Journal of Computational Science 24 (2018) 232–254

Contents lists available at ScienceDirect

Journal of Computational Science

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

GPU parallel implementation for asset-liability management in insurance companies

José L. Fernández [a], Ana M. Ferreiro-Ferreiro [b], José A. García-Rodríguez [b], Carlos Vázquez [b,*]

[a] University Autónoma of Madrid, Spain
[b] University of A Coruña, ITMATI and CITIC, Spain

ARTICLE INFO

Article history:
Received 22 July 2016
Received in revised form 30 April 2017
Accepted 10 May 2017
Available online 26 May 2017

Keywords:
Asset liability management (ALM)
Life insurance
Monte Carlo balance sheet projection
Parallelization
Multi-CPU
GPUs

ABSTRACT

In this work we present a stochastic asset liability management (ALM) model for a life insurance company together with its numerical simulation, based in a Monte Carlo balance sheet projection, and we carry out its efficient parallel computation using graphics processing units (GPUs) hardware. The liabilities of the company consist of a portfolio comprising with-profit life insurance policies, that evolve according to the policyholder saving account, surrender and biometric models. On the asset side, we mainly consider bonds, equity and cash, so that appropriate stochastic models are considered for their evolution. We consider some innovations with respect to literature in the modeling of the surrenders of the policyholders. Another important innovative aspect comes from the implementation of ALM in the new high performance computing architectures provided by GPUs technology. Numerical results illustrate the high speed up of the calculus by using GPUs and the coherence of the computations (asset evolution, default probabilities and so on).

1. Introduction

Asset liability management (ALM) is a broad denomination for the models that are used to forecast the evolution of a company along time, jointly projecting its assets and liabilities portfolios and computing the predicted cash inflows and outflows in the future. Such a company can be either a bank, an insurance company, or more generally any financial institution, a state pension fund or even a non financial corporation. Depending on the business model of the company, the specific definition of the underlying models for the assets and liabilities may vary. ALM tools aim to cover the liquidity and interest rate risks to ensure the solvency of the company, i.e. its capability to meet all its financial obligations. Another additional relevant goal is to increase the company profit. Thus, ALM can be generally understood as a management tool to maximize the investment returns, while minimizing the reinvestment risks. These models have a particular relevance in the insurance industry, because one central problem in the insurance business is precisely to guarantee the solvency of the company. Some ALM models for life insurance have been presented, for example, in [1] and the references therein. Additional references are [2–8], for example. Their numerical solution can be carried out by using Monte Carlo techniques [1]. Also high dimensional integration with sparse grids can be used to reduce the computational cost [9]. For certain simple cases, following Ito theory, one can obtain Kolmogorov partial differential equations. These equations can be solved using finite differences, see [5,10] or [8], for example.

Traditionally, the analysis of the cash flows has been computed for some previously designed (adverse) scenarios (feasible economic situations) to stress the ALM model of the company. However, nowadays the importance of stochastic ALM models for insurance companies has increased, mainly due to new regulations and a stronger competition. With Solvency II, see [11,12], insurance companies are allowed, and even encouraged, to develop their own in house ALM models and simulators to asses their risks. In the case of banks, the ALM is also required to manage liquidity risk in the Basel III regulation. The increase in computational power, thanks to the modern hardware architectures, has also made feasible the computation of more accurate approximations of the portfolio evolution using models of increasing complexity.

Our ALM model mainly consists in two coupled models: one for the asset portfolio and one for the liabilities.

Liability model. On the side of the liabilities, our portfolio comprises only with-profit life insurance policies, which are contracts

* Corresponding author.
E-mail address: carlosv@udc.es (C. Vázquez).

https://doi.org/10.1016/j.jocs.2017.05.010
1877-7503/

between the insurer company and the insured (policyholder). The policyholder pays a monthly premium, while the insurer pays monthly a variable rate benefit in a policyholder saving account, and also pays a benefit, either at maturity (or policy expiration) or if the insured dies before the expiry date of the policy. For the liability model, several issues are taken into account, three of the most important being: the saving account, the surrender and the mortality of the policyholder. In order to compute the projected liabilities cash outflows, some of them are known on beforehand, as they do not depend on stochastic economic variables: for example, the final payments due to the maturity of the policies. Others are uncertain and depend on the market evolution or on the stochastic behavior of policyholders biometry.

As we are considering with-profit policies, also known as participating life insurance contracts, that allow the policyholder to participate in the earnings of the company, the premiums paid by the policyholder are incorporated into a policyholder saving account, equipped with an interest rate guarantee and a surplus associated to the participation in the company benefits. So, we need to incorporate a model for the policyholder account evolution. In this respect, we follow the usual procedure in European countries, where the interest rate guarantee takes the form of a cliquet option (see [13], for example). Recently, some attention has been addressed to the optimization of the interest rate guarantee by the insurance company (see [14] or [13], for example). In particular, we mainly follow [13] concerning the model for the policyholder saving account evolution. Note that as we merge the policies of the same characteristics in a model point, actually we consider a model for the model point saving account.

We also include a model for the possibility of surrender the policy before maturity. We assume that surrender can happen when the rates in the market are higher than the profit offered by the company at some moment. So, the surrender model is parameterized by considering at any time the difference between the profit of the company and the profit provided by risk free bonds (see [15–17], for example).

We also consider a biometric model for the benefit that the company must pay in case of death of the policyholder before the policy's maturity.

Asset model. On the side of the asset, the portfolio involves fixed interest rate instruments (bonds), equities and cash. Insurance companies have a conservative investing strategy, and thus the biggest part of the monthly incomes are indexed to fixed rate assets. We assume that the remaining part is invested in equity. The bond and equity returns are related to the corresponding stochastic models of the interest rates and the asset value evolution, respectively.

Asset allocation model. We note that the assets portfolio is not constant in time, its composition evolves according to the investment strategy of the company. Thus, the asset portfolio is restructured (rebalanced) monthly to maintain the proportion of each asset (cash, equity and bonds) equal to a given target proportion in the asset mix. For this purpose, the investment strategy decides whether to buy or sell assets, and which kind of bonds or equities must be sold or bought. This target asset mix can be parameterized and represents an input for the ALM model. So, at each rebalance date we ensure that there is enough money to meet the obligations with the policyholders and the company's shareholders, and also to fit the pre-specified asset mix. In the sense of [18], we pose a simplified fixed asset mix modeling approach.

We note that in the stochastic programming setting, this is not understood as a dynamic asset allocation model, as the paths are not interconnected. The stochastic programming approach is mainly based on an event tree for the involved random variables, thus looking for the optimal decision for each node of the event tree, given the information available at that point. The optimal policy fits the conditions at each state and optimizes the long term. In the present work, we just rebalance the portfolio to maintain a target asset mix, however we do not solve the optimal allocation problem. Our objective is just to illustrate the advantages of GPU computations to improve the computational performance of the simple model, which does not include the dynamic optimal allocation through stochastic programming. For this approach, we address the readers to the books [19,20] or the references [21,18], for example. We are are exploring to incorporate the stochastic programming approach to the here proposed GPU implementation. In this respect, a useful starting reference could be the work in [22], that uses parallel computing techniques based on a PVM protocol for ALM with stochastic programming in a real Dutch pension fund.

Table 1
Asset and liability portfolios for a life insurance company.

Asset	Liability
Bonds Equities Cash	Life insurance policies

Balance sheet projection. The balance sheet of the company must be computed for any given economic scenario of the previously defined models. Each scenario is given by the evolution of several economic variables. If we know all the values of the variables at any time in the scenario, we can supply them to the assets and liabilities models, compute the surrendered policies, etc. Next, we compute the cash flows and the value of the assets and liabilities portfolios.

Numerical method and parallel implementation. The numerical method to compute the projected balance sheet of the company is based on a Monte Carlo framework. We generate thousands of economic scenarios provided by the stochastic models for the evolution of interest rates and assets.

Computing the balance sheet of the company for all times at each scenario can be a quite demanding computational task, that can take even a day or more, mainly depending on the number of policies in the portfolio, the number of scenarios, the temporal length of the forecast and the number of time steps per scenario. Taking all this into account, the model has been parallelized in multi-CPU clusters using OpenMP and also in heterogeneous systems, more precisely using GPUs. The current implementation has been developed for Nvidia GPUs using CUDA. We show performance results for the Multi-CPU and GPU implementations.

The plan of the article is as follows. In Section 2 we describe a with-profit life insurance company and the composition of the assets and liabilities portfolios. In Section 3 we describe the computations related to the balance sheet of the book of the company. In Section 4 we introduce the models for the asset, the liabilities and the asset allocation model. In Section 5 we describe the numerical scheme, while in Section 6 we present its implementation and its parallelization using GPUs. Finally, in the last section we show some results with the GPU implementation of a test example.

2. Insurance company: with-profit life insurance product

For any company we have two portfolios, an asset portfolio (a set of resources with the expectation that they will provide future benefits) and a liability one (debts or obligations that arise during the course of business operations). In the case of our with-profit life insurance company, the asset portfolio comprises bonds, equities and cash (see Table 1), while the liability portfolio contains with-profit life insurance policies.

Although there are many different types of insurance contracts, the basic principle is similar. A company (insurer) agrees to pay out money, which is referred as *benefits*, at specified times, upon the

occurrence of specified events. In return, the person purchasing insurance (insured or policyholder), agrees to make payments of prescribed amounts to the company. These payments are known as *premiums*. The contract between the insurer and the insured is referred to as the *insurance policy*.

2.1. Liability portfolio

In this section the characteristics of the insurance product are specified. In the Solvency II framework only the guaranteed part of the insurance product is considered, so that the financial bonus will not be included. Our liability portfolios are only composed of with-profit life insurance policies. A with-profit life insurance policy is a contract between the insurer company and the insured, where the insurance pays monthly a variable rate benefit to the policyholder, and also pays a benefit at maturity (or policy expiration) or before the policy expiry date, if the insured dies.

We consider discrete time steps and define the period k as $[t_k, t_{k+1}]$. In each period, we assume that premiums are paid at the beginning while benefits are paid at the end. Administrative costs are included in the premium.

The guaranteed part of the policy at period k is defined by the following notations and characteristics:

- Premium: π_k denotes the payment of the policyholder at the beginning of period k, if still alive.
- Death benefits: D_k is the guaranteed payment to an insurance holder at the end of period k, if the person dies during that period.
- Savings account of the policyholder: P_k is the value of the account where the policyholders premiums are saved and the participation on the profit of the company is included. Following [13], we consider an interest rate guarantee with cliquet style.

There can be hundreds of thousands of such policies in a liability portfolio. They can be computed one by one (stand alone computation) or alternatively it is possible to group similar policies in buckets, so called model points, to reduce the computational cost of the calculus (see [1], for example). For example, a model point could be the group of all the policies of policyholders between 30 and 35 years old, with a 10 years expiration date. A more detailed study of how to build the model points can be seen, for example, in [23].

We group the policies with the same age and maturity, so that T_i is the month of maturity of all the policies in the model point m_i and a_i is the age of all the policyholders in m_i. Our liability portfolio is given by the set of model points, $\mathcal{P} = \{m_i / m_i \text{ is a model point}\}$.

2.2. Asset portfolio

The asset portfolio involves fixed rate bonds, equities and cash.

- Bonds: insurance companies have a conservative investing strategy, and thus the largest part of the monthly incomes corresponds to fixed rate assets (bonds). We denote by n the number of different maturities of the bonds in the asset portfolio and we assume a set of increasing maturities $T_0, T_1, \ldots, T_n$. Moreover, for $i = 1, \ldots, n$, let N_i be the total nominal/principal of bonds with maturity T_i and let c_i denote the total of coupons associated to bonds with maturity T_i. Note that at position 0 we always store the nominal whose maturity is the current date. Moreover, for all the bonds with the same maturity we add their nominal values and coupons.
- Equity: the remaining part is invested in equity (stocks).
- Cash: is the cash account of the company.

Table 2
Sketch of the balance sheet.

Balance sheet	
Assets	Liabilities
Value of bonds	Reserves
Value of equity	
Value of cash	
P&L	

Moreover, we can define the value of the monthly discount as:

$$d_k = \frac{1}{\left(1 + \frac{r_k}{12}\right)^k}, \tag{1}$$

where r_k is the interest rate up to time T_k (zero coupon curve). Starting from $\hat{d}_0 = 0$, the sum of monthly discounts up to month k (accumulated discount), $\hat{d}_k$, is:

$$\hat{d}_k = \hat{d}_{k-1} + d_k. \tag{2}$$

3. Balance sheet

In financial accounting, a balance sheet is a summary of the financial balances of a corporation or any business organization. A balance sheet can be understood as a "snapshot of a company's financial condition", taking into account the values of the assets and the liabilities at a particular closing date t. There is a large amount of standard calculus that a company computes to understand its financial position at a certain date (see Table 2). In this section we explain the overall structure of the balance sheet and the usual actuarial calculus corresponding to the balance sheet of our asset-liability portfolio (see [24], for example).

3.1. Computations related to the liabilities

3.1.1. Liability cash flows

These are the cash flows related to the policyholder benefits. At each month, the company must make some periodic payments to their policyholders. Some of them are known on beforehand, while others can depend on the market situation. Let us assume, at the beginning of the k period, the following notations:

- $n^d_{i,k}$ is the nominal corresponding to the number of deaths for the model point m_i.
- $n^s_{i,k}$ is the nominal corresponding to the number of surrenders for the model point m_i.
- $n^a_{i,k}$ is the nominal corresponding to the number of alive policies for the model point m_i.

Maturity payments: These are the payments the company must make for all those policies that reach maturity at time k:

$$M_k = \sum_{m_i \in \mathcal{P}} m^i_k, \tag{3}$$

where $m^i_k = r_k n^a_{i,k}$

Death payments: These are the payments that the company must make for all those policies whose policyholders died during the period k, that is

$$D_k = \sum_{m_i \in \mathcal{P}} n^d_{i,k} r_k p_d, \tag{4}$$

where p_d is the death benefit (usually equal to 100%).

In our case, we will know the nominal associated to dead policyholders, because they will follow a biometric model, given by a life table (see Section 4.2.3).

Surrender payments: The **surrender value** of an insurance product is the cash amount offered to the policyholder by the insurance company upon cancellation of the contract. The surrender payments are the payments that the company must make to all those policyholders who surrender before maturity at period k, which is given by:

$$S_k = \sum_{m_i \in \mathcal{P}} s_k^i, \tag{5}$$

where $s_k^i = r_k n_{i,k}^s p_s$ denotes the payment if the policyholder surrenders at model point m_i, with p_s being the guaranteed payment if the policyholder surrenders at the end of period k.

The number of surrenders will be given by a model through a surrender rate depending on the market interest rates (that is, one of the stochastic components of our ALM model). The surrender rate will be calculated at each time depending on the scenario evolution (see Section 4.2, for details).

Policy benefits: The total policy benefits, PB_k, are given by $PB_k = M_k + D_k + S_k$.

3.1.2. Supposed/estimated/approximated liability cash flows

These are the supposed liabilities cash flows, without taking into account the possible surrenders. They are an estimation of the real liability cashflows.

The supposed liability cash flow for each model point m_i at time t_k, $l_i^F(t_k)$, is the quantity that the company must pay to the policyholders of m_i due to death or maturity. It is given by:

$$l_i^F(t_k) = \begin{cases} n_{i,k}^d p_d & \text{if } t_k < T_i, \\ n_{i,k}^a & \text{if } t_k = T_i, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

We say supposed, because they are fixed (as the number of deaths can be taken from the company life tables and the maturities are known), and we are not computing the stochastic part of the payments that are given by the surrenders. These fluxes do not depend on the simulation (scenario). They just depend on t_k and the company's life tables, so the vector is constant for all the scenarios.

Therefore, the supposed liability cash flows of the company, L_k^F, are given by:

$$L_k^F = \sum_{m_i \in \mathcal{P}} d_k l_i^F(t_k). \tag{7}$$

3.1.3. Reserves

The actuarial reserves allow to compute the money we need at a given time t_k in order to be able to discharge our future obligations. The actuarial reserve at month k for the model point m_i, v_k^i, is given by the addition of the discounted supposed future cash flows for $j > k$, that is:

$$v_k^i = \sum_{j=k+1}^{M} d_j l_i^F(t_j), \tag{8}$$

where $l_i^F(t_j)$, $j = k+1, \ldots, M$, is given by:

$$l_i^F(t_j) = \begin{cases} n_{i,j}^d p_d & \text{if } t_j < T_i \\ n_{i,j}^a & \text{if } t_j = T_i \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, $n_{i,j}^a = n_{i,j-1}^a p_i$ and $n_{i,j}^d = n_{i,j-1}^d (1 - p_i)$ are recursively defined in terms of the survival rate p_i introduced in the biometric model in the forthcoming Section 4.2.2.

Thus, the **total reserves** at time t_k are the sum of all the reserves for all the model points:

$$V_k = \sum_{m_i \in \mathcal{P}} v_k^i,$$

with $V_0 = 0$. Moreover, V_k is the amount that the insurer needs at time k in order to ensure that the obligations associated to the contract can be met. Note that in (8) we use the technical discounts given by (1).

The **variation of reserves** at month k is given by $\Delta V_k = V_k - V_{k-1}$.

3.1.4. Duration of the liability

Duration is the most commonly used risk measure of an investment. When we buy a bond, one of the most important elements (in addition to credit risk, commissions, etc.) is what experts call the *bonds duration*. The duration of a bond is a measure of the weighted average maturity of all flows to be paid by that bond. We use the Macaulay duration formula, which in the case of the liabilities is given by:

$$L_k^D = \frac{\frac{1}{12}\sum_{i=1}^{\max(n,M-k)} i \cdot d_i \cdot L_{k-1+i}^F}{\sum_{i=1}^{\max(n,M-k)} d_i \cdot L_{k-1+i}^F}, \tag{9}$$

where L_k^F is given in (7) and the discount factors are given by (1).

3.2. Computations related to the asset

We have to compute the value of the asset portfolio, which includes a broad category of diversified investment portfolios containing bonds, equities and cash. The value of the asset portfolio, A_k^v, is given by:

$$A_k^v = B_k^v + E_k^v + C_k^v.$$

3.2.1. Value of bonds portfolio

The value of the bonds portfolio is given by:

$$B_k^v = \bar{N}_k + \bar{\bar{c}}_k, \tag{10}$$

where $\bar{N}_k$ is the value of the nominal/principal

$$\bar{N}_k = \sum_{i=1}^{n} d_i \cdot N_i, \tag{11}$$

and $\bar{\bar{c}}_k$ is the value of the coupon

$$\bar{\bar{c}}_k = \frac{1}{12} \sum_{i=1}^{n} \hat{d}_i \cdot c_i \cdot N_i. \tag{12}$$

3.2.2. Value of equity

The value of the equity can be recursively obtained by:

$$E_k^v = E_{k-1}^v \cdot (1 + E_k^p), \tag{13}$$

where the relative equity profitability:

$$E_k^p = \frac{\hat{S}_k - \hat{S}_{k-1}}{\hat{S}_{k-1}}, \tag{14}$$

can be computed from the simulation of the equity price, $\hat{S}_k$, based on the forthcoming equity model (20), which assumes no dividends associated to the equity investment.

3.2.3. Financial incomes (revenue, profits or turnover) and value of the cash

In business, the revenue or turnover is the income that a company receives from its business activities, usually from the sale of goods and services to customers. The financial revenues (or incomes) at month k, I_k, are the incomes that come from the bonds investment, that is

$$I_k = C_k + A_k, \tag{15}$$

where:

- C_k denotes the revenues associated to coupon payments at month k

$$C_k = \sum_{i=1}^{n} N_i c_i.$$

- A_k denotes the *amortization*, that is the nominal whose maturity is in the current month k, that is, $A_k = N_0$.

$$C_k^v = C_{k-1}^v \cdot \left(1 + \frac{r_{k-1}^1}{12}\right) + E_k^p + I_k, \tag{16}$$

where r_{k-1}^1 is the one month rate at time t_{k-1} and E_k^p, C_k and A_k are the benefits, the coupon revenues of the period and the amortization, respectively.

3.2.4. Duration of the assets

For the assets duration, A_k^D, we also use the Macaulay duration formula, given by:

$$A_k^D = \frac{\frac{1}{12}\left[\sum_{i=0}^{n} i \cdot N_i \cdot d_i + \sum_{i=0}^{n} i \cdot c_i \cdot \hat{d}_i\right]}{B_k^v}, \tag{17}$$

if $B_k^v \neq 0$, and zero otherwise.

3.3. Profit and loss (P&L)

The profit and loss account (P&L) is a primary financial statement that summarizes the revenues, the costs and the expenses incurred during a specified period of time. The P&L statement is also known as a "statement of profit and loss", an "income statement" or an "income and expense statement". The profit or loss for the period is added to the accumulated profit or loss balance in the balance sheet. The P&L usually contains two columns, one for the current accounting period and one for the prior accounting period. The analysis of the P&L results very relevant for accountants to asses how the company is performing.

The P&L at month k, PL_k, is given by:

$$PL_k = \Delta V_k + I_k - (M_k + D_k + S_k),$$

so that the accumulated value of the P&L at month k, A_k^{PL}, is given by:

$$A_k^{PL} = A_{k-1}^{PL} + PL_k,$$

starting from $A_0^{PL} = 0$.

4. ALM model

As indicated in the introduction, our ALM model can be understood as a fixed mix model, based on two models (see Table 3): one for the assets portfolio and another one for the liabilities portfolio. Assets and liabilities evolve together in time, interacting each other.

Table 3
ALM model.

ALM model	
Asset models	Liability models
Bond interest rate model	Policyholder account model
Stock interest rate model	Surrender model
Asset allocation model	Biometric model

- Asset model: it mainly consists of the models for the evolution of the interest rates and the assets.
- Liability model: this is the model for the evolution of the liabilities. It includes a model for the policy holder account, for the surrender and the mortality biometric model.
- Asset allocation model: this model couples the assets with the liabilities, thus fixing their interaction while they evolve in time in the balance sheet projection. It parameterizes the company strategy for restructuring the assets, buying or selling them depending on the market conditions, in order to be able to face their obligations and maintain a fixed asset mix to obtain profit/benefits.

4.1. Asset models

4.1.1. Bonds interest rates models and equity model

We pose stochastic models for the interest rates associated to bonds and the profitability of stocks.

Interest rate model associated to bonds: For the interest rate associated to bonds several models can be chosen. For example, in [25] the LIBOR market model has been chosen. In our case, for the bond dynamics we use the one factor Black and Karasinski (BK) model (see [26]). This model represents a generalization of the Hull–White (HW) one, which assumes that the instantaneous short rate process evolves as the exponential of an Ornstein–Uhlenbeck process with time dependent coefficients. Therefore, it avoids the presence of negative interest rates. However, this is not the current case in some economic regions (EU, for example), so that alternative models (Vasicek, for example) could be used if negative interest rates are allowed.

In the BK model we assume that, under the risk-neutral measure $\mathbb{Q}$, the logarithm of the instantaneous spot rate, $\ln r(t)$, evolves according to

$$d\ln r(t) = [\theta(t) - \alpha(t)\ln(r(t))]dt + \sigma(t)dW(t), \tag{18}$$

where $r(0) = r_0$. The deterministic terms $\theta(t)$, $\alpha(t)$ and $\sigma(t)$ represent the mean reversion level, the mean reversion speed and the instantaneous volatility of $\log r(t)$, respectively. The term $dW(t)$ denotes the increment of a standard Brownian motion under the risk-neutral measure $\mathbb{Q}$.

If we introduce the long term rate $\mu(t)$, we can consider:

$$\theta(t) = \alpha(t)\ln\mu(t) \text{ and } \alpha(t) = \frac{\theta(t)}{\ln\mu(t)},$$

so that (18) can be written as follows:

$$d\ln r(t) = \alpha(t)[\ln\mu(t) - \ln r(t)]dt + \sigma(t)dW(t). \tag{19}$$

In our case, we consider constant parameters α, σ and μ.

In order to simulate the BK model we use the Euler–Mayurama discretization scheme for (19), see [27]:

$$\begin{aligned}\ln r(t_{k+1}) &= \ln r(t_k) + \alpha\cdot[\mu - \ln r(t_k)]\Delta_k \\ &\quad + \sigma\sqrt{\Delta_k}(\rho\cdot Y(t_k) + \sqrt{1-\rho^2}\cdot Z(t_k)).\end{aligned}$$

Next, by taking exponentials, we get:

$$\begin{aligned}r(t_{k+1}) &= \exp\{\ln r(t_k) + \alpha[\mu - \ln r(t_k)]\Delta_k \\ &\quad + \sigma\sqrt{\Delta_k}(\rho\cdot Y(t_k) + \sqrt{1-\rho^2}\cdot Z(t_k))\},\end{aligned}$$

where ρ denotes the instantaneous correlation between the interest rate and the economy, while Y and Z are independent $\mathcal{N}(0, 1)$.

Equity model: In order to model the evolution of the equity/stock price we assume that the stock price process, $S(t)$, follows a Geometric Brownian Motion. We use "hats" for the variables and the parameters of the equity model. Therefore, the process $(\hat{S}(t))$, $t \in \mathbb{R}^+$ satisfies:

$$d\hat{S}(t) = \hat{\mu}\hat{S}(t)dt + \hat{\sigma}\hat{S}(t)d\hat{W}(t),$$

where the volatility $\hat{\sigma} > 0$ and the drift $\hat{\mu} \in \mathbb{R}$ are constant parameters, while $d\hat{W}(t)$ denotes the increment of a standard Brownian motion.

The log-Euler discretization of the stock model reads as follows:

$$\hat{S}(t_k) = \hat{S}(t_{k-1}) \cdot \exp\{\hat{\mu} \cdot \Delta_k + \sqrt{\Delta_i} \cdot \hat{\sigma} \cdot (\hat{\rho} \cdot Y + \sqrt{1 - \hat{\rho}^2} \cdot \hat{Z})\}, \quad (20)$$

where $\hat{\rho}$ is the correlation between the asset and the economy, while Y and $\hat{Z}$ are independent $\mathcal{N}(0, 1)$.

4.1.2. Shareholders benefits

We also have a model for the company's shareholders benefits. At the beginning of each period we take a fixed percentage of the benefits from the cash account that is distributed among the company shareholders.

4.2. Liability model

In order to compute the projected cash outflows of the liabilities, we note that some of these values are known on beforehand, as they do not depend on stochastic economic variables (for example, the final payments due to the maturity of the policies). Others are stochastic and depend on the evolution of the market or on the stochastic behavior of policyholders biometry.

4.2.1. Policyholder saving account model

As we are considering with-profit life insurance policies, we have to model how the policyholder participates in the benefits of the company and incorporates this participation in the policyholder account. The annualized profitability, A_k^p, of the company asset portfolio at month k can be computed as follows:

$$A_k^p = 12 \cdot \left(\frac{\hat{A}_k^v}{A_{k-1}^v} - 1 \right). \quad (21)$$

Note that $\hat{A}_k^v$ is the asset value before the dynamic asset allocation and we assume positive benefits.

Following [13], the policyholder account earns an interest rate which is the maximum of the technical interest rate, $\hat{r}$, and a fraction α of the annual return on the insurer's investment portfolio. More precisely, the policyholder account, P_k, is updated at time t_k according to the formula:

$$P_k = P_{k-1}(1 + \max(\hat{r}, \alpha A_k^p))^{1/12}. \quad (22)$$

4.2.2. Surrender model

We have a model for the surrender of the policy before maturity. More precisely, we consider the possibility that some policyholders choose to surrender the policy before maturity. For example, this happens if the rates in the market are higher enough than the earnings offered by the policy at some moment. So, the surrender model is parameterized considering the difference between the earnings given by the insurance company and the return of risk free bonds at the market (for surrender modeling see, for example, [17,28]).

Thus, in order to compute the percentage of the policy owners that surrender their policy, we compare the current interest rates with the earnings offered by the insurance company. Thus, we introduce Δr_k as:

$$\Delta r_k = \max(A_k^p - \bar{r}_k, 0), \quad (23)$$

where $\bar{r}_k$ is an averaged interest rate obtained from the BK model (see Section 5.1) and A_k^p is given by (21).

In Table 4, I_i^t represents the threshold interval i, and q_{ij}^s denotes the non-surrender rate of the interval i for the maturity interval j. In our example, $I_1^t = [0, 3\%]$, $I_2^t = [3\%, 9\%]$ and $I_3^t = [9\%, +\infty]$.

Therefore, if the difference between the earnings of the policy and the average interest rate falls in the threshold interval I_i^t, then q_{ij}^s denotes the percentage of non-surrender policies associated to the maturity interval I_j^m.

Table 4
Monthly surrenders in terms of non-surrender rates associated to maturities and differences between the benefits of the insurance portfolio and the interest rates in the market.

		Maturities threshold		
		I_1^m	I_2^m	I_3^m
Prof. threshold	I_1^t	q_{11}^s	q_{12}^s	q_{13}^s
	I_2^t	q_{21}^s	q_{22}^s	q_{23}^s
	I_3^t	q_{31}^s	q_{32}^s	q_{33}^s

Table 5
Life table generation.

Month/age	a_1	a_2	a_3	...
k	l_k^1	l_k^2	l_k^3	...
$k+1$	$p_1 l_k^1$	$p_2 l_k^2$	$p_3 l_k^3$	...

4.2.3. Biometric model (life tables)

The biometric model is required to compute the benefit that the company must pay in case of death of the policyholder before maturity.

Given the biometric model, we obtain the life tables (see Table 5) by introducing l_k^i as the percentage of the initial nominal of the model point i that stays alive until time k, which is computed by the recursive formula

$$l_{k+1}^i = p_i \cdot l_k^i,$$

where p_i is a constant survival rate which depends on the bucket a_i and $l_0^i = 100\%$.

Conversely, the death table can be computed from the life table as follows

$$d_k^i = l_{k-1}^i - l_k^i.$$

In our case, the survival rates, p_i, are constant for each age (for more complex life models see [25]).

4.3. Dynamic asset allocation model

As previously indicated, the assets portfolio is not constant in time, its composition can evolve according to the investment strategy of the company and how much conservative the composition of the investment portfolio is. Thus, the asset portfolio is restructured by reinvesting the incomes according to a pre-specified investment strategy that decides, depending on the economic situation of the company, whether to buy or sell assets, and which kind of assets (bonds, equities, etc.) must be sold or bought.

This investment strategy can be parameterized and provides an input for the ALM model. So, at each time step (usually a month), we must rebalance the asset portfolio to ensure that there is enough money to meet the obligations with the policyholders and the company's shareholders, and to maintain the target asset position. This

target asset mix is given by a certain proportion of stocks and bonds in the asset portfolio. More precisely, let p_c, p_e and p_f be the respective target percentages of cash, equity and bonds in the asset portfolio. The main part of the investment must be done in bonds, so that p_c and p_e must be small while p_f is close to one. For instance, the values of p_c, p_e and p_f could be 10%, 5% and 85%, respectively.

The asset mix depends on the financial objectives and on the market conditions. The asset mix also determines the portfolio returns because equities and bonds perform differently over time. So, our asset allocation model can be understood as an automated investment strategy to be followed by the company at each time step. In order to rebalance our asset portfolio we make the following steps:

4.3.1. Evaluation of the bonds position with respect to the target asset portfolio

First, we compute the difference between our target bonds position, $\hat{B}_k$, and the position at time k:

$$\overset{\Delta}{B}_k = \hat{B}_k - B_k,$$

where $\hat{B}_k = A_k^v \cdot p_f$, with A_k^v being the value of the asset portfolio (see Section 3.2).

4.3.2. Computation of the gap between the durations of assets and liabilities

In order to build the asset investment strategy, we need to compare the duration of the assets (17) and the duration of the liabilities (9). The gap between both durations is:

$$G_k = A_k^D - L_k^D.$$

4.3.3. Decision of the reinvestment strategy

Taking into account the values of B_k^Δ and G_k we build the asset allocation strategy which is detailed in Section 5.3. The main points are:

1. If $B_k^\Delta < 0$, we have more liability than asset and we need to sell bonds to meet our obligations, until we balance both the asset and the liability portfolios. We have to decide the duration of the bonds we have to sell.
2. If $B_k^\Delta > 0$, we have surplus, so that we can buy bonds. Again, depending on the gap between durations we decide the maturity of the bonds to buy (with shorter or longer maturities).

In Section 5.3 we specify a practical example of this model.

5. Numerical method: MC balance sheet projection

So far, we have shown how to compute the balance sheet of an insurance company, at only one given time step. Thus, only the usual actuarial calculus to compute the balance for the books of the enterprise has been described. As everything was assumed to be known at that time, no simulations were needed.

Next, we are going to start simulating (predicting the future) the balance sheet evolution for the economic scenarios in time $[T_0, T]$; this is the so called *projection of the balance sheet*. These economic scenarios are given by the evolution of several economic variables, like bonds interest rates or assets values, for example, along a certain time interval $[T_0, T]$. Thus, we can create the feasible economic scenarios by taking into account all the previously described biometric, interest rates and asset models.

The increase in the computational power, mainly due to new hardware architectures like GPUs, makes feasible the computation of more accurate approximations of the portfolio evolution by means of models of increasing complexity in a huge number of scenarios. In our case, the numerical method to compute the projected balance sheet of the company is based on a Monte Carlo framework (see [29]). We generate (hundreds of) thousands of economic scenarios to simulate the evolution of stochastic interest rates and stock prices. These scenarios are defined in the time interval $[T_0, T]$, which is divided into N monthly spaced subintervals $\{[t_k, t_{k+1}], k = 0, \ldots, M-1\}$, with $t_0 = T_0$ and $t_M = T$.

Table 6
Coupons, discount factors and sum of discount factors at time t_k.

Nominal	$N_{k,0}$	$N_{k,1}$	$N_{k,2}$	...	$N_{k,n-1}$	$N_{k,n}$
Coupon	$c_{k,0}$	$c_{k,1}$	$c_{k,2}$	...	$c_{k,n-1}$	$c_{k,n}$
Discount curve	$d_{k,0}$	$d_{k,1}$	$d_{k,2}$	...	$d_{k,n-1}$	$d_{k,n}$
Sum of discounts	$\hat{d}_{k,0}$	$\hat{d}_{k,1}$	$\hat{d}_{k,2}$	...	$\hat{d}_{k,n-1}$	$\hat{d}_{k,n}$

In this simulation setting, note that Table 6 varies at each time step. Indeed, bonds evolve in time according to the scenario evolution: not only the nominal changes, because when doing asset allocation we can buy or sell bonds, but also the coupons curve and the discount factors. More precisely, we obtain a different discount factor curve for each time step, that is given by the interest rate model.

In order to compute the projection of the balance sheet we must make all the previous computations at times k and, depending on the situation of the company, we must restructure the asset portfolio. Let us assume that we already know all the positions of the portfolios at the beginning of month t_k of a given scenario. In order to advance from time t_k to time t_{k+1} we must perform the following steps (Fig. 1):

- Firstly, all the zero coupon (and discount) curves, and asset values are computed following the corresponding stochastic models.
- Secondly, in order to know the situation of the company to obtain the balance, we must compute the balance sheet. This includes computing all the variables studied in Section 3.
- Once we understand the current situation of our portfolio, we can decide whether to buy or sell bonds if needed. So, the third step is to apply our asset allocation model.
- Finally, we advance to the next time step t_{k+1}.

In the following subsections we detail the process. For this purpose, let us assume that we are in a fixed scenario, and we know the position (the value of all the portfolios) of the company at time t_{k-1} and we want to advance to time t_k.

5.1. Scenario generation

In order to obtain each scenario, first we apply the models to compute the interests rates at time t_k. At each step k, we simulate three interest rates by using BK model for the maturities 5, 10 and 20 years. For this purpose, for each maturity a set of particular parameters (α, σ, μ and r_0) in the BK model is chosen. The average of these rates provides the averaged rate $\bar{r}_k$ used in (23). Moreover, the interest rates for the other maturities are obtained by using a piecewise linear interpolation. In this way, we obtain the zero coupon curve at time t_k which clearly depends on k. By using these interest rates, we compute the discount curve at time t_k. The value of the discount at each month and the sum of discounts at month k are given by (1) and (2), respectively. These values are displayed in Table 6.

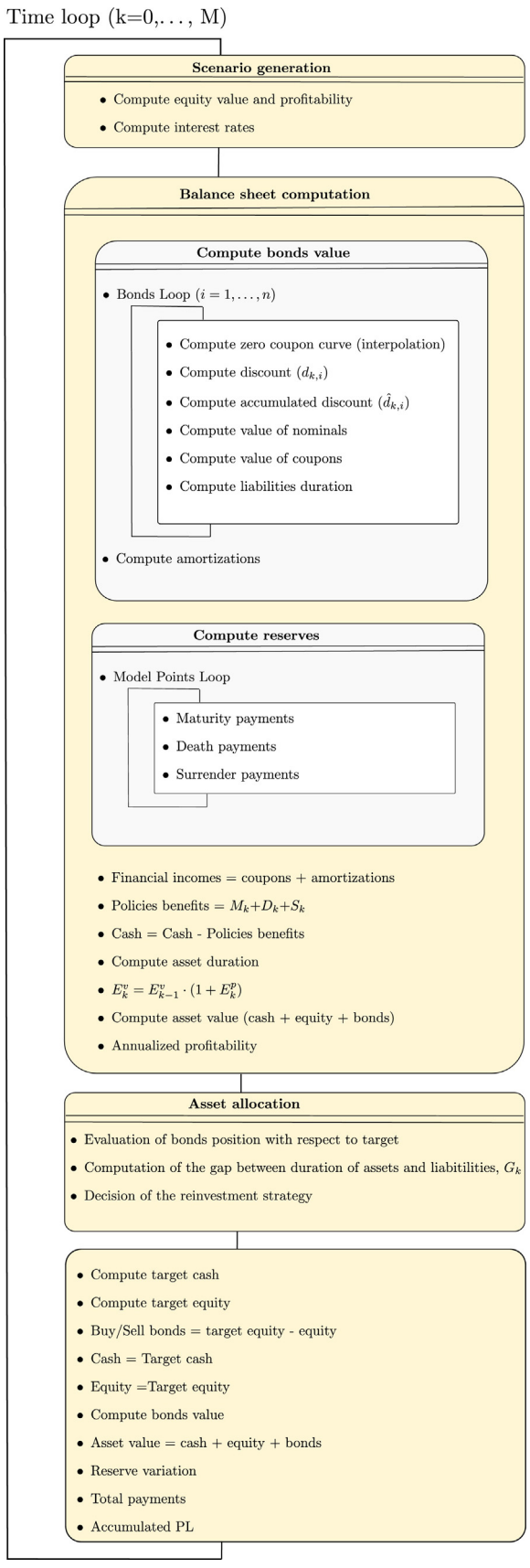


Fig. 1. ALM flowchart per scenario.

5.2. Balance sheet computation (before rebalance/asset allocation)

Essentially, we need to obtain a picture of the company situation, by computing the value of the assets and liabilities and their durations. These are the parameters to be used to take decisions in order to rebalance our assets. We will supply these values to the asset allocation model.

1. We have to compute the value of the asset, that is:
   - The discounted value of the bonds portfolio, that is given by addition of the market value of the principals and the market value of coupons (10).
   - The equity value, given by (13) and by the equity's profitability (14).
   - The cash account value, given by (16).
   - The financial incomes, that is, the addition of coupons and amortization (see (15)).
2. We have to compute the value of the liability, that is:
   - Maturity payments (see (3)).
   - Surrender payments: to compute these payments we apply our surrender model (see (5)).
   - Death payments: to compute these payments we apply our biometric model (see (4)).
3. Finally, we compute the durations of both the asset and the liabilities.

5.3. Asset allocation (asset rebalance)

So, once that we have the whole "picture" of the economic situation of the company provided by the balance sheet (and which is mainly given by the values and durations of assets and liabilities), we can take decisions to rebalance our asset portfolio in order to satisfy our target investment strategy. So, we detail the main ideas in Section 4.3.

Although the strategy can be easily modified, in the numerical examples we choose the following:

1. Note that if $B_k^\Delta < 0$, we have to sell bonds. Moreover, if $|\Delta B_k| < |B_k^\Delta|$ then the company has entered bankruptcy (default), because it needs to sell more asset than it owns in order to meet the insurance company obligations.

In other case, we sell bonds depending on the gap of durations between the asset and the liability. In order to decide if we sell bonds of short or long maturities, we have to compute the difference

$$\Delta B_k^v = B_k^v - B_{k,0}^v,$$

where B_k^v is given by (10) and $B_{k,0}^v$ is the value of the bonds whose maturity is today, i.e.:

$$B_{k,0}^v = N_0 \cdot d_0 + \frac{1}{12} c_0 \cdot \hat{d}_0 \cdot N_0. \tag{24}$$

Note that here the subindex 0 refers to the current time t_k.

- If $G_k < 0$, the duration of the liabilities is greater than the duration of the asset, so we must short sell bonds. To achieve that, we travel the bonds portfolio, starting in the beginning and we search for the first nonzero nominal (it cannot be the nominal of time zero, because it corresponds to the bonds that will be amortized today). We travel the bonds portfolio onwards and we sell all the bonds until we balance both the asset and the liability portfolios. The nominal that we must maintain in the corresponding bond is given by:

$$N_k = \frac{B^v_{k,0} + \Delta B^v_k \cdot N_k}{B^v_{k,0}},$$

where:

$$\Delta B^v_k = \overset{\Delta}{B}_k + \sum_{j=1}^{p} B^v_{k,j},$$

the index p being the first one such that $B^{\Delta}_k + \sum_{j=1}^{p} B^v_{k,j} > |B^{\Delta}_k|$.

- If $G_k > 0$, the duration of the liabilities is shorter than needed, so that we must sell bonds in the long term. In order to accomplish this, we travel the bonds portfolio in the opposite direction (backwards), that is, we travel the portfolio backwards and we sell bonds until we balance both the asset and liability portfolios. In this case, the nominal that we must maintain in the corresponding bond is given by:

$$N_k = \frac{B^v_{k,0} + \Delta B^v_k \cdot N_k}{B^v_{k,0}},$$

where:

$$\Delta B^v_k = \overset{\Delta}{B}_k + \sum_{j=1}^{p} B^v_{k,j},$$

the index p being the first one such that:

$$\Delta B^v_k + \sum_{j=1}^{p} B^v_{k,j} > |\overset{\Delta}{B}_k|.$$

2. If $B^{\Delta}_k > 0$, we have to buy bonds. More precisely:

- If $G_k < -5$ then the duration of the liability is longer, and thus we need more duration in the asset, so we must buy bonds with 10 years maturity. The value of the nominal of the bond that we must buy is B^{Δ}_k.
- If $G_k > -2$ then the duration of the liabilities is shorter than the one of the asset, so we need less duration on the asset and therefore we need to buy bonds with maturity 1 year. In this case, the nominal of the bond to be bought is B^{Δ}_k.
- If $-5 < G_k < -2$ then we buy bonds with maturity 5 years. The nominal of the bond to be bought is B^{Δ}_k.

5.4. Outputs of the simulation

A series of results need to be stored in vectors due to its particular importance for actuaries to understand what has happened to the company along each scenario. Thus, at each time step, we allow to store a bunch of vector results, that can be grouped into: Market Value results, Liabilities, Cash flows and the P&L of the company (see Table 7). These results provide a whole vision of the company evolution on each scenario. We also compute statistical measures for each of those variables, such as the best and worst scenarios for each variable, and certain quantile (that can be selected by the user of the software library).

Table 7
ALM actuarial outputs.

Market value	Before rebalance	Portfolio value
		Bonds value
		Cash value
		Equity value
	After rebalance	Portfolio value
		Bonds value
		Cash value
		Equity value
Liability value		
Cash flows	Bonds	Buy/sell bonds
		Coupons
		Amortizations
	Equity	Buy/sell equities
	Liabilities	Maturity payments
		Death payments
		Surrender payments
Profit and loss (P&L)		Reserves variation
		Variation of market value of bonds
		Variation of market value of equity
		Financial incomes
		Benefits

6. Implementation details

Computing the balance sheet of the company at all times for each scenario can be a quite demanding computational task, that can take even a day or more, mainly depending on the number of policies, the number of scenarios, the temporal duration of the forecast and the number of time steps per each scenario Therefore, the model has been parallelized in multi CPU clusters using OpenMP and also in heterogeneous systems, more precisely using GPUs. Thus, the code has been parallelized both in Multi-CPU and GPUs.

6.1. GPUs

GPUs are similar to vector machines (with SIMD paradigm: Single instruction -kernel- applied to multiple data) and offer a double advantage:

1. **Execution parallelism**. A GPU is a **Many-core** architecture with a huge number of computing cores (up to 2880 in current Nvidia Kepler architecture, size 2012, for example, in the Nvidia GTX Titan Black). GPUs follow a **SIMD** paradigm: the GPU can execute a program (kernel) in a lot of computing threads at a time.
2. **Parallel memory access**. GPUs are able to fetch many data in a single cycle from RAM memory: 336 GB/sec in a GTX Titan Black.

The here presented GPU implementation has been developed for Nvidia GPUs by using the CUDA API. As a physical layout, NVIDIA GPUs are organized as Streaming Multiprocessors (SM) with simple scalar processors (SP) on a chip. The SMs access device memory as a shared resource, although it can be considered as independent, even if grouped in Texture Processor Clusters (TCP).

Unlike the SIMD execution model (Single Instruction, Multiple Data) used for general data-parallel programming, the NVIDIA model is SIMT (Single Instruction, Multiple Threads): the code execution unit is called a kernel and is executed simultaneously on all SMs by independent blocks of threads; each thread is assigned to a single processor and executes within its own execution environment (instruction address and register state), but they run the same instruction at a time. In order to efficiently execute hundreds of threads in parallel, the SM is hardware multi threaded. The SM also implements the CUDA __syncthreads() synchronization barrier with a single instruction. Once a block has its threads assigned to a SM, it is further divided by the SIMT multi threaded instruction unit into 32-threads units called warps. Each SM manages a pool of up to 32 warps, although only one of these warps will be actually executed by the hardware at any time instant.

In order to support computing and C/C++ language needs, the SM implements memory load/store instructions in addition to graphics

texture fetch and pixel output. During the kernel execution, the load/store instructions access three read/write memory spaces:

- local memory: per-thread, private, for temporary data (implemented in external DRAM);
- shared memory: for low-latency access to data shared by cooperating threads in the same SM (implemented on chip);
- global memory: for data shared by all threads of a computing application (implemented in external DRAM).

In addition to these memories, each SM contains an important number of registers to be used to store instruction operands.

There is an API for programming Nvidia GPUs called CUDA (Compute Unified Device Architecture) [30], which mainly consists of: some drivers for the graphics card, a compiler and a language, that is basically a set of extensions for the C/C++ language, that allows to control the GPU (the memory transfer operations, the work assignment to the processors and the processors/threads synchronization).

CUDA provides all the means of a parallel programming model with the particularity of two types of shared memory: the on-chip shared memory that can be shared by threads of a block executing on a SM and the global memory that is independently accessed by the blocks running on the GPU.

Due to these execution model and memory hierarchy, GPUs support two levels of parallelism:

- An outer fully-parallel loop level that is supported at the grid level with no synchronization. Thread blocks in a grid are required to be executed independently. It must be possible to execute them in any order, in parallel or in series. This independence requirement allows thread blocks to be scheduled in any order across any number of cores, thus enabling programmers to write a scalable code.
- An inner synchronous loop level at the level of thread blocks, where all threads within a block can cooperate and synchronize.

The memory of the GPU (device memory) is completely separated from the host memory and all transfers between the two memories need to be instructed explicitly by the programmer and must be carefully designed because of the memory bandwidth and latency characteristics.

Inside the device, threads are able to access data from multiple memory spaces. As previously presented, each thread block has a shared memory visible to all threads of the block and with the same lifetime as the block. All threads from any grid have access to the same global memory, which is persistent across kernel launches by the same application. Each transfer between these memory spaces must be also explicitly managed. Transferring data between different types of memory inside the device is also important because of different access patterns and the specific size and latency of the memory.

6.2. C++ OO implementation on multi-CPUs and GPUs

The whole code, both the C++ sequential code, the parallel OpenMP and the GPU implementations have been built from scratch. In this way, their performances can be compared as they are exactly the same code, excluding the parallel techniques introduced in the parallel versions.

- The C++ CUDA code contains around 20,000 lines of code including suitable helper functions: for management like reading model parameters, reading input data (initial asset and liability portfolios), writing the results and post processing in terms of statistical measures. In this paper we only focus on discussing the computing aspects.
- It is a multi platform library. It can be built as a dynamic library available for Unix/Linux and Windows (Windows dll, Dynamic-link library) that can be integrated in a web application by using the.NET framework, for example.
- The library is fully object oriented, both in the CPU and in the GPU codes. All the objects for the simulation are created in the CPU, and in the GPU when the simulation is performed in the GPU. They are the same objects and share most of their attributes and methods. When computing in the GPU, the CPU and GPU objects are mirrored so that it is easy to return the computed values in the GPU to their corresponding mirrored object in the CPU.
- In the library, we try to avoid repeating code when possible. Thus, we make extensive use of host-device functions (pieces of code that are shared between CPU and GPU). Note that in most of the code snippets shown in the paper, the methods are preceded by a `__host__ __device__` (these methods) are common for both objects regardless they are instantiated in the CPU or in the GPU. Thus the code has only be written once. This is a key feature for code management, maintenance, and reusability, quality and maintenance because in this way we have not to rewrite a function for OpenMP in the CPU and for CUDA for the GPU.
- Extensive use of Thrust.
- "On the fly" random number generation. Currently using Curand (L'Ecuyer's MRG32k3a).

Note that the code snippets included in this paper are not the exact copies of the code: many details have been omitted for the clarity of the explanation and confidentiality agreements. Only the main characteristics of the code are retained.

All the code is object oriented and is written in a bunch of C++ classes, the main ones will be presented in the following sections.

Table 8
Some of the global vectors, reserved at the GPU global RAM memory at the beginning of the calculus.

Global vectors
Nominal
Coupon
Discount
Sum of discounts
Period Interest
Asset Duration
Payments
Liability Duration
Liability Cash Flow
RF Value Before
RF Value After
Equity Before
Equity After
Cash Before
Cash After

6.2.1. Parallel implementation

In the parallel implementation each economic scenario corresponds with a computing thread, see Fig. 2. At each thread we compute the whole balance sheet projection for the corresponding scenario. Thus, the code for the balance sheet projection is defined in a `kernel` function with the keyword `__global__` , see Listing 1 . Inside this kernel, several device functions are called, as we will see in the forthcoming subsections.

Most of the required pointers are stored in the global memory of the device (GPU). A summary of the used vectors is given in Table 8. In the GPU all these vectors are accessed in a coalesced way. For example, the value of the nominal at time *i* and scenario *j* is stored in the position `index(i,j,NS)` of the device vector `nominal`, with *NS*

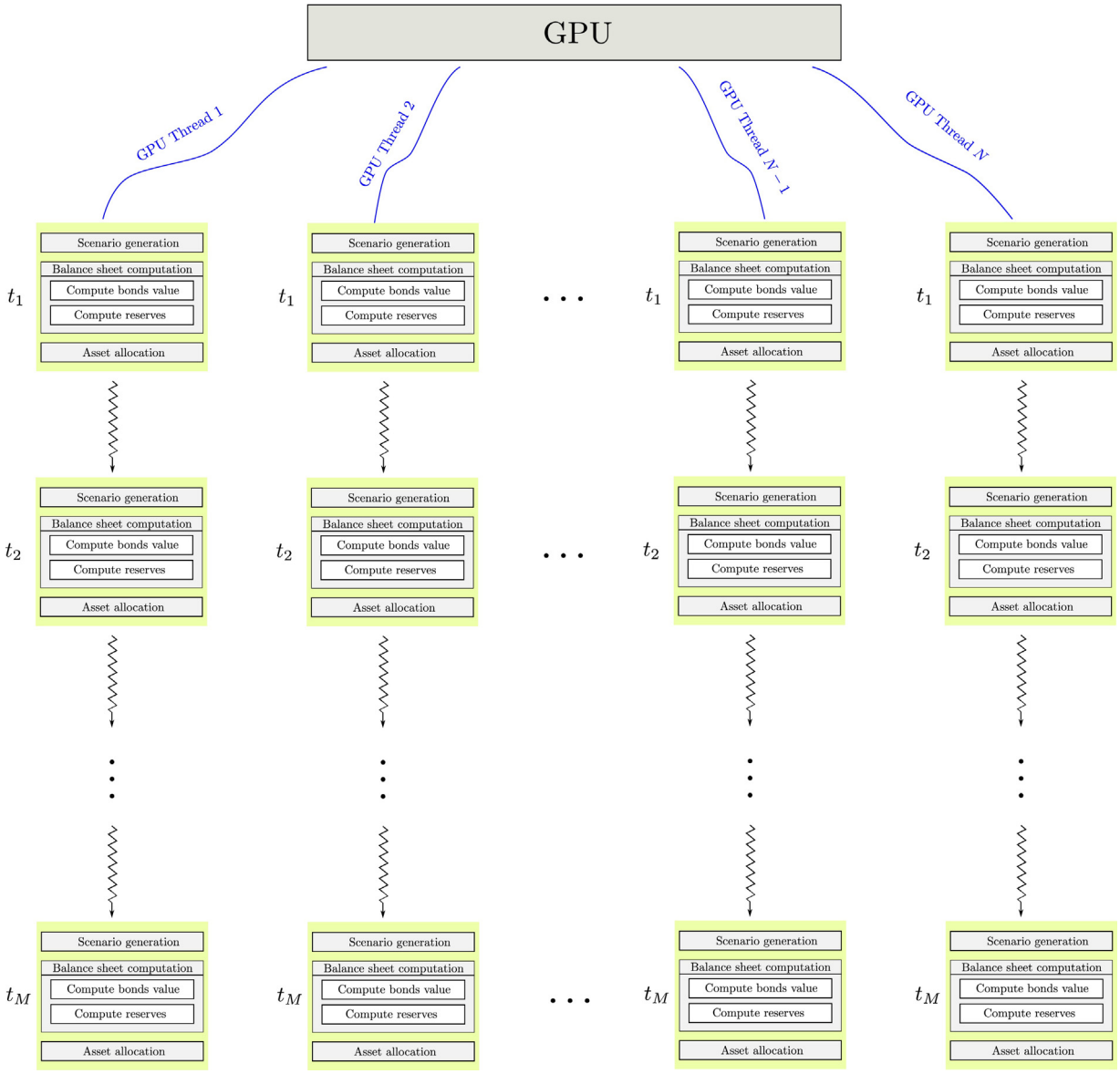


Fig. 2. GPU parallel implementation flowchart.

Table 9
Age and nominals of the model points in the liabilities portfolio. The set of maturities is $\{i, i = 1, \ldots, 29\}$.

Age	Nominal for maturity i
30	1250
31	1375
32	1500
35	1625
38	1750
39	1814
40	1875
42	1938
45	1975

Table 10
Target asset mix.

Cash	Equity	Bonds
0.1	0.2	0.7

Table 11
Parameters of the interest rates model for 1, 10 and 20 years.

	θ	σ	μ	r_0
1	0.2	0.2	0.04	0.02
10	0.25	0.2	0.1	008
20	0.25	0.2	0.12	0.09

being the number of scenarios and where the value of the position is given by $i * NS + j$, see Listing 2 .

6.2.2. Asset class

This class includes the following attributes and methods (see Listing 3).

Attributes: The asset portfolio, with the bonds and equity structures/objects. This class consists, mainly, of four tables (pointers): one for the bonds nominal, one for the coupons, one for the discounts and one for the sum of discounts.

We use a circular/cyclic structure for storing the bonds tables (including coupon, rates, discounts and sum of discounts). We keep a pointer that advances to the right. New bonds enter at the begin-

Table 12
Parameters of the equity model.

Volatility	Drift	Correlation	Initial condition
0.4	0.06	0.5	1

Table 13
Speed-up results.

	1 CPU	2 CPU	4 CPU	8 CPU	GPU
Time	1432m90s	732m68s	362m76s	108m92s	23m38s
Speedup	1	1.98	3.95	7.92	61.27

ning of the table, so that we avoid copying the structure for each new time step or having a over dimensioned table with length $M+n$ for each scenario, which would be inefficient for the parallel GPU simulation.

Methods: One method for each of the calculus presented in Section 3.2, as well as one method for each of the models related to the liabilities.

In Listing 4 we show the code for computing the interest rates array at each time by using the three simulated interest rates (with 5, 10 and 20 years period) and a piecewise linear interpolation.

In Listing 5 we show the code for computing the value of the bonds portfolio using Eq. (10), where the value of the bonds with maturity i is computed using Listing 6 (see Eq. (24)).

Most of the functions are device and host functions, so most of the code is shared between the CPU and GPU classes.

6.2.3. Liability class

We have a liability class, see Listing 7 . This class includes the liability portfolios, as well as the management of all the models and computations related to liabilities. Therefore, this class includes the following attributes and methods.

Attributes: The liabilities portfolio, with all the policies. The policies are grouped in a structure containing all the model points.

Methods: This class contains one method for each of the calculus presented in Section 3.1, as well as one method for each of models related to the asset.

In Listing 8 we show the code of the method for computing the surrender rate with the Surrender Model described in Section 4.2.2.

At the beginning of each time step, `t_k`, the zero coupon discount curve is recomputed in the ALM kernel by using the function in Listing 9 . At a later step, the reserves are computed by calling the function `Reserves_AllMP`.

The reserves for all model points are computed by the function of Listing 10 . This function performs a loop for all the model points, and for each model point the function of Listing 11 is called.

In Listing 11 we show the function that computes the reserves per model point at each time `t_k`, as well as the liability fluxes from time `t_k` to the end, at each time step `t_k`. In order to compute the market value of the reserves we need to use the zero coupon curve. Note that computing the reserves for each model point requires performing the time loop from `t_k` to the end. If the interest rate were constant, we can precompute a table with all the required values before launching the computing kernel.

In Listing 12 we show part of the time loop kernel, with the calls to the previous functions.

6.2.4. ALM class

This class models our company, and thus includes both the assets portfolio and the liabilities portfolios (Listing 13).

Attributes: This is the main class. Among others, an ALM object of the ALM class contains two objects: one from the Asset class and one from the Liability class.

Methods: Among the methods of this class, we have one method for computing the balance sheet projection, see Listing 1. This method performs the Monte Carlo simulation. In our case, we distribute all the scenarios among the GPU cores. This function corresponds to our CUDA kernel, as we can see in the `__global__`

This kernel is called when we invoke the `compute()` method of the ALM class. Inside this method we decide whether to call this kernel or a CPU OpenMP kernel, depending on whether the user has chosen to make the computation in GPU or in CPU. This option is stored in the attribute `compute_device` of the ALM class.

In Listing 14 the code for computing the bonds value is shown. In order to save computation time, we compute in the same function

Table 14
[{(Listing 1)}] Kernel for the balance sheet projection.

```
__global__ void kernel_balance_sheet_projection(const scalar*  d_rescue_rate,
                                                const          Device_PopulationLife d_SPopulation,
                                                const          Device_ModelPoints d_SModelP,
                                                const scalar*  d_Reserves_MP,
                                                const          RatesModel d_SRates,
                                                ALM*           alm)
{
  // Thread index: id of the scenario
  long int id_S = blockIdx.x * blockDim.x + threadIdx.x;
  if (id_S<ALM->NS ){
    for (int k=1;k<=alm->NT;k++){
    ....
    // Call of all ALM device functions
    ....
  }
}
```

Table 15
[{(Listing 2)}] Macro for the coalesced memory access.

```
#define index(I,J,D)((I)*(D)+J)  //D is the number of scenarios
```

Table 16
[{(Listing 3)}] Asset class with its main attributes and methods.

```
/*************************************************************/
/*                          Asset Class                      */
/*************************************************************/
class Asset {
public:
  __device__ __host__ scalar Bonds_Value ( scalar* RatesEvol, int month, int id_Scenario
      );
  __device__ __host__ void compute_asset_duration (  scalar* RatesEvol,
                                                     int month,
                                                     long id_Scenario,
                                                     scalar valorRF );
  __host__   __device__ void asset_allocation ( const int &id_Scenario,
                                                scalar *RatesEvol,
                                                int &month,
                                                scalar &BondsValue );
  __device__ __host__ scalar bvi        ( const int &id_Scenario, int k );
  __device__ __host__ scalar compute_Bv ( const int &id_Scenario );
};
```

Table 17
[{(Listing 4)}] Device function for bonds rates linear interpolation.

```
/*
 * Bonds interest rates linear interpolation
 */
__device__ __host__ double ALM::bond_rate_interpolation(int month,double* RatesEvol){
    scalar month_rate;
    int a,b,N;
    // Month to year conversion
    scalar year=month/12.;
    // Month inside interval [a,b]
    a=int(1);
    b=int(this->SRates.num_terms);
    N=int((a+b)/2);
    while ((b-a)>1){
        if ( this->SRates.Terms[N-1]>=year) {
            b=N;
        }
        if  ( this->SRates.Terms[N-1]<year) {
            a=N;
        }
        N=int((a+b)/2);
    }
    month_rate = exp(log(RatesEvol[a-1]) + (log(RatesEvol[b-1]) -
                log(RatesEvol[a-1])) * ((year - this->SRates.Terms[a-1]) /
                (this->SRates.Terms[b-1] - this->SRates.Terms[a-1])));
    return month_rate;
}
```

the duration of the liability and the interest rate for the present time period.

In Listing 15 we show the code for computing the asset duration. This is done after calling the method for computing the value of bonds and the profits.

7. Numerical results

In this section we show the numerical results and the performance of the parallel implementations by using OpenMP for Multi-CPU and CUDA for GPUs.

Table 18
[{(Listing 5)}] Bonds value host-device code. Bv is the value of all bonds at the actual time k and bvi is the value of one bond with maturity *i*.

```
__device__ __host__ double ALM::Bv(Asset* asset, const int& id_Scenario)
{
    double Bv=0.;
    for (int i=0;i<asset->bonds->maturity;i++){
        Bv+=bvi(asset,id_Scenario,i);
    }
    return Bv;
}
```

Table 19
[{(Listing 6)}] Value of a bond with maturity *i*, bvi.

```
__device__ __host__ double ALM::bvi ( Asset* asset, const int& id_S, int i )
{
    int NS=this->NumSim;  // Number of scenarios
    // Bonds tables
    double* nominal  = asset->bonds->nominal;
    double* coupon   = asset->bonds->coupon;
    // Discount tables
    double* disc     = asset->bonds->disc;
    double* sum_disc = asset->bonds->sum_disc;
    return           nominal[ index(k,id_S,NS) ] * disc   [ index(k,id_S,NS) ] +
             1./12*( nominal[ index(k,id_S,NS) ] * coupon [ index(k,id_S,NS) ] *
                     sum_disc[index(k,id_S,NS) ] );
}
```

Table 20
[{(Listing 7)}] Liability class with its main attributes and methods.

```
/*************************************************************/
/*                    Liability Class                        */
/*************************************************************/
class Liability {
public:
  __device__ __host__ void compute_surrender_rate( scalar& pctje_nonresc_till_month,
                                                   scalar  AvgRate,
                                                   scalar  annualized_profitability,
                                                   int     id_Scenario,
                                                   int     month );
  __device__ __host__ scalar compute_nonrescue_pctje ( scalar base,
                                                       scalar rates_dif );
  void compute_reserves ( const PoblacionVivir &SPoblacion,
                          const ModelPoints &SModelP,
                          const scalar *Reserve_MP,
                          const scalar *h_vt_cap,
                          const Surrender_Rate &Scancel,
                          const scalar *cash_flow);
  __device__ __host__ scalar compute_market_liability_value ( scalar* RatesEvol,
                                                              int month,
                                                              int id_Scenario );
};
```

7.1. Test example

In our test example the parameters of the ALM simulation are the following. For the asset portfolio we start with an empty bonds portfolio, no equity and 6.5×10^8 cash. Before starting the simulation, we invoke the asset allocation procedure in order to automatically fill the bonds portfolio. The liability portfolio is given by Table 9. More precisely, the first column of Table 9 contains the

Table 21
[{(Listing 8)}] surrender rate method.

```
/*************************************************************/
/*                          Surrender rate method             */
/*************************************************************/
__device__ __host__ void Liability::compute_surrender_rate(scalar& pctje_nonresc_till_month,
                                                           scalar AvgRate,
                                                           scalar annualized_profitability,
                                                           int id_S,
                                                           int month)
{
    scalar rates_dif;
    scalar NonRescueRate;
    scalar a,b;
    int    ind;
    scalar base_case;
    scalar tasa_noresc;
    scalar pend;
    scalar pctje_nonresc_till_month0;
    rates_dif=max(AvgRate-annualized_profitability,0.);
    if (month<=60){
        base_case=this->SNonresc.base_case[0];
        NonRescueRate= calcula_pctje_no_resc(base_case,rates_dif);
    }
    else if (month>120){
        base_case=this->SNoresc.base_case[2];
        NonRescueRate= compute_pctje_non_resc(base_case,rates_dif);
    }
    else{
        base_case=this->SNoresc.base_case[1];
        NonRescueRate= compute_pctje_non_resc(base_case,rates_dif);
    }
    pctje_noresc_till_month0=pctje_noresc_till_month;

    /* At the first month, the rescue rate is the Base Case*/
    if(month==1){
        this->rescue_rate[id_simulacion]=this->SNoresc.base_case[0];
        pctje_noresc_till_month=pctje_noresc_till_month0-this->SNonresc.base_case[0];
    }
    else{
        pctje_noresc_till_month=pctje_noresc_till_month*NonRescueRate;
        this->rescue_rate[id_simulacion]=pctje_nonresc_hasta_month0-pctje_noresc_till_month;
    }
}
```

ages of the policyholders in the model point and the second one the total nominal of their policies for each maturity. The nominal structure is the same for all maturities, which range from 1 to 29 years.

Furthermore, Table 9 is repeated forty times, so that the total number of model points in the simulation is 10440 (although some of them are repeated, each one is treated as a different model point in the simulation).

The asset allocation strategy is given by Table 10, the parameters in the BK model are in Table 11 and those ones for the equity model are in Table 12.

The time horizon of the simulation is 30 years (360 months) and the maximum duration of bonds that can be bought is 20 years. The number of scenarios is 100,000 with 360 time steps per scenario. In the following test we consider $\alpha = 0.3$ and the guaranteed technical interest rate $\hat{r} = 0.03$.

According to [1] we consider three sample products:

- p_0: pure savings account without death benefit and no surrender possibility.
- p_1: a endowment insurance with death benefit.
- p_2: a endowment insurance with death benefit and surrender option with no surrender fee.
- p_3: a endowment insurance with death benefit and surrender option with 10% surrender fee.

Table 22
[{(Listing 9)}] Zero coupon bond curve interpolation.

```
1
2  /*************************************************************/
3  /*      Zero coupon bond interpolation                       */
4  /*************************************************************/
5    __device__ __host__ scalar Asset::ZCBC_Generation    (Liability liability,
6                                                          ALM  alm;
7                                                          scalar* RatesEvol,
8                                                          int id_Scenario){
9  for (int jk=0;jk<this->NumTimes;jk++){
10       if(jk >= k-1) this->liablility->flux[jk*this->NumSim+id_Scenario]=0.;
11       if(jk==0){
12         asset->zero_coupon_disc_curve[jk*this->NumSim+id_Scenario]=1.;
13       }
14       else{
15         // zero copupon bond interpolation at eath time step of the simulation
16         this->asset->zero_coupon_disc_curve[jk*this->NumSim+id_Scenario]=
17                         1./pow(scalar(1.)+interpola_tipo_anual(jk+1,RatesEvol)/12.,scalar(jk+1));
18       }
19 }
```

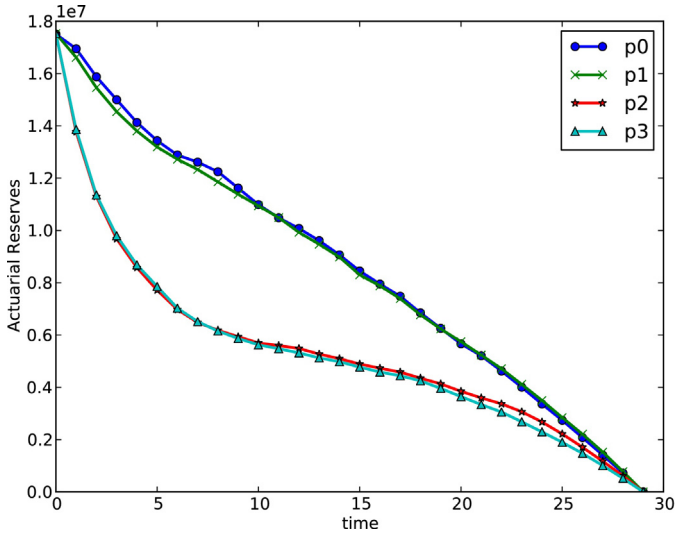


Fig. 3. Average evolution with time of the actuarial reserves for the products p_0, p_1, p_2, p_3.

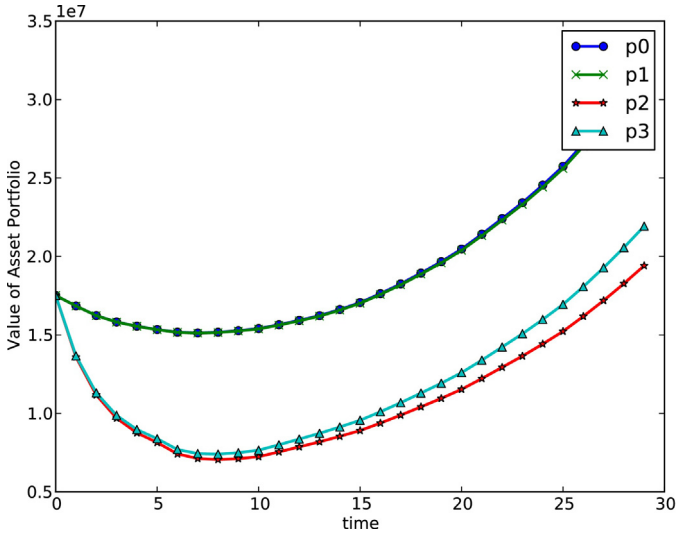


Fig. 4. Evolution with time of the average value of the asset portfolio for the products p_0, p_1, p_2, p_3.

In Fig. 3 we show the expectation of the reserves value at each month, $\mathbb{E}(V_k)$. As we can see in Fig. 3, the value of the reserves decreases when we introduce the death benefit and the surrender option.

In Fig. 4 we show the expectation of the asset value at each month, $\mathbb{E}(A_k^v)$. Note that the asset value increases when increasing the surrender fee. In the absence of death benefits and surrender option the asset value is higher.

In Fig. 5 we show the probability of default in the company for the products p_0, p_1, p_2 and p_3. As expected, the default probability increases with the presence of death benefits and surrenders. Also, if we increase the surrender fee then the default probability decreases.

In Fig. 6 we show the accumulated value of the P&L. The small jumps appearing at certain dates are due to the choice of a toy example where bond maturities are concentrated at those specific dates.

In Fig. 7 we show the results for the product p_3 when considering different values for α, $\alpha \in \{0.25, 0.3, 0.35, 0.4\}$, for the participation in benefits. As expected, the profit decreases when increasing the value of α, and the default probability increases with α.

7.2. Performance results

The hardware test platform is a dual Quad-Core Xeon E5530 server with 16 GB of RAM and two Nvidia GTX Titan Black GPUs, with 6 GB of RAM.

In Table 13 we show the execution times and speedups for the Multi-CPU and the GPU implementations. Note that the speedup is given by:

$$\text{Speedup} = \frac{\text{time using 1 CPU}}{\text{computing time}}$$

Table 23
[{(Listing 10)}] Reserves for all model points.

```
/*************************************************************/
/*     Reserves for model points inside the kernel           */
/*************************************************************/
  __device__ __host__ scalar Liability::Reserves_AllMP (Asset asset,
                                                        ALM  alm;
                                                        scalar* rates,
                                                        int age,  // Age
                                                        int mat,  // Maturity (years)
                                                        int t_k,  // Current simulation time
                                                        int id_S,
                                                        scalar capitalization_rate){
  scalar Reserve_MPi; // Reserve for model point i
  // model points loop: ages and maturities
  for (int age=0;age<numEdad;age++){
    for (int mat=0;mat<NumMat;mat++){
      col=je*NumMat+jv;
      pos=k*(NumAges*NumMat)+col;
      remaining_pol0=vt_remaining_policies[col];
      aux_surrender_payment=pol_rest0*(tasa_resc*aux_h_vt_cap*Scancelar.porresc);
      aux_death=SPoblacion.DeathTable[(k-1)*numEdad+je];
      aux_death_payment=remaining_pol0* ( aux_death * aux_h_vt_cap );
      remaining_pol11=pol_rest0*((1.-aux_death)*non_surrender_rate);
      vt_remaining_policies[col]=remaining_pol1;
      // Compute the reserves for this model point
      Reserve_MPi=this->Reserves_1MP(rates,age,mat,vt_des,vt_des,k,remaining_pol11,id_Scenario,
          aux_h_vt_cap);
      Total_Reserves1 += remaining_pol1*aux_h_vt_cap*Reserve_MPi;
      aux_mat_payments = ( k == int( SModelP.vt_venc[jv]*12) )?1.:0.;
      aux_mat_payments   *= ((aux_h_vt_cap*pol_rest1));
      maturity_payments  -= aux_mat_payments;
      surrender_payments -= aux_surrender_payments;
      death payments     -= aux_death_payments;
      }
  }
```

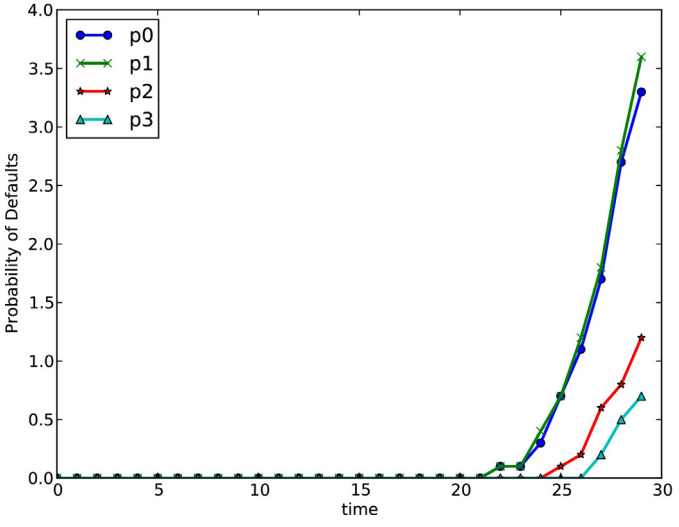


Fig. 5. Evolution with time of the default probabilities for the products p_0, p_1, p_2, p_3.

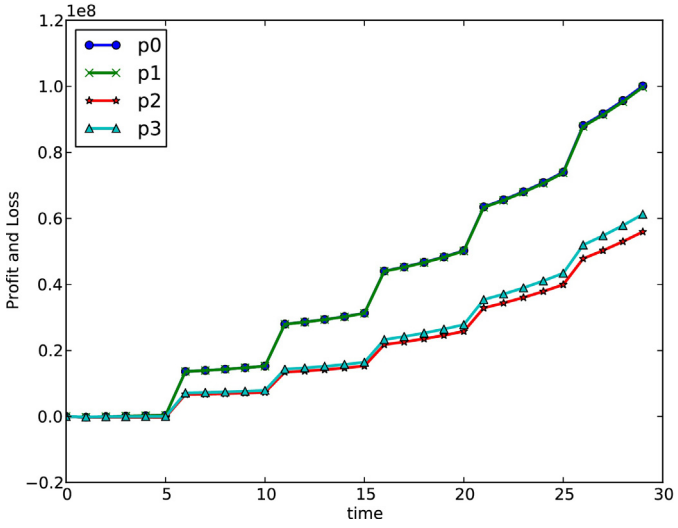


Fig. 6. Evolution with time of the average accumulated P&L for the products p_0, p_1, p_2, p_3.

Table 24
[{(Listing11)}] Reserves per model point.

```
/*************************************************************/
/*                  Reserves for 1 model point               */
/*************************************************************/
__device__ __host__ scalar Liability::Reserves_1MP (Asset asset,
                                                    ALM  alm;
                                                    scalar* rates,
                                                    int age,  // Age
                                                    int mat,  // Maturity (years)
                                                    int t_k,  // Current simulation time
                                                    int id_S,
                                                    scalar factor_capitalizacion){
    scalar dif;
    int maturity=this->MP.maturities[mat]*12;;
    scalar temp=0.;
    scalar temp_sum=0.;
    for(int j=t_k;j<=alm->NT;j++){
      temp=1./asset->ZCDC[index(j-t_k,id_S,alm->NS)];
      if (j<maturity){
        dif=this->LifeTable[index(j-1,age,this->MP.NumAges)] - this->LifeTable[index(j,age,this->MP.
            NumAges)];
        temp *= dif*this->deathbonus;
      }
      else if (j== maturity){
        temp *= this->LifeTable[index((j),age,this->MP.NumAges)];
      }
      else{
        temp*=0;
      }
      this->flux[index( j-1,id_Scenario,this->simulacion->NumSim)] += capitalization*temp*this->MP.
          MPTable[index(age,mat,this->MP.NumVenc)];
      temp*=asset->ZCDC[index(j-t_k,id_Scenario,alm->NS)];
      if(j!=t_k) temp_sum+=temp;
    }
    return temp_sum;
}
```

Table 25
[{(Listing 12)}] Time loop kernel.

```
  ...
for (int k=1;k<=this->NumTimes;k++){   // time loop in the ALM simulation
  ...
  this->asset->ZCBC_Generation (liability,this, RatesEvol, id_Scenario);
  ...
  this->liability->Reserves_AllMP (asset,alm, t_k, id_Scenario, capitalization_rate){
} // End of time loop
```

8. Conclusions

In this work we have developed an efficient GPU and multiCPU parallel implementation of a stochastic asset liability management model for a (with-profit) Life Insurance Company.

The library has been developed in an object oriented way using C++ and CUDA, for the Nvidia GPUs version, and using OpenMP for the Multi-CPU version. The most usual characteristics of a with profit life insurance policy have been considered, like interest rate guarantee with cliquet option style, mortality benefits, surrender option with surrender fee, bonds investment, etc. In this ALM model the evolution of the assets and liabilities portfolios are coupled through an asset allocation model. As particularly illustrated in Fig. 3, we notice the effects of the surrender and the surrender fee in the reserves, which are overestimated if surrender is not considered.

We have obtained a rather good speedup. Furthermore, GPUs are more power efficient than CPUs, in the sense that they require

Table 26
[{(Listing 13)}] ALM class with its asset and liabilities objects, and its main methods.

```
/*************************************************************/
/*                          ALM Class                        */
/*************************************************************/
class Asset;
class Liability;
class ALM {
public:
  /***************** ALM atributes *************************/
  int NS; // Number of scenarios
  int NT; // Number of time steps per scenario
  // Pointers to asset and liabilities objects
  Asset*    asset;
  Liabilty* liability;
  string compute_device; // Compute device to use (CPU or GPU)
  int    alm_numproc;    // #CPU processors to use in OpenMP, if compute_device=="CPU"
  /***************** ALM methods *************************/
  ALM();
  ALM(Asset &asset, Liability &liability);
  ~ALM();
  /*  Methods shared by CPU and GPU code */
  __device__ __host__ scalar bond_rate_interpolation(int month,double* RatesEvol);
  __device__ __host__ scalar device_simulate_equity(const int idSim, int id_t,
                                                    scalar rendimiento_equity,
                                                    scalar dt);
  __device__ __host__ void device_Hull_White(const int idSim,
                                             int id_t,
                                             scalar* RatesEvol);
  __device__ __host__ scalar npv_deflactor(scalar* fluxes,scalar* deflactors);
  void compute();  // Method that performs the whole simulation
  void  compute_flows(Asset &asset,Liability &asset);
  __host__  void compute_asset_rebalance(const int &id_Scenario,
                                         scalar    *RatesEvol,
                                         int       &month,
                                         scalar    &BondsValue);
};
```

much less Watts per Gigaflop, and we can also use a cluster of GPUs, what increases our computational power. All the implementation is Object Oriented and has been carried out in C++, so GPUs allow a reusable code development, which is of key importance for the scalability of the library to more complex business models. The more important ALM commercial libraries for insurance are also fully C++ object oriented. Furthermore, most of the multi-CPU and GPU code is shared, between both versions, which reduces the development time and the possibility of introducing errors in the code. The library is multi-platform and has been tested both in Windows and Unix/Linux environments.

From previous paragraphs, we can conclude that GPUs have proved to be clearly a more efficient tool than conventional CPUs for the implementation of ALM models for insurance companies. Particularly, if we scale the speed up results in Table 13 then computations that would require two months in 1 CPU can be performed in one day in a GPU.

Although the ALM model has been chosen complex enough to illustrate the advantages of the GPUs implementation, several pending economic aspects to be closer to a fully realistic model will be considered by the authors in a forthcoming research. The first one, already pointed in the article, concerns to the implementation in GPUs of the stochastic programming approach in order to optimize the asset portfolio. Note that in [22] an example of ALM parallelization with an old PVM protocol has been addressed. The second main issue is related to the treatment when the insurance company enters into default. Although in our examples, the probability of default is very small, the strategy in case of default is not properly addressed. In [13], the authors propose that if default occurs the assets are paid out to policyholders, who probably will invest them in a saving account.

Acknowledgements

This work is related to a project between Instituto Tecnológico de Matemática Industrial (ITMATI) and the consultancy company Analistas Financieros Internacionales (AFI). This work has been partially funded by Spanish MINECO (Research Project MTM2013-47800-C2-1-P). We also would like to thank Paul MacManus, Samuel Soláns, José Antonio Puertas, Iratxe Galdeano and Janire Alonso for their most valuable comments and help.

Finally, the authors would like to thank three anonymous referees whose remarks highly helped to the improvement of the article.

Table 27
[{(Listing 14)}] Value of bonds method.

```
/*************************************************************************
 * Value of bonds = Principals value + Coupons value
 *************************************************************************/
__device__ __host__ scalar ALM::FR_Value (scalar* RatesEvol, int month, int id_Scenario){
    scalar FRValue=0.;
    scalar discount, sum_discount;
    scalar valor_principal=0, value_coupon=0;
    scalar rate;
    scalar sum_cashflow_temp;
    sum_cashflow_temp=0.;
    scalar temp_dur_pasivo=0.;
    int i_mod; // index module max maturity
    int index; // index of the time we are accesing, for coalesced access
    int index_FP; // Liability cash flow index
    scalar temp_dur_asset1,temp_dur_asset2;
    asset->asset_duration[index(0,id_Scenario,this->NumSim)]=0.;
    discount =1.;  sum_discount=0.;
    asset->period_interert[index(0,id_Scenario,this->NumSim)]=
            Portfolio->Bonds_nominal[index(mes%240,id_Scenario,this->NumSim)]*
            Portfolio->Bonds_coupon[index(mes%240,id_Scenario,this->NumSim)];
    for (int i=1;i<=Portfolio->Bonds_duration;i++){
      index_FP=(month!=0)?(month+i-2):(month+i-2+1);
      i_mod=(mes+i)%(Portfolio->bonds_duration+1);
      index=(i_mod)*this->NumSim+id_Scenario;
      tipo=this->anual_rate_interpolation(i,EvolTipos);
      discount=1./pow((1.+tipo/12.),i);
      Portfolio->disc[index]=discount;
      sum_discounts+=discount;
      Portfolio->sum_discount[index]=sum_discount;
      // Nominal and copupon multiplied by discount factor
      principal_value+= discount*Portfolio->Bonds_nominal[index];
      coupon_value+=Portfolio->Bonds_nominal[index]*Portfolio->RF_cupon[index]*
           sum_discounts;
      // Duration of liablilities computation
      if( (index_FP) <portfolio->bonds_duration){
          liability->duration[index(0,id_Scenario,this->NumSim)]+=
                                (pasivo->flujo_pasivo[index_FP]*(i)*descuento/12.);
          temp_liability_dur+= liability->cash_flow[index_FP]*discount;
          sum_flow_temp+=liability->cashflow[index_FP];
      }
      asset->period_interest[index(0,id_Scenario,this->NumSim)]+=Portfolio->Bonds_nominal
           [index]*Portfolio->Bonds_coupon[index];
  }
  liability->duration[index(0,id_Scenario,this->NumSim)]= asset->duration[index(0,
       id_Scenario,this->NumSim)]/temp_dur_liability;
  coupon_value/=12.;
  asset->period_interest[index(0,id_Scenario,this->NumSim)]/=12;
  BondsValue=coupons_value+principals_value;
  asset->amortizaciones[index(0,id_Scenario,this->NumSim)]=Portfolio->FR_nominal[index(
       month,id_Scenario,this->NumSim)];
  return FRValue;
}
```

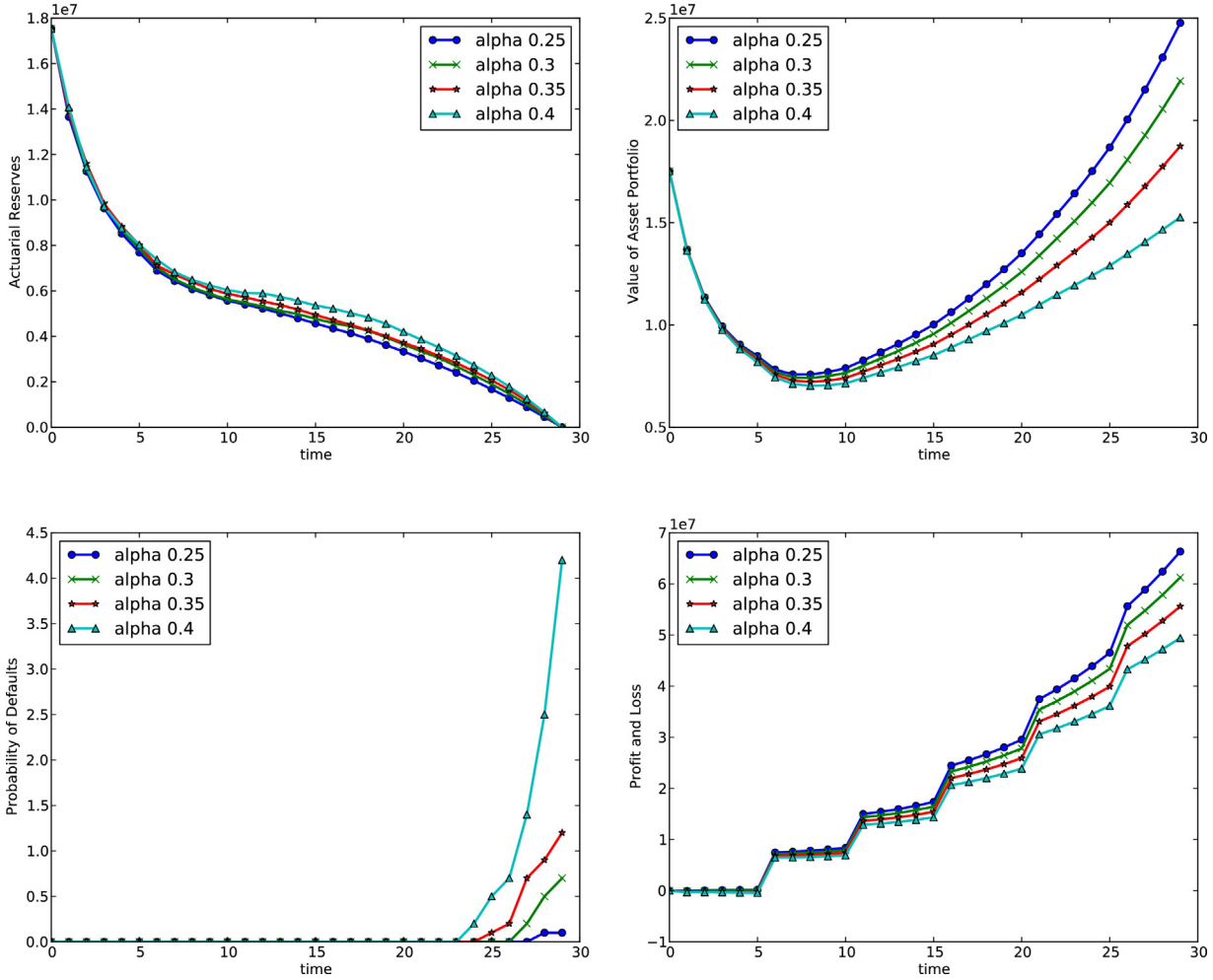


Fig. 7. Results for the product p_3 with different values of α: $\alpha = 0.25$, $\alpha = 0.3$, $\alpha = 0.35$, $\alpha = 0.4$.

Table 28
[{(Listing 15)}] Asset duration.

```
/*****************************************************************
 * Asset duration:
 * we compute asset duration after computing the bonds value.
 * It is the same kind of loop
 *****************************************************************/
__device__ __host__ void Simulation::compute_asset_duration(scalar* RatesEvol,
                                                            int month,
                                                            long id_Scenario,
                                                            scalar BondsValue)
{
  scalar discount, discount_sum;
  scalar principal_value=0., coupons_value=0.;
  scalar rate;
  scalar sum_cashflow_temp;
  scalar temp_asset_dur1,temp_asset_dur2;
  int i_mod; // index module 241
  temp_asset_dur1=0.; temp_asset_dur2=0.;
  suma_descuento=0.;
  for (int i=0;i<=portfolio->bonds_duration;i++){
    i_mod=(month+i)%(portfolio->bonds_duration+1);
    tipo=annual_rate_interpolation(i,RatesEvol);
    discount=1./pow((1.+rate/12.),i);
    if(i!=0) sum_discount+=discount;
    temp_asset_dur1 += (Portfolio->FR_nominal[index(i_mod,id_Scenario,this->NumSim)]*
        discount+\
            Portfolio->Bonds_coupon[index(i_mod,id_Scenario,this->NumSim)]*suma_descuento
                )*i/12;
    temp_asset_dur2 += Portfolio->FR_nominal[index(i_mod,id_Scenario,this->NumSim)]*
        discount+\
            Portfolio->Bonds_coupon[index(i_mod,id_Scenario,this->NumSim)]*sum_discount;
  }
  scalar temp_dur_activo3;
  if(fabs(temp_dur_activo2)>1e-6){
      temp_asset_dur3=temp_dur_activo1/temp_dur_activo2;
  }
  else{
      temp_asset_dur3=0.;
  }
  if((Portfolio->cash[index(0,id_Scenario,this->NumSim)]+BondsValue)>1e-6){
      asset->duracion_activo[index(0,id_Scenario,this->NumSim)]=
              (temp_asset_dur3*BondsValue+ Portfolio->cash[index(0,id_Scenario,this->
                  NumSim)]*0) /
              (Portfolio->cash[index(0,id_Scenario,this->NumSim)]+BondsValue);
  }
  else{
    asset->asset_duration[index(0,id_Scenario,this->NumSim)]=0.;
  }
}
```

References

[1] T. Gerstner, M. Griebel, M. Holtz, R. Goschnick, M. Haep, A general asset-liability management model for the efficient simulation of portfolios of life insurance policies, Insur. Math. Econ. 42 (2008) 704–716.

[2] L. Ballotta, S. Haberman, N. Wang, Guarantees in with-profit and unitized with-profit life insurance contracts: fair valuation problem in presence of the default option, Insur. Math. Econ. 73 (2006) 97–121.

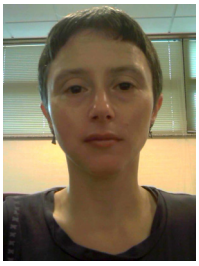
[3] D. Bauer, R. Kiesel, A. Kling, J. Rub, Risk neutral valuation of participating life insurance contracts, Insur. Math. Econ. 39 (2005) 171–183.

[4] P. Hieber, R. Korn, M. Scherer, Analyzing the effect of low interest rates on the surplus participation of life insurance policies with different annual interest rate guarantees, Eur. Actuar. J. (2015) 1–18, http://dx.doi.org/10.1007/s13385-014-0102-3.

[5] B. Jensen, P. Jørgensen, A. Grosen, A finite difference approach to the valuation of path dependent life insurance liabilities, Geneva Pap. Risk Insur. Theory 26 (2001) 57–84.

[6] K. Miltersen, S. Persson, Guaranteed investment contracts: distributed and undistributed excess returns, Scand. Actuar. J. 23 (2003) 257–279.
[7] T. Møller, M. Steffensen, Market-valuation Methods in Life and Pension Insurance, Cambridge University Press, 2007.
[8] A. Tanskanen, J. Lukkarinen, Fair valuation of path-dependent participating life insurance contracts, Math. Econ. 33 (2004) 595–609.
[9] T. Gerstner, M. Griebel, M. Holtz, Efficient deterministic numerical simulation of stochastic asset-liability management models in life insurance, Insur. Math. Econ. 44 (2009) 434–446.
[10] M. Milevsky, T. Salisbury, Financial valuation of guaranteed minimum withdrawal benefits, Insur. Math. Econ. 38 (2006) 21–38.
[11] A. Sandström, Handbook of Solvency for Actuaries and Risk Managers: Theory and Practice, Chapman and Hall/CRC, 2010.
[12] M.V. Wuthrich, M. Merz, Financial Modeling, Actuarial Valuation and Solvency in Insurance, Springer, 2013.
[13] A. Braun, M. Fischer, H. Schmeiser, How to derive optimal guarantee levels in participating life insurance guarantees, 2016 (preprint).
[14] H. Schmeiser, J. Wagner, A proposal on how the regulator should set minimum interest rate guarantees in participating life insurance contracts, J. Risk Insur. 9999 (2014) 1–28.
[15] A. Bacinello, Pricing guaranteed life insurance participating policies with annual premiums and surrender option, North Am. Actuar. J. 7 (2003) 1–17.
[16] A. Bacinello, Fair valuation of a guaranteed life insurance participating contract embedding a surrender option, J. Risk Insur. 70 (2003) 461–487.
[17] A. Bacinello, Modelling the surrender conditions in equity-linked life insurance, Insur. Math. Econ. 37 (2005) 270–296.
[18] R. Kouwenberg, Scenario generation and stochastic programming models for asset liability management, Eur. J. Oper. Res. 134 (2001) 279–292.
[19] S.A. Zenios, W.T. Ziembla, Handbook of Asset and Liability Management. I: Theory and Methodology, Elsevier, UK, 2006.
[20] S.A. Zenios, W.T. Ziembla, Handbook of Asset and Liability Management. II: Applications, Elsevier, UK, 2007.
[21] G. Consigli, M.A.H.T. Dempster, Dynamic stochastic programming for asset liability management, Ann. Oper. Res. 81 (1998) 131–161.
[22] J. Gonzio, R. Kouwenberg, High performance computing for asset liability management, Oper. Res. 49 (2001) 879–891, 279-292.
[23] N. Jansen, Model Points for Asset Liability Models (Master Thesis), Maastricht University Faculty of Economics and Business Administration, 2008.
[24] S.D. Promislow, Fundamentals of Actuarial Mathematics, Wiley, 2015.
[25] D. Andersson, A risk and capital requirement model for life insurance portfolios, Master-uppsats, Umeå universitet/Institutionen för matematik och matematisk statistik.
[26] D. Brigo, F. Mercurio, Interest Rate Models – Theory and Practice, Springer, 2007.
[27] P.E. Kloeden, E. Platen, Numerical Solution of Stochastic Differential Equations, Springer Verlag, 1992.
[28] A. Grosen, P. Jorgensen, Fair valuation of life insurance liabilities: the impact of interest rate guarantees, surrender options and bonus policies, Insur. Math. Econ. 26 (2000) 37–57.
[29] P. Glasserman, Monte Carlo Methods in Financial Engineering, Springer, 2003.
[30] Nvidia, NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110 (Whitepaper), Nvidia, 2012.

José L. Fernández. Full professor in Mathematical Analysis at the Faculty of Mathematics in the Autonomous University of Madrid. He got a Ph.D. in Mathematics from Washington University at St. Louis. During his stay in USA, he has been main researcher of 2 grants from the National Science Foundation and a Special Grant in 1986. He has also been main researcher of a EU project involving research groups from universities in 6 European countries. He has more than 50 scientific publications. Main research areas are related financial mathematics and complex analysis. He has been editor in chief of Revista Matemática Iberoamericana for a long time. He has advised 8 Ph.D. defended thesis. Also he has been Associate Member of the company Analistas Financieros Internacionales (AFI), a Spanish financial consultancy firm with worldwide activity, in which he got a lot of experience in the collaboration with the financial and insurance sectors, and also was the Director of the AFI Financial School and the AFI Master on Quantitative Finance. In 2009 h3 was nominated Doctor Honoris Causa by Universidad de La Laguna (Spain) and in 2015 he was one of the three first awarded with the medal of the Spanish Royal Society of Mathematics.



Ana M. Ferreiro-Ferreiro. Associate Professor of Applied Mathematics at the Department of Mathematics of the University of A Coruña since 2006, member of the research group "Models and numerical methods in engineering and applied sciences", M2NICA and affiliated Researcher of Technological Institute for Industrial Mathematics, ITMATI. She obtained her M.Sc. degree in Mathematics from University of Santiago de Compostela, Spain, and in 2006 received the Ph.D. degree in mathematics from University of Sevilla, Spain. Her research interests include mathematical modeling, numerical methods, high-performance computing and parallel computing. She has worked in numerical schemes for solving hyperbolic equations arising in fluid mechanics and her current research interests are financial mathematics, numerical methods for global optimization and numerical simulation using HPC hybrid architectures with GPUs accelerators. Her scientific activity has produced more than 15 publications in relevant journals.

José A. García-Rodríguez. Associate Professor of Applied Mathematics at the Department of Mathematics of the University of A Coruña since 2006, member of the research group "Models and numerical methods in engineering and applied sciences", M2NICA and affiliated researcher of Technological Institute for Industrial Mathematics, ITMATI. He obtained his M.Sc. degree in Mathematics from University of Málaga, Spain, and received the Ph.D. degree in mathematics from the same university in 2005. He has worked in numerical schemes for solving hyperbolic equations arising in fluid mechanics and its parallel implementation. His current research interests are financial mathematics, numerical methods for global optimization and numerical simulation using HPC hybrid architectures with GPUs accelerators. He has participated in projects and technology transfer contracts with enterprises and public institutions. His scientific activity has produced more than twenty publications in relevant journals.

Carlos Vázquez. Full professor in Applied Mathematics at the Faculty of Informatics in the University of A Coruña (Spain) since 2000.Formerly he was associated professor at universities of A Coruña, Vigo and Santiago, and invited professor at universities of Lyon and Bologna. He has a B.Sc. in Mathematics (5 year degree), a B.Sc. in Economics (5 year degree) and a Ph.D. in Applied Mathematics from University of Santiago. He got a Doctorat Troisieme Cicle at University Claude Bernard de Lyon (France). He is coauthor of more than 90 articles in scientific journals (80 in JCR) related to modeling, numerical methods and scientific computing for problems arising in finance, insurance, fluid and solid mechanics, tribology, glaciology and biology. Main researcher of national and international grants, and involved in more than 15 contracts with industry. He has been invited speaker in more than 30 international congresses and author of more than 120 national and international congress communications. He has advised 11 Ph.D. defended thesis. He currently belongs to editorial boards of 7 international scientific journals. He is also main researcher of the group "Models and numerical methods in engineering and applied sciences", M2NICA, and affiliated researcher of Technological Institute for Industrial Mathematics, ITMATI.