The Journal of Supercomputing (2023) 79:11827–11843
https://doi.org/10.1007/s11227-023-05124-3

Parallelisation of decision-making techniques in aquaculture enterprises

Mario Ibáñez[1] · Manuel Luna[2] · Jose Luis Bosque[1] · Ramón Beivide[1]

Accepted: 15 February 2023 / Published online: 3 March 2023

Abstract

Nowadays, the *Artificial Intelligent (AI)* techniques are applied in enterprise software to solve *Big Data* and *Business Intelligence (BI)* problems. But most AI techniques are computationally excessive, and they become unfeasible for common business use. Therefore, specific high performance computing is needed to reduce the response time and make these software applications viable on an industrial environment. The main objective of this paper is to demonstrate the improvement of an aquaculture BI tool based in AI techniques, using parallel programming. This tool, called AquiAID, was created by the research group of Economic Management for the Sustainable Development of Primary Sector of the Universidad de Cantabria. The parallelisation reduces the computation time up to 60 times, and the energy efficiency by 600 times with respect to the sequential program. With these improvements, the software will improve the fish farming management in aquaculture industry.

Keywords Aquaculture · Parallelism · Artificial intelligence · Decision-making · Distributed systems

✉ Mario Ibáñez
mario.ibanez@unican.es

Manuel Luna
lunamanuel@uniovi.es

Jose Luis Bosque
bosquejl@unican.es

Ramón Beivide
ramon.beivide@unican.es

[1] Department of Computer Engineering and Electronic, Univerisidad de Cantabria, Santander, Spain

[2] Department of Bussines Administration, Universidad de Oviedo, Oviedo, Spain

1 Introduction

At the present time, due to the development of Information Technology, companies are capable of generating a large amount of information, much more than they are able to process. This is why it is considered that we live in the 'Big Data era', where traditional data processing methods have become obsolete, and companies have to look for new methods to take advantage of so much information.

For companies, processing information into knowledge is fundamental. With this knowledge, company managers are able to make decisions based on data and empirical statistics [1], obtaining a greater benefit for their companies. For this reason, more and more companies are becoming part of the Big Data movement, and are applying analytical techniques within their organisations [2].

This processing of information to provide decision-makers with the knowledge they need to improve their decision-making is known as Business Intelligence (BI) [3, 4]. BI is a concept that encompasses everything from the generation and collection of information, to the easy and interactive access of knowledge by decision-makers.

The methods used in BI range from simple metrics from the data to the use of complex Artificial Intelligence (AI) methods for information processing [5–7]. Because of the complexity that derives from all aspects of BI, many companies need specific tools that enable them to deal with the knowledge needs they demand. But due to most AI techniques are very computationally expensive, they become unfeasible in common business use. Therefore, specific High Performance Computing (HPC) techniques, such as parallelisation, are needed to reduce the response time and make the use of these applications viable at an industrial level.

In this context, the research group Economic Management for the Sustainable Development of the Primary Sector of the Universidad de Cantabria (IDES), developed an AI-based decision support tool for companies in the aquaculture sector, which is part of the European project Mediterranean Aquaculture Integrated Development (MedAID) [8] for improving the competitiveness of Mediterranean companies in the aquaculture sector. This tool, called AquiAID,[1] has a very high computational cost, and therefore, huge execution times for real problems, so it is not usable in practice. Therefore, the main objective of this article is to demonstrate the improvement of the AquiAID software development, through the parallelisation of the code in distributed memory clusters, using the message-passing programming paradigm.

Parallelisation will improve the scalability of the program, reducing its execution time and energy consumption, allowing the application to be viable for business use. In addition, detailed experiments will be carried out to evaluate the parallel implementation. The results of the experiments will be used to obtain metrics and measurements that define the strengths of the parallel version. In particular parallelisation

[1] https://www.ides.unican.es/aquiaid-2.

reduce the computation time up to 60 times, and the energy efficiency by 600 times with respect to the sequential program.

The remainder of the article is divided into the following sections. Section 2 explains the basic concepts of aquaculture and the Particle Swarm Optimisation algorithm. Section 3 explains in detail the analysis, design and implementation phases of the parallel program. Section 4 presents the performance of experiments together with the verification and validation of the results. Section 5 gives the final conclusions and possible future directions for the development of the tool.

2 Background

2.1 Aquaculture

Aquaculture is the activity of producing and fattening aquatic organisms in their living environment [9]. Aquaculture is used to increase the production of fish, molluscs, crustaceans and aquatic plants through specific feeding and protection from predators.

At present, the use of bottom fishing has declined [10] due to its high impact on the environment and its low efficiency. This type of fishery, while maintaining similar catch numbers in recent years, has been overtaken by the growth of aquaculture. Aquaculture has increased its share of marine species over the last thirty years, from 10% to the current 45%. It is a viable and sustainable alternative for the production and sale of marine species.

At this time, most aquaculture in developed countries is carried out by mainly multinational companies. In this respect, it is important to emphasize the difficulty of managing an aquaculture enterprise [11]. This is due to the multitude of systems and paradigms that such a company has to deal with. Companies should monitor and obtain data on spawning, growth, care, waste, diseases, genetics, etc. All these aspects will have their impact on business scheduling, from the costs of feeding, temperature, oxygen and humidity control of the marine species, to the time of harvesting, transport, location and dates of sales and restocking.

Due to the large number of all these factors, many companies are looking to increase their profits with the help of specialised schedulers. These schedulers are currently very varied, but a solution based on artificial intelligence [12] could give good results if the right methods are used. In this work, a scheduler from the AquiAID project based on a well known artificial intelligence technique called Particle Swarm Optimisation (PSO) is used.

2.2 Particle swarm optimisation

The Particle Swarm Optimisation [13] is a meta-heuristic method based on the behaviour of certain insect species, which act in the form of swarms. These insects, like bees or ants, randomly forage for food in nearby areas. When they find interesting areas with

enough food, they share it with the rest of the swarm. PSO-type algorithms act in a similar way.

The objective of these algorithms is to maximise or minimise a value for a function *f*, of which there is no other possibility than to try random points of this function, here dubbed *particles*, to find the global minimum or maximum. This function *f* is also known as a *fitness function*, because it evaluates how close a particle to the optimal solution is. The dimensions of a particle $(\vec{x}_1, ..., \vec{x}_n)$ are the needed parameters to have a complete solution of the problem.

The pseudocode of PSO type algorithms can be seen in Algorithm 1.

Algorithm 1 Particle Swarm Optimisation Pseudocode

$M \in \mathbb{R}^2 \leftarrow Initialise\_matrix\_population()$
for $i = 0$ *to* $max\_iterations$ **do**
  for *Each particle* $\overrightarrow{p}$ *in* M **do**
    $fp \leftarrow f(\overrightarrow{p})$
    if fp *is better than* $f(\overrightarrow{local\_Best})$ **then**
      $\overrightarrow{local\_Best} \leftarrow \overrightarrow{p}$
    end if
  end for
  $\overrightarrow{global\_Best} \leftarrow best\_particle(M)$
  for *Each particle* $\overrightarrow{p}$ *in* M **do**
    $\overrightarrow{v} \leftarrow \overrightarrow{v} + c1 * \overrightarrow{random\_vector} * (\overrightarrow{local\_Best} - \overrightarrow{p}) + c2 * \overrightarrow{random\_vector} * (\overrightarrow{global\_Best} - \overrightarrow{p})$
    $\overrightarrow{p} \leftarrow \overrightarrow{p} + \overrightarrow{v}$
  end for
end for
return best_particle(M)

The computational difficulty of the algorithm depends on two parameters, the number of particles and dimensions. More particles mean that there are more solutions to evaluate in every iteration, and too, dimensions increase the evaluating time of a particle. In general increase this numbers involves in better solutions from the problem, but rise the execution time.

This is why PSO algorithms are quite suitable for data-parallel processing. In particular, when searching in a space with hundreds of dimensions, it is necessary to have a large number of particles in order to find a feasible solution. Moreover, by the nature of the algorithm itself, it is quite favourable for the distribution of its computations, since the particles are usually independent of each other, and it allows most computations to be performed on independent computers.

3 Design and implementation

3.1 Sequential description

First, the behaviour of the sequential application is described, in order to have a clear idea of the parallelisable parts, and the different techniques employed.

The formulation of the problem is complex due to there are two different models used in the program. The first one is the Economical submodel.

Implementation of PSO in AquiAID follows the generic iterative structure of Algorithm 1. However, some calculations are added to improve the efficiency of the searches, and also, some semantics are attached to the dimensions of the particles to allow a better scheduling.

The first step in the algorithm is generating the population matrix M. Each of the particles is made up of a number of *cages*, which represent the fish cages in which a culture is carried out by a company. A particle is a finite set of cages, and the dimensions of the particle are linked to the number of cages. Therefore, the algorithm uses a matrix ($p \times c$) to represent the set of particles and cages, where each row is a particle and each column represents a cage. Therefore, each element of the matrix is a duple particle-cage (p_i, c_j).

Once the matrix of particles and cages has been initialised, an iterative process starts with the computation of the fitness function and constraints for each particle. In the generic algorithm only the fitness function is specified as the value to be calculated, but in these case there is also the calculation of the constraints.

First one, the fitness function is defined by the next equation:

$$F(X) = \frac{d^-(X)}{d^-(X) + d^+(X)}$$

The parameters of this function are the next one:

- X: represents a particle of the PSO algorithm. Every particle has four elements per particle-jail which are:
  - The seeding date.
  - Fingerling initial mass.
  - Feed used for fattening.
  - Harvesting date.
- $d^-(X)$: distance from the anti-ideal solution of criteria values of particle X.
- $d^+(X)$: distance from the positive ideal solution of criteria values of particle X. $d^-(X)$ and $d^+(X)$ are functions which use the Technique of Order Preference by Similarity to Ideal Solution (TOPSIS), to calculate the distance of every particle.

And now the constraint is the one defined below:

$minp(w) \leq Prod_w(X) \leq maxp(w)$ *with* $w = 1;...;$ *Number of total weeks*: commercial or operational constraints for week w.

where

- $Prod_w(X) = \sum_{cage=1}^{N} Harvest_x(cage, w)$: this represents the sum of amounts harvested in week w according to particle X. The returned value of $Harvest\_x(cage, w)$ it could be money, weight, amount of fish, etc. So with only one function we can create different constraints depends on our goal.
- $minp(w)$: minimum production in week w.
- $maxp(w)$: maximum production in week w.

These constraints are validated for each of the particles, and allow determining whether a particle meets a series of minimum requirements to be considered among the best ones. This greatly improves the search capacity of the algorithm, because through this check, particles which the solution is impossible to realise at the practical level, are ignored.

Once the fitness and constraint values have been calculated, the best local and global particle is calculated like on the Algorithm 1. Then the velocity and position of the particles are updated. Finally, the algorithm finishes if there no are pending iterations, or if the improvement of the fitness of the best particle between the last two iterations does not exceed a certain threshold.

The AquiAID program has more complexity is more complex than previously description, but for the goal of this manuscript, it's enough with the described functions. For more information about the problem description and models used to simulate a aquaculture company check the next reference [14, 15].

With this description of the sequential program, it can be seen that the implemented algorithm is well suited for parallelisation due to the fine-grained degree it allows, and the relative independence of the calculations performed on the data structures. With all this in mind, the following sections explain the phases of the parallelisation of the program.

3.2 Data a task decomposition

As described in the previous section, this algorithm is very suitable for domain or data parallelisation. Therefore, the first step in the parallelisation must be analysed the data structures to carry out the decomposition. Namely, is essential to determine which data structures will be distributed among the processes, and the minimum workload to be used.

In the implementation of PSO in AquiAID, there are two ways to carry out this partitioning, with particles or with particle-cages. Particles are typical elements of PSO algorithms as defined in Sect. 2.2. They are usually the most common elements where parallelisation is performed, due to the computational cost of their fitness function. Their parallelisation will consist of creating independent MPI [16] processes that receive a given number of particles and will be in charge of calculating the scores of each one. After obtaining the results of the particles, a communication phase will take place where the partial results of each process are shared, and the respective measures are calculated in order to continue with a new iteration of the algorithm.

This particle-based parallelisation is simple and fast, allowing to obtain good results in a short development time. But it limits the scalability of the algorithm, since the parallelism kernel implies in this case one particle per process. Therefore, it is not possible to use more processors than particles, and the number of particles is usually in the hundreds.

For this reason, it is necessary to use a different decomposition which would allow greater scalability. The new partitioning will consist on make use of *particle-cage blocks*. These blocks are made up of a set of continuous particle-cage. In first instance, it is important to know that it is possible to divide the particles into particle-cages and to distribute it among the processes. It is achievable because there is no dependency between the particle-cages of a particle, so they can be distributed among the processes.

This independence between particle-cages is largely due to the fitness and constraints functions. In this implementation, both functions of the particle are divided into the calculation of every individual particle-cage. Then, all the partial results are merged to compute the particle results. Thanks to this individual computation it is possible to divide the computation of a particle between the different processes.

As a final point, the computational output of the particle-cages scores can be shared between the processes. This allows to calculate the fitness and constraints of the particles and the final metrics of the algorithm iterations. As in the case of the particles, MPI makes it possible to share the particle-cage data among the processes and to perform distributed operations on them.

Parallelising in terms of particle-cages becomes more complex, because the PSO algorithm defines operations in terms of particles, and not subsets of the particle dimensions, such as particle-cages. But the finer-grained parallelisation that particle-cages allow, improve the efficiency of the final application.

The latter, after analysing the execution of the program, the computation of the fitness of a particle is the most computationally limiting part. Parallelising based on particles allow to divide the computation time of this fitness by the total number of particles. However, when the fitness function is divided in the calculation by particle-cages, the computation time of the fitness is divided by the number of particle-cages, which is always greater than the number of particles.

Given these alternatives, and because the main objective is to achieve the greatest scalability of the program, the decision is to parallelise according to particle-cage blocks. For this, it is necessary to divide the particle-cage blocks between the processes, and to create all the necessary structures to parallelise the PSO algorithm, as will be described in the following sections.

3.3 Description of parallel algorithm

This section explains the design and implementation of the parallel algorithm. Figure 1 shows the general scheme of the parallelisation algorithm, where each number correspond to a phase that will be explained afterward. The initial phase (1) is to initialize the matrix of particle-cages with random number.

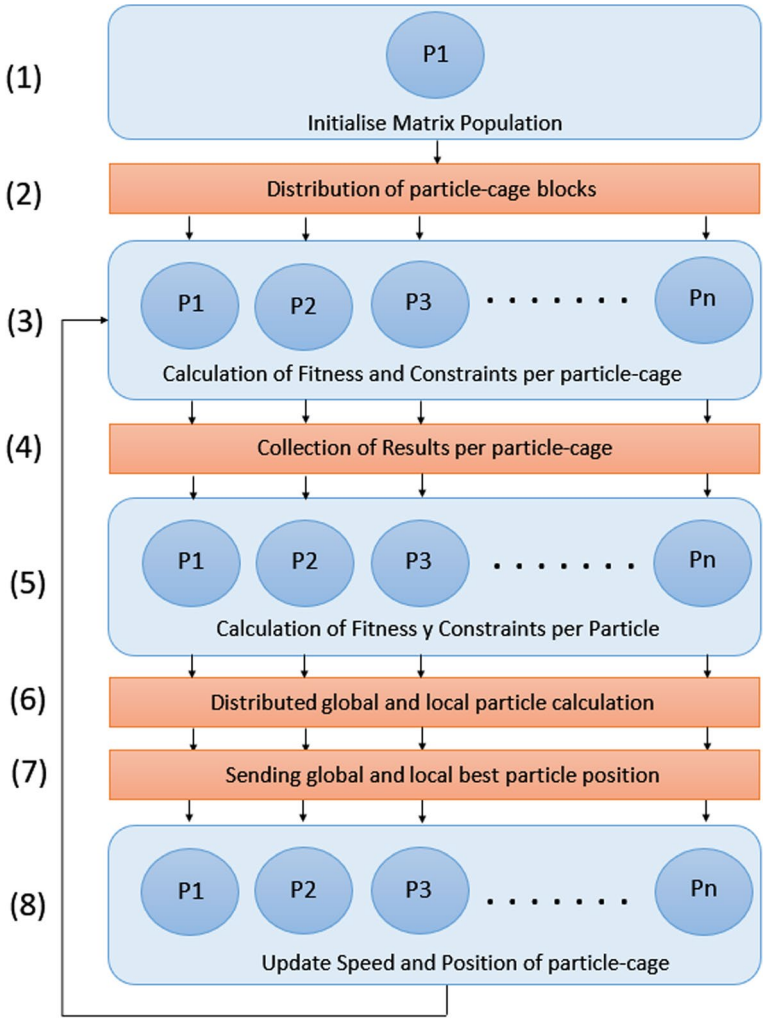


Fig. 1 Scheme of the Parallelisation AquiAID

Then, in the second step (2), this initial matrix has to be partitioned in the particle-cage blocks, and distributed among all the processes, like it is observed in Fig. 2. In this figure, there is a single example of the partition and distribution of the matrix. In the example, the matrix is made up by two particles (p_1, p_2) and six cages $(c_1, ..., c_6)$, conforming a total of twelve particle-cages. Moreover, there are three processes (P1,P2,P3) and the size of particle-cage blocks is performed as the ratio between the total particle-cages (twelve) and the number of processes (three). This operation gives a total of three blocks with four particle-cages:

- Block 1 (brown): from p_1c_1 to p_1c_4.

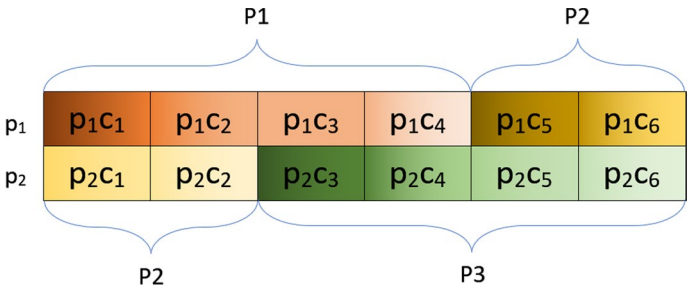


Fig. 2 Example of the distribution of the particle-cage

- Block 2 (yellow): from p_1c_5 to p_2c_2.
- Block 3 (green): from p_2c_3 to p_2c_6.

Then, particle-cage blocks are assigned in order to processes, block one to P1, block two to P2 and block three to P3.

With this, a workload balance between the processes is guaranteed, because the computational load of the particle-cages is regular, that is, they all perform similar number of operations [17, 18]. This distribution is static, once it has been decided which particle-cage block are computed by each of the processes, the distribution is not changed throughout the execution.

With the distribution of particle-cage blocks done, every process calculate the fitness and constraints of theirs particle-cages (3). Once the processes have the results of the particle-cage blocks, it is necessary to compute the fitness of a complete particle, performing a communication and synchronisation phase (4). This step is complex because a process may or may not have all the particle-cages of a given particle. If a process has all the particle-cages of the particle, it should be the process that computes the results of the particle without any communication.

In contrast, if it does not have all, it can be said that it shares the particle with other processes. Therefore, in order to perform as little computation as possible, one of the processes which share the particle, should be responsible for computing its results. The rest of the processes send the data of their particle-cages to the responsible one in order to calculate the values for the particle, like in Fig. 3.

to this task a group communicator is generated for every shared particle, for example in previous figure, there are two group communicators, COMM1 and COMM2, each one for different particles. The fitness, constraints and values of the particle-cages are sent to the processes responsible. In the figure, P2 and P3 send to process responsible P1, and in COMM2, P5, P6 and P7 send to responsible P4. This communication is done through collective calls from each of the groups, allowing a very efficient and fast communication between all the processes.

The responsible processes receive the final data and compute the results of the particles which it is responsible for (5). Then, the responsible processes calculate their best local particle. First of all, it is checked that the particles satisfy all the constraints, based on the results obtained. If a particle satisfies all the constraints

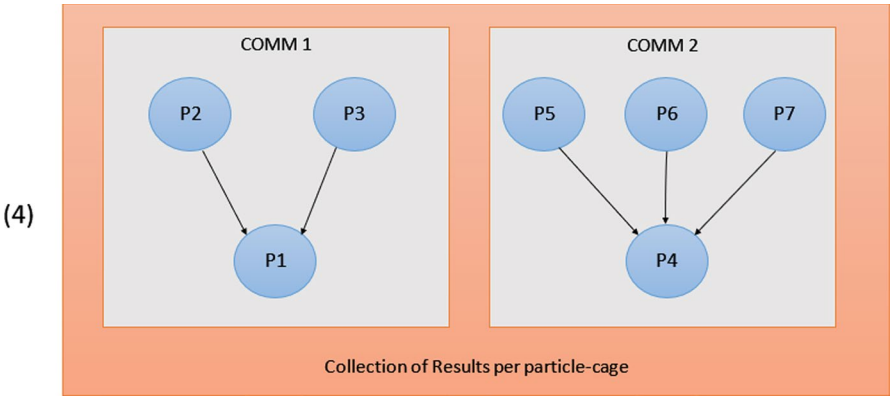


Fig. 3 Sending data to responsible processes

then it is checked if its current position is the best one it has had in all the iterations; if so, its old local position is replaced by the new best position. After updating the best position of each particle, the processes individually select the best particle they have.

The next step (6) computes the best global particle using an AllReduce MPI call. The best particles of all the responsible processes are sent to all the processes, indicating also the owner process. The particle with the best fitness is selected as the best overall particle. The communication patron in the MPI call is similar as shown in Fig. 3. But in this case, the communications is between the responsible processes.

In the last communication step (7), the owner of the best particle sends the position of the particle to the rest of the processes. With this, all processes have the best global particle. Furthermore, the processes that share particle-cages still lack the position of the shared particle, since it is only known by responsible one. To resolve this, the communicators of the shared groups are used, and the responsible sends the particle data to the rest of the processes, like is shown in Fig. 4. This figure represents the last communication step in the same example as Fig. 3. In Fig. 4, the responsible processes send the results of particles to the rest of the group communicator, in COMM1, P2 and P3 receive from P1, and in COMM2 P5, P6 and P7 receive from P4.

The final step (8) is to calculate a velocity vector that updates the positions of the particle-cages. The calculation of the velocity vector, as shown in Algorithm 1, consists of the combination of three different vectors:

- The previous velocity of the particle, in the particle-case of the initial iteration the velocity is initialised randomly.
- The position of the best part.
- The best position of the particle, to which the velocity vector corresponds.

These three vectors, in the sequential algorithm, are implemented to be applied to each of the particles. However, since the distributed model works with

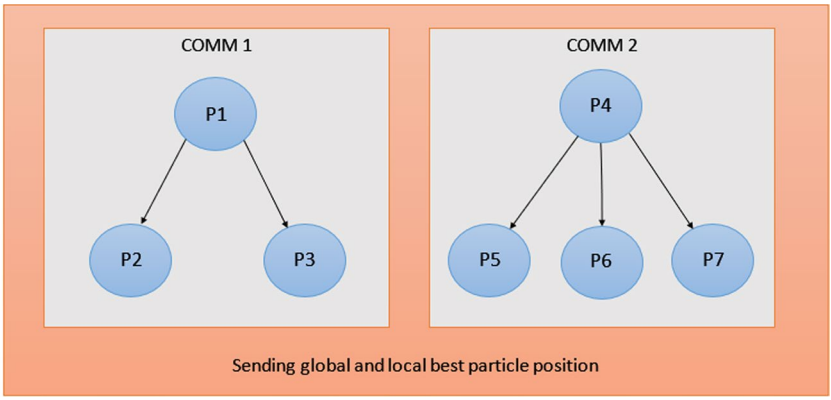


Fig. 4 Sending data from responsible processes to the rest of group processes

particle-cages, the data structures must be modified in order to adapt the different vectors to the exact dimensions of each process.

As can be seen in Fig. 1, this calculation finish one iteration of the loop and recalculate the value of the cages, sending the results, etc. With all the explanations, the PSO algorithm and its parallelisation finish.

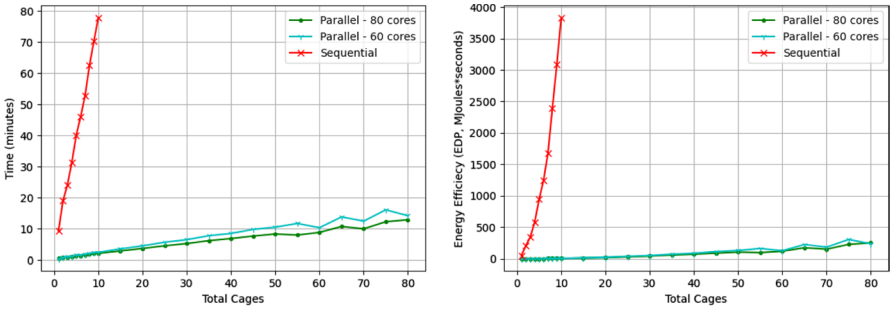
4 Evaluation

4.1 Experimental setups

This section presents an experimental evaluation of the parallel application. This evaluation aims to test the improvements of the parallel application with respect to three key metrics: performance, energy efficiency and scalability.

The performance evaluation is performed by comparing the response time of the sequential version with that of the parallel version, for different combinations of the input parameters (number of cells and number of particles). With regard to energy efficiency, it should be noted that the parallel program makes use of a greater number of computational resources, which imply an increase in the power consumption. This can represent an important economic cost that, must be taken into account when implementing a decision-support system such as AquiAID. Nevertheless, the energy is the product of power consumption times execution time. Therefore, the increase in power can be compensated by a reduction in execution time, such that the energy consumption of the parallel application would actually be lower than the sequential one. For this analysis, the metric used will be the energy efficiency, measured through the Energy-Delay Product (EDP) [19, 20].

Finally, the scalability of the application will be evaluated, providing information on its behaviour as the number of processors in the parallel system increases [21, 22]. This metric allows determining whether the application is able to adequately



(a) Response Time (b) Efficiency Energy (EDP)

Fig. 5 Performance and Energy of the sequential vs. parallel program

take advantage of the computational resources, even when their number is very large. The metric used will be the speedup of the parallel application as the number of cores in the parallel system increases.

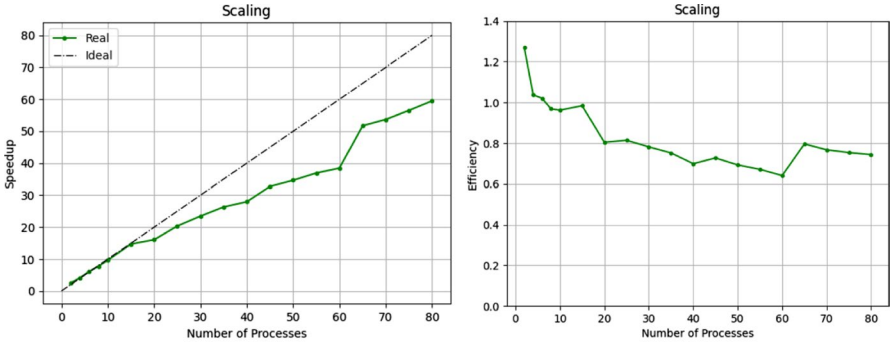
The platform used is a cluster consisting of four servers, each with two 2.2GHz Intel(R) Xeon(R) Silver 4114 processors, ten cores per processor, 32GB of memory per node and 1Gb/s Ethernet interconnection. Therefore, the system has a total of 80 cores, which means that up to 80 MPI processes can be executed. The Slurm resource manager [23] is used, which guarantees an adequate distribution of tasks to the cores. Therefore, the measurements are collected from the Slurm database, which stores information on both the performance and the energy consumption.

Another aspect to be determined is the input parameters, the number of cages and particles, which characterise each experiment. Regarding the number of particles, there are many studies concerning this variable depending on the nature of the problem [24–28], which allow to conclude that an appropriate number of particles for a problem of these characteristics is one hundred. The number of cages is determined by the company to be analysed. For the performance and energy experiments, values between 10 and 80 cages have been used, which correspond to average farms. For scalability, 200 cages have been used, which is a large but completely realistic farm size.

4.2 Experimental results

The first experiment consists of measuring the execution time and the energy consumed for both the sequential and parallel versions using 60 and 80 cores. Due to the excessive time consumed by the sequential version, the data obtained have been only with values between 1 and 10 cages per particle. Once these values were obtained, the EDP was calculated, and the results are shown in Fig. 5, where the X axis represents the product of the number of particles and the number of cages.

Figure 5a shows the improvement of the parallel implementation in execution time. As can be seen, the parallel version considerably reduces the execution time with respect to the sequential version. In the same time that the sequential



(a) Speedup (b) Parallel Efficiency

Fig. 6 Strong scaling of the parallel version as the system size increases

program performs an experiment with 1 cage (and 100 particles), the parallel program with 80 cores is able to compute 60 cages, and 40 cages with 60 cores. This allows schedules that would last for days in the sequential program, to only a couple of hours if run in parallel with 60 or 80 cores. Therefore, the parallel version achieves speedups between 60 and 40, using 80 and 60 cores, respectively.

Regarding energy efficiency, Fig. 5b shows the EDP of the three executions (less is better). This metric takes into account both the energy consumption and the execution time. As can be seen in the figure, the parallel version is much more energy efficient than the sequential version, with a gain of 600 times the EDP of the sequential version in the 10-cage experiment. These gains are due both to the significant reduction in response time and to a significant reduction in the energy consumption, with values up to 20 times smaller for 10 cages. Therefore, it can be concluded that the increase in power consumption in the parallel version, due to the use of more computational resources, is largely outweighed by the reduction in execution time, consuming less energy in absolute terms and obtaining a much better energy efficiency.

The scalability of the parallel version will be discussed in the following paragraphs. In particular, the strong scaling will be analysed, taking a fixed problem size (100 particles and 200 cages) and the execution time is obtained as the number of cores in the system increases. Figure 6a presents the speedup obtained as the ratio between sequential and parallel execution time, compared to the ideal speedup, while Fig. 6b shows the parallel efficiency. The first conclusion that can be drawn from these results is that the application has good scalability, as the speedup always grows, and exploits the system properly, with efficiency values around 0,75.

Nevertheless, the figures present different segments, which are analysed below. To do this, it should be remembered that the system is composed of four nodes with 20 cores each. The results for the first 15 cores show an increase in speedup proportional to the increase in the number of cores, matching the ideal line. Additionally, from the first 5 cores happens that speedup is superlineal, with a value of efficiency above 1 (1.05). This event is due to the distribution of data between the cores, which

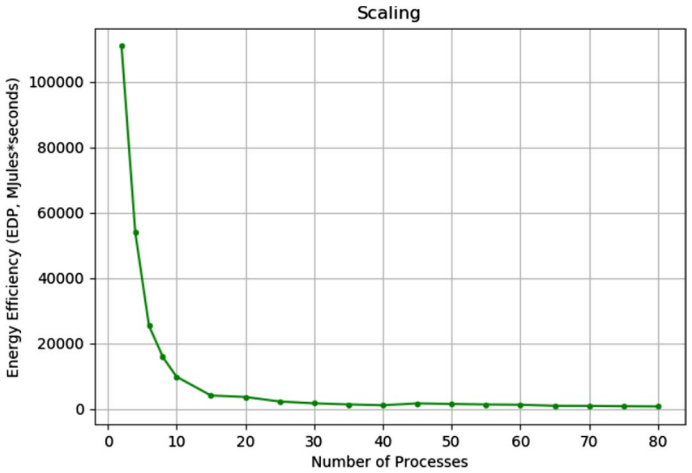


Fig. 7 Energy efficiency of scaling experiments

reduce the load of caches improving the execution time. For 20 cores the increase in speedup is damped and diverges from the ideal line. Since all processes are running on the same node, this indicates that there is contention in the access to the main memory of the node. This effect is also showed in the parallel efficiency, which drops to 0.8.

From 20 cores upwards, the processes are executed on different nodes, so that communication overheads are imposed, which is reflected in a more moderate increase in speedup, and the reduction of parallel efficiency. This corresponds to the gradual deviation observed in the speedup obtained compared to the ideal. Additionally, it is observed that the increase in speedup and parallel efficiency between 20 and 25 cores has a much steeper slope than between 15 and 20. This is due to the fact that a new compute node is added, and the processes are distributed equally between the two nodes, reducing memory contention. Between 25 and 40 cores, the increase of speedup becomes more and more moderate, which correspond to a reduction of parallel efficiency, but this is caused due to the problems of memory contention and communications, until 45 cores are reached. At that point, the third compute node is added, and the processes are again distributed among the nodes. This behaviour is repeated with 65 nodes.

The speedup and parallel efficiency have a great scalability and performance. And this is supply also with the energy efficiency shown in Fig. 7, which concludes that consumption and execution time have a similar behaviour forming in the efficiency the most usual curve in this type of scaling. As the execution time and energy efficiency follow a similar scaling pattern, it can be concluded that the parallelised algorithm scales correctly, even with many cages as in this experiment.

5 Conclusions

Nowadays, the era of Big Data provides companies with a huge amount of data, which they have to process and turn into useful information. If this information is incorporated into the decision-making process, it can help to improve the efficiency of the companies in multiple aspects. But this is a complex process that requires metaheuristics and artificial intelligence techniques to be truly useful and efficient. These techniques are generally very computationally heavy, so that in order to be applied to real business cases they require the use of high-performance computing.

In this context, this paper presents the design, implementation and optimisation of a metaheuristic algorithm, the Particle Swarm Optimisation (PSO), to aid decision-making in aquaculture enterprises. The optimisation has been developed by parallelising the algorithm using the paradigm of concurrent processes communicating through message passing using the MPI library, on a cluster of multicore nodes.

The design of the parallel algorithm has focused on maximising its scalability, even at the cost of increasing its complexity. To this end, a domain decomposition has been made as fine-grained as possible, achieving a gran-fine parallelisation. This has been done so that the proposed solution can be applied to any size of company, and make maximum use of resources, even for very large computational systems.

The experiments in Sect. 4 clearly show that the program is now able to perform real cases. With the parallel version it is possible to carry out examples 50 to 60 times larger than those achieved with the original version. In addition, energy consumption does not become a constraint when running in parallel due to the high energy efficiency obtained with the parallel program. With these improvements it is possible for a company to use this scheduler software to manage fish farming, without the need to wait days for a result. This makes the program useful for its intended use, and allows both companies and researchers to continue to improve the long road of detail and improvement of aquaculture management applications.

As for possible future work, the possibility of implementing the same algorithm using GPUs is being considered, as the high data parallelism and good scalability suggest that it could offer very good results.

Author Contributions Sequential AquiAID: Manuel Luna Design and implementation of parallel application: Mario Ibáñez and Jose Luis Bosque. Evaluation: Mario Ibáñez. Paper writing: All authors. Funding: Research projects where the Principal researchers are Ramón Beivide and Jose Luis Bosque. All authors reviewed the manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work has been supported by the Spanish Science and Technology Commission under contracts PID2019-105660RB-C22 and TED2021-131176B-I00 and FPU21/03110.

Availability of data and materials The sequential implementation is available in https://www.ides.unican.es/aquiaid-2/. Parallel version will be publishing soon.

Declarations

Conflict of interest There is not any financial interest in this paper.

Ethical Approval Not applicable

References

1. Mora M, Forgionne GA, Gupta JN (2002) Decision making support systems: achievements. Trends and Challenges For. IGI Global, Pennsylvania, USA
2. Papathanasiou J, Ploskas N. (2018) Multiple criteria decision aid. Methods, Examples and Python Implementations. Springer Cham. ISBN978-3-319-91646-0
3. Hardoon DR, Shmueli G (2013) Getting started with business analytics: insightful decision-making, vol 1. CRC Press, USA
4. Howson C (2013) Successful business intelligence, 2nd edn. McGraw-Hill Education, USA
5. Shmueli G, Patel NR, Bruce PC (2011) Data mining for business intelligence: concepts, techniques, and applications in microsoft office excel with xlminer. Wiley, USA
6. Kou G, Yang P, Peng Y, Xiao F, Chen Y, Alsaadi FE (2020) Evaluation of feature selection methods for text classification with small datasets using multiple criteria decision-making methods. Appl Soft Comput 86:105836
7. Kou G, Xu Y, Peng Y, Shen F, Chen Y, Chang K, Kou S (2021) Bankruptcy prediction for smes using transactional data and two-stage multiobjective feature selection. Decis Support Syst 140:113429
8. Cidad M, et al. (2018) Assessment of mediterranean aquaculture sustainability. Deliverable 1.2 of the Horizon 2020 project MedAID (GA number 727315), published in the project web site on 21.12.2018. https://archimer.ifremer.fr/doc/00515/62630/
9. Luna M, Llorente I, Cobo A (2020) Aquaculture production optimisation in multi-cage farms subject to commercial and operational constraints. Biosyst Eng 196:29–45
10. Stankus A (2021) State of world aquaculture 2020 and regional reviews: Fao webinar series. FAO Aquac Newsl 63:17–18
11. Tidwell JH (2012) Aquaculture production systems. Wiley, Boston
12. Winston PH (1992) Artificial intelligence. Addison-Wesley Longman Publishing Co., Inc, Boston, USA
13. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of ICNN'95-international Conference on Neural Networks, vol. 4, pp 1942–1948 . IEEE
14. Luna M, Llorente I, Cobo Á (2019) Integration of environmental sustainability and product quality criteria in the decision-making process for feeding strategies in seabream aquaculture companies. J Clean Prod 217:691–701
15. Luna M, Llorente I, Cobo A (2020) Aquaculture production optimisation in multi-cage farms subject to commercial and operational constraints. Biosys Eng 196:29–45
16. Pacheco P, Malensek M (2021) An introduction to parallel programming, 2nd edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA
17. Bosque JL, Toharia P, Robles OD, Pastor L (2013) A load index and load balancing algorithm for heterogeneous clusters. J Supercomput 65(3):1104–1113
18. Nozal R, Pérez B, Bosque JL, Beivide R (2019) Load balancing in a heterogeneous world: Cpu-xeon phi co-execution of data-parallel kernels. J Supercomput 75(3):1123–1136
19. Laros JH III, Pedretti K, Kelly SM, Shu W, Ferreira K, Vandyke J, Vaughan C (2013) Energy delay product. Energy-efficient high performance computing. Springer, Heidelberg, pp 51–55

20. Castillo E, Alvarez L, Moretó M, Casas M, Vallejo E, Bosque JL, Beivide R, Valero M (2018) Architectural support for task dependence management with flexible software scheduling. In: IEEE International Symposium on High Performance Computer Architecture, HPCA, Vienna, Austria, pp 283–295
21. Bosque JL, Robles OD, Toharia P, Pastor L (2011) Evaluating scalability in heterogeneous systems. J Supercomput 58(3):367–375
22. Bosque JL, Perez LP (2004) Theoretical scalability analysis for heterogeneous clusters. In: 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid). Chicago, Illinois, USA. IEEE Computer Society, New York, pp 285–292
23. Yoo AB, Jette MA, Grondona M (2003) Slurm: Simple linux utility for resource management. Workshop on job scheduling strategies for parallel processing. Springer, Berlin, pp 44–60
24. Trelea IC (2003) The particle swarm optimization algorithm: convergence analysis and parameter selection. Inf Process Lett 85(6):317–325
25. Bratton D, Kennedy J (2007) Defining a standard for particle swarm optimization. In: IEEE Swarm Intelligence Symposium, pp 120–127
26. Shi Y, Eberhart RC (1999) Empirical study of particle swarm optimization. In: Proceedings of the 1999 Congress on Evolutionary computation-CEC99 (Cat. No. 99TH8406), vol. 3, pp 1945–1950 . IEEE
27. Li-Ping Z, Huan-Jun Y, Shang-Xu H (2005) Optimal choice of parameters for particle swarm optimization. J Zhejiang Univ Sci A 6(6):528–534
28. Piotrowski AP, Napiorkowski JJ, Piotrowska AE (2020) Population size in particle swarm optimization. Swarm Evol Comput 58:100718