Bus Inf Syst Eng 62(1):5–20 (2020)
https://doi.org/10.1007/s12599-019-00626-y

RESEARCH PAPER

Efficient Model Points Selection in Insurance by Parallel Global Optimization Using Multi CPU and Multi GPU

Ana Maria Ferreiro-Ferreiro · José Antonio García-Rodríguez · Luis A. Souto · Carlos Vázquez

Received: 21 February 2019 / Accepted: 19 August 2019 / Published online: 9 December 2019

Abstract In the insurance sector, Asset Liability Management refers to the joint management of the assets and liabilities of a company. The liabilities mainly consist of the policies portfolios of the insurance company, which usually contain a large amount of policies. In the article, the authors mainly develop a highly efficient automatic generation of model points portfolios to represent much larger real policies portfolios. The obtained model points portfolio must retain the market risk properties of the initial portfolio. For this purpose, the authors propose a risk measure that incorporates the uncertain evolution of interest rates to the portfolios of life insurance policies, following Ferri (Optimal model points portfolio in life, 2019, arXiv:1808.00866). This problem can be formulated as a minimization problem that has to be solved using global numerical optimization algorithms. The cost functional measures an appropriate distance between the original and the model point portfolios. In order to solve this problem in a reasonable computing time, sequential implementations become prohibitive, so that the authors speed up the computations by developing a high performance computing framework that uses hybrid architectures, which consist of multi CPUs together with accelerators (multi GPUs). Thus, in graphic processor units (GPUs) the evaluation of the cost function is parallelized, which requires a Monte Carlo method. For the optimization problem, the authors compare a metaheuristic stochastic differential evolution algorithm with a multi path variant of hybrid global optimization Basin Hopping algorithms, which combines Simulated Annealing with gradient local searchers (Ferreiro et al. in Appl Math Comput 356:282–298, 2019a). Both global optimizers are parallelized in a multi CPU together with a multi GPU setting.

Keywords Model points portfolio · Risk functional · Hybrid optimization algorithms · Differential evolution · Basin hopping · Monte Carlo simulation · HPC · Multi CPU · Multi GPU

1 Introduction

The motivation of this work arises from a very relevant problem in finance, and particularly for life insurance companies: the so-called asset liability management (ALM). ALM consists of the joint management of assets and liabilities portfolios to ensure the future wealth and profitability of the insurance company (for example, see Corsaro et al. 2010; Fernández et al. 2018; Gerstner et al. 2008; Schmeiser and Wagner 2014; Corlosquet-Habart et al. 2015 and the references therein). For this purpose, it is important to compute the balance sheet projection: the joint projection of the future cash flows of assets and liabilities portfolios.

Traditionally, the projected cash flows have been computed for some previously designed scenarios 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 (Sandström 2010;

Accepted after two revisions by the editors of the special issue.

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s12599-019-00626-y) contains supplementary material, which is available to authorized users.

A. M. Ferreiro-Ferreiro · J. A. García-Rodríguez · L. A. Souto · C. Vázquez (✉)
Department of Mathematics and CITIC, University of A Coruna, Faculty of Informatics, 15071 A Coruña, Spain
e-mail: carlosv@udc.es

M. Merz and Wuthrich 2010) 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, allows the computation of more accurate approximations of the portfolio evolution with models of increasing complexity.

Computing these projections for the original portfolios, usually comprising a high number of policies (hundreds of thousands) on the liabilities side, usually leads to a highly demanding computational time task or is even prohibitive in reasonable time schedules. With this in view, insurance companies are allowed to compute these projections by replacing the original policies portfolio with some suitable representative set of contracts, usually known as the related model points (see Corlosquet-Habart et al. 2015). Thus, previous to the ALM computations, the model points selection is applied on the liabilities side to obtain a reduced portfolio of policies which mimics the much larger real portfolio. Next, the ALM balance sheet projections are computed by means of Monte Carlo simulation, taking into account the more manageable model points portfolio. The alternative consideration of each policy in the original portfolio would become prohibitively time-consuming and memory-consuming.

This new portfolio could be understood as a compressed version of the original one that should retain the same risk properties (see EIOPA2010 2010, for further details). In order to measure to what extent these risk properties are retained in this replacement, we need to choose an appropriate functional. This risk functional is usually defined in terms of the fluctuation of some underlying stochastic risk factors inside a time horizon. The interest rate term structure and the mortality trend of the population are two of the main factors when dealing with the valuation and the risk management of life insurance products (Jalen and Mamon 2009). For example, in Denuit and Trufin (2015) the choice of the model points aims to control the impact on Tail-Var and related risk measures. More recently, in Goffard and Guerrault (2015) an alternative method to group policies in model points is related to the Best Estimate Liabilities. As indicated in Goffard and Guerrault (2015), model points are usually incorporated in commercial actuarial software and information systems, such as MG-ALFA (see MG-ALFA 2019) and GGY-AXIS (see GGY-Axis 2019). In Ferri (2019), a first attempt to introduce the proposed mathematical and computational methodologies for the automatic selection of model points based on the minimization of a risk functional is addressed. In this case, the main underlying risk factor is the interest rate, although the methodology can be extended to incorporate mortality risk, which also is relevant in life insurance portfolios (Jalen and Mamon 2009), or additional risk factors. We have chosen a LIBOR marked model for the stochastic evolution of interest rates, but alternative models in the literature can be used.

Once the risk measure functional has been chosen, its value has to be approximated by using a Monte Carlo algorithm to simulate the involved risk factors. Thus, finding the structure of the model points portfolio can result in a very hard computational problem. More precisely, finding the new model points portfolio can be posed as a global optimization problem, where we have to minimize the distance between both portfolios in terms of the chosen risk measure.

Multi CPUs and GPUs settings have been widely used for accelerating the implementation of Monte Carlo numerical methods, particularly in finance. For example, in Fernández et al. (2018) a parallel Monte Carlo ALM balance sheet projection was performed in GPUs; in Ferreiro et al. (2014) SABR/LIBOR models for the evolution of interest rates including stochastic volatility smile are considered for the pricing and calibration of interest rate derivatives and parallelization is addressed by means of GPUs and multi CPUs; in Corsaro et al. (2010) ALM balance sheet projections are obtained by a parallel implementation of Monte Carlo methods in multi CPUs; in Lee et al. (2012) a study on the advantages of GPUs to perform massively parallel simulations of advanced Monte Carlo methods is presented; and recently in Leitao and Oosterlee (2017) GPUs technology is successfully applied to modern Monte Carlo type methods for multi-dimensional Bermudan options pricing.

Due to their stochastic nature, global optimization methods are also good candidates for parallelization. For example: the parallelization using hybrid architectures (accelerators, GPUs) for image recognition, parallel Differential Evolution and the GPU parallel cost function was presented in Casella et al. (2018); in Zhu (2011) a massively parallel DE-pattern search was implemented in GPUs; in Tasoulis et al. (2004) a parallel Differential Evolution (DE) method was implemented in multi CPU; in McCarty and McGuire (2018) a parallel multi CPU version of the Monotonic Basin Hopping for thrust optimization was presented; in Ferreiro et al. (2013) the authors proposed a parallel version of the Simulated Annealing (SA) algorithm; and in Ferreiro et al. (2019a) a parallel multi path version of Basin Hopping (BH) for multi CPUs or multi GPUs was presented.

The plan of this paper is as follows. In Sect. 2 we recall the insurance problem and introduce the risk functional for a given interest rate model. More precisely, we describe the model points portfolio selection problem, we present the definition of the risk measure and we show the numerical discretization of the cost function using a Monte Carlo

numerical method. Section 3 is devoted to the HPC implementation of the optimization problem. We show the efficient GPU implementation of the cost function, and we briefly describe the global optimization algorithms (DE and a multi path BH algorithm) and its parallel implementation in a multi CPU framework. In this paper we apply these algorithms to the optimization of the risk functional measure, for obtaining the model point portfolio. In Sect. 4, we show the numerical experiments. First, we illustrate the parallel performance of the cost function. Second, we show some examples with known solution to validate the convergence of the optimization algorithms. After validating the methodology, we finally apply the technique of the model points portfolio generation to a general insurance liability portfolio.

In Appendix 1, we recall the LIBOR market model that represents one of the building blocks of our risk functional. In Appendix 2, we show the used biometric survival model. In Appendix 3 we show the code listings.

2 Optimization Problem

In this section we describe the global optimization problem. In Sect. 2.1 we introduce the cost function, while Sect. 2.2 presents the numerical discretization for its evaluation.

First, we describe the financial setting of the problem that can be framed in the life insurance policies portfolios. The cost function is given by the risk measure associated with the model points representation of a portfolio consisting of with-profit life insurance policies, i.e., policies that may pay a benefit to the policy holder. Thus, the portfolio is a set of $I \times J$ with-profit life insurance policies corresponding to a set of ages $\mathcal{X} = \{x_1, \ldots, x_I\}$ and maturities $\mathcal{Y} = \{y_1, \ldots, y_J\}$, so that $\mathcal{X}$ and $\mathcal{Y}$ are two sets of real numbers, such that $x_i \geq 0$ and $y_j \geq 1$, for $i = 1, \ldots, I$ and $j = 1, \ldots, J$. These contracts pay a lump sum benefit in case of the death of the policy owner, provided that it occurs until a specific date that is defined in the contract.

We assume that we are dealing with policies that are unaffected by credit risk, i.e., the insurance company always guarantees the entire benefit that is provided for in the contract. On the other hand, we do not analyze the revenues received by the insurance company and thus we do not take into account the premiums stream of the contract nor any further expenses that are the responsibility of the client.

2.1 Cost Function: Risk Functional

For computing the Model Points Risk Estimation we apply the theory presented in Ferri (2019). For this purpose, we start from a given policy portfolio $v = (v_{ij})$ with $I \times J$ policies, as described above. Next, we fix a set of insurance policies portfolios $\mathcal{W}$ containing policies with fixed L ages and M maturities, with L and M smaller than I and J, respectively. We refer to any element $w = (v_{lm}) \in \mathcal{W}$ as one model points portfolio in that set.

Moreover, we understand the model points risk functional $R(w|v)$ as the error that occurs when the original portfolio v is replaced by the model points portfolio $w \in \mathcal{W}$. Such an error is assessed as the average changes of the difference between the two portfolios in terms of the stochastic fluctuation of the interest rate risk term structure. The model point $w^*(v)$ that minimizes this functional is understood as the best representation of the original portfolio v preserving the risk associated to the underlying stochastic interest rates evolution.

As we are considering the term structure as the only risk factor, we must select a model for the evolution of interest rates. It is important to notice that this choice is by no means a limitation in the application of our methodology, as any stochastic model can be plugged into the developed software toolbox and its Monte Carlo simulation can benefit from the parallel implementation. Although in the present work we consider a LIBOR market model (Brigo and Mercurio 2006), any short rate model (Vasicek, CIR, Black-Karasinski,...) could be used. Furthermore, if we aim to incorporate the recent presence of negative rates in yield curves then suitable recent models like shifted LIBOR (Dutra-Lopes and Vázquez 2019), shifted SABR or free boundary SABR (Antonov et al. 2015) can be considered.

Once we fix an interest rates model, the model points risk functional induced by a portfolio v over $\mathcal{W}$ admits the form:

$$R(w|v) = \mathbb{E}\left\{ \int_{\mathcal{I}} \left\| \sum_n \hat{B}_n(t)\left(R_n^*(v) - R_n^{**}(w)\right) \right\|_W^2 (1-t)dt \right\}, \quad \text{for any } w \in \mathcal{W}, \tag{1}$$

where:

- $\mathcal{I} = (0, 1)$, which corresponds to one year period.
- The expressions for $R_n^*(v)$ and $R_n^{**}(w)$ are given by:

$$R_n^*(v) = \sum_{i,j} v_{ij} S_T(x_i, T_n) \mathbb{1}_{\{T_n \leq y_j\}},$$

$$R_n^{**}(w) = \sum_{l,m} w_{lm} S_T(x_l, T_n) \mathbb{1}_{\{T_n \leq y_m\}}.$$

- $\hat{B}_n(t)$ is linked to the discounted bond price for a given interest rates model. For example, in the case of the LIBOR model we have $\hat{B}_n(t) = \varepsilon_n(t)\tilde{B}_n(t)$ [see Appendix 1, Eqs. (10) and (11)],

– $S(x_i, T_n)$ is the survival rate, which is understood as the proportion of those individuals labelled by $x_i \in \mathcal{X}$ that survive to the age $x_i + T_n$. The survival rate can be computed using past survival tables or from a mortality model. In our case we use the model of Appendix 2.

Note that when replacing LIBOR with an alternative interest rate model, we just take it into account when computing the term $\hat{B}_n(t)$. By using expression (1) we consider the error in norm L^2, which fits with the chosen optimization methods that require differentiability. In case we considered the more robust for outliers L^1 norm, then alternative methods should be used.

With the previous notations, the selection of model points policy portfolio w^* for an original portfolio v is posed as the global optimization problem

$$w^*(v) = \operatorname{argmin}_{w \in \mathcal{W}} R(w|v). \tag{2}$$

Note that the model points risk functional is identified as the cost function or objective function in the optimization literature. The parameters to optimize are the nominals of each model point.

In view of the nature of the global optimization problem (2), we need to propose efficient numerical methods to evaluate the risk functional $R(\cdot \mid v)$ [given by expression (1)] and also efficient optimization methods to minimize its value.

2.2 Monte Carlo Numerical Discretization

In order to discretize the cost functional (1), the involved expectation is computed by Monte Carlo simulation, so that each simulation requires the computation of the evolution of the interest rates according to the chosen model. In this paper we choose the LIBOR market model (see Appendix 1), thus following the ideas in Brigo and Mercurio (2006), taking logarithmic rates and using Ito lemma in (7), the forward rate F_n dynamics satisfy the following equation with deterministic diffusion coefficient:

$$\begin{aligned} d\ln F_n(t) = \sigma_n(t) \sum_{k=1}^{n} \frac{\varrho_{nk}\tau_k\sigma_k(t)F_k(t)}{1+F_k(t)\tau_k} dt \\ - \frac{\sigma_n(t)^2}{2} dt + \sigma_n(t) dW_n(t), \end{aligned} \tag{3}$$

where F_n denotes the forward rate with maturity T_n, $n = 1, \ldots, N$, $\tau_n = T_n - T_{n-1}$ is the associated accrual, σ_n is the volatility of F_n, and ρ_{nk} is the correlation between forward rate F_n and F_k.

Next, applying the classical Euler–Maruyama scheme for the time discretization of (3), we get

$$\begin{aligned} \ln \hat{F}_n(t+\Delta t) = \ln \hat{F}_n(t) + \sigma_n(t) \sum_{k=1}^{n} \frac{\rho_{nk}\tau_k\sigma_k(t)\hat{F}_k(t)}{1+\tau_k\hat{F}_k(t)} \Delta t \\ - \frac{\sigma_n(t)^2}{2} \Delta t + \sigma_n(t)(\hat{W}_n(t+\Delta t) - \hat{W}_n(t)), \end{aligned} \tag{4}$$

for $n = 1, \ldots, N$, where $\hat{F}_n(t)$ is the approximation of the forward rate $F_n(t)$ and $\hat{W}_n(t+\Delta t) - \hat{W}_n(t) \equiv \sqrt{\Delta t}\mathcal{N}(0,1)$ simulates the increment of the multidimensional Wiener process $dW_n(t)$ at time t. The discretization is performed over a uniform mesh defined by the mesh nodes $t_q = q\Delta t$, for $q = 0, \ldots, N_t$ where Δt denotes the constant time step in the Euler-Maruyama scheme, which exhibits strong convergence of order 1/2 in Δt.

Next, we describe the discretization of expression (1) for the functional $R(w|v)$. More precisely, if we denote by $\hat{R}(w|v)$ its approximation, then

$$\begin{aligned} \hat{R}(w|v) &= \frac{1}{N_p} \sum_{p=1}^{N_p} \left(\frac{1}{N_t} \sum_{q=1}^{N_t} ||R_{pq}(w|v)||^2 (1-t_q) \right) \\ &= \frac{1}{N_p} \sum_{p=1}^{N_p} \left(\frac{1}{N_t} \sum_{q=1}^{N_t} R_{pq}(w|v) \cdot C \cdot R^t_{pq}(w|v)(1-t_q) \right), \end{aligned}$$

where C denotes the correlation matrix, N_t is number of time steps, N_p the number of simulations and R_{pq} is defined by:

$$R_{pq}(w|v) = \sum_{n=1}^{N} (R^*_{pq}(v) - R^{**}_{pq}(w)) v_{pq}(T_n), \tag{5}$$

with the vector $v_{pq}(T_n)$ given by

$$\begin{aligned} v_{pq}(T_n) &= \epsilon^p_n(t_q)\tilde{B}^p_n(t_q) \\ &= -\tilde{B}^p_n(t_q) \sum_{k=1}^{n} \frac{\tau_k}{1+\tau_k F_n(t_q)} \Sigma_k(t_q), \end{aligned}$$

with

$$\tilde{B}^p_n(t_q) = \frac{\tau_n \sigma_n(t_q) F^p_n(t_q)}{1+\tau_n F^p_n(t_q)},$$

where index p is associated to a particular simulation of forward LIBOR rates and discounted bond price, index q is related to time t_q and index n is related to maturity T_n in the tenor structure. Moreover, we have used the notation

$$R^*_{pq}(v) = \sum_{i,j} v_{ij} S_T(x_i, T_n)(T_n - y_j),$$

$$R^{**}_{pq}(w) = \sum_{l,m} w_{lm} S_T(x_l, T_n)(T_n - y_m),$$

where v_{ij} denotes the nominal of contracts with age x_i and maturity y_j in the original portfolio, while w_{lm} denotes the analogous in the model points portfolio. The pseudocode for the risk functional discretization is shown in Algorithm 1.

Algorithm 1: Cost function. Pseudocode.

Data: Δt, N_t, Euler time step size and number of time steps
Data: t_i, $i = 1, \ldots, N_t$, Times for Euler Scheme
Data: N_{Paths}, Number of Monte Carlo paths
Data: M, Number of tenors
Data: T_k, $k = 0, \ldots, M$, Tenor structure
Data: $F_k(0)$, $k = 0, \ldots, M$, Forward initial rates
Data: Initial volatilities, $\sigma_k(0)$
Data: Input file containing original policies portfolio
Data: $numP$, Number of policies in the portfolio
Data: $N_p(i)$, $A_p(i)$, $E_p(i)$, $i = 1, \ldots, numP$, number of contracts, age and expiration of each police of the portfolio
Data: Structure of model points portfolio:
Data: $numMP$, Number of model points
Data: $N_{mp}(i)$, $A_{mp}(i)$, $E_{mp}(i)$, $i = 1, \ldots, numMP$, number of contracts, age and expiration of each police of the model points portfolio
Data: a, b, data for survival model
Data: β, data for correlations
$S(age, T, t)$: Force of mortality function L, build correlation matrix
$L = CC^T$, Perform Cholesky decomposition
$B(t_0, t)$, Discount factor from t_0 to t.
/* Parallel loop */
for $j = 1$ *to* N_{Paths} **do**
  $Rj = 0$
  for $i = 1$ *to* $N_t - 1$ **do**
    `/* Compute vector for the Brownian motions */`
    for $k = 1$ *to* M **do**
      Generate the Brownian step $\Delta B_k(t_i)$
    end for
    $\Delta W(t_i) = C \Delta B(t_i)$
    `/* Compute vector of Forward rates */`
    for $k = 1$ *to* M **do**
      Simulate $[F_k(t_i)$ using log-Euler scheme (4)
      $ln\, F_k(t_i) = \ln F_k(t_{i-1}) + \mu_k(t_i)(t_i - t_{i-1}) + \sigma_k(t_i) F_k(t_i) \Delta W_k(t_i)$
      where $\mu_k(t_i)$
      $\mu_k(t_i) = \sigma_k(t_i) F_k(t_i) \sum_{h=1}^{k} \frac{\varrho_{kh} \tau_h \sigma_h(t_i) F_h(t_i)}{1 + F_h(t_i)\tau_h}$.
    end for
  end for
  for $i = 1$ *to* $N_t - 1$ **do**
    $p(t_i, k) = 1$
    for $k = 1$ *to* M **do**
      Compute the discounted bond prices
      $B(t_i, k) = B(t_i, k) \cdot \dfrac{1}{1 + F_k(t_i)\tau_k}$
      Compute the diffusive component of the discounted bond price $\nu_{ij}(T_k) = \dfrac{\tau_k}{1 + \tau_k F_k(t_i)} \sigma_k(t_i) F_k(t_i)$
    end for
    $Ri = 0$
    for $k = 1$ *to* M **do**
      $sum_p = 0$
      for $l = 1$ *to* $NumP$ **do**
        $sum_p += N_P l[l] * S(A_P[l], T_k, t_i) * (T_k <= E_p[l])$
      end for
      $sum_{mp} = 0$
      for $l = 1$ *to* N_{MP} **do**
        $sum_{mp} += N_{MP}[l] * S(A_{MP}[l], T_k, t_i)(T[k] <= E_{MP}[l])$
      end for
      $Ri = Ri + (sum_p - sum_{mp}) * (-B(t_i, k) * \nu_{ij}(T_k))$
    end for
    $Rj += Ri * L * Ri * (1 - t_i)$
  end for
end for
$R = 0$
for $j = 1$ *to* N_{Paths} **do**
  $R += Rj$
end for
$R = (1/NPaths) * (1/N_t) * R$

3 HPC Numerical Implementation

The optimization of this cost function is a hard problem mainly for two reasons :

- On one hand, we have to evaluate a a very expensive cost function, because it relies on the discretization of the risk functional using a Monte Carlo method, which is known to have a high computational cost. Each Monte Carlo path involves the simulation of the interest rate values, and additionally a computationally intensive loop in the policies, corresponding to the original portfolio.
- On the other hand, the resulting optimization problem is a global one, and requires performing a large number of evaluations of this costly cost function. Thus, we can not use fast local optimizers, because they get stuck into local minima.

Thus, each evaluation has to be performed as fast as the available software and hardware computational tools can allow for. Also the numerical global optimization algorithms must be efficient.

As we are using a stochastic global optimizer to minimize the cost function, which in turn is evaluated by Monte Carlo methods, a double level of parallelization can be applied. Actually, we face the problem of the parallel implementation of two nested Monte Carlo type algorithms: one for the stochastic optimizer paths, and one for the cost function evaluation Monte Carlo paths. Hybrid hardware architectures with accelerators, such as FPGAs or GPUs are well suited for handling this double level of parallelism. Thus, we can carry out the search threads/paths of the global searcher using multiple CPU threads, while the calls to the cost function at each of those threads is offloaded to one accelerator (GPUs in our case) per CPU thread, that is responsible for carrying out the Monte Carlo discretization for the evaluation of the cost function. In this setting we take advantage of all the available resources of the machine, using as many CPU threads as available accelerators.

Therefore, in this section we describe the two main tasks to achieve an HPC implementation of the optimization numerical methods. In Sect. 3.1, we show the GPU parallel implementation of Monte Carlo method for computing the cost function (1). In Sect. 3.2, we discuss the proposed numerical methods for the global optimization problem (2), which require very efficient and fast evaluations of the functional discretization, and we also describe the parallel multi CPU implementation of the optimizers.

3.1 Cost Function GPU Implementation

As the cost function is computed via Monte Carlo, it involves a large computational cost. However, it also offers the opportunity of using massively parallel computing techniques for the evaluation of the cost function. In particular, many core architectures like GPUs are well suited for performing these Monte Carlo simulations (see, for example, Fernández et al. 2018; Ferreiro et al. 2014; Corsaro et al. 2010; Lee et al. 2012 or Leitao and Oosterlee 2017).

Thus, we have carried out the parallel implementation of the cost function by using GPUs and the CUDA API. Moreover, a parallel random number generator algorithm for GPU architectures has been used for parallelizing Monte Carlo simulation. More precisely, in this case we use the CURAND library. In this GPU setting, the cost function is mapped to a GPU kernel, so that each Monte Carlo path is computed by a different computing thread in the GPU. We have paid special attention to handle all the memory accesses in a coalesced way, which is the best suited memory access pattern for this problem, in order to take advantage of the wide memory access bus in the GPUs.

We would like to emphasize that the whole code has been implemented from scratch in C++, including the LIBOR rates simulator and the Monte Carlo technique for computing the risk functional. In Appendix 3, a skeleton of the code for computing the cost function can be seen in Listings 1, 2 and 3. Listing 2 shows the cost function with the calls to the GPU kernels. In Listing 2 we present a summary with the code of the kernel for computing the evolution of the LIBOR interest rate model (see Appendix 1). In Listing 3 a sketch of the kernel for computing the cost function (1) (discretized in (5)) is shown.

3.2 Multi-CPU and Multi-GPU Parallel Global Optimization Algorithms

Obtaining the model points portfolio turns out to be a very difficult problem, as it involves solving a global optimization problem of a high dimension. More precisely, the dimension of the searching space is given by the number of policies in the model points portfolio.

In this section we discuss the efficient parallel numerical implementation of the global optimization algorithms we propose for solving this problem. As they are highly computationally demanding, we also propose to take advantage of parallel computing techniques, which are specially well-suited for the kind of numerical algorithms we are handling.

For solving global optimization problems, stochastic algorithms are usually required. They have the advantage that they can deal with complex problems, discarding local optima and avoiding getting stuck in these local solutions. However, their main disadvantage is associated with their slow convergence, due to their stochastic nature. One example of this kind of algorithm is Simulated Annealing

(SA) (see Ferreiro et al. 2013 and references therein for details). On the other hand, deterministic local optimization algorithms are faster, their disadvantage being that they cannot be guaranteed to escape from local minima. Some examples of these local algorithms are Pattern Search, Nelder Mead or gradient based methods like NCG, BFGS, L-BFGS Liu and Nocedal (1989) and L-BFGS-B (Byrd et al. 1995).

In order to deal with the high computational cost of global optimization algorithms, variants of those algorithms tailored to its HPC implementation either in multi CPU or GPU have been studied (for example, see Casella et al. 2018; Zhu 2011; Tasoulis et al. 2004; McCarty and McGuire 2018; Ferreiro et al. 2013 or Ferreiro et al. 2019a). In the present work we propose the efficient use of parallel implementations of two currently well-known global optimization algorithms:

- A parallel implementation of a Differential Evolution (DE) gradient free algorithm.
- A parallel implementation of a multi-path variant of the Basin Hopping (BH) algorithm, using a L-BFGS-B as local optimizer.

DE is a metaheuristic genetic algorithm for global optimization, originally presented in Storn and Price (1997). It creates *np* solution candidates or "population individuals" and makes them evolve by mixing random movements and the information provided by the rest of the candidates.

There are several variations of the DE algorithm depending on the interaction between the individuals, or, as it is usually called, the "mutation". In this article we implemented the DE/best/1/bin mutation, as this is the one that acts by default in the SciPy library (library 2019). The DE/best/1/bin variation is as follows:

$$x_m = x_b + F(x_{r_1} - x_{r_2}), \tag{6}$$

x_m being the trial or mutated vector, x_b the candidate with the lowest cost function so far, F the mutation factor, and x_{r_1}, x_{r_2} two randomly selected different individuals. In this way, each individual will be mainly influenced by the best solution so far (if F is small), with some stochastic behaviour related to the variance of the population. That is, the closer all the individuals are to the global minimum (or a local minimum if the method got stuck there), the easier to achieve convergence, even for a large value of F.

Again, following the SciPy implementation, we use the Latin Hypercube Sampling (LHS) algorithm to generate the starting points, which avoids the possibility of creating two candidates too close to each other.

Moreover, since the evaluations of the cost function for each population can be performed independently one from each other, this algorithm can also be parallelized both with multi-CPU or GPU. Since the cost function is a GPU kernel then we use OpenMP in the DE method, so that each OpenMP thread uses a different GPU, thus allowing a two-level parallelization. In Algorithm 2 we show the pseudocode for the parallel DE method. First, we generate the initial configuration with the LHS algorithm. Next, for each individual of the population, we generate a new trial member according to the chosen mutation and ensure its components stay in the range [0,1]. Then, we scale the mutated individual to the desired range and evaluate the cost function. If the value of the cost function for the new member is lower than the already recorded, we save this new configuration; otherwise we maintain the previous one. We repeat this process until the stop criterion is satisfied.

Algorithm 2: *Differential Evolution* pseudocode.

```
y = LHS();
Iteration number: k = 0;
Maximum iteration number: K;
Initial position: x_0 = x* = y;
while (k < K) do
    /* Parallel loop */ ;
    for i=0:np do
        y_i = mutate(x_i^k);
        EnsureConstraint(y_i);
        Scale(y_i, ȳ_i);
        test_i^k = f(ȳ_i);
        if test_i^k < v_i^k then
            x* = x_i^{k+1} = y_i;
            v_i^k = test_i^k;
            /* Atomic operation */;
            if test_i^k < v_b^* then
                v_b = test_i^k;
                x_b = y_i;
            end if
        else
            x* = x_i^{k+1} = x_i^k;
        end if
    end for
    k = k + 1;
end while
```

Another possible technique to obtain global optimization algorithms comes from mixing both kinds of algorithms (stochastic global with local ones), thus obtaining the so called hybrid algorithms. Hybrid algorithms can benefit from the global convergence properties of the stochastic ones and from the speed of convergence of the local optimization algorithms (see Fig. 1, for a sketch of the behavior of hybrid algorithms). One example of hybrid algorithms is the Basin Hopping (BH) algorithm (see Wales and Doye 1997; Ferreiro et al. 2019b as well as references therein). In a BH algorithm, first a SA is used for sampling the searching space by randomly generating neighbors. Next, local gradient algorithms are applied to

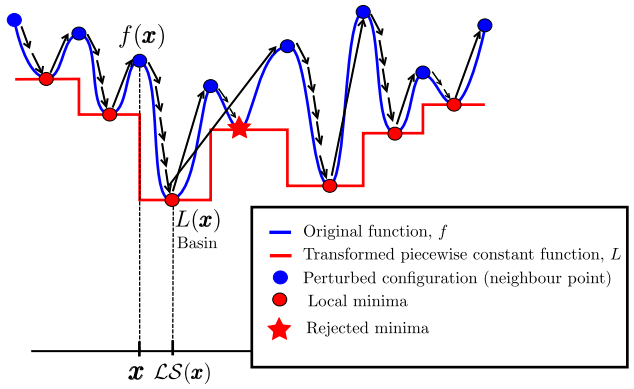

Fig. 1 Sketch of Basin Hopping algorithm

capture the minima starting from the generated points by the stochastic sampler. These algorithms can be seen as the minimization of the simplified piecewise constant function L, where $L(\boldsymbol{x})$ corresponds to the value of the function f evaluated at the local minimum returned by the local optimizer operator, starting from the point $\boldsymbol{x}$, $\mathcal{LS}(\boldsymbol{x})$ (see Fig. 1).

In this article we use a multi path version of BH, that was presented in Ferreiro et al. (2019a). We will refer to this algorithm as BH_M. In this version of BH, a number M of search paths are computed at each step of the algorithm, computing several gradient searches at this step. Before advancing to the next step, the best minimum is gathered from all the paths, and it is used as starting point for all the search paths in the next step of the algorithm. This BH_M version has several advantages: it improves the convergence speed and the success rate of the classical BH, and furthermore it has the advantage that all these search paths can be computed at the same time using different computing threads (CPU or GPU threads), so that the algorithm is highly parallelizable. We refer to Ferreiro et al. (2019a) for more details. The pseudocode of the BH_M algorithm is shown in Algorithm 3.

The whole parallel optimization routines, DE and parallel BH_M, have been implemented from scratch in C++, in the case of BH_M following the previous works (Ferreiro et al. 2013, 2019a).

Note that we are using a GPU implementation of the cost function. So, when we mix the multi CPU implementation of the optimization algorithm with the GPU implementation of the risk function, we end up in a multi GPU setting for the whole HPC implementation for solving the problem. Thus, the multi CPU threads correspond to the search paths or families of the global search algorithms; the GPU threads are used for the computation of the Monte Carlo scenarios for the valuation of the cost function; and the number of the CPU threads that can be used is equal to the number of the available accelerators (GPUs). In our case, we have implemented the OpenMP multi CPU version in one single machine with 4 GPUs (see Fig. 2).

Algorithm 3: Synched multi L-BFGS-B Basin Hopping (BH_M), pseudocode.

$\boldsymbol{y}$ = sampled from a uniform random distribution in D;
Number of successive rejections: $j = 0$;
Iteration number: $k = 0$;
Initial position: $\boldsymbol{x}_0 = \boldsymbol{x}_* = \mathcal{LS}(\boldsymbol{y})$;
Cooling factor and Boltzman constant: ρ, k_b ;
Minimum temperature: $\mathcal{T}_{min}$;
Temperature at each iteration: $\mathcal{T}$;
while $(j < J)$ ***or*** $(\mathcal{T} < \mathcal{T}_{min})$ **do**
  for l=1:M **do**
    for i=1:N **do**
      $\boldsymbol{y}_k^l$ = random uniform in $B(\boldsymbol{x}_k^l, \boldsymbol{r}_k)$;
      $\boldsymbol{u}$ = random uniform in $[0, 1]$;
      $\Delta = L(\boldsymbol{y}_k^l) - L(\boldsymbol{x}_*^l)$;
      if $\boldsymbol{u} < \exp(-\Delta/k_B\mathcal{T})$ **then**
        $\boldsymbol{x}_*^l = \boldsymbol{x}_{k+1}^l = \mathcal{LS}(\boldsymbol{y}_k^l)$;
        $j = 0$;
      else
        $j = j + 1$;
      end if
      $k = k + 1$;
    end for
  end for
  Synchronization: $\boldsymbol{x}_{best} = min(\boldsymbol{x}_*^l)$;
  for l=1:M **do**
    $\boldsymbol{x}_*^l = \boldsymbol{x}_{best}$;
  end for
  Update radius $\boldsymbol{r}_k$;
  $\mathcal{T} = \rho \cdot \mathcal{T}$;
end while

4 Numerical Tests

In this section we show some examples to asses the performance and the accuracy of the proposed methodology.

In Sect. 4.1 we present a test to asses the parallel performance of the GPU implementation. We also compare this performance with the performance of a multi CPU prototype that we developed prior to the final GPU implementation.

In Sect. 4.2 we show an example to validate the proposed model points selection technique: this experiment is a synthetic test with known solution, so that the correct operation of the technique can be assessed comparing with the exact solution. Furthermore, we also use this example to show a comparison between the global DE method and the hybrid multi path BH, BH_M, with L-BFGS-B as local optimizer, as proposed alternatives to solve the problem. More precisely, we will present some graphs with the evolution of the cost function's value with respect to the

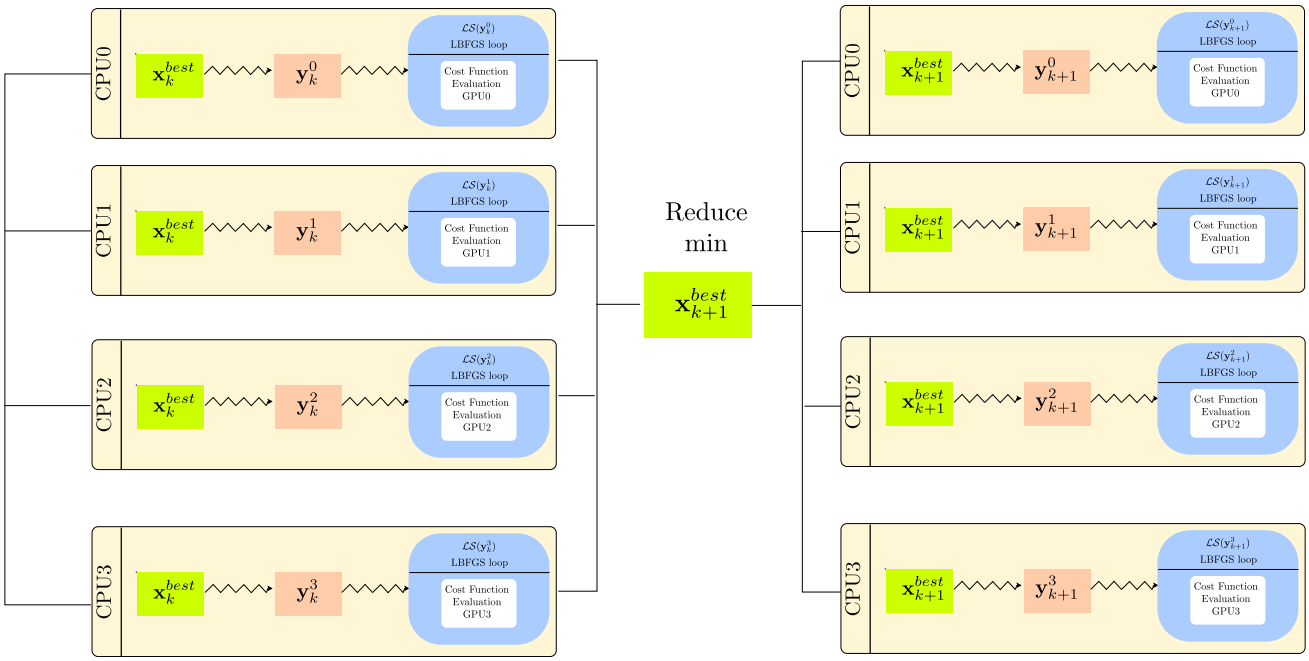

Fig. 2 Sketch of the parallel global optimization BH_M algorithm, using multi CPU and multi GPU with 4 CPUs. y_k^i represents the neighbour of x_k^{best} in thread i at step k

number of evaluations. Note that the number of evaluations is closely related to the computational cost.

Finally, in Sect. 4.3 we show in Example 2 an application to a real world scenario, with unknown solution.

The parameters of the LIBOR model are the same for all tests, and can be checked in Appendix 1.

Concerning the hardware configuration, all tests have been performed in a hybrid architecture server with 16 GB of RAM, 12 CPU cores (two Intel Xeon E5-2620 v2 at 2.10 GHz) and 4 Nvidia GeForce GTX TITAN Black GPUs (Kepler architecture).

4.1 Performance of the Parallel Implementation of the Risk Functional Evaluation

As we mentioned before, the computational cost of the proposed optimization algorithms is closely related to the cost of the risk functional's evaluation, which is performed a large number of times during the optimization procedure. Therefore, as a prior step to the presentation of numerical examples, we show the performance of the multi CPU and GPU implementations for the risk functional calculation.

For this purpose, we consider 10,000 policies in the original portfolio and 10 policies in the model points portfolio. For building the 10,000 policies of the original portfolio, we consider the 10 policies in Table 1, and we repeat each one 1000 times. In this way, we end up with an original portfolio containing 10,000 policies (although only of the 10 different types in Table 1).

Recall that the cost function is stochastic and its computation requires a Monte Carlo simulation technique. Therefore, the test in this section has been performed by using a different number of paths for the Monte Carlo simulation in the computation of the cost function. Thus, we illustrate the effect in the speed-up for different numbers of paths.

Table 2 and Figs. 3, 4 show the obtained speed-up for the parallel implementation of the risk functional when different numbers of Monte Carlo paths and different number of CPU cores or GPU are being used. As illustrated in Figs. 3 and 4, the speed-up increases when the number of Monte Carlo paths increases. Therefore, parallelization becomes more interesting and efficient for a large number of Monte Carlo paths, which is the usual situation.

Table 1 Original portfolio for Example 1

Age	Maturity	Nominals
20.0	50.0	50,000
25.0	45.0	100,000.0
30.0	40.0	150,000.0
35.0	35.0	200,000.0
40.0	30.0	250,000.0
45.0	25.0	300,000.0
50.0	20.0	350,000.0
55.0	15.0	400,000.0
60.0	10.0	450,000.0
65.0	5.0	500,000.0

Table 2 Computation times in seconds and speedups with different numbers of CPUs and using the GPU

Paths	Hardware	Time (s)	Speedup
256	1 CPU	309.57	1.00
	2 CPUs	156.68	1.98
	4 CPUs	81.81	3.78
	8 CPUs	41.00	7.55
	12 CPUs	29.12	10.63
	1 GPU	7.08	43.72
1024	1 CPU	1218.84	1.00
	2 CPUs	631.52	1.93
	4 CPUs	327.17	3.73
	8 CPUs	164.21	7.42
	12 CPUs	115.04	10.59
	1 GPU	7.09	171.91
4096	1 CPU	5008.62	1.00
	2 CPUs	2616.86	1.91
	4 CPUs	1349.40	3.71
	8 CPUs	675.19	7.42
	12 CPUs	455.30	11.00
	1 GPU	9.91	505.41

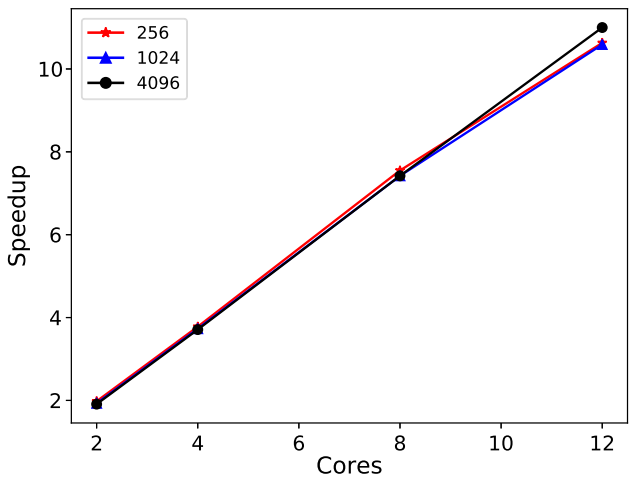

Fig. 3 Speedups of multi CPU parallel implementation for different number of Monte Carlo paths

4.2 Example 1: Analytical Test: Repeated Policies Classification

In this example we validate the proposed technique for a problem with known solution. Thus, we compute the model points portfolio by using the two previously discussed global optimization algorithms: the pure global DE algorithm and the hybrid BH_M algorithm, so that we can compare the performance of both algorithms for the major problem we pose.

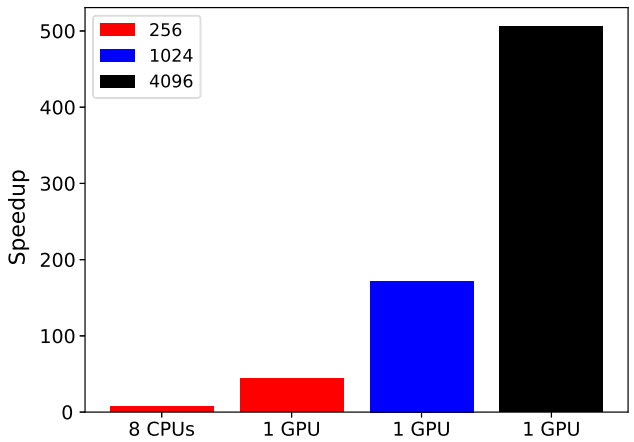

Fig. 4 Speedups of multi CPU and GPU parallel implementation for different number of Monte Carlo paths

In this example we will use the same starting portfolio as in Sect. 4.1. We built that portfolio by repeating the policies in Table 1 1000 times. Although they are repeated (actually, there are only 10 types of policies), each policy is understood as a different individual one, from the computational point of view. Concerning the model points portfolio, we try to represent the previously described original portfolio with a 10 model points portfolio with the same structure (ages and maturities) as the one in Table 1, so that the problem consists of finding the nominals in the model points for that model points portfolio that better represents the original portfolio with respect to the risk functional. The solution to the resulting optimization problem is known, because clearly the analytical solution consists of multiplying the nominals in Table 1 by 1000. Moreover, when choosing the model points portfolio in this way the corresponding value of the cost function is equal to zero.

As we can see in Fig. 5, DE reaches the value 7.2554×10^{-15} of the cost function, while the BH_M method with the L-BFGS-B gradient local optimizer reaches the value 1.2061×10^{-16}, the exact solution being equal to zero. Moreover, in Table 3 the obtained values for the number of contracts in the model points portfolio are shown. We note that these values are rounded to the eighth decimal digit, thus matching those corresponding to the exact solution. These roundings explain the small difference between the computed solution and the exact solution zero in the cost function. On the other hand, the computational time was around 3900 seconds both for DE and BH_M, as they were stopped when reaching a maximum number of cost function evaluations, which is the larger part of the computational cost in the computation.

By using this example with an analytical solution, we have checked whether the proposed technique is able to

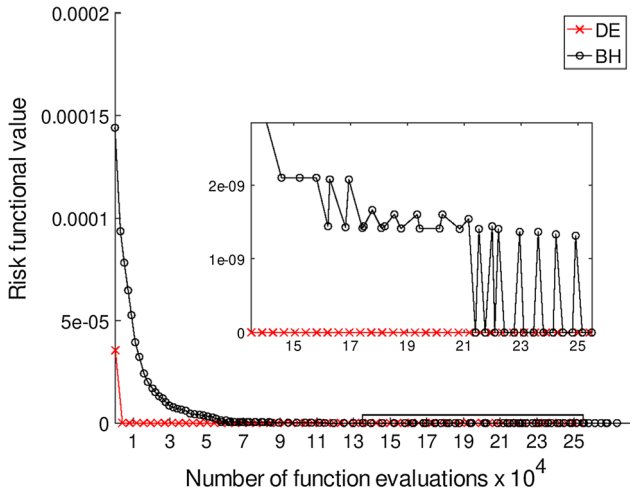

Fig. 5 Convergence of the DE and BH algorithms for Example 1. The box contains a zoom of the values for the last 100,000 evaluations

Table 3 Obtained solution in Example 1 with DE or BH_M (results are rounded to the sixth decimal place)

Age	Maturity	Nominals
20.0	50.0	50,000,000
25.0	45.0	100,000,000
30.0	40.0	150,000,000
35.0	35.0	200,000,000
40.0	30.0	250,000,000
45.0	25.0	300,000,000
50.0	20.0	350,000,000
55.0	15.0	400,000,000
60.0	10.0	450,000,000
65.0	5.0	500,000,000

classify repeated policies in their corresponding buckets, which is a desirable property of the risk function.

We would like to emphasize that in the optimization algorithms we have imposed very strict stopping criteria in the involved numerical methods to guarantee a very small error with respect to the analytical solution (high accuracy). This leads to long computational times, even though we use some parallel computing tools. Also, looking at the convergence graph in Fig. 5 we note that DE results are more efficient than BH_M, for this problem.

Moreover, in Table 4 we show the speedup in terms of the number of CPUs for DE and BH, which confirms the expected linear speedup in both methods, since they allow massive parallelization with one synchronization step per iteration, and very similar computational times for a fixed number of functional evaluations.

Table 4 Speedup for the DE and BH algorithms with respect to CPUs in Example 1

	DE		BH	
CPUs	Time (s)	Speedup	Time (s)	Speedup
1	15,624.2	1.00	15,840.7	1.00
2	7812.5	2.00	7992.3	1.98
4	3924.3	3.98	4032.1	3.93

4.3 Example 2: Real Scenario

In this example we present a realistic synthetic case. More precisely, in this test the original portfolio consists of 10,000 different policies, and we want to represent it with 5 different model points portfolios with 10, 20, 30, 40 and 45 model points, respectively. In the supplementary online information we include an Excel file that contains the data of the portfolio policies and another file showing the cash flows.

Each of the model points portfolios is given by a grid of ages and maturities. For example, the 45 model points portfolio is given by a grid of 9 ages and 5 maturities, with ages varying from 30 to 70 with step of 5 years, and maturities ranging from 5 to 25 with of step 5 years, thus accounting for that total of 45 model points.

This is a more difficult test than the analytical one in Example 1. As DE was more efficient than BH_M in the analytical example studied in Sect. 4.1, we decided to perform Example 2 only with the first optimization algorithm.

The required computing time by the DE algorithm was 4067.77 seconds for the 20 model points portfolio. The obtained final value of the model points risk functional is 8.56347×10^{-12} (see Fig. 6). Moreover, the computed nominals are shown in Table 5 and Fig. 7 (left). For the 45 model points portfolio, the computing time was 4897.69 seconds. The obtained final value of the model points risk functional is 4.39475×10^{-12}. Moreover, the computed nominals are shown in Table 6 and Fig. 7 (right).

The global component of the hybrid algorithm is of great importance for this example, which prevents from getting stuck at a local minimum, as it may happen with a local optimization method.

5 Conclusions

We have developed a computationally efficient methodology for building an equivalent model points portfolio starting from the bulk life insurance policies' portfolio. For

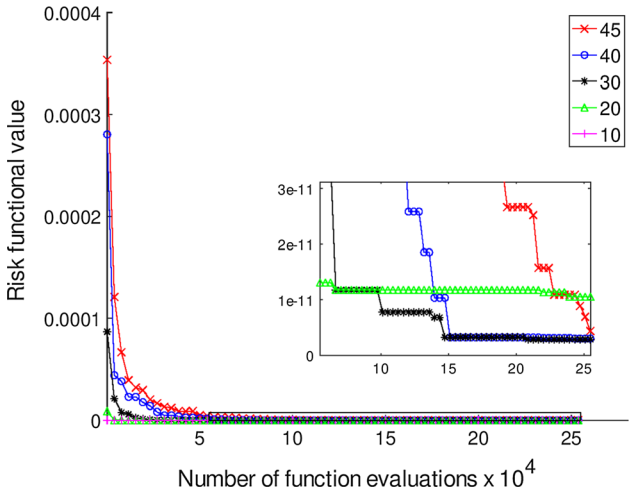

Fig. 6 Convergence of the DE algorithm for Example 2. The box contains a zoom of the values for the last 100,000 evaluations

Table 5 Obtained solution with DE in Example 2 with 20 model points

	Maturities			
	5	10	15	20
Ages				
30	0.2636054	0.5426405	0.0331077	0.3297138
35	0.2724662	0.2482779	0.0048624	0.2420600
40	0.0892925	0.0442299	0.0529424	0.0901912
45	0.3736702	0.1934909	0.0575758	0.0979045
50	0.6107136	0.0931458	0.0400420	0.1345385

Nominals are divided by 10^9

this purpose, the problem can be formulated as a global optimization problem with a very high computational cost, due to the size of the original portfolio. Furthermore, the optimization problem is of a high dimension and the risk

Table 6 Obtained solution with DE in Example 2 with 45 model points

	Maturities				
	5	10	15	20	25
Ages					
30	0.0955700	0.3256259	0.0883524	0.0229972	0.0043578
35	0.0592411	0.1198595	0.1226493	0.0509578	0.0158965
40	0.3659527	0.3177687	0.0591830	0.1009888	0.0162018
45	0.1035843	0.1218179	0.0357225	0.0739401	0.0347449
50	0.3763608	0.0764832	0.0586362	0.0175613	0.0299867
55	0.1298100	0.1005898	0.0164415	0.0095639	0.0161440
60	0.2330816	0.0265396	0.0243602	0.0412993	0.0234670
65	0.0052912	0.0352017	0.0493717	0.0072890	0.0022997
70	0.0450234	0.0624398	0.0820887	0.0696481	0.0114133

Nominals are divided by 10^9

functional to be minimized has to be evaluated a large number of times by means of Monte Carlo simulation techniques. Therefore, we propose and build up from scratch a multi-CPU implementation with a GPU accelerator for the functional evaluation. As illustrated with the numerical examples, we can obtain a high speed up in functional evaluations and the model points portfolio can be obtained in a reasonable computing time for real synthetic originally large policies portfolios. In this way, a relevant problem arising in the insurance sector can be solved by the efficient use of available HPC technologies.

Appendix 1: LIBOR Market Model

In this section, following Brigo and Mercurio (2006) we describe the risk-free dynamics of the discounted bond price, when considering the LIBOR Market Model governing the time evolution of the forward rates.

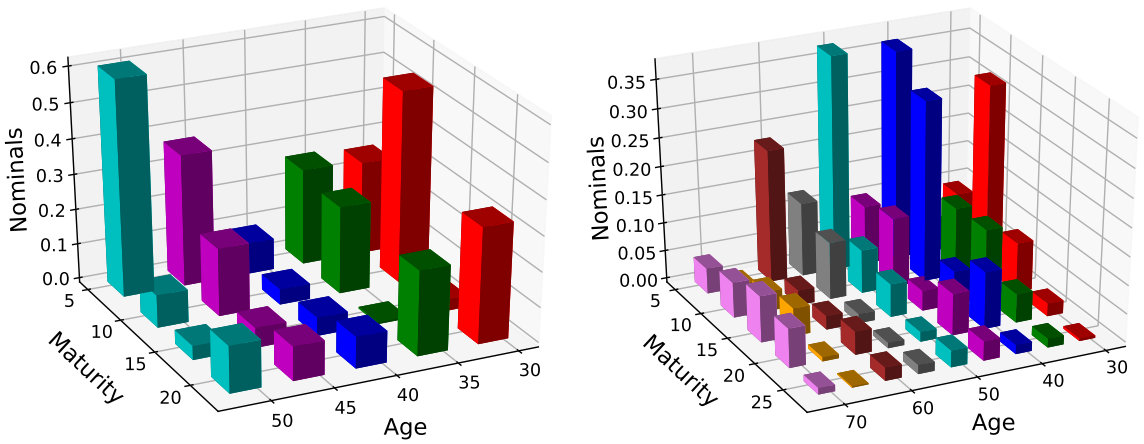

Fig. 7 Solution for Example 2 for 20 model points (left) and 45 model points (right)

Let N be a positive integer and hence define the finite set $\mathcal{T} = \{T_0, T_1, \ldots, T_N\}$ to be a fixed tenor structure, with $T_0 = 1$ and $T_0 < T_1 < \cdots < T_N$, so that T_n corresponds to a specific maturity time. For $n = 1, \ldots, N$, let $\tau_n = T_n - T_{n-1}$ be the corresponding accruals. Moreover, we set $\mathcal{I} = (0, 1)$ to be the unit interval on the real line, which corresponds to the period of one year.

We use $B_n(t)$ to denote the risk-neutral discounted price at time $t \in \mathcal{I}$ of a (zero-coupon) bond expiring at the tenor date T_n, for any $n = 0, \ldots, N$. Moreover, we denote by $F_n(t)$ the value at time $t \in \mathcal{I}$ of the LIBOR forward rate associated to the accrual period $(T_{n-1}, T_n]$, for $n = 1, \ldots, N$. Therefore, $F_n(t)$ satisfies the following condition:

$$B_n(t)(1 + F_n(t)\tau_n) = B_{n-1}(t).$$

Hence, for any $n = 1, \ldots, N$ we can write

$$B_n(t) = B_0(t) \prod_{k=1}^{n} \frac{1}{1 + F_k(t)\tau_k}, \qquad \text{for any } t \in \mathcal{I}.$$

It is worth to be highlighted that since $t < T_n$, for any $t \in \mathcal{I}$ and $n = 0, \ldots, N$, the price $B_n(t)$ is always well defined.

We will consider the price $B_0(t)$ of the bond expiring at the tenor date $T_0 = 1$, for $t \in \mathcal{I}$, as the reference numeraire process. Hence, we denote by $\mathbb{Q}$ the forward measure related to T_0, i.e., the martingale measure associated to the numeraire process $B_0(t)$, for $t \in \mathcal{I}$.

Next, we fix a N-dimensional Wiener process $W(t) = (W_1(t), \ldots, W_N(t))$, for $t \in \mathcal{I}$, defined on the suitable complete probability space $(\Omega, \mathscr{F}, \mathbb{Q})$, and we write $\varrho = (\varrho_{nk})_{nk}$ to denote the corresponding (positive defined) correlation matrix, i.e.,

$$dW_n(t)dW_k(t) = \varrho_{nk}dt.$$

In particular, we shall assume constant correlation coefficients given by the usual parameterization:

$$\varrho_{nk} = \exp(-\beta \mid T_n - T_k \mid),$$

with $\beta = 0.01$ in the numerical examples. Note that these coefficients will correspond to the correlation between LIBOR forward rates.

For any given $h = (h_1, \ldots, h_N) \in \mathbb{R}$, we define the following norm:

$$\|h\|_W = \left\{ \sum_{n,k=1}^{N} \varrho_{nk} h_n h_k \right\}^{1/2}.$$

For each $n = 1, \ldots, N$, let $\sigma_n(t)$, for $t \in \mathcal{I}$, be a given deterministic function, representing the volatility of F_n. According to the LIBOR Market Model, given any fixed $n = 1, \ldots, N$, the corresponding forward rate F_n is a martingale with respect the risk-neutral measure induced by the numeraire B_n. When we consider the same measure $\mathbb{Q}$ (associated to the numeraire B_0) to write the dynamics of all F_n, then Girsanov theorem implies that the risk-neutral dynamics of the process $F_n(t)$, for $t \in \mathcal{I}$ is given by

$$dF_n(t) = \mu_n(t)dt + \sigma_n(t)F_n(t)dW_n(t), \tag{7}$$

jointly with some given initial condition $F_n(0)$, where, at any time $t \in \mathcal{I}$, the drift component $\mu_n(t)$ is completely determined by following identity:

$$\mu_n(t) = \sigma_n(t)F_n(t) \sum_{k=1}^{n} \frac{\varrho_{nk}\tau_k\sigma_k(t)F_k(t)}{1 + F_k(t)\tau_k}.$$

Concerning the modeling of volatilities, in this article we choose the widely used parameterization:

$$\sigma_n(t) = [a + b(T_n - t)] \exp[(T_n - t)] + d$$

Furthermore, in the numerical examples we have chosen the constant parameters: $a = 0.07$, $b = 0.2$, $c = 0.6$ and $d = 0.075$.

For any fixed $t \in \mathcal{I}$, set $\mu(t) = (\mu_1(t), \ldots, \mu_N(t))$ and thus define $\Sigma(t)$ to be the matrix whose components are given by

$$\Sigma_{nk}(t) = \sigma_n(t)F_n(t)\delta_{nk}, \quad \text{for any } n, k = 1, \ldots, N, \tag{8}$$

where δ_{nk} denotes the Kronecker delta. Moreover, we shall write $\Sigma_n(t)$ to denote the nth row of the matrix $\Sigma(t)$, for any $n = 1, \ldots, N$. Then, when setting $F(t) = (F_1(t), \ldots, F_N(t))$, we may regard (7) as a N-dimensional dynamics by means of the following compact form notation:

$$dF(t) = \mu(t)dt + \Sigma(t)dW(t), \tag{9}$$

jointly with the initial condition $F(0) = (F_1(0), \ldots, F_N(0))$.

Concerning the tenor structure of the LIBOR model, in all the article we consider 100 tenors, with maturities ranging from 1 to 100 and initial rates given by $F_1(0) = 0.01$, $F_2(0) = 0.02$, $F_3(0) = 0.03$, $F_4(0) = 0.04$ and $F_n(0) = 0.05$, for $n \geq 5$.

Moreover, for any $n = 1, \ldots, N$ we shall write

$$\tilde{B}_n(t) = \frac{B_n(t)}{B_0(t)}, \quad \text{for any } t \in \mathcal{I},$$

to denote the discounted price processes associated to the bond expiring at the tenor date T_n.

The following result provides the risk-free dynamics for the discounted price of any bond expiring at some tenor date in $\mathcal{T}$. For any $n = 1, \ldots, N$, the discounted bond price process $\tilde{B}_n(t)$ admits the dynamics

$$d\tilde{B}_n(t) = -\varepsilon_n(t)\tilde{B}_n(t)dW(t), \tag{10}$$

where we set

$$\varepsilon_n(t) = \sum_{k=1}^{n} \frac{\tau_k}{1 + F_k(t)\tau_k} \Sigma_k(t). \tag{11}$$

Appendix 2: Biometric Survival Model

$S(x_i, T_n)$ is the survival index which is understood as the proportion of those individuals labelled by $x_i \in \mathcal{X}$ that survive to the age $x_i + T_n$. The survival index can be computed using past survival tables or from a model.

In our case we use the model:

$$S(x_i, T_n) = \exp\left\{ -\int_1^{T_n} \mu(s, x_i + s) ds \right\}, \quad \text{for any } x_i \in \mathcal{X} \text{ and } n = 1, \ldots, N, \tag{12}$$

which yields the proportion of those individuals with age x_i that survive to the age $x_i + T_n$, and where $\mu(s, x_i + s)$ denotes the force of mortality at time $s \geq 0$ related to the class of individuals labelled by $x_i \in \mathcal{X}$. In this respect, we assume that $\mu(s, x_i + s)$ is a deterministic observable function, for any $x_i \in \mathcal{X}$ and $s \geq 0$.

In particular, we consider a Gompertz-type law modeling the force of mortality (Gompertz 1825), by setting

$$\mu(s, x_i + s) = a(s) \exp\{(x_i + s) b(s)\}, \quad \text{for any } s \geq 1 \text{ and } i = 1, \ldots, I,$$

where $a(s)$ and $b(s)$ are deterministic functions for $s \geq 1$, which are considered to be observables. Throughout, we write S_T to denote the derivative of S in its second variable, which is given by

$$S_T(x_i, T_n) = -S(x_i, T_n)\mu(T_n, x_i + T_n).$$

Concerning the force of mortality, we consider the Gompertz type law modeling with constant parameters, i.e., $\mu(x) = a \exp(bx)$, where we will take $a = 0.0003$ and $b = 0.06$.

Appendix 3: Code Listings

In this section, we include the code snippets, illustrating the GPU parallelization.

Listing 1 Code for the risk functional evaluation, with the calls to the corresponding kernels.

```
double LMM::risk_functional(double *nom){

  kernel_init_rand<<< , >>>(seed,state);

  mp_rep.Set_Nom(nom);

        kernel_libor<<< , >>>();
        kernel_functional<<< , >>>();

        cudaMemcpy();

        double Integral = discounted_average();

  return Integral;
}
```

Listing 2 CUDA code with the kernel for the LIBOR interest rate scenario simulator.

```
__global__ void kernel_libor()
{
    int tid = threadIdx.x + blockIdx.x*blockDim.x;
    int index = threadIdx.x;

        for(int step=1;step<dimTimeLine;step++)
    {
      BrownianIncrements();
      for(int rate=0;rate<dimTenor;rate++)
      {
        ComputeVol();
        ComputeDrift();
        LogEulerScheme();
      }
    }
}
```

Listing 3 CUDA code for the kernel for the risk functional evaluation.

```
template<int firstTime>
__global__ void kernel_functional(){
        int tid = threadIdx.x + blockIdx.x*blockDim.x;
        int index = threadIdx.x;

  double Integral = 0.;
  for(int j=0;j<dimTimeLine;j++)
  {
    for(int i=0;i<dimTenor;i++)
    {
      double dif = OriginalPortfolio() - ReducedPortfolio();
      for(int k=0;k<=i;k++){
        p[index(k,tid,NumPath)] -= DiffusiveComponent(dif);
      }
    }
    Integral += vt_Mat_Vt(dimTenor,
                          NumPath,
                          d_CORR,
                          &p[index(0,tid,NumPath)])*(1.-tj);
  }
  return Integral;
}
```

References

Antonov A, Konikov M, Spector M (2015) The free boundary SABR: natural extension to negative rates. Social Science Research Network (SSRN)

Brigo D, Mercurio F (2006) Interest rate models—theory and practice with smile, inflation and credit. Springer, Berlin

Byrd RH, Lu P, Nocedal J, Zhu C (1995) A limited memory algorithm for bound constrained optimization. SIAM J Sci Comput 16:1190–1208

Casella A, Falco ID, Della Cioppa A, Scafuri U, Tarantino E (2018) Exploiting multi-core and GPU hardware to speed up the registration of range images by means of differential evolution. J Parallel Distrib Comput

Corlosquet-Habart M, Gehin W, Janssen J, Manca R (2015) Asset liability management for banks and insurance companies. Wiley, Hoboken

Corsaro S, Angelis PD, Marino Z, Perla F, Zanetti P (2010) On parallel asset-liability management in life insurance: a forward risk-neutral approach. Parallel Comput 36:390–402

Denuit M, Trufin J (2015) Model points and Tail-VaR in life insurance. Insur Math Econ 64:268–272

Dutra-Lopes S, Vázquez C (2019) Real world scenarios with negative interest rates based on the LIBOR Market Model. Appl Math Financ 25:466–482

EIOPA2010 (2010) Quantitative impact studies V: technical specifications. Technical Report, European Comission, Brussels

Fernández JL, Ferreiro AM, García-Rodríguez JA, Vázquez C (2018) GPU parallel implementation for asset-liability management in insurance companies. J Comput Sci 24:232–254

Ferreiro AM, García-Rodríguez JA, López-Salas J, Vázquez C (2013) An efficient implementation of parallel simulated annealing algorithm in GPUs. J Glob Optim 57:863–890

Ferreiro AM, García-Rodríguez JA, López-Salas JG, Vázquez C (2014) SABR/LIBOR market models: pricing and calibration for some interest rate derivatives. Appl Math Comput 242:65–89

Ferreiro AM, García-Rodríguez J, Souto L, Vázquez C (2019a) Basin hopping with synched multi L-BFGS local searches. Parallel implementation in multi-CPU and GPUs. Appl Math Comput 356:282–298

Ferreiro AM, García-Rodríguez JA, Vázquez C, Costa e Silva E, Correia A (2019b) GPU parallelization of two-phase optimization algorithms. Math Comput Simul 156:67–90

Ferri E (2019) Optimal model points portfolio in life insurance. arXiv:1808.00866

Gerstner T, Griebel M, Holtz M, Goschnick R, Haep M (2008) A general asset-liability management model for the efficient simulation of portfolios of life insurance policies. Insur Math Econ 42(2):704–716

GGY-Axis (2019). https://www.ggy.com/. Accessed 15 Feb 2019

Goffard LO, Guerrault X (2015) Is it optimal to group policyholders by age, gender, and seniority for bel computations based on model points? Eur Actuar J 5:165–180

Gompertz B (1825) On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies, in a letter to Francis Baily, Esq. FRS & c. Philos Trans R Soc Lond 115:513–583

Jalen L, Mamon R (2009) Valuation of contingent claims with mortality and interest rate risks. Math Comput Model 49:1893–1904

Lee A, Yau C, Giles MB, Doucet A, Holmes CC (2012) On the utility of graphics cards to perform massively parallel simulation of advanced monte carlo methods. J Comput Graph Stat 19:769–789

Leitao A, Oosterlee C (2017) Modern Monte Carlo methods and GPU computing. In: Novel methods in computational finance. Mathematics in industry, vol 25, Springer, Heidelberg, pp 627–637

Library S (2019). http://www.scipy.org. Accessed 15 Feb 2019

Liu DC, Nocedal J (1989) On the limited memory method for large scale optimization. Math Program B 45:503–528

McCarty SL, McGuire ML (2018) Parallel Monotonic Basin Hopping for low thrust trajectory optimization. In: Conference paper, AIAA SciTech Forum. https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20180004586.pdf. Accessed 15 Feb 2019

MG-ALFA (2019). http://www.milliman.com/mg-ala/. Accessed 15 Feb 2019

Sandström A (2010) Handbook of Solvency for actuaries and risk managers: theory and practice. Chapman and Hall/CRC, London

Schmeiser H, Wagner J (2014) A proposal on how the regulator should set minimum interest rate guarantees in participating life insurance contracts. J Risk Insur 82:659–686

Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 11:341–359

Tasoulis D, Pavlidis N, Plagianakos V, Vrahatis M (2004) Parallel differential evolution. In: IEEE congress on evolutionary computation, Portland, Oregon, vol 2, pp 2023–2029

Wales DJ, Doye JPK (1997) Global optimization by Basin-Hopping and the lowest energy structures of Lennard–Jones clusters containing up to 110 atoms. J Phys Chem A 101:5111–5116

Wuthrich MV, Merz M (2013) Financial modeling, actuarial valuation and solvency in insurance. Springer, Heidelberg

Zhu W (2011) Massively parallel differential evolution-pattern search optimization with graphics hardware acceleration: an investigation on bound constrained optimization problems. J Glob Optim 50:417–437