4 and 5, respectively. Since the speculative versions follow different paths depending on the number of threads used, we show the likelihood obtained using each different degree of parallelism. Table 4 contains for comparative purposes the starting likelihood value associated to the input (Initial row). The starting value for the HJ search process is the likelihood value obtained by the corresponding configuration (either sequential or speculative using the same number of threads) of the SA algorithm shown in Table 4. Notice that the likelihood value is better the lower it is, thus the search algorithms reduce it. We can see that some models, like HAKE, begin with values near the optimum point found, while others, such as TUSK, can be strongly optimized by Gadget. The likelihood value is usually improved when using the parallel speculative version.

8. Conclusions

The aim of this work has been to speedup the Gadget program to reach a reliable model in a reasonable execution time, getting profit from the new multicore architectures. First, the sequential code was analyzed and optimized so that the most important bottlenecks were identified and reduced. Then, the SA and H&J algorithms, used to optimize the model provided by Gadget, were parallelized using OpenMP. Two different versions were implemented for each algorithm, the reproducible one, which yields the same result as the sequential version, and the speculative version, which can exploit more parallelism. It must be stressed that all the parallel algorithms proposed are totally general and can thus be applied to other optimization problems. As expected, the speculative version provides better results for all the analyzed examples, achieving a speedup of 22.6 (6.7 with respect to the optimized sequential version) for the hybrid SA-H&J algorithm and the HAKE model on a 24-core server. Moreover, the speculative version not only reduces significantly the execution time, but it also obtains a better likelihood score.

OpenMP is nowadays the standard de facto for shared memory parallel programming and it allows the efficient use of today's mainstream multicore processors. The OpenMP versions of the Gadget software developed in this work will allow researchers to make a better use of their computer resources. They are publicly available under GPLv2 license at www.github.com/hafro/gadget.

Acknowledgements

This work has been partially funded by the Galician Government (Xunta de Galicia) and FEDER funds of the EU under the Consolidation Program of Competitive Research Units (Network ref. R2014/041) and by European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no. 613571. Authors gratefully thank CESGA (Galicia Supercomputing Center, Santiago de Compostela, Spain) for providing access to the HP ProLiant XL230a Gen9 server and Santiago Cerviño from IEO for his support.

References

[1] J.J. Pella, P.K. Tomlinson, A Generalized Stock Production Model, Inter-American Tropical Tuna Commission, 1969.
[2] K. Patterson, R. Cook, C. Darby, S. Gavaris, L. Kell, P. Lewy, B. Mesnil, A. Punt, V. Restrepo, D.W. Skagen, et al., Estimating uncertainty in fish stock assessment and forecasting, Fish Fish. 2 (2) (2001) 125–157.
[3] G. Einarsson, Competitive coevolution in problem design and metaheuristical parameter tuning (M.Sc. Thesis.), University of Iceland, Iceland, 2014 http://hdl.handle.net/1946/18534.
[4] A. Punt, B. Elvarsson, Improving the performance of the algorithm for conditioning implementation simulation trials, with application to North Atlantic fin whales, IWC Document SC/D11/NPM1, 2011, 7 pp.
[5] J. Begley, D. Howell, An Overview of Gadget, the Globally Applicable Area-Disaggregated General Ecosystem Toolbox, ICES, 2004.
[6] J. Begley, Gadget user guide, 2012 http://www.hafro.is/gadget/files/userguide.pdf.
[7] L. Taylor, J. Begley, V. Kupca, G. Stefansson, A simple implementation of the statistical modelling framework Gadget for cod in Icelandic waters, Afr. J. Mar. Sci. 29 (2) (2007) 223–245.
[8] H. Björnsson, T. Sigurdsson, Assessment of golden redfish (*Sebastes marinus* L.) in Icelandic waters, Sci. Mar. 67 (S1) (2003) 301–314.
[9] B. Elvarsson, L. Taylor, V. Trenkel, V. Kupca, G. Stefansson, A bootstrap method for estimating bias and variance in statistical fisheries modelling frameworks using highly disparate datasets, Afr. J. Mar. Sci. 36 (1) (2014) 99–110.
[10] E. Andonegi, J.A. Fernandes, I. Quincoces, X. Irigoien, A. Uriarte, A. Pérez, D. Howell, G. Stefánsson, The potential use of a Gadget model to predict stock responses to climate change in combination with Bayesian networks: the case of Bay of Biscay anchovy, ICES J. Mar. Sci.: J. Cons. 68 (6) (2011) 1257–1269.
[11] S. McCully, F. Scott, L. Kell, J. Ellis, D. Howell, A novel application of the Gadget operating model to North East Atlantic porbeagle, Collect. Vol. Sci. Pap. ICCAT 65 (6) (2010) 2069–2076.
[12] D. Howell, B. Bogstad, A combined Gadget/FLR model for management strategy evaluations of the Barents Sea fisheries, ICES J. Mar. Sci.: J. Cons. 67 (9) (2010) 1998–2004.
[13] Report of the working group on the assessment of southern shelf stocks of hake, monk and megrim, Tech. Rep. ICES CM 2010/ACOM:11, International Council for the Exploration of the Sea, 2010.
[14] Report of the benchmark workshop on deep-sea stocks (wkdeep), Tech. Rep. ICES CM 2014/ACOM:44, International Council for the Exploration of the Sea, 2014.
[15] Report of the benchmark workshop on redfish management plan evaluation (wkredmp), Tech. Rep. ICES CM 2014/ACOM:52, International Council for the Exploration of the Sea, 2014.
[16] Report of the inter benchmark process on greenland halibut in ices areas I and Ii (ibphali), Tech. Rep. ICES CM 2015/ACOM:54, International Council for the Exploration of the Sea, 2015.
[17] A. Corana, M. Marchesi, C. Martini, S. Ridella, Minimizing multimodal functions of continuous variables with the 'Simulated Annealing' algorithm, ACM Trans. Math. Softw. (TOMS) 13 (3) (1987) 262–280.
[18] W.L. Goffe, G.D. Ferrier, J. Rogers, Global optimization of statistical functions with simulated annealing, J. Econom. 60 (1) (1994) 65–99.
[19] R. Hooke, T.A. Jeeves, Direct search solution of numerical and statistical problems, J. ACM 8 (2) (1961) 212–229.
[20] D.P. Bertsekas, Nonlinear Programming, Athena scientific, 1999.
[21] OpenMP website, http://openmp.org/.
[22] D.R. Greening, Parallel simulated annealing techniques, Phys. D: Nonlinear Phenom. 42 (1) (1990) 293–306.
[23] E.E. Witte, R.D. Chamberlain, M. Franklin, et al., Parallel simulated annealing using speculative computation, IEEE Trans. Parallel Distrib. Syst. 2 (4) (1991) 483–494.
[24] D.J. Ram, T. Sreenivas, K.G. Subramaniam, Parallel simulated annealing algorithms, J. Parallel Distrib. Comput. 37 (2) (1996) 207–212.
[25] E. Onbaşoğlu, L. Özdamar, Parallel simulated annealing algorithms in global optimization, J. Glob. Optim. 19 (1) (2001) 27–50.
[26] D.-J. Chen, C.-Y. Lee, C.-H. Park, P. Mendes, Parallelizing simulated annealing algorithms based on high-performance computer, J. Glob. Optim. 39 (2) (2007) 261–289.
[27] A. Bevilacqua, A methodological approach to parallel simulated annealing on an SMP system, J. Parallel Distrib. Comput. 62 (10) (2002) 1548–1570.
[28] M. Lazarova, Parallel simulated annealing for solving the room assignment problem on shared and distributed memory platforms, in: 9th Intl. Conf. on Computer Systems and Technologies and Workshop for PhD Students in Computing, CompSysTech '08, 2008, pp. 18:II.13-18:1.
[29] J. Ma, K.p. Li, L.y. Zhang, The adaptive parallel simulated annealing algorithm based on tbb, in: 2nd Intl. Conf. on Advanced Computer Control (ICACC 2010), Vol. 4, 2010, pp. 611–615.
[30] N. Safaei, D. Banjevic, A.K. Jardine, Multi-threaded simulated annealing for a bi-objective maintenance scheduling problem, Int. J. Prod. Res. 50 (1) (2012) 63–80.
[31] A. Debudaj-Grabysz, R. Rabenseifner, Recent advances in parallel virtual machine and message passing interface, in: 12th European PVM/MPI Users' Group Meeting (EuroPVM/MPI 2005), Ch. Nesting OpenMP in MPI to Implement a Hybrid Communication Method of Parallel Simulated Annealing on a Cluster of SMP Nodes, 2005, pp. 18–27.
[32] MPI Forum, Message Passing Interface, http://www.mpi-forum.org.
[33] Z.J. Czech, W. Mikanik, R. Skinderowicz, 8th Intl Conf on Parallel Processing and Applied Mathematics (PPAM 2009), Ch. Implementing a Parallel Simulated Annealing Algorithm, 2010, pp. 146–155.
[34] J.E. Dennis Jr., V. Torczon, Parallel implementations of the nelder-mead simplex algorithm for unconstrained optimization, in: 1988 Los Angeles Symposium-OE/LASE'88, International Society for Optics and Photonics, pp. 187–191.
[35] J.E. Dennis Jr., V. Torczon, Direct search methods on parallel machines, SIAM J. Optim. 1 (4) (1991) 448–474.
[36] L. Coetzee, E.C. Botha, The parallel downhill simplex algorithm for unconstrained optimisation, Concur. Pract. Exp. 10 (2) (1998) 121–137.
[37] P.D. Hough, T.G. Kolda, V.J. Torczon, Asynchronous parallel pattern search for nonlinear optimization, SIAM J. Sci. Comput. 23 (1) (2001) 134–156.
[38] Á. B̋urmen, T. Tuma, Sprouting searchâĂŤan algorithmic framework for asynchronous parallel unconstrained optimization, Optim. Methods Softw. 22 (5) (2007) 755–776.
[39] C. Audet, J.E. Dennis Jr., S.L. Digabel, Parallel space decomposition of the mesh adaptive direct search algorithm, SIAM J. Optim. 19 (3) (2008) 1150–1170.
[40] G. Stefánsson, A. Jakobsdóttir, B. Elvarsson, J. Begley, Paramin, A Composite Parallel Algorithm for Minimising Computationally Expensive Functions, 2004 http://www.hafro.is/gadget/files/paramin.pdf.
[41] A. Geist, PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing, MIT press, 1994.

Sergio Vázquez is a master's student at the Computer Architecture Group (GAC) in the Departamento de Electrónica e Sistemas of the Universidade da Coruña, Spain. He received the B.S. in Computer Science in 2015 and is now studying the Master's degree in Computer Engineering, both of them also in the Universidade da Coruña. Before being a M.S. student, he worked on the application of HPC techniques to Machine Learning algorithms. Currently, he is devoted to the optimization and parallelization of algorithms of diverse nature.

María J. Martín received the B.S. (1993), M.S. (1994) and Ph.D. (1999) degrees in physics from the Universidade of Santiago de Compostela, Spain. Since 1997, she has been on the faculty of the Departamento de Electrónica e Sistemas of the Universidade da Coruña, where she is currently an Associate Professor of Computer Engineering. Her major research interests include parallel algorithms and applications and fault tolerance for parallel applications. Her homepage is http://gac.udc.es/~mariam

Basilio B. Fraguela received the M.S. and the Ph.D. degrees in computer science from the Universidade da Coruña, Spain, in 1994 and 1999, respectively. He is an associate professor in the Departamento de Electrónica e Sistemas of the Universidade da Coruña since 2001. His primary research interests are in the fields of programmability, high performance computing, analytical modeling, and compiler transformations. His homepage is http://gac.udc.es/~basilio

A. Gómez is the Projects and Applications manager at CESGA and holds a Ph.D. in Physics from the University of Santiago de Compostela. He has worked for several industrial and IT companies, mainly in distributed systems programming, design and management. Since 2001, he is working at CESGA, where he has participated in several ICT European and National research projects. He has published more than 65 technical and scientific publications in journals and conferences. His research interests are focused on the Cloud for HPC, performance of parallel applications, the development of medical physics software tools, and the improvement of the usability of computing resources.

Aurelio Rodríguez received his Ph.D. in Physical Chemistry from the University of Basque Country in 2001. After that he was a visiting researcher in several European universities. Currently he is a researcher and scientific applications technician at Galicia Supercomputing Center (CESGA) in Spain and he is involved in several HPC projects with participants from different countries.

Bjarki Þór Elvarsson, Ph.D, is a statistician at the Fisheries advisory section at the Marine Research Institute in Reykjavík, Iceland. Main research interests are related to the statistical properties of fisheries stock assessment models, length based models in particular. He has participated in work related to the management of various fish stocks and marine mammals in the North Atlantic.