



Accelerating an algorithm for perishable inventory control on heterogeneous platforms



Alejandro Gutierrez-Alcoba^{a,*}, Gloria Ortega^b, Eligius M.T. Hendrix^a, Inmaculada García^a

^a Department of Computer Architecture, Escuela de Ingenierías, c/ Dr Ramos, University of Málaga, Málaga, 29071, Spain

^b Informatics Department, University of Almería, Agrifood Campus of Int. Excell. (ceiA3), Almería, 04120, Spain

HIGHLIGHTS

- Optimization algorithm for perishable inventory control problems.
- Design of parallel implementations for distributed and shared memory platforms.
- Task distribution modelling for the inventory control problem on multi-GPU clusters.
- Workload balancing through heuristic for the Bin-Packing problem.

ARTICLE INFO

Article history:

Received 26 October 2015

Received in revised form

20 October 2016

Accepted 21 December 2016

Available online 28 December 2016

Keywords:

Perishable inventory control

GPU computing

Heterogeneous computing

Optimization

Monte-Carlo simulation

Bin-Packing problem

ABSTRACT

This paper analyses and evaluates parallel implementations of an optimization algorithm for perishable inventory control problems. This iterative algorithm has high computational requirements when solving large problems. Therefore, the use of parallel and distributed computing reduces the execution time and improves the quality of the solutions. This work investigates two implementations on heterogeneous platforms: (1) a MPI-PTHREADS version; and (2) a multi-GPU version. A comparison of these implementations has been carried out. Experimental results show the benefits of using parallel and distributed codes to solve this kind of problems.

Furthermore, the distribution of the workload among the available processing elements is a challenging problem. This distribution of tasks can be modelled as a Bin-Packing problem. This implies that the selection of the set of tasks assigned to every processing element requires the design of a heuristic capable of efficiently balancing the workload statically with no significant overhead. This heuristic has been used for the parallel implementations of the optimization for perishable inventory control problem.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

The main objective of this work consists of determining up to what extent the use of heterogeneous platforms (multicore and multi-GPU) accelerates the solution process of a novel optimization algorithm for an inventory control problem of perishable products. Our goal is to study how to take advantage of the computing capacity of these architectures to obtain more accurate solutions when large problems are considered, which imply high computational demand, keeping a reasonable response time.

The perishable inventory control problem presented in this paper is defined over a finite horizon of T periods of a perishable product in which a fixed percentage of the non-stationary stochastic demand has to be satisfied according to a so-called fill rate service level requirement. The perishable product has a fixed shelf life of J periods. In the modelling of this problem, it is supposed that $J < T$ and the dynamics of the inventory follows the FIFO issuance (first in, first out), in which the oldest product is issued first. Items of age J cannot be used in the next period and are considered waste. The goal is to minimize the cost related to the production, distribution, storage and waste. [8] describes an algorithm based on Monte-Carlo simulation of the demand, specifically designed to solve this problem. This algorithm is able to determine the optimal order policy and the corresponding order quantities efficiently. However, the computational burden grows exponentially in the time horizon.

* Corresponding author.

E-mail addresses: agutierrez@uma.es (A. Gutierrez-Alcoba), gloriaortega@ual.es (G. Ortega), eligius@uma.es (E.M.T. Hendrix), igarciaf@uma.es (I. García).

<http://dx.doi.org/10.1016/j.jpdc.2016.12.021>

0743-7315/© 2016 Elsevier Inc. All rights reserved.

Currently, extended High Performance Computing architectures are heterogeneous platforms composed of distributed memory systems, where every processing element (node) has a multicore architecture with a certain number of cores [10]. These heterogeneous platforms offer higher peak performance compared to traditional CPUs being both energy and cost efficient. Nevertheless, programming for heterogeneous environments is a tedious task and has a long learning curve. In this context, parallel implementations should be modified in order to be executed over such heterogeneous architectures. Therefore, it is necessary to have a good knowledge of both, the algorithm to parallelize and the computational resources used for the implementation [13]. Furthermore, accelerators, such as GPUs, FPGAs, Intel Xeon Phi coprocessors and so on, can be included on these architectures.

Inventory control problems for perishable products have been studied since the seventies. Recently several new inventory models for controlling perishable items have appeared. For instance, [14] considers an inventory system for perishable products where demand follows a Poisson distribution, service interruptions, retrial demands and negative customers are considered. The study obtains a (s, S) policy, i.e. an order is placed whenever inventory drops below a level s . In [18], a MILP (Mixed-Integer Linear Programming) approximation for an order up to level policy is presented. For practical cases, the solutions obtained by this approximation are less than 5% higher than those obtained by the optimal policy. Authors in [15] consider a joint dynamic pricing and inventory control policy for stochastic inventory and perishable products. They obtain the optimal dynamic policy by solving a Hamilton - Jacobi - Bellman equation. Authors in [5] study a joint pricing and a dynamic production policy for perishable items, with the addition that shortages are not allowed, deriving the optimal sales price and designing algorithms to compute them. However, to the best of our knowledge, no study has been done with respect to the use of heterogeneous platforms for computer-intensive inventory control models for deriving optimal order policies for perishable product inventory control. Related to the acceleration of the Monte-Carlo simulation, required for the model described in this paper, recent works can be found in the literature in order to efficiently compute this simulation on heterogeneous platforms [11,16].

In this study, the algorithm to solve the perishable inventory control problem has been implemented on Multi-GPU clusters, as an example of heterogeneous platforms. In a Multi-GPU cluster, there are several distributed memory nodes, where every node is composed of a shared memory multicore architecture and one or more GPUs. Each GPU has a separate memory space. Therefore, if data dependencies among GPUs occur, communication via PCI-Express is needed. The main advantages of using Multi-GPU computing are as follows: (1) the use of massively parallel platforms (GPUs) facilitates speeding up the most computationally intensive tasks, because these devices have enough computational power to calculate vectorial computation schemes; and (2) when the problem to solve is large enough, several distributed memory nodes can be used (with/without GPUs). In the literature, heterogeneous computing has been used to accelerate the execution of a wide variety of numerical models [17,23]. For the inventory control problem, the use of parallel and distributed platforms improves the accuracy of the results; parallel computation allows executions with a larger number of simulations to solve a particular problem in a reasonable runtime.

The parallel computational model associated to this problem can be described in terms of a set of independent tasks. However, the computational burden associated to each task is different and therefore a workload balancing problem may appear when the workload is distributed in a blind way. The problem of assigning tasks to processing elements is well-known in the literature as

the Bin-Packing problem [6]. Because this problem is NP-hard, several heuristics have been developed to distribute the workload efficiently among the available platforms.

The rest of the paper is organized as follows. Section 2 discusses the perishable inventory control problem. In Section 3, the implemented sequential algorithm to solve the problem is outlined. Section 4 describes the details of the implemented parallel versions and several heuristics for distributing the workload among the available processing elements are described and evaluated. Section 5 studies and analyses the results of running various implementations on heterogeneous platforms. Finally, conclusions are drawn in Section 6.

2. Description of the model

The basis of the implementations presented in this work is an algorithm developed in Matlab to solve a MINLP (Mixed Integer NonLinear Programming) problem, [8]. The algorithm sets a schedule, along a finite number of periods T , for the quantities that must be provided of a product in order to satisfy the demand under a β service level requirement implying that for every period and in terms of the expected value, more than a fraction β of the demand is satisfied. This is equivalent to at most a fraction $(1 - \beta)$ of demand is lost due to a stock-out. The shelf life of the product after which the product perishes and becomes waste is $J < T$ periods. Furthermore, items are served following a FIFO rule: products are served starting from the oldest ones.

The optimization problem consists of finding the quantities of the product that must be provided at every period such that the restrictions are met and the value of the objective function is minimized. These quantities have to be determined at the beginning of the planning horizon, following a static uncertainty strategy over demand: the order quantities have to be defined for all T periods before realization of demand. If the decision maker is able to adapt the production over the periods as the demand is observed, one may use another strategy. An approach to this scenario is discussed in [9].

The model is specified as follows.

Indices

t	period index, $t = 1, \dots, T$
j	age index, $j = 1, \dots, J$, with J the shelf life of the product

Data

d_t	Demand at every period following normal distributions with mean $\mu_t > 0$ and variance $(cv \times \mu_t)^2$ given by a coefficient of variation cv , equal at every period.
k	Ordering cost, $k > 0$
c	Unit cost, $c > 0$
h	Holding cost, $h > 0$
w	Waste cost, can be negative, but $w > -c$
β	Required service level, $0 < \beta < 1$

Variables

$Q_t \geq 0$	Order quantity in period t . Q represents the vector (Q_1, \dots, Q_T)
$Y_t \in \{0, 1\}$	Indicates if order takes place in period t . $Y_t = 1$ if and only if $Q_t > 0$. Y denotes vector (Y_1, \dots, Y_T)
\mathbf{X}_t	Lost sales at period t
\mathbf{I}_{jt}	Inventory of age j at the end of period t , $I_{j0} = 0$, $\mathbf{I}_{jt} \geq 0, j = 1, \dots, J$.

The notation $E(\cdot)$ is used to express the expected value of a stochastic variable (in bold face) and $(\cdot)^+ = \max(\cdot, 0)$.

The objective function to be minimized depends on the vector $Q = (Q_1, \dots, Q_T)$ and can be defined as follows:

$$f(Q) = \sum_{t=1}^T \left(C(Q_t) + E \left(h \sum_{j=1}^{J-1} \mathbf{I}_{jt} + w \mathbf{I}_{Jt} \right) \right), \quad (1)$$

where

$$C(x) = k + cx, \quad \text{if } x > 0, \text{ and } C(0) = 0. \quad (2)$$

The inventory of products of age j at the end of period $t = 1, \dots, T$ follows the FIFO rule:

$$\mathbf{I}_{jt} = \begin{cases} \left(Q_t - (\mathbf{d}_t - \sum_{j=1}^{J-1} \mathbf{I}_{j,t-1})^+ \right)^+ & j = 1, \\ (\mathbf{I}_{j-1,t-1} - \mathbf{d}_t)^+ & j = J, \\ \left(\mathbf{I}_{j-1,t-1} - (\mathbf{d}_t - \sum_{i=j}^{J-1} \mathbf{I}_{i,t-1})^+ \right)^+ & \text{Otherwise.} \end{cases} \quad (3)$$

The service level requirement can be expressed as follows:

$$E(\mathbf{X}_t) \leq (1 - \beta)\mu_t, \quad t = 1, \dots, T. \quad (4)$$

The value of the lost sales in each period t is given by

$$\mathbf{X}_t = \left(\mathbf{d}_t - \sum_{j=1}^{J-1} \mathbf{I}_{j,t-1} - Q_t \right)^+. \quad (5)$$

The expected value of the lost sales is a function known as the *loss-function*, that in general does not have a closed-form expression. Some approximations to this function can be found in the literature in [12,20,21,24]. Monte-Carlo simulation has been used for this model in order to obtain a good estimation of the *loss-function*. Having set the conditions detailed before, the problem of finding the quantities that have to be ordered to minimize the objective function (1) and at the same time finding the timing vector $Y \in \{0, 1\}^T$ for the optimal policy, is a MINLP (Mixed Integer NonLinear Programming) problem. As will be shown, the computational model associated to this problem exhibits appropriate properties for being implemented on heterogeneous high performance computers architectures.

3. Sequential algorithm

This section describes the algorithm for solving the inventory control problem of Section 2. Algorithm 1 represents the highest level of abstraction for the solution method. Firstly, the method generates all timing vectors in set $\{0, 1\}^T$ that are feasible for the values of T and J . A timing vector $Y \in \{0, 1\}^T$ is considered infeasible if it contains a series of more than J sequential zeros (J or more periods without placing an order), since demand cannot be met. For a specific timing vector Y , Algorithm 2 determines, for each period, the order quantity of product $Q(Y)$ that minimizes objective function (1).

Algorithm 1 performs an exhaustive evaluation over the set of feasible timing vectors (except for the cases discarded by the lower bound check), to find the optimal timing vector Y^* and, at the same time, the vector of the optimal order quantities $Q^* = Q(Y^*)$ with associated cost $f(Q^*)$. Each vector Y is composed of a number of cycles: a cycle is a set of periods in which a replenishment is set in the first period of the cycle, to cover the demand for all periods of the cycle. In any vector Y , a cycle is identified by a period with value 1 and a sequence of zeros from none to $J - 1$.

For the determination of the optimal quantities, $Q(Y)$, the method uses a table of so-called base order quantities $\hat{q}_{R,t}$ with the required quantity at period t to cover the demand of a cycle of R periods when inventory is zero. This quantity is the order quantity for those order periods where no inventory is available. Let $m = \sum Y_t$ be the number of orders, then Algorithm 2 starts by identifying m -dimensional vector A with the order moments and vector B with the last period of each cycle.

For each replenishment period, that is, any period t in which $Y_t = 1$, Algorithm 2 calculates the minimum order quantity that

Algorithm 1 *Ally()*: Finds the optimal timing vector (Y^*) calculating the optimal cost of all feasible timing vectors Y

```

1: Generate all feasible  $Y$ 
2: for all  $Y$  do
3:    $Q_Y = \text{MINQ}(Y)$ ; # Algorithm 2
4:   Determine  $f(Q_Y)$ 
5: end for
6: return  $Y^*$ ;  $Q^*$ ;  $f(Q^*) = \text{mincost}$ 

```

Algorithm 2 *MinQ(Y)*: $Q(Y)$ optimization

```

1: Generate vectors  $A$  and  $B$  for  $Y$ 
2: for  $i = 1$  to  $m$  do
3:   if  $A_i = 1$  or  $B_{i-1} - A_{i-1} = J$  then # No Invent.
4:      $Q_{A_i} = \hat{q}_{(B_i - A_i), A_i}$ ;
5:   else
6:     solve  $f_{\text{loss}}(Q_{A_i}, A_i, B_i) = (1 - \beta)\mu_{B_i}$ 
7:   end if
8: end for
9: return  $Q$ ;

```

guarantees that the service level is fulfilled for that cycle. For each cycle, the optimal order quantity Q_{A_i} is the one that makes that the lost sales are equal to the lower bound of (4) in period B_i . When there is no inventory, Q_{A_i} can be taken as the base quantity [8]. When inventory is not zero, Monte-Carlo simulation of the inventory is used to give an accurate approximation. Function *floss* (q, a, b) in Algorithm 3 handles the simulation and returns the approximation of the expected value of lost sales for the last period of the cycle when q units are ordered at period a for a cycle with $(b - a) + 1$ periods. An external function, like the secant method in [8], can iteratively look for the value of q for which (4) is fulfilled with equality.

Algorithm 3 *floss(q, a, b)*: Monte-Carlo method approximating $E(\mathbf{X})$ based on N sample paths $d_{t,n}$

```

1: for  $n = 1$  to  $N$  do
2:   Simulate  $I_{j,t,n}$ ,  $t = a, \dots, b$ ;  $j = 1, \dots, J$ 
3:    $x_{bn}$  loss in sample  $n$  at end period  $b$ ,  $\sum_{n=1}^N x_{bn}$ 
4: end for
5: Average loss  $X_b = \frac{1}{N} \sum_{n=1}^N x_{bn}$  approximates  $E(\mathbf{X})$ 
6: return  $X_b$ ;

```

4. Parallel implementations

The order of complexity of Algorithm 1, is related to the number of feasible timing vectors Y , which depends on the values of J and T . Independently of the value of J , the number of feasible cases to process increases exponentially with the value of T . This means that the complexity is $O(e^T)$. Moreover, in Algorithm 2, the optimal value of the $Q(Y)$ vector is found for a specific feasible Y ; its complexity depends on the number of iterations needed to solve the equation in line 6 which is bounded by T . Evaluation of *floss* in Algorithm 3 (Monte-Carlo simulations) requires N simulations of the inventory for every period in the cycle and all possible ages of the product $(1, \dots, J)$.

Therefore, the order of complexity of the whole method to find the optimal timing vector Y and the optimal quantities, is approximately $O(N \cdot T \cdot e^T)$.

Algorithm 3 requires simulation to obtain approximations of the *floss* function. This function consumes most of the computational runtime of the algorithm. A higher number of demand paths

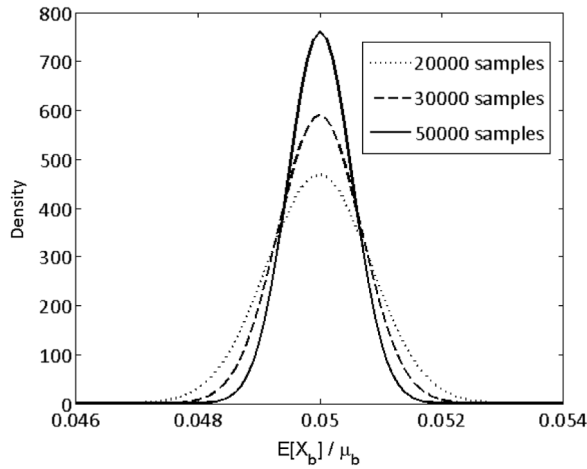


Fig. 1. Density functions of the estimation of the percentage of the expected value of $\frac{X_b}{\mu_b}$ according to the Algorithm 3 using $N = 20000, 30000, 50000$ sample paths.

in the Monte-Carlo simulation provides more accurate approximations of the lost sales (*floss* function). The standard deviation of the estimate $\text{std}(X_b)$ reduces with N according to $\text{std}(X_b) = \frac{\text{std}(x_{bn})}{\sqrt{N}}$. Fig. 1 shows the density distribution of estimating the percentage of lost sales X_b , when X_b is estimated based on $N = 20000, 30000, 50000$ sample paths for a case that analytically gives the value $\frac{E(X_b)}{\mu_b} = 0.05$, that is, the lost sales is 5% of the expected demand μ_b .

Therefore, approximating the expected lost sales (X) accurately requires a sufficiently high number N of sample paths. This represents the main computational workload of the problem.

From a parallel point of view, Algorithm 1 can be decomposed into independent tasks, as there are hardly dependencies in the computation associated to each Y . So, the loop “for” in Algorithm 1 can be executed in parallel. However, the computational burden associated to each Y is different and requires distributing vectors Y among the available processing elements in a balanced way. To do that, an estimation of the workload for each Y is needed.

The workload associated to a feasible timing vector Y (Algorithm 2) can be estimated. In this way, the most time-consuming task of this approach (the Monte-Carlo simulations) can be taken into account to find the optimal solution. More precisely, the Monte Carlo simulation for an order depends on the length of the replenishment cycle which varies between 1 and J periods. Following Algorithm 2, for cycles in which there is no left inventory, the quantity of product needed is already known from the so-called basic quantities. After that, only one Monte-Carlo simulation must be carried out. For any other period, an external function iteratively performs Monte-Carlo simulation up to a certain accuracy. With the method used in [8], the number of iterations of Algorithm 3 needed to solve the equation line 6 of Algorithm 2 depends on the accuracy chosen and an average value of K iterations can be assumed. The workload of any vector Y can be taken as the accumulation of Monte-Carlo simulations per period that are needed, as sketched in Algorithm 4.

Now that the parallel decomposition of the problem has been analysed, the details of the implemented approaches are described in the following sections. Section 4.1 describes the computational platform and programming interfaces for the MPI-Pthreads implementation and Section 4.2 gives the Multi-GPU implementation. In Section 4.3, the heuristics used to distribute the workload among processing elements are discussed. Implementations described in 4.1 and 4.2 have been carried out on a heterogeneous platform which consists of a multi-GPU cluster (composed of several nodes with multicores and GPU devices). The exploitation of a

Algorithm 4 Rank(Y): Workload approximation for Y

```

1: Generate vectors  $A$  and  $B$  for  $Y$ 
2: for  $i = 1$  to  $m$  do
3:    $\text{length\_cycle} = B_{i-1} - A_{i-1}$ ;
4:   if  $A_i = 1$  or  $\text{length\_cycle} = J$  then # No Invent.
5:      $v += \text{length\_cycle}$ ;
6:   else
7:      $v += \text{length\_cycle} \cdot K$ 
8:   end if
9: end for
10: return  $v$ ;

```

heterogeneous platform has two main advantages: (1) larger problems can be solved; and (2) it can be done in less runtime.

- **MPI-Pthreads:** obtains the parallelism of the nodes and the multicore processors available in the cluster. Therefore, programming with POSIX Threads (Pthreads¹) and distributed programming based on Message Passing Interface (MPI) are used [3,22].
- **Multi-GPU:** uses GPUs in order to parallelize the Monte-Carlo simulation, which is the most computationally demanding task of the problem. For this, the CUDA² interface is used. In this case, the implementation uses MPI and CUDA.

4.1. MPI-Pthreads implementation

Focusing now on the MPI-Pthreads implementation, parallelism has been exploited on two levels: at node level (distributed memory) and at multicore level (shared memory). On one hand, at node level, and thanks to its portability, Message Passing Interface (MPI) [22] has become a standard for multiple-processor programming of code that runs on a variety of machines. On the other hand, there are multiple ways of parallelizing routines in shared memory models. One standard library is POSIX threads (or Pthreads), which supplies a unified set of C routines facilitating the use of threads in codes [3].

A hybrid parallelization (MPI and Pthreads) of the optimization algorithm for perishable inventory control problem described in Section 3 has been implemented. At the beginning of Algorithm 1, the set of timing vectors Y are distributed among cores following the rules of a heuristic designed for balancing the workload which is discussed in Section 4.3. This MPI-Pthreads implementation has been tested in a Bullx cluster and obtained results are described in Section 5.

4.2. Multi-GPU implementation

The Multi-GPU version has been based on the exploitation of several GPUs for the parallelization of the Monte-Carlo method, computed by the *flossGPU* function (see Algorithm 5). In this implementation, each MPI process can open one or two threads and every thread initializes the CUDA interface. Timing vectors Y are distributed among cores in the same way as in the MPI-Pthreads implementation (following the rules of a heuristic designed for balancing the workload which is discussed in Section 4.3). Only Monte-Carlo simulations are computed on the GPU, the remaining tasks are computed by the CPU (cores). The N sample paths in the Monte-Carlo simulation have been implemented to run independently. At the same time, the calculation of the inventory level at each age (3) depends only on

¹ <https://computing.llnl.gov/tutorials/pthreads/>

² <https://developer.nvidia.com/cuda-toolkit>

Algorithm 5 *flossGPU*(q, a, b): Monte-Carlo method for obtaining an approximation of $E(\mathbf{X})$

```

1: for  $t = a$  to  $b$  do
2:   for  $n = 1$  to  $N$  do #GPU
3:     Update  $I_{j,t,n}, j = 1, \dots, J$  and  $x_{tn}$ 
4:   end for
5: end for
6: Average loss  $X_b = \frac{1}{N} \sum_{n=1}^N x_{bn}$  approximates  $E(\mathbf{X})$ 
7: return  $X_b$ ;

```

the inventory level of the previous period. Then, proceeding by periods, a CUDA kernel is responsible to compute N simulations of the J ages of the inventory in parallel.

In Algorithm 5, Monte-Carlo simulation (*flossGPU* function) is performed on one or several GPUs. The external loop simulates the periods, while the inner loop computes N samples on the GPU. This way, each GPU launches N threads in parallel, computing the same sequence of instructions over different input data. Thus, the programmer can consider the GPU as a set of SIMT (Single Instruction, Multiple Threads). Each GPU thread stores its partial computation in the shared memory. To generate the approximation of X , one reduction of the N values computed and stored in shared memory has to be included. One additional issue has been the optimization of the occupancy on the GPU. The occupancy determines how well the hardware is kept busy with the goal of hiding latencies, by switching between active warps, due to memory operations and paused warps. Occupancy is closely related to the thread block size (BS) and the number of registers and shared memory size used by a kernel. Therefore, a good choice of BS will improve the performance. Experimental results of Section 5 have considered the best BS size (512) to optimize the performance of the GPU code.

Section 5 presents the experimental results of both implementations, MPI-PTHREADS and Multi-GPU.

4.3. Heuristics for the Bin-Packing problem

The problem for distributing the workload of Algorithm 1 among processors can be modelled as a Bin packing problem. The Bin-Packing problem is a combinatorial optimization problem (NP-complete) and for the inventory problem, it can be described as follows: Given a set of L independent runs of the same algorithm with workload $0 < w_i < C, i = 1, \dots, L$, for each independent run i of Algorithm 2 (the feasible timing order vectors) and a set of P processors (Bins), distribute the runs of the algorithm among the processors $p = 1, \dots, P$ such that the maximum workload assigned to a processor is as low as possible. See [6] for a general formulation of the Bin-Packing problem.

Due to the hardness of finding optimal solutions for this kind of problems, some heuristics capable of finding acceptable solutions in a reasonable time are usually considered, some of them are based on evolutionary computation, e.g. [1]. Here, we have tested three heuristics (H1, H2 and H3) which are able to provide approximate solutions to the workload balancing problem. These heuristics attempt to distribute the workload w_i equally over the available cores.

1. (H1) is based on Round Robin scheduling: Sort w_i from high to low and assign to p following the pattern $(1, \dots, P, P - 1, \dots, 1, 1, \dots)$.
2. (H2): For $i = 1, \dots, L$ assign w_i to processor p with the lowest gathered workload.
3. (H3): Similar to heuristic H2, but in this case w_i is sorted previously from high to low values.

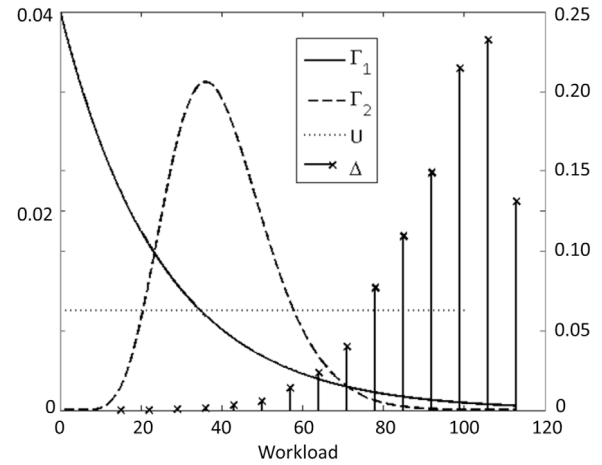


Fig. 2. Probability density functions of $\Gamma(10, 4)$, $\Gamma(1, 25)$, $U(0, 100)$ used to generate the 4000 samples combined with the distribution of the inventory instance, Δ .

To evaluate the heuristics, three instances called Γ_1 , Γ_2 and U were generated by taking $L = 4000$ weights w_i at random from gamma distributions $\Gamma(10, 4)$ and $\Gamma(1, 25)$ and from the uniform distribution $U(0, 100)$, respectively. Moreover, an instance Δ was taken from a real workload distribution of the inventory control problem with $T = 15$ and $J = 3$. Fig. 2 sketches the corresponding distributions.

To calculate the workload balancing assigned to each processor, the coefficient of Gini (G) [7] has been used in this work following the examples in [4,19,2]. This coefficient has been widely used in economy to measure the degree of inequality and wealth distribution for large populations. The Gini index varies between 0 representing complete equity and 1 if all the wealth (workload) of the population belongs to only one individual (processor) [7]. Let W_p be the workload assigned to processor $p, p = 1, \dots, P$ sorted in ascending order. The Gini coefficient G is

$$G = \frac{2 \sum_{p=1}^P p W_p}{P \sum_{p=1}^P W_p} - \frac{P+1}{P}. \quad (6)$$

Table 1 summarizes the behaviour of the heuristics, considering the Gini coefficient (G) for the samples (compared to a blind distribution of the workload (HR column)). Clearly, this table shows that heuristic H3 is, at least, better than heuristics H1 and H2 in an order of magnitude, and that a random workload assignment (HR) is at least two orders of magnitude worse than H3 and one for H1 and H2. From the data in Table 1, can be concluded that heuristic H3 presents the best results, being able to balance the workload almost perfectly.

5. Experimental results

For the evaluation of the implementations of Algorithm 1 to solve the perishable inventory control problem, a Bullx cluster composed of eight nodes with a total of 128 cores has been used. In particular, each node contains two Intel Xeon E5 2650 and a total of 16 cores. The eight nodes are interconnected by a QDR/FDR InfiniBand port embedded on the motherboard. Four of the nodes have two GPUs (total of 2×4 NVIDIA Tesla M2070). The main characteristics of the GPUs are described in Table 2. The experiments have been compiled with NVIDIA CUDA (6.5 version), gcc compiler (4.8.1 version) with -O2 as the optimization option and OpenMPI as the MPI library.

Table 1

Gini coefficient in 10^{-6} of the workload distribution of heuristics H1, H2 and H3 versus a random allocation HR. $P = 8, 16, 32, 64$, weights w_i from Γ_1, Γ_2 and uniform distribution, Δ is an empirical distribution.

	P	H1	H2	H3	HR
Γ_1	8	130	480	12	3,700
	16	250	1200	52	7,600
	32	580	2000	140	13,000
	64	3200	3900	1500	21,000
Γ_2	8	690	1000	19	23,000
	16	1200	2100	19	39,000
	32	3300	4100	78	61,000
	64	8000	7600	100	79,000
U	8	130	750	7.4	12,000
	16	180	1400	8.6	18,000
	32	220	2400	39	26,000
	64	370	4100	54	40,000
Δ	8	43	190	30	3,200
	16	220	460	230	4,500
	32	300	1000	320	7,100
	64	550	2000	380	9,800

Table 2

Characteristics of the GPUs considered for the evaluation.

	Tesla M2070
Peak performance (double prec.) (GFLOPs)	515
Peak performance (simple prec.) (GFLOPs)	1030
Device memory (GB)	5.24
Clock rate (GHz)	1.2
Memory bandwidth (GBytes/sec)	150
Multiprocessors	14
CUDA cores	448
Compute Capability	2
DRAM TYPE	GDDR5

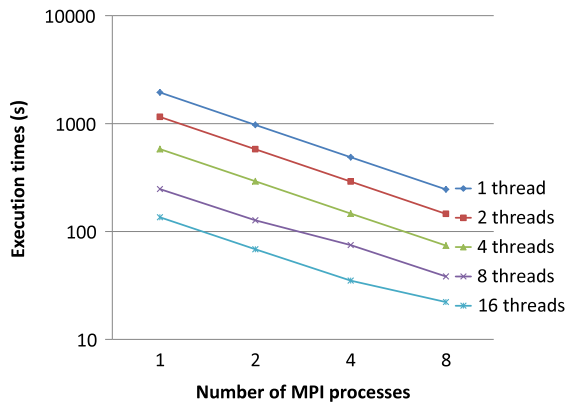


Fig. 3. Executions time, in seconds, of the MPI-Pthreads implementation using H3 heuristic.

In order to test the parallel implementations, an inventory control problem of perishable products based on $T = 15$ and $J = 3$ has been considered. It comprises a total of $L = 5768$ feasible timing vectors Y . For Monte-Carlo simulation, $N = 200000$ samples are used in Algorithm 3.

Fig. 3 illustrates the values of the execution time for the MPI-Pthreads implementation using 1, 2, 4, 8 and 16 threads and 1, 2, 4 and 8 MPI processes. A maximum of eight nodes has been considered. At every node, only one MPI process is executed and the number of Pthreads launched per node varies from 1 to 16 (one thread per physical core). The number of cores P in which the heuristic balances the workload associated to L feasible timing vectors Y of the problem is equal to the number of MPI processes \times the number of threads.

Fig. 4 shows the speed up of the MPI-Pthreads implementation using H3 ranging from 1.7 for the version with a single MPI

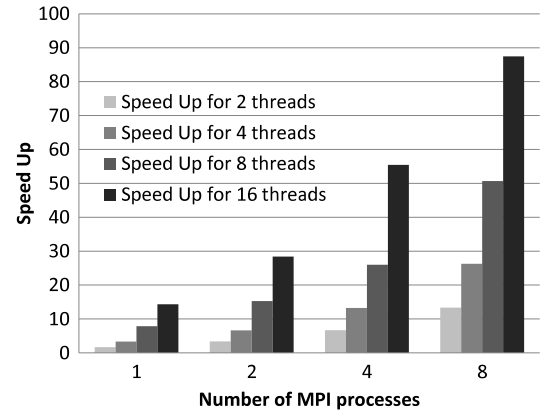


Fig. 4. Speed Up of MPI-Pthreads implementation versus the sequential code, considering H3 heuristic.

Table 3

Execution time, (in seconds), of the multi-GPU version. Total: total execution runtime; GPU (s): runtime (in seconds) of the *flossGPU* function; % GPU: percentage of the total execution time devoted to Monte-Carlo function (*flossGPU* function) using GPU computing; AF: acceleration factor of the implementation 1 GPU (1 MPI, 1 thread) versus the 2 GPU, 4 GPU, 6 GPU, and 8 GPU versions.

	Total (s)	GPU (s)	% GPU	AF
1 GPU (1 MPI, 1 thread)	185.91	56.38	30.33	–
2 GPU (1 MPI, 2 threads)	93.50	28.32	30.29	1.99
4 GPU (2 MPI, 2 threads)	44.69	14.18	31.74	4.16
6 GPU (3 MPI, 2 threads)	33.05	9.45	28.59	5.63
8 GPU (4 MPI, 2 threads)	24.07	7.07	29.39	7.72

process with 2 threads to 87.5 for 8 MPI processes with 16 threads each. The performance increases with the number of MPI processes. So, the best results in terms of performance are obtained for 8 MPI (8 nodes). The Y-axis of the figure highlights how the number of threads in a MPI process affects the total performance of the algorithm. To be more precise, for the same number of MPI processes, duplication of the number of threads (and cores) offers an acceleration factor of nearly 2.

Table 3 shows the executions time, (in seconds), of the multi-GPU version using 1, 2, 4 and 8 GPUs. Notice that the information in brackets in the first column identifies the mapping on the cluster of each version. For instance, 6 GPU (3 MPI, 2 threads) is mapped using 3 MPI processes and 2 threads. Moreover, every thread is associated to one core and one GPU device. It can be observed that the acceleration factor ranges from $1.99\times$ and $7.72\times$ for 2 GPU and 8 GPU, respectively. Therefore, the speed up is approximately linear. Column “% GPU” represents the percentage of the total execution time devoted to Monte-Carlo function (*flossGPU* function) using GPU computing, and it can be observed that it is mostly equal to one third of the total runtime.

Comparing results in Fig. 3 (MPI-Pthreads implementation) with data in Table 3 (multi-GPU version), it can be observed that the runtime when 1 MPI and 1 thread are considered for H3 (1941.46 s) is much higher ($10\times$ approximately) than the runtime for 1 MPI, 1 thread and 1 GPU (185.91 s). This case illustrates the power of the GPU computation to accelerate this kind of problems. Moreover, focusing our attention on using 4 MPI, 2 threads and 8 GPUs of Table 3, the total runtime (24.07 s) is very similar to the total runtime for 8 MPI and 16 threads of the MPI-Pthreads version (22.19 s). Therefore, in this work, the total runtime of the optimization algorithm for a perishable inventory control problem has been accelerated by means of two parallel implementations MPI-Pthreads and multi-GPU. The results are similar for the configurations with the highest number of nodes, threads and GPUs considered; i.e. for 8 MPI and 16 threads (for the MPI-Pthreads version) and for 4 MPI, 2 threads and 8 GPU (for the multi-GPU version).

6. Conclusions

In this paper, two parallel implementations of an optimization algorithm for a perishable inventory control problem have been studied. A MPI-PTHREADS version designed to exploit the parallelism in both multi-core and distributed platforms; and a multi-GPU version which uses GPU computing to accelerate the most computationally intensive task (Monte-Carlo simulation). The MPI-PTHREADS parallelization using a static workload balancing based on the H3 heuristic has shown a good scalability. The results have shown that MPI-PTHREADS implementation has a good scalability when increasing both the number of processes (nodes) and threads (cores). Therefore, using 8 MPI processes and 16 threads, the performance has been increased in a factor of 87 versus the sequential code. Finally, parallelization of the Monte-Carlo function (*GPUloss* function) with 8 GPUs speeds up the running time with a factor of 81 versus the sequential code. It has been shown that both parallel implementations, MPI-PTHREADS and multi-GPU, can considerably accelerate the optimization algorithm for the perishable inventory control problem studied in this work.

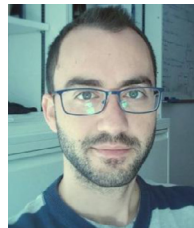
Implemented software for the perishable inventory control on heterogeneous platforms is freely available through the following website: <https://sites.google.com/site/hpcoptimizationproblems/inventory-problem>.

Acknowledgments

Alejandro Gutierrez-Alcoba is a fellow of the Spanish FPI programme. This paper has been supported by The Spanish Ministry (TIN2015-66680) and Junta de Andalucía (P11-TIC-7176), in part financed by the European Regional Development Fund (ERDF).

References

- [1] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, *ACM Comput. Surv.* 35 (3) (2003) 268–308.
- [2] A. Burkimsher, I. Bate, L. Indrusiak, Scheduling HPC workflows for responsiveness and fairness with networking delays and inaccurate estimates of execution times, in: F. Wolf, B. Mohr, D. an Mey (Eds.), *Euro-Par 2013*, in: LNCS, vol. 8097, Springer, Berlin Heidelberg, 2013, pp. 126–137.
- [3] D. Butenhof, Programming with POSIX Threads, in: *Professional Computing Series*, Addison-Wesley, 1997.
- [4] D.G. Feitelson, Workload modeling for performance evaluation, in: *Performance Evaluation of Complex Systems: Techniques and Tools*, Performance 2002, in: Tutorial Lectures, Springer-Verlag, London, UK, UK, 2002, pp. 114–141.
- [5] L. Feng, J. Zhang, W. Tang, Optimal inventory control and pricing of perishable items without shortages, *IEEE Trans. Autom. Sci. Eng.* 13 (2) (2016) 918–931.
- [6] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, USA, 1979.
- [7] C. Gini, Measurement of inequality of incomes, *Econ. J.* 31 (2014) 124–126.
- [8] A. Gutierrez-Alcoba, E.M.T. Hendrix, I. García, G. Ortega, K.G.J. Pauls-Worm, R. Haijema, On computing order quantities for perishable inventory control with non-stationary demand, in: O. Gervasi, et al. (Eds.), *ICCSA 2015*, Part II, in: LNCS, vol. 9156, Springer, Cham, 2015, pp. 429–444.
- [9] A. Gutierrez-Alcoba, R. Rossi, B. Martin-Barragan, E.M.T. Hendrix, A simple heuristic for perishable item inventory control under non-stationary stochastic demand, *Int. J. Prod. Res.* (2016) 1–13. <http://dx.doi.org/10.1080/00207543.2016.1193248>.
- [10] J.L. Hennessy, D.A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, 2011.
- [11] S.-H. Hung, M.-Y. Tsai, B.-Y. Huang, C.-H. Tu, A platform-oblivious approach for heterogeneous computing: A case study with monte carlo-based simulation for medical applications, in: *Proceedings of 2016 FPGA*, ACM, New York, NY, USA, 2016, pp. 42–47.
- [12] A. Kurawarwala, H. Matsuo, Forecasting and inventory management of short life-cycle products, *Oper. Res.* 44 (1996) 131–150.
- [13] A. Lastovetsky, Heterogeneity in parallel and distributed computing, *J. Parallel Distrib. Comput.* 73 (12) (2013) 1523–1524.
- [14] P.V. Laxmi, M. Soujanya, Perishable inventory system with service interruptions, retrieval demands and negative customers, *Appl. Math. Comput.* 262 (2015) 102–110.
- [15] S. Li, J. Zhang, W. Tang, Joint dynamic pricing and inventory control policy for a stochastic inventory system with perishable products, *Int. J. Prod. Res.* 53 (10) (2015) 2937–2950.
- [16] S. Miranda, J. Feldt, F. Pratas, R.A. Mata, N. Roma, P. Tomás, Efficient parallelization of perturbative monte carlo QM/MM simulations in heterogeneous platforms, *Int. J. High Perform. C* (2016) 1–13. <http://dx.doi.org/10.1177/1094342016649420>.
- [17] G. Ortega, J. Lobera, I. García, M. Arroyo, E. Garzón, Parallel resolution of the 3D Helmholtz equation based on multi-graphics processing unit clusters, *Concurr. Comput.* 27 (13) (2015) 3205–3219.
- [18] K.G.J. Pauls-Worm, E.M.T. Hendrix, A. Gutierrez-Alcoba, R. Haijema, Order quantities for perishable inventory control with non-stationary demand and a fill rate constraint, *Int. J. Prod. Econ.* (2015) 1–9. <http://dx.doi.org/10.1016/j.ijpe.2015.10.009>.
- [19] Z. Ren, J. Wan, W. Shi, X. Xu, M. Zhou, Workload analysis, implications, and optimization on a production hadoop cluster: a case study on taobao, *IEEE Trans. Serv. Comput.* 7 (2) (2014) 307–321.
- [20] R. Rossi, S.A. Tarim, S. Prestwich, B. Hnich, Piecewise linear lower and upper bounds for the standard normal first order loss function, *Appl. Math. Comput.* 231 (2014) 489–502.
- [21] S.K.D. Schrijver, E.-H. Aghezzaf, H. Vanmaele, Double precision rational approximation algorithm for the inverse standard normal first order loss function, *Appl. Math. Comput.* 219 (3) (2012) 1375–1382.
- [22] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, *MPI-The Complete Reference, Volume 1: The MPI Core*, MIT Press, Cambridge, MA, USA, 1998.
- [23] S. Tabik, A. Villegas, E.L. Zapata, L.F. Romero, Optimal tilt and orientation maps: a multi-algorithm approach for heterogeneous multicore-GPU systems, *J. Supercomput.* 66 (1) (2013) 135–147.
- [24] G.R. Waissi, D.F. Rossin, A sigmoid approximation of the standard normal integral, *Appl. Math. Comput.* 77 (1) (1996) 91–95.



Alejandro Gutierrez-Alcoba gained both an M.Sc. in Mathematics and an M.Sc. in Computer Science in 2011 from the University of Málaga (Spain). In 2014, he began his Ph.D. at the department of Computer Architecture at the University of Málaga. He is a member of the TIC-146 Supercomputing-Algorithms research group of the University of Almería. His research focuses on the use of high performance computing (HPC) techniques for dynamic Global Optimization problems.



Gloria Ortega (<https://sites.google.com/site/gloriaortegalopez/>) received the Bachelor's degree in computer science and the M.S. and Ph.D. degrees from the University of Almería (Spain), in 2009, 2010, and 2014, respectively. From 2009, she has been working as a member of the TIC-146 Supercomputing-Algorithms research group. Currently, she has a Post-doctoral fellowship at the Informatics Department and her current research work is focused on High Performance Computing and Optimization. Some of her research interests include the study of strategies for load balancing the workload on heterogeneous systems and the parallelization of optimization problems.



Eligius M.T. Hendrix (<https://sites.google.com/site/eligiusshendrix/>) is a European researcher and professor with 30 years of experience in mathematical modeling and optimization questions. His work involved working with Master and Ph.D. students on a wide variety of practical problems in environmental and food science. His research is on the question how to use the mathematical structure of an optimization application in order to derive specific solution methods and algorithms. Recently, more focus is on how to adapt algorithms such that they can exploit modern computer structures. His affiliation is with Wageningen University and the last eight years at the Universidad de Málaga due to a scholarship. He is active in organizing workshops and has published over 60 journal articles on a wide variety of mathematical analysis in environmental and logistical questions.



Inmaculada Garcia received a B.Sc. degree in physics in 1977 from the Complutense University of Madrid, Spain, and a Ph.D. degree in 1986 from the University of Santiago de Compostela, Spain. From 1977 to 1987, she was an assistant professor, associate professor during 1987–1997, between 1997 and 2010 full professor at the University of Almería and since 2010 full professor at the University of Málaga. She was head of the Department of Computer Architecture and Electronics at the University of Almería for more than 12 years. During 1994–1995, she was a visiting researcher at the University of Pennsylvania, Philadelphia. She has been the head of the Supercomputing-Algorithms research group from 1995 until 2010. Her research interest lies in the field of high performance computing applications and parallel algorithms for irregular problems related to image processing, global optimization, and matrix computation.