Fast Calculations of Portfolio Credit Losses and sensitivities

Interim Literature Review

Arvind Nayak

Student number:	5374707	
Project duration:	November 2020 -	
Supervisors:	Dr. ir. Fang Fang,	ING
	Dr. ir. Xiaoyu Shen,	ING
	Prof. dr. ir. Cornelis Vuik,	TU Delft
	Prof. dr. B.D. (Drona) Kandhai,	ING

ING

TU Delft

Abstract

Computing portfolio credit losses and associated risk sensitivities is crucial for the financial industry to help guard against unexpected events. Quantitative models play an instrumental role to this end. As a direct consequence of their probabilistic nature, portfolio losses are usually simulated using Monte Carlo copula models, which in turn play a decisive role in their measurement of risk metrics such as the Value-at-Risk (VaR). Semi-analytical numerical methods are alternatives to the Monte Carlo simulations to compute the distribution of the portfolio credit losses, the VaR and the VaR sensitivities. We find that numerical approaches such as the COS method, based on a Fourier cosine series expansion are superior to the Monte Carlo based computations in terms of both, the computational speed and the accuracy. [29] have demonstrated these results, using the examples of Gaussian and Gaussian-T copula models on the CPU. In this study, we extend that scope and critically examine two modelling approaches for improving the computing efficiency of the COS method in multifactor models. First, we investigate and validate, a parallel algorithm for the COS method. Additionally, we study the effect of dimensionality reduction techniques to approximate the factor copula models, in their own light. In this process, we demonstrate the suitability of COS algorithm for parallelization on the GPU and highlight the performance improvements over existing methods of an optimal algorithm that we develop using the above techniques.

Contents

1 Background

Financial institutions are engaged in borrowing and lending of monetary products, which means they face two principal risks, namely market risk and credit risk. While market risk is associated with unexpected changes in the prices of market ingredients, credit risk corresponds to losses resulting from the failure of an obligor to make a payment. The source of risk for a bank from credit risk is so significant that market risk is analogised to be *'a pimple on the back of the elephant that is credit risk'*.[18] In order to reduce exposure, banks are involved in trading activities of such credit risky instruments. Thus, decision makers seek for methods and technologies that are practically useful for managing their credit portfolios and for enhancing their profits from trading these instruments.In the wake of the Global Financial Crisis of 2007-08, financial authorities have become committed to improve the existing directives for risk management processes and the related capital buffers. This is linked to the banks' obligation to protect against unexpected losses that may arise from widespread defaults throughout their portfolios. The Basel Committee on Banking Supervision (BCBS from now on) updated large parts of the regulations for financial markets. The biggest change in decades for market risk, is the Fundamental Review of the Trading Book (FRTB), to be in effect from 2022, but has already pushed banks to be compliant with it [cf. 4, 26].The main goal of the FRTB is to put appropriate capital charges on risks in the trading book. Previous regulation gave the opportunity to gain regulatory arbitrage by shifting credit related products from the banking book to the trading book and vice versa. The FRTB requires a different treatment of credit, and a sharper defined boundary between the trading and banking books. The FRTB regulation has not only consequences for capital calculation, but as well effects the granularity of reporting. Previously, reporting took place at company-level, under FRTB it has to be done at trading-desk level. A milestone example is represented by the competition between the Value at Risk (VaR) and Expected Shortfall (ES). The VaR was adopted as the official market risk measure in 1996 by BCBS, [see 31]. Then,the academic community pointed out its drawbacks, such as the lack of sub-additivity property.[21]

Historically, the Asymptotic Single Factor (ASRF) model described by Gordy [13] has been used for determining capital charges for credit risk. This single factor model has been built on the foundations of the work by Merton [25] and Vasicek [35], and by modifying the model it can also be applied to default risk in trading portfolios. The use of factor models is a popular tool to model correlations in large portfolios. The work of Vasicek [35] led to the widespread use of the Large Homogeneous Pool (LHP) model, which at first was used to model defaults in loan portfolios. In the LHP model lies the implicit assumption that defaults are correlated through a bivariate Gaussian copula. The use of the Gaussian copula method became conventional after the publication of the work of Li [23]. Research objective, questions and model xthe correlation of defaults. This led to a widespread application of the Gaussian copula in the world of finance. However, after the 2008 global financial crisis the bivariate Gaussian copula received heavy criticism. The main critique was on the lack of tail dependence implied by the Gaussian copula. Other copulas like the Student-t copula and the Clayton copula exist, which are known to imply fatter tails. Burtschell et al compare some of the factor copula models used in the context of CDO pricing. The BCBS, 2016a [cf 4, Pg.20] does give financial institutions the freedom of developing their own default models, as long as they are compliant to the BCBS' requirements for internal models on market risk. This in combination with the global introduction of FRTB regulation brings momentum to (re-)develop default models.

The BCBS also sets new regulations on the calculation of capital resulting from the risk of default: the Default Risk Charge (DRC). The size of the capital buffer is intended to reflect the level of credit quality that the bank affords its clients. It is determined in terms of a confidence interval, up to which the bank claims it can absorb losses and remain solvent. As a result highly rated banks maintain large capital buffers to justify their rating. DRC under the internal model approach of FRTB is defined as the 99.9% Value-at-Risk (VaR) of the total loss distribution of the trading book positions that are subject to issuer risk. A review of the entire FRTB regulation for the DRC can be found in [5]. The recognized industry standard for measuring a portfolio's credit risk is the Monte Carlo (MC) simulation method. However, it is worth noting that a very high quantile level, 99.9% for DRC, MC computed empirical quantile can suffer from high variance estimation error. A huge number of heavy simulations are then required to get convergence. This implies a time intensive task which can be difficult to set up, especially for a large trading portfolio. The volume of banks data calculation is increasing each year with scale and complexity of different financial products. Subsequently, there is a need for fast and more importantly, accurate numerical methods to reliably price and quantify risks in the banks' portfolios.

In [29], MC simulation is replaced by a semi-analytical method to compute the distribution of the portfolio credit losses, the VaR, and the VaR sensitivities in factor copula models. In essence, it directly recovers the cumulative distribution function of the portfolio loss via the COS method, a method based on the Fourier-cosine series expansion. Fourier coefficients are extracted from the characteristic function of the portfolio loss which can be pre-calculated by means of numerical integration. Both the computational speed and the accuracy of this method are observed to be superior to MC simulation. To further improve the computing efficiency, this thesis will focus in two directions. Firstly, we aim to design an efficient parallel algorithm for the computation of characteristic function and to compare the performances. Moreover, we consider an interesting exercise to approximate the factor copulas, which are in practice of high dimensionality by lower dimensional models using dimensional reduction techniques such as Principal Component Analysis (PCA).
In chapter 2, we lay the groundwork of formal terms and definitions in portfolio credit risk management along with its associated risk measures and attributions. Furthermore, we explain the regulatory framework of the Default Risk Charge introduced by BCBS, that banks adhere to and the guidelines within which we create our models. Finally, a brief overview of High performance computing relevant in the case of numerical simulations is explained. Chapter 3 gives the mathematical representation of different Credit Risk models, i.e. the mathematical definitions of portfolio default loss, measures under different multifactor copulas. We review the COS method and show how risk allocations ES and VaR are derived from it. Chapter 4, will list some of our numerical experiments on two different portfolios, starting from a simple test case to a more practical example. The computation speed and accuracy will be tested and compared to the 'workhorse' MC. Through these examples, we expect that the associated challenges, specifically the computational complexity due to *"curse of dimensionality"* will become apparent with the existing Monte Carlo and COS methods. This will help motivate our discussions on the optimized implementations of these algorithms. We discuss the feasibility of various other improvement strategies in Chapter 5 and finally conclude by formulating our research aims.

2 Preliminaries

2.1. Credit portfolio losses

The primary objective of Credit Risk Management is to improve the ability of banks to identify optimal credit portfolios. It is thus, important for financial institutions to possess tools for pricing and managing their trading portfolios. Let us now see how this can be quantified. On an obligor level, the three basic components of credit risk are:

- exposure at default (ϵ_n), the total amount that the bank is exposed at the time of default of the n-th obligor. The bank can calculate ϵ_n at any specified time.
- loss given default (υ_n), this factor quantifies the losses incurred by the bank when there is a default of the n-th obligor. υ_n is calculated separately from the ϵ_n because the bank is unlikely to lose all the value that they may be exposed to. There can be some recovery from each obligor based on a certain Recovery rate (R_n). So the loss given default then becomes,

  $$\upsilon_n = A_n(1 - R_n) \tag{2.1}$$

  where A_n is the notional amount of the n-th obligor.
- probability of default of the n-th obligor (PD_n) within a fixed time horizon.

ϵ_n is usually assumed to be deterministic. υ_n is a random variable and its uncertainty arises from PD_n. Suppose the random variable x_n represents the creditworthiness of the n-th obligor. The n-th obligor is said to have defaulted if x_n falls below a certain default threshold $\xi_n(T)$, where T can be taken as fixed time horizon for bond maturity. $\xi_n(T)$ is based on the Merton's structural model of default [25]. Suppose the asset value of the borrower V follows a geometric Brownian motion with initial value V_0, drift μ and volatility σ, so that $dV_t = \mu V_t dt + \sigma V_t dx_t$, where x_t is a Brownian motion process representing the creditworthiness of an obligor. The asset value at horizon T can be given as

$$V_T = V_0 exp(V_0 + \mu T - \frac{1}{2}\sigma^2 T + \sigma\sqrt{T}x_T)$$

. The obligor defaults if $V_T < B$. Hence $\xi_n(T)$ is formulated as,

$$\xi_n(T) = \frac{\ln B_n - \ln V_n - \mu_n T + 0.5\sigma_n^2 T}{\sigma_n\sqrt{T}}$$

The probability of default for obligor n at time T can then be formalized as

$$PD_n(T) = P(\tau_n \leq T) = P(x_n \leq \xi_n(T)) \tag{2.2}$$

Let us now consider a credit portfolio containing N obligors. We consider the risk/loss of the $n-th$ obligor as a random variable $L_n\colon \Omega \to \mathbb{R}$, where we work on a probability space $(\Omega, \mathcal{F}, \mathbb{P})$. The dependence on T can be suppressed for simplicity, since typically T is chosen to be one unit. The loss incurred by obligor n is given by,

$$L_n = \epsilon_n \upsilon_n \mathbb{1}_{x_n \leq \xi_n(T)} \tag{2.3}$$

It follows that the portfolio loss is given by

$$L = \sum_{n=1}^{N} L_n = \sum_{n=1}^{N} \epsilon_n \upsilon_n \mathbb{1}_{x_n \leq \xi_n(T)} \tag{2.4}$$

where $\mathbb{1}_A$ is the indicator function which equals 1 when A is true and zero otherwise.

2.2. Quantitative Risk Measures

In order to manage credit risk it is imperative to quantify credit risk on a portfolio level. Financial institutions calculate the required capital to shield against possible extreme losses and hence can determine whether they are adequately compensated for the risk incurred. These are often measured by Value at Risk (VaR) which is the quantile of the portfolio loss distribution at a given confidence level. Aggregation of credit risk from individual obligors to a portfolio level involves specification of the correlations among obligors. Factor models are utilized in common practice , in which the obligors itself are independent but are dependent on some common factors, e.g systemic risks.(c.f Chapter 3) We discuss this at length in Chapter 3. VaR for a given confidence level α is the α-quantile of the portfolio loss distribution L and therefore is given by,

$$VaR_\alpha = \inf\{x : P(L \leq x) \geq \alpha\} \tag{2.5}$$

VaR has faced criticism to not being *statistically coherent*[cf. 1]. In particular VaR does not satisfy subadditivity,i.e., the VaR of a portfolio can be larger than the sum of the VaRs of its subportfolios.

2.2.1. Risk Allocation

After computing the overall risk distribution of a portfolio, an equally important task as to quantify the portfolio level credit risk is to measure how much each obligor in a portfolio contributes to the total risk, i.e., the risk contributions of single exposures. The Euler principle of risk allocation has been shown (e.g. in [8]) to be the only compatible principle to the subadditivity of a risk measure. First, the Euler decomposition is homogeneous, meaning that the risk attribution of a portfolio scales in proportion to the size of the portfolio. Second, the decomposed risk across obligors or sub-portfolios sums up to the risk measure of the portfolio.
Decomposition of ES or VaR boils down to solving conditional expectations of losses of obligors or sub-portfolios, conditioned on the tail event that characterizes the risk measure. We consider the following definition for allocating ES by the Euler principle:

$$CES_n = \mathbb{E}\left[\mathbb{1}_{x_n} \leq \xi_n L_n | L \geq VaR_\alpha\right] \tag{2.6}$$

Both ES and CES_n scale in proportion to the size of the portfolio.

$$ES = \mathbb{E}\left[\sum_n \mathbb{1}_{x_n} \leq \xi_n L_n | L \geq VaR_\alpha\right] = \sum_n CES_n \tag{2.7}$$

Similar to Conditional ES, Conditional VaR decomposes VaR by the Euler principle for risk allocation. Consider the definition,

$$CVaR_\alpha = \mathbb{E}\left[\mathbb{1}_{x_n \leq \xi_n} \cdot L_n | L = VaR_\alpha\right] \tag{2.8}$$

such that

$$VaR = \mathbb{E}\left[\sum_n \mathbb{1}_{x_n \leq \xi_n} \cdot L_n | L = VaR_\alpha\right] = \sum_n CVaR_n \tag{2.9}$$

2.3. Regulatory Framework

In this section we very briefly describe the Default Risk Charge (DRC) regulation introduced by the Fundamental Review of the Trading Book (FRTB). In January 2016, BCBS revised the market risk framework in FRTB [4]. Detailed regulations regarding DRC under IMA (Internal Model Approach) are given in paragraph 186 of [4]. The FRTB in general aims to minimise regulatory arbitrage, improve both the standardised approach and internal model approach, introduce a more granular framework, and increase transparency. The DRC "*captures default risk of credit and equity trading book exposures with no diversification effects allowed with other market risks*" (BCBS, 2016a). As stated, banks could both use the standardised approach as an internal models approach. Compared to the currently in use Incremental Risk Charge (IRC) framework, the main change of the DRC is that credit migrations are not taken into account anymore. This ensures that the variability of the VaR is reduced under the new default risk measure. The standardised Default Risk Charge is calibrated to the credit risk treatment in the banking book. This reduces the potential discrepancies in capital requirements for similar risk exposures in the banking and trading books. The model is based on a large number of Monte Carlo simulations of portfolio default loss scenarios, and the DRC figure corresponds to a VaR at 99.9% confidence level.

The design of the DRC model, we describe here is infact an interpretation of the DRC framework as published in BCBS instructions [4]. The model has been refined based on several consultative documents subsequently issued by the Basel committee. As a consequence of the need to increase the number of risk factors for DRC, a multi-factor Gaussian copula model was developed. This is basically an extension of the onefactor Gaussian copula model currently adopted for IRC. To estimate the DRC, we first need to define a set of factors that represent the geographical regions and sectors of the economy. For each of these regional and sectoral factors an index is chosen that encodes their performance. The correlations between time series corresponding to these factors are then measured.

We will now briefly take a look at the procedure of calculating a total financial outcome of one scenario. First, for each sector and each region, a standard normal random number is generated. We store these generated values and start processing each issuer. For each issuer we generate one more standard normal that represents its idiosyncratic contribution, then find the region and sector that matches this particular issuer and select the corresponding random numbers. Then, all three random numbers xescribing thex issuer (one for the sector, one representing the creditworthiness of an obligor for the region and one for the idiosyncratic) are correlated. We thus get the credit-change index of the issuer we are currently evaluating. Based on matching a threshold obtained from the PD_n of issuer and L_n we find whether the financial impact of the default of the issuer should be added to a total portfolio loss L. In cases were no default occurs the contribution of issuer is set to zero. By going through all the issuers and all the scenarios, the vector with the cumulative financial impact across all issuers per scenario, L, is created and subsequently sorted. From this vector a 99.9% quantile is taken, which is the DRC value. Note that the scope of the DRC model excludes the migration charge, thus no rating transition events occur.

2.4. High Performance Computing

In the context of financial calculations, the Monte Carlo method has a relatively slow rate of convergence, and thus becomes a trade off between number of simulations and the required error tolerance. A number of optimizations exist to increase the rate of convergence, some of them being improved numerical algorithms such as the COS method which we later discuss. The purpose of this section is to introduce the reader to High Performance Computing as an alternative. This type of optimization alongside the COS method forms one of the focuses of this work. Eventually, we would like to explore the advantages of both higher fidelity and fast algorithms.

An upgraded hardware may result in a higher CPU rate for instance or the use of advanced compiler can result in better code optimization. A less straightforward but promising approach consists of parallelizing the application. If utilized to a certain potential, parallelization can ensure higher acceleration. It may be that the code be required to be refactored, (e.g. CUDA, openCL). In other cases, few added lines could be enough to enable a parallel execution of a portion of a code, in general for/while-loops (e.g. OpenMP, OpenACC). Nonetheless, parallelizing with the introduced methods requires a know-how of our used hardware and the specifics of our application.

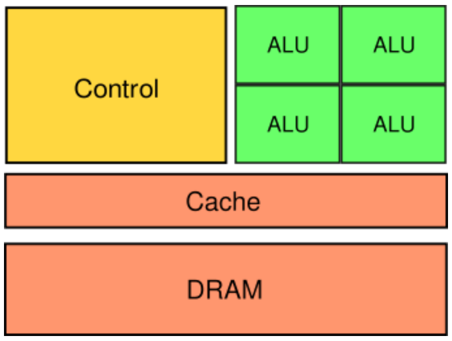


Figure 2.1: Shared Memory Architecture

High performance systems can be classified following the memory architecture. The two main architectures are the shared-memory and the distributed-memory systems. For the shared-memory architecture, a set of processors shares the same memory contingent. Current popular architectures such as multi-core CPUs belong to this group. OpenMP can handle parallelization tasks in such systems. With the help of few simple compiler directives (`#pragmas`) surrounding sequential for-loops can divide automatically the work between available cores and every core processes a part of the loop. The communication among processors is very simple since they all share the same memory contingent. The maximum available number of cores and the memory capacities for this system are, however, too low to cover large-scale problems.

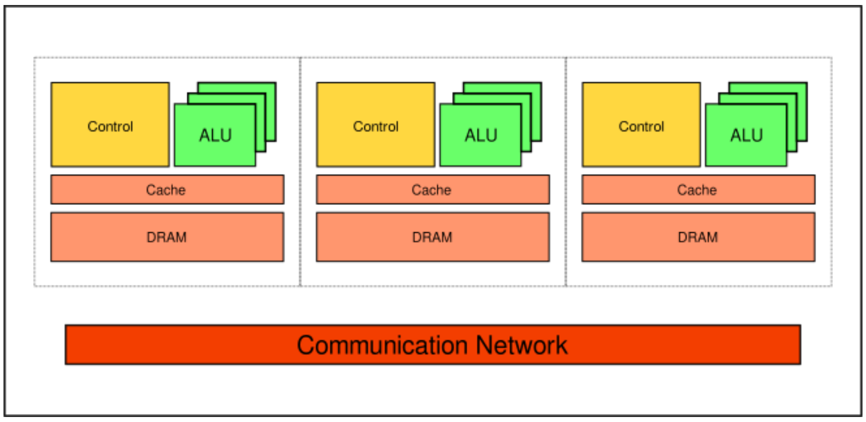


Figure 2.2: Distributed Memory Architecture

In the distributed-memory configuration, better known as clusters, every processor has its own memory. The communication between processors occurs through the Message Passing Interface (MPI). A decomposition of the computational domain is essential for the parallelization on distributed-memory systems. Every processor contributes to the solution of the simulation by solving a part of the computational domain. A high number of cores (e.g. cluster of CPUs) could speed up the whole process significantly. But the parallelization increases also the programming burden, since the designer has to distribute the computational work among the available CPU processors and regulate the communication. For a realistic application, a large number of cores is essential and MPI is the most implemented paradigm on today HPC systems. Hybrid systems, consisting of a cluster of shared-memory systems, are getting more and more popular. The most powerful HPC systems nowadays are hybrid, where the first benchmark measurement of exaflops performance was made [refer 32]. Hybrid systems use in every cluster node not only standard CPUs but also accelerators such as Field Programmable Gate Arrays (FPGAs) and Graphics-Processing Units (GPUs).

2.4.1. GPU vs. CPU

Before investigating the details of GPU computing, it is perhaps worthwhile to discuss the drivers behind the recent shift towards GPU computing specifically as well as towards massive parallelism in

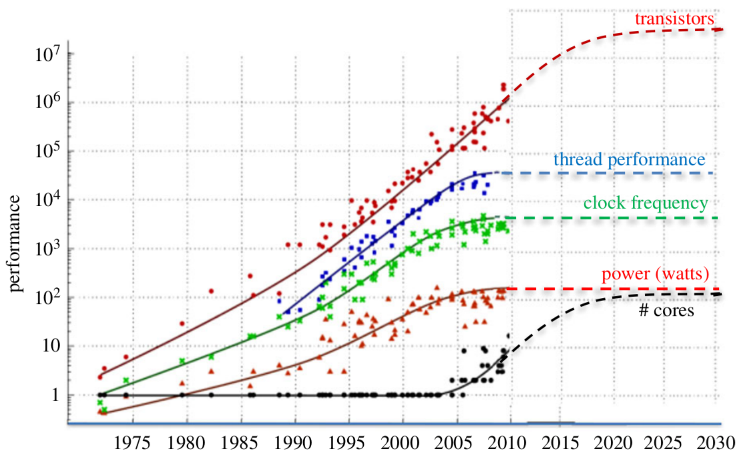


Figure 2.3: Different characteristic trends for general purpose processor technologies

Source: Figure from Kunle Olukotun, Lance Hammond, Herb Sutter, Mark Horowitz extended by John Shalf [see 28]. (Online version in colour.)

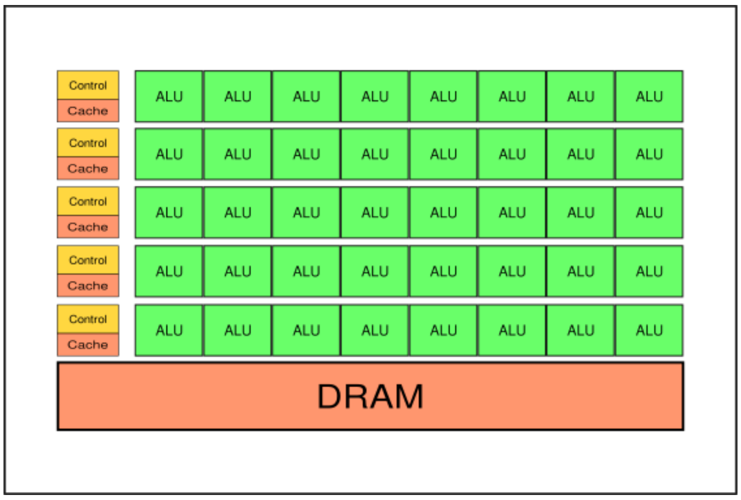


Figure 2.4: Massively parallel architecture (GPU)

general. Moore's Law and Dennard Scaling have guided the development of the semiconductor industry. Traditionally, the performance gains realised in computing have been attributed to increasing amounts of transistors on integrated circuits (ICs) as well as the frequency with which these transistors could switch. Hence, Moore's law states that the number of transistors doubles every 1.5-2 years, and thus increasing the logic and computing power has been true since its inception circa 1965. Dennard scaling described that the energy consumption of a chip would stay in proportion to the size of the chip. Shrinking transistor sizes (i.e., reducing the process size) allowed us to increase the computing capabilities of the device without consuming more energy. Moore's Law and Dennard Scaling charted a promising future for computers. As we reduce the transistor size, the performance of computing devices would exponentially increase over time. But it has been predicted that within a decade (see Figure 2.3), the technological underpinnings for the process that Gordon Moore described will come to an end, as lithography gets down to atomic scale.[28] As the transistor sizes is approaching the practical limitations of physics, reducing transistor sizes can hardly improve performance. To guide future semiconductor research, we need to revisit these historical development trends and find new directions that can drive performance improvement. GPUs offer a way to continue accelerating applications as it has been witnessed already in fields such as graphics and supercomputing[see 15, chap 4].
GPUs can be considered as a shared-memory system since a number of multi-processors are connected to the same device memory. Notice the architectural differences between a traditional CPU

and a GPU as depicted in Figures 2.1 & 2.4. A typical GPU contains hundreds of cores almost entirely dedicated to single and double-precision arithmetic operations, called Arithmetic Logic Units (ALUs). Compared to a CPU core, a GPU core is less powerful. The conventional multi-core CPU relies mainly on a large and fast memory cache, which serves a small number of cores, in order to reduce the instructions execution time (latency) to a minimum. A significant portion of the transistors is dedicated to the instruction flow control (instruction pipelining, branch prediction and similar tasks). The CPU cache has a spacial and a temporal locality as data is more likely to be kept in the cache if it is often used (temporal) and if it is close to an often used data (spatial). For a CPU, the computational power is provided by a high-speed processor optimized for serial operations and low latency. A GPU includes, however, a large number of these cores, which combined result in a higher computational power. Hence, the second advantage is the specialization of the GPU cores. The GPU cache is much smaller and disposes of spatial locality. The memory bandwidth in CPUs can be hoarded by requirements from the operating system and I/O devices, which can make it more difficult for it to increase. This is reflected in a performance gap between CPUs and GPUs in in terms of arithmetic throughput (FLOPS: FLoating OPerations per Second) and memory bandwidth (amount of data transferred per second). [see 19, p.10]

2.4.2. GPU Computing

Today's GPUs evolved from graphics cards installed in most computers starting from the eighties [cite some source]. A graphics card is a complex electrical circuit that processes graphical data sent from the CPU to render and visualize it on the monitor with increasing quality and refreshing rates. The primary idea was to offload the CPU from rendering images on the display and use a dedicated hardware instead. A high pressure on graphics cards for fast refreshing of pictures (mainly for video-gaming) caused the spectacular increase of the computational power reflected by the large number of cores packed in one card. The high computational power attracted scientists and engineers looking for low-cost high-performance alternatives to speed up their numerical calculations in different applications. However, to use these first graphics cards, scientists had to adapt their problems to the partially programmable GPU, which implied a change to the data storage and the programming language. The term GPGPU, which stands for General Purpose Graphics Processing Unit, was established for this type of use of the graphics card. In response to this emerging demand, NVIDIA released in 2007 the first fully programmable open graphics processing units in a C-based programming language called CUDA. At the same time, AMD released its programmable GPU with OpenCL. The programming model CUDA is specialized for NVIDIA GPUs whereas OpenCL is preferred on AMD GPUs.

2.4.3. CUDA C programming

This subsection explains briefly some concepts of GPU programming implemented in the CUDA C or CUDA language. An extensive and detailed treatment of the topic can be found in the CUDA user manual [1]. CUDA is based on the C programming language with a minimal set of extensions to handle the parallel execution and the memory organization. The main computing systems involved in CUDA programming are the *host*, which is the traditional CPU in a personal computer and the the *devices*, which are the massively parallel processing devices, typically the GPUs. Many software programs have sections that may exhibit *data parallelism*, a term used for a case that allows arithmetic operations to be safely performed on different parts of the data structure in parallel. A CUDA program can help accelerate these sections of programs by running them on CUDA compatible GPUs, which as we discussed in 2.4.1 have massive number of arithmetic units. Hence a CUDA program contains both host code and device code. The functions running on the GPU are called kernels and are executed by threads, which are organized in grids of blocks distributed among the multiprocessors. CUDA kernels are launched as follows:

```
kernel_name <<<nB,nT>>>(args);
```

where, nB he number of thread blocks launched and nT the number of threads per block. A kernel is usually executed by $nB \times nT$ threads, also known as a grid; though a run with one block of one thread is equivalent to a serial run on a CPU. Each thread is executed by a core of the GPU and a thread block can be computed on a streaming multiprocessor. Several concurrent blocks can reside

[1]https://docs.nvidia.com/cuda/index.html, retrieved December 2020

on one streaming multiprocessor depending on the blocks' memory requirements and the processor's memory resources. The GPU starts threads always as a multiple of the warp size, in which the input blocks are divided into thread wide units. The division is implementation specific, though currently the warp size is 32 contiguous threads per block. Therefore, the number of requested threads per block is recommended to be a multiple of 32 otherwise, the last warp is not fully used and its extra threads are inactive but consume, nevertheless, registers and shared memory space. When all threads of a kernel complete their execution, the corresponding grid eliminates, and the execution continues on the host until another kernel is launched. Multiple kernels can execute on a device at one time.
Let us illustrate a typical CUDA programming structure using the simple example of scalar vector addition (SAXPY), a popular benchmark in computing, $\mathbf{y} \leftarrow \mathbf{y} + \alpha\mathbf{x}$ where $\mathbf{x}, \mathbf{y} \in \mathbb{R}^N$ and $\alpha \in \mathbb{R}$. First it is helpful to review the CPU-only implementation. (see Listing 2.1)

Listing 2.1: sequential SAXPY in C

```
// compute saxpy
void saxpy(int N, float a, float *h_x, float *h_y)
{
  for (i=0; i<N; i++) h_y[i] = h_y[i] + a*h_x[i];
}

int main(void)
{
  // memory allocation for a , h_y and h_x
  // I/O to read h_x and h_y, N elements each and scalar a

  ...

  saxpy(N,a,h_x,h_y);

}
```

We prefix the variable names that are mainly processed by the host with `h_` and that by the device with `d_`. The implementation is straightforward. The `saxpy()` function uses a for loop to iterate through the `N` vector elements and the *i*th iteration `h_y[i]` receives the sum of `a*h_x[i]` and `h_y[i]`. Since the variables are passed by reference, when the function `saxpy()` returns, the new contents of `h_y` can be accessed.
A standard way to execute vector addition in parallel is to replace the sequential loop with a grid of threads (see Listing 2.2).

Listing 2.2: Revised parallel SAXPY for CUDA

```
#include <cuda.h>
...
__global__
void saxpy(int N, float a, float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if(i<N) y[i] = a*x[i] + y[i];
}

 int main(void)
{
  // I/O for host variables x, y and N
  // Allocate d_x, d_y pointers to device memory

  ...

  //copy x,y to device memory
  cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
  cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);

  // Perform SAXPY , warp size is 256
  saxpy<<<ceil(N/256.0),256>>>(N, a, d_x, d_y);

  //copy back result to host memory
  cudaMemcpy(y, d_y, N, cudaMemcpyDeviceToHost);
```

```
    //free device pointers
    cudaFree(d_x); cudaFree(d_y);

    ...
}
```

Notice the use of keyword `__global__` before the function declaration. This instructs the compiler to generate a kernel function, executed on the device but callable from the host. Other keywords such as `__device__` and `__host__` define purely device and host functions respectively, callable only inside the particular system. `saxpy()` kernel is launched using `N/256.0` thread blocks and `256` threads per block. There exists a set of predefined variables `threadIdx, blockIdx, blockDim` that assign each thread with a unique global identifier. Every thread treats a small subset of elements, and the programmer must ensure the proper use of an index inside the kernel. Kernel calls are asynchronous since the control is returned to the host immediately after the call, which can then proceed with the control flow. A synchronization or a memory copy can force the CPU to wait for the completion of the last called kernel. Finally, CUDA supports a set of runtime API functions, for managing data in the device memory. In the above example, `cudaMemcpy()` function is used for data transfer between device and host memories. `cudaFRee()` is then used to free the object from device global memory.

It should be emphasised that the scalar vector addition example used above is trivial. Infact, the overhead of allocating device memory, bandwidth between host and device, and final memory deallocation will likely make the resulting code slower than the its sequential counterpart. Relative to the floating point operations performed in this example, data processing requires more work. Hence an important point to note is that, GPU code optimization would not always accelerate every algorithm. A careful understanding of the use case and the device architecture is therefore, a must to reach good performance. We will discuss this in later sections for our excercise. Several popular numerical libraries now exist in CUDA optimized for production, such as CUBLAS [2], CUSPARSE [3], CUFFT [4]. For brevity, we omit talking about other available functionalities here. An in depth discussion for other CUDA features can be found in the official CUDA manual and in textbooks such as [19].

2.5. Useful Parallel Algorithms

The optimization of diverse mathematical algorithms used in financial mathematics marked with the use of high performance computing has been promising. The increased performance as well as the scalability provided by GPUs means that institutions are able to increase business at the same time as having tighter control on their exposure. We observe the pervasive use of GPU computing in several areas of quantitative finance. Most popular being the pricing of financial instruments such as options [see 30, 33, 36]. In simulating credit risk losses of large portfolios, which is the focus of this work GPU algorithms have implemented Monte Carlo methods. Monte Carlo simulations are very computationally expensive owing to the slow convergence properties of the tails of the loss distribution; they require no less than a CPU server farm consisting of several hundred cores running over several hours. The motivation of GPU acceleration is to investigate the extent to which a single GPU can replicate the performance of a typcial multicore CPU and, if so, what new capabilities does a GPU implementation offer. Monte Carlo analysis are embarrassingly parallel processes and hence an good fit for GPU both algorithmically and architecturally, Moreover, the onboard memory provides both the capacity and bandwidth to complement the frequent processing of large portfolio data sets. We now look at a few core algorithms that are useful for implementing the Monte Carlo simulation. [10, 27]

2.5.1. Random Number Generation

A core feature of the Monte Carlo simulation is the random number generator requiring almost $O(10^{15}) \sim O(2^{50})$ samples with good statistical quality. The GPU architecture also places additional requirements on the generator: the ability to generate different substreams on parallel nodes with no correlations between substreams on different nodes. If we use the same generator in all parallel processes, the effective generator over the whole process will produce stretches of identical values. A master-worker

[2]https://docs.nvidia.com/cuda/cublas/index.html
[3]https://docs.nvidia.com/cuda/cusparse/index.html
[4]https://docs.nvidia.com/cuda/cufft/index.html

generator, that generates random numbers on one thread and distributes them to other processes, causes considerable overhead or may again cause correlation between processes. An alternative is to set up independent generators with different parameter choices. One thread may generate the starting point for the other thread's generated sequence. A popular random number generator is the *Mersenne twister*(MT19937). It has been adapted to allow parallel streams of uncorrelated numbers but may require lots of memory. As a result, the former may not be ideal for the GPUs.
Two different GPU optimized generators, the pseudo random and the quasi-random number generators have been benchmarked on the GPU and the CPU. The first is a pseudo-random number generator which uses an implementation of L'Ecuyer's multiple recursive random number generator (mrg32k3a) described in [22]. This generator ports well onto the GPU because of its inherent structure of long streams and substreams, providing a substantially large period of 2^{191}. The latter is a quasi-random number generator which uses a Sobol sequence implementation described in [6]. The block and grid structures on the GPU complement the Sobol implementation. Using pseudo-random numbers, offers a rate of convergence $O(N^{-\frac{1}{2}})$, whereas with quasi-random numbers this rate of convergence can improve to as much as $O(N^{-1})$. Making use of the latter scheme would require fewer simulations to achieve the same convergence properties and effectively offer a speed-up.

2.5.2. Sorting Loss Distribution

Calculating the capital buffer based on the analysis of the portfolio losses requires the loss vector to be sorted, a process which is particularly efficient. Performance benefits can be gained from the careful ordering of the portfolio data, which results in better arrangement in memory and hence permitting efficient data requests. Sort algorithms are supplied within the Thrust library [5]. This library has several performance optimized algorithms- merge or radix sort.

2.5.3. Numerical Integration

Integration is one of many types of numerical computations that is highly suitable for parallel processing. Due to the linearity of the integration operator, no communications among the processors are required during computation, one can achieve high parallel efficiency. In addition, it scales well with many workers. At the end of computation, a many-to-one, collective, communication is required to collect the integral sums from all the processors and compute the final integral sum. In context of GPUs, this approach implies that the points or the subregions of discretized domain processed at once is equal to the number of threads. A GPU has to hold the state of each thread, hence the number of threads executed simultaneously maybe limited by insufficient GPU resources. Popular approaches with summation operations can be applied to numerical integration tasks easily. These include the parallel sum reduction which uses a tree-based approach within each thread block to compute individual sums. A schematic Fig. 2.5 shows this hierarchical approach. Let us consider without loss of generality, a scalar valued function $h(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$ to be integrated over a domain D partitioned as $\bigcup_i^n D_i$:

$$I = \int_D h(\mathbf{x}) d\mathbf{x} \approx \Sigma_{i=1}^n \overbrace{\int_{D_k} h(\mathbf{x}) d\mathbf{x}}^{H_k}$$

[5]http://code.google.com/p/thrust/

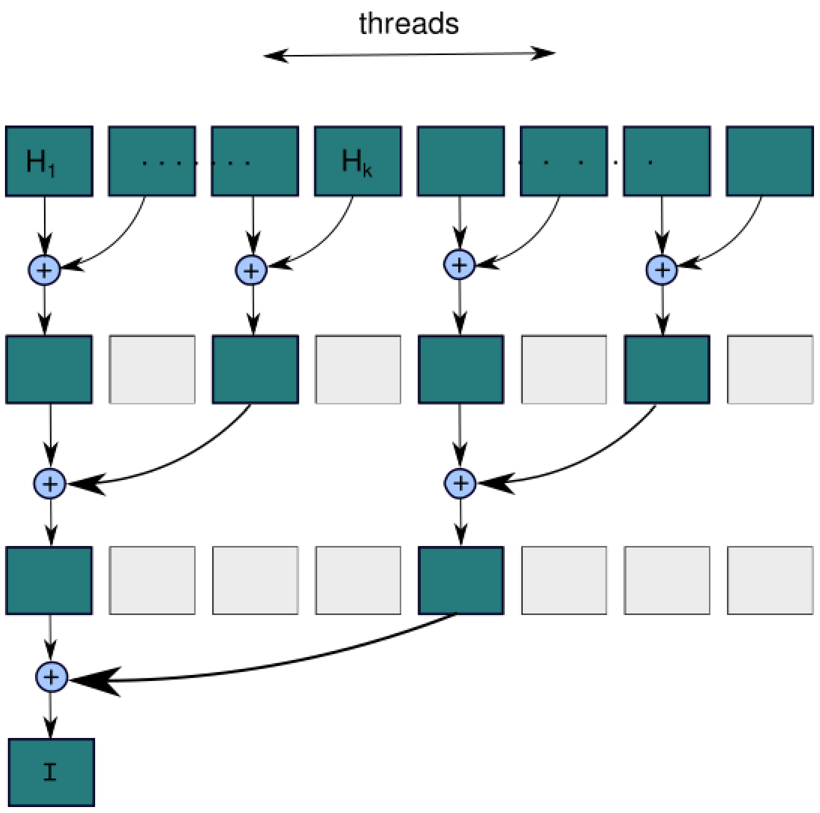


Figure 2.5: With each operator +, the parallel reduction algorithm repeatedly groups each computation of a subintegral, H_k in pairs. Each pair is computed in parallel with others, halving the overall array size in one step. The process is repeated until a single element exists

3 Modeling Portfolio Loss Distributions

In section 2.1 we introduced the concept of credit portfolio losses and various defined measures that are beneficial for determining the optimal portfolio. We now outline a few mathematical approaches to model these portfolio losses. Models for estimating credit risk has progressed for a long time. There are now three main approaches to modelling portfolio loss distributions of debt instruments: the “Merton-style” approach, the purely empirical econometric approach, and the actuarial approach. Each of these approaches has, in turn, produced several models. As described in 2.1, our concentration in this thesis will be on models based on structural default approach of Merton[25] in a *static* context. A key challenge in all such models always has been, to explain the relationship between default event, including on the dependence between defaults, Loss given defaults and Obligor default probabilities. Since direct modelling of the pairwise dependencies becomes unfeasible as credit portfolios become larger. Models follow different approaches to reduce this computational complexity by utilizing several kinds of approximations. The earliest credit risk measure models that were published, are popular industrial examples such as CreditMetrics by J.P. Morgan, KMV’s Portfolio Manager, CreditRisk+ by Credit Suisse. These models, though are slight variants of each other, essentially use a factor based approach to introduce default dependence via a few common market variables and obligor dependent idiosyncratic variables. We refer to [9] for a detailed description of the above. Before understanding what factor based models are, let us give a brief overview of an important tool used for representing correlations.

3.1. Copulas

Copulas are tools to describe the interrelation of several random variables. Let us consider an explanatory example: what can be a simple way to map a one dimensional random variable, X, to a one dimensional standard normal, Y? The answer is built on a transformation that takes either one to a standard uniform random variable, U. Standard uniform means that the probability density of U is $h(u) = 1$ if $0 \leq u \leq 1$, and $h(u) = 0$ otherwise. The copula construction is based on the above mapping. The cumulative distribution function (CDF), $F(x) = Pr(X \leq x)$ maps any random variable, X to a standard uniform. If X is a random variable whose CDF is F , then $U = F(X)$ has the standard uniform distribution. This goes both ways. If U is standard uniformly distributed then, X can be found by solving $F = U(X)$. An N-dimensional copula is a distribution function on $[0, 1]^N$ with standard uniform distributions:

$$C(u) = C(u_1, u_2, \dots, u_N) \tag{3.1}$$

Theorem 3.1.1 (**Sklar’s theorem (1959)**). *Consider an N-dimensional CDF F with marginals $F_1, \dots, F_N$. There exists a copula $C : [0, 1]^N \to [0, 1]$, such that*

$$F(x_1, \dots, x_N) = C(F(x_1), \dots F(x_N)) \tag{3.2}$$

$\forall x_i \in [-\infty, \infty]$, $i = 1, \dots, d$. *If F_i is continuous $\forall i = 1, \dots, d$ then C is uniquely determined; otherwise C is uniquely determined only on $RanF_1 \times \dots \times RanF_N$, where $RanF_i$ denotes the range of CDF F.*

According to Theorem 3.1.1, we can extract a unique representation of copula C explicitly, in terms of F and its continuous margins by calculating

$$C(u_1, u_2, \ldots, u_N) = F(F_1^{-1}(u_1), F_2^{-1}(u_2), \ldots, F_N^{-1}(u_N)) \tag{3.3}$$

where $F_1^{-1}, \ldots, F_N^{-1}$ are (generalised) inverses of $F_1, \ldots, F_N$.

Li [23] used the copula function conversely. The copula function links univariate marginals to their full multivariate distribution. For N given univariate marginal distribution functions $F_1(x_1), F_2(x_2), \ldots, F_N(x_N)$ with $x_i = F_i^{-1}(u_i)$ the joint distribution function C is defined as

$$\begin{aligned} C(F_1(x_1), F_2(x_2), \ldots, F_N(x_N), \Sigma) &= Pr[U_1 \leq F_1, U_2 \leq F_2, \ldots, U_N \leq F_N] \\ &= F(x_1, x_2, \ldots, x_N) \end{aligned} \tag{3.4}$$

where Σ is correlation matrix of $U_1, U_2, \ldots, U_N$ With given marginal functions, we can construct the joint distribution through copulas accordingly. Thus, with given individual distribution (e.g., creditworthiness over a 1-year horizon) of each credit asset within a portfolio, we can obtain the joint distribution and default correlation of this portfolio through copula function. In our methodology, we do not use copula function directly. We describe the concept of factor copula for further improvement to form the default correlation. If there are substantial number of instruments (for instance, N > 1,000) in our portfolio, to capture obligor-obligor interactions, we may be required to store a $N \times N$ correlation matrix, hence scalability can be a challenge. Factor copulas can be used to avoid constructing a high-dimensional correlation matrix. We can attempt to explain a large part of interactions in terms of a smaller set of market variables.

3.1.1. Linear Factor Copulas

Factor copula describes dependence structure between random variables not from the perspective of a certain copula form but from the factors model. Factor copula models have been broadly used for derivative pricing as well as computing banks' capital charge requirements. The main concept of factor copula model is that under a certain macro environment, credit default events are independent to each other. And the main causes that affect default events come from potential market economic conditions. Therefore a variation in the N-dimensional creditworthiness vector $\mathbf{X} = (x_1, \ldots, x_n, \ldots, x_N)$, can be explained in terms of the variation of a smaller set of market economic conditions or systematic risk factors, say $\mathbf{Z} = (Z_1, \ldots, Z_d)$. Formally we define a linear factor model as follows.

Definition 3.1.1 (**Linear Factor Copula model**). The random vector **X** is said to follow a $d-$factor model if it can be decomposed as

$$\mathbf{X} = \boldsymbol{\beta}^T \mathbf{Z} + \mathbf{b} \circ \boldsymbol{\epsilon}$$

where, $\circ$ represents the Hadamard product of vectors and

- $\mathbf{Z} \in \mathbb{R}^d$ with $d < N$ and a covariance matrix that is positive definite.
- $\boldsymbol{\epsilon} \in \mathbb{R}^N$ is a random vector of idiosyncratic risk factors, which are uncorrelated.
- $\boldsymbol{\beta} \in \mathbb{R}^{d \times N}$ & $\mathbf{b} \in \mathbb{R}^N$ are the constant *factor loadings* matrices.

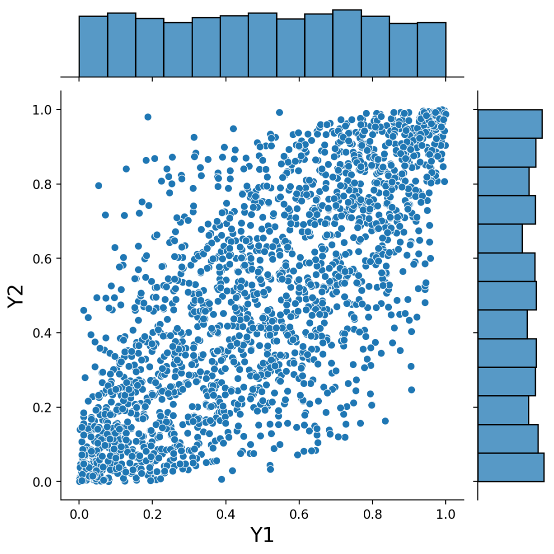
3.1.2. Examples

Gaussian copula A $d-$factor Gaussian copula model assumes that the correlations among x_n are imposed by linking them to $d \times 1$ **Z** vector of independent standard normal variables, for $n = 1, \ldots, N$.

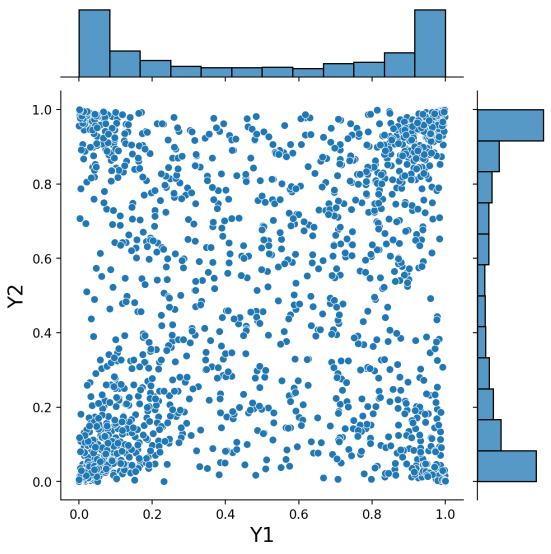
$$x_n = \beta_n^T \mathbf{Z} + b_n \epsilon_n, \tag{3.5}$$

where $\mathbf{Z} = (Z_1, \ldots, Z_d)^T$ with $Z_n \sim N(0,1)$, $\beta_n = (\beta_{n,1}, \ldots, \beta_{n,d})^T$, $b_n = \sqrt{1 - \sum_{i=1}^d \beta_{n,i}^2}$ and $\epsilon_n \sim N(0,1)$. Using the result of the above Gaussian assumptions in the context of modeling credit portfolio losses as described in Section 2.1, the default threshold of x_n is given by

$$\xi_n = N^{-1}(p_n)$$



(a) bivariate Gaussian copula with correlation=0.5



(b) bivariate T-copula with correlation=0.5 and $\gamma = 1$

Figure 3.1: Visualization of bivariate copula on a unit cube $[0,1]^2$ with corresponding marginal distributions. Note the differences of the distributions in the tail.

T Copula As an alternative to the Gaussian copula, the correlations of defaults can be modelled by a Student-t copula. Banking regulations published by the Basel group [4] suggested two ways of modelling for the correlations via a Student-t copula for credit risk, either using a Student-t copula for all issuer risk factors, or using a Student-t copula for the systematic risk factors. Here we discuss the second case. Particularly, we model the systematic risk factors as a multivariate-t random vector and the idiosyncratic factors as Gaussian random variables. This model choice gives another layer of difficulty since the default thresholds of the obligors do not have analytical solutions. The standard Student-t copula $C^t_{\gamma,\Sigma}$ given by

$$C^t_{\gamma,\Sigma}(u_1,\dots,u_d) = \int_{-\infty}^{t_\gamma^{-1}(u_1)} \dots \int_{-\infty}^{t_\gamma^{-1}(u_d)} \frac{\Gamma\left(\frac{\gamma+d}{2}\right)}{\Gamma(\gamma/2)\sqrt{(\pi\gamma)^d \mid \Sigma \mid}} \left(1 + \frac{\mathbf{X}^T\Sigma^{-1}\mathbf{X}}{\gamma}\right)^{-\frac{\gamma+d}{2}} d\mathbf{X}$$

If a random vector **X** has the t copula $C^t_{\gamma,\Sigma}$ as the component-wise probability transformed random vector and the student-t distribution as the component-wise marginal distribution with the same degree of freedom γ, then it follows a multivariate student-t distribution with γ degrees of freedom. Specifically, the d-dimensional random vector **X** is said to have a multivariate t distribution with γ degrees of freedom, the mean vector μ and a positive-definite dispersion matrix Σ, denoted by $\mathbf{X} \sim t_d(\gamma,\mu,\Sigma)$. Another convenient representation for simulation is

$$\mathbf{X} = \sqrt{W}\mathbf{Z} \tag{3.6}$$

where $\mathbf{Z} \sim N_d(0,\Sigma)$, and W is independent of Z and has an inverse gamma distribution i.e,

$$W \sim I_g(\gamma/2,\gamma/2) \tag{3.7}$$

Hence incorporating the dependence of the (3.6) to modify the systematic factors of x_n we find,

$$x_n = \sqrt{W}\beta_n^T\mathbf{Z} + b_n\epsilon_n \tag{3.8}$$

The marginal distribution of x_n is a convolution of a Student t distribution and a Gaussian distribution, while the dependence among x_n solely depends on the common factors **W**, **Z** and the idiosyncratic factors ϵ_n remain independent.

3.1.3. The Vasicek model

The first copula model for portfolio credit risk was given by Li [23]; his model is based on the Gaussian copula. The one-factor Gaussian copula model offers analytic tractability by the assumption that the underlying portfolio of assets is large and homogeneous, hence a special case of the examples described above. This approach by [35] is also referred to as the Large Homogeneous Pool (LHP) Model. Although it relies on the Gaussian distribution and is often criticized for being simplistic, there have been extensions to this model.[cite] For now, we follow the standard assumptions. Without loss of generality, we set loss value at default ($\epsilon_n v_n$) (Ref. eqn 2.3) constant for all obligors. The Vasicek model evaluates the default of an obligor in terms of the evolution of its asset value over a fixed time horizon $T > 0$, based on the firm value model of Merton (Ref. section 2.1). With β constant for all obligors, Equation (3.5) for a single factor Gaussian copula for the n-th obligor reduces to,

$$x_n = \beta Z + \sqrt{1-\beta^2}\epsilon_n, \tag{3.9}$$

where Z and ϵ_n are i.i.d standard normal random variables. Other assumptions of the Vasicek model are that all PD_n are equal and ϵ_n equals 1. The probability of default conditional on the common factor $Z = z$ can be given by,

$$p_n(z) = \mathbb{P}\left[\epsilon_n < \frac{\xi_n - \beta Z}{\sqrt{1-\beta^2}} | Z = z\right] = N\left(\frac{N^{-1}(PD_n) - z\beta}{\sqrt{1-\beta^2}}\right) \tag{3.10}$$

The resulting loss distribution is $L = \sum_{n=1}^{N} \mathbb{1}_{x_n \leq \xi_n} \sim Bin(N, p_n)$. As a consequence of the Central Limit theorem,

$$\lim_{n\to\infty} L/n = p(Z) = N\left(\frac{N^{-1}(PD_n) - Z\beta}{\sqrt{1-\beta^2}}\right) \quad a.s.$$

The analytical asymptotic Vasicek formula is helpful for calibrating our results which we will see in Chapter 4.

The methodology for measuring credit risks in debt instruments is in a state of rapid development. Here, we have only presented only a few of them. We refer to Laurent and Gregory [20] and Burtschell, Gregory and Laurent [7], that provide comparative analyses of various copula-based factor models. Typically factor copulas are simulated using the Monte Carlo method. An alternative approach are the whole branch of semi-analytic methods, which are based on on a combination of numerical integration and analytic methods. These models are significantly faster than Monte Carlo calculations. Hull and White introduced a bucketing approach[], Andersen and Sidenius [2] and Jackson [17] proposed a recursive method valid in more general cases. Saddle-point approximations were analyzed by Huang and Oosterlee [16]. Fourier transform methods were analyzed by Laurent and Gregory [20] and Grundke [14]. These methods suggest using Fourier transforms of densities , that is characteristic functions of the random variables, to compute the distribution of the sum of the independent assets. This alternative class of methods based on Fourier transforms, leads us to our main approach in this thesis for our models, the COS method which we describe in Section 3.4. A recent numerical approach proposed by Colldeforns-Papiol et al [8] is based on the Haar wavelets basis instead of a Fourier basis to compute the inversion of the characteristic function.

3.2. Monte Carlo methods

For computing a general cumulative loss distribution function, denoted by F_L and its corresponding risk measures, the industry standard is using Monte Carlo methods. First we need to draw independent and identically distributed replicates of the random variable L, and then given a set of n i.i.d samples

$L^{(1)}, \dots, L^{(n)}$, we require a method to estimate the risk measures.

Algorithm 1: Monte Carlo method for single factor Gaussian copula

```
Data: number of simulations $n$, factor loadings □ $\in \mathbb{R}^N$, obligor specific properties $\mathbf{PD}$ & $\mathbf{l} \in \mathbb{R}^N$
Result: Portfolio Loss vector $\mathbf{L} \in \mathbb{R}^n$
for i=1,..,n do
    $L^{(i)} = 0$;
    Generate $z \sim \mathcal{N}(0,1)$;
    for j=1,...,N do
        Generate $\epsilon \sim \mathcal{N}(0,1)$;
        $x_j = \beta_j z + \sqrt{1-\beta_j^2}\epsilon$;
        if $x_j < N^{-1}(PD_j)$ then
            $L^{(i)} = L^{(i)} + l_j$;
        end
    end
end
```

The *Crude Monte Carlo* estimator of VaR comes from the quantile estimator explained in traditional statistics. From our definition of VaR (Ref. eqn 2.5) the analytical distribution is replaced with the equivalent empirical distribution function $\hat{F}_L$,

$$VaR_\alpha = \inf x : \hat{F}_L(x) \geq \alpha$$

where, $\hat{F}_L(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{(L^{(i)} \leq x)}$ is the empirical distribution function of the iid sorted samples $L^{(i)}$ such that $L^{(1)} \leq, \cdots \leq L^{(n)}$. For Expected Shortfall (eqn. 2.7), the estimator is just the an expectation.

$$ES = \frac{1}{N(1-\alpha)} \sum_{i=1}^{N} L^{(i)} \mathbb{1}_{L^{(i)} \geq VaR_\alpha}$$

By the virtue of central limit theorem, the VaR and ES estimators have asymptotically normal distributions. An efficient variant of the Monte Carlo based methods is the Importance Sampling (IS) approach. Since the canonical Monte Carlo estimators generally require a very large sample size for desired lower tolerances, IS approach in practice can give better performance with lower sample sizes. However, we do not discuss this approach in our study. Refer to [See 24, Chapter 8] for a detailed treatment.

3.3. Fourier Transform Techniques

The essence of the Fourier transform is that it represents a function as a continuous superposition of periodic functions. Often we will obtain an analytic expression for the Fourier transform but not be able to analytically find the original function, and instead numerical schemes are used to invert the transformation.However , the amount of computational effort is small compared to that required to compute option prices via other means such as Monte Carlo simulation. A suggested approach is using Fourier transforms of densities ,that is characteristic functions of the random variables, to compute the distribution. This method relies on the fact that if x_n are independent random variables then

$$\mathbb{E}\left(e^{i\xi \sum_n x_n}\right) = \prod_n \mathbb{E}\left(e^{i\xi x_n}\right) \tag{3.11}$$

This corresponds to the fact that the Fourier transform of a convolution is the product of the Fourier transforms.We therefore compute the characteristic function of the random variable expressing the contribution of each asset to the loss distribution , take their product, and take the inverse Fourier transform to compute the loss distribution. Given the individual characteristic functions , it is simple to take the product. The loss distribution is then calculable via an inverse Fourier transform. A detailed analysis is given in the paper by Laurent and Gregory. [20]

3.4. The COS method

The previous section briefly outlined the general Fourier transform approach. Our work concentrates on the related idea of the *COS method* [11]. The algorithm is based on the usage of a cosine series on a truncated finite interval. In contrast to the complete Fourier transform, the cosine series converges rapidly for smooth functions on a bounded interval. Thus one needs to compute only a small number of terms. We will also see that the cosine coefficients of the density can be easily computed from its characteristic function , and so whenever there is a closed-form , they are simple to find.

3.4.1. Characteristic Function of the Portfolio Loss

We use ϕ_L to denote the characteristic function of the portfolio loss L described in Section 2.1. It then follows that

$$\begin{aligned}\phi_L(z;\omega) &= \mathbb{E}\left[\mathbb{E}\left[e^{i\omega\sum_{n=1}^{N} l_n\cdot\mathbb{1}_{x_n\le\xi_n}}|Z=z\right]\right]\\ &= \mathbb{E}\left[\prod_{n=1}^{N}\mathbb{E}\left[e^{i\omega l_n\cdot\mathbb{1}_{\epsilon_n\le\alpha_n(z_n)}}|Z=z\right]\right]\end{aligned}\tag{3.12}$$

and, $\alpha_n(z)=\frac{\xi_n-\beta_n z}{b_n}$

$$\begin{aligned}\phi_L(z;\omega) &= \mathbb{E}\left[\prod_{n=1}^{N}\left[1+P(\epsilon_n\le\alpha_n(z))(e^{i\omega l_n}-1)\right]\right]\\ &= \int_{\mathbb{R}} f_Z(z)\prod_{n=1}^{N}\left[1+P(\epsilon_n\le\alpha_n(z_n))(e^{i\omega l_n}-1)\right]dz.\\ &= \int_{\mathbb{R}}\Phi(z;\omega)dz.\end{aligned}\tag{3.13}$$

$f_{\mathbf{Z}}(\mathbf{z})$ denotes the joint probability density function of the systematic factors. The above integral can be computed by using any standard numerical integration scheme. Clenshaw-Curtis quadrature rule is preferred because of its exponential convergence on a smooth integrand as well as fast computational performance - the discretized integral can be solved via *Fast Fourier Transform* (FFT) algorithm, with $O(J\log(J))$ complexity where J is the total number of quadrature points.

3.4.2. Recover the CDF

The main idea of the COS method is that the density of a continuous random variable can be reconstructed from a Fourier cosine series. If the portfolio loss is a continuous function over a real domain, we could fix a range $[l_a,l_b]$ that is sufficiently broad to cover a large enough probability level, and consider the finite K terms Fourier cosine expansion of the density $f_L$of the portfolio loss L within the range as

$$f_L\approx\tilde{f}_K(x)\equiv\frac{A_0}{2}+\sum_{k=1}^{K}A_k\cos\left(k\pi\frac{x-l_a}{l_b-l_a}\right)\tag{3.14}$$

Whilst we have the Fourier cosine series, the Fourier cosine coefficients can be directly extracted from the characteristic function.

$$A_k=\frac{2}{l_b-l_a}\Re\left\{\phi_L(\frac{k\pi}{l_b-l_a})\cdot e^{-i\frac{k\pi l_a}{l_b-l_a}}\right\}\tag{3.15}$$

For computing the CDF F_L, it follows from (3.14) that yields,

$$F_L(x)\approx\tilde{F}_K(x)\equiv\frac{A_0}{2}(x-l_a)+\sum_{k=1}^{K}A_k\frac{l_b-l_a}{k\pi}\sin\left(k\pi\frac{x-l_a}{l_b-l_a}\right)\tag{3.16}$$

Here in the integration of the density function we again truncate the integration range and the last equation is obtained by interchanging the order of summation and integration. It can be seen that the CDF is expressed in the form of sine series.

3.4.3. Gibbs phenomenon

Since $f_L(x)$ in our case will always be a probability mass function, the corresponding Fourier cosine density expansion $\tilde{f}_K(x)$ will not converge to it in cases of highly non-smooth functions for example, step functions. The rate of convergence drops to first order and spurious oscillations develop near the discontinuities known as *Gibbs phenomenon*. The spurious effects do not die out as the number of the Fourier terms increases and also causes a slow convergence rate of the Fourier series at other continuous locations. Several techniques have been developed to mitigate or remove the Gibbs phenomenen, which includes applying spectral filters, change of basis for example using wavelets series expansion and edge detection algorithms [12]. In our study, we follow the approach of filtering. The main idea of applying spectral filters is to let the obtained series coefficients decay faster. Modifying eqn. (3.14), we obtain the COS density function after applying a spectral filter as,

$$\tilde{f}_K^{\sigma}(x) = \frac{A_0}{2} + \sum_{k=1}^{K} A_k \sigma(k/K) \cos\left(k\pi \frac{x - l_a}{l_b - l_a}\right) \tag{3.17}$$

where the spectral filter $\sigma(\eta) \in \mathcal{C}^{q-1}[-\infty, \infty]$ with the following properties,

- $\sigma(\eta) = 0$ for $|\eta| > 1$
- $\sigma(0) = 1$ and $\sigma(1) = 0$
- $\sigma^m(0) = \sigma^m(1) = 0 \ \forall m \in [1, \ldots, q-1]$

Analogously we get the COS CDF approximation using the now modified filtered density distribution eqn. (3.17),

$$\tilde{F}_K^{\sigma}(x) = \frac{A_0}{2}(x - l_a) + \sum_{k=1}^{K} A_k \sigma(k/K) \frac{l_b - l_a}{k\pi} \sin\left(k\pi \frac{x - l_a}{l_b - l_a}\right) \tag{3.18}$$

Error convergence of Fourier series expansion with spectral filters has been studied in literature, such as [34]. The analysis in shows that, to restore convergence in the Fourier series after applying a spectral filter, the function itself has to be at least piecewise continuous. However, it is already proven in [29] that, though f_L is not piecewise continuous, the recovered CDF function in the eqn. (3.18) still converges to the true CDF. The proof is based on the observation that the recovered expression is nothing but a finite linear combination of the Fourier series expansions of some piecewise constant functions.

3.4.4. Computation of Risk Measures

In Section 2.2, we have already seen the definitions of various credit risk measures. Here, we see how they are calculated under the COS method. The computation of VaR and ES are trivial. Since, once we have recovered the CDF, i.e $\tilde{F}_K(x)$, the q-th quantile can be solved numerically, $\tilde{F}_K(x) = q$. ES can be evaluated by for e.g. a numerical integration scheme.
As we have seen in Section 2.2.1, there exist standard decompositions of ES and VaR of the portfolio loss that follow the properties of Euler risk allocation. From equations (2.6) and (2.8) the decomposition of VaR and ES essentially is a problem of solving a conditional loss distribution. This is again solvable by applying the COS to conditional characteristic function. It follows from eqn (2.6),

$$\begin{aligned} CES_n &= l_n P(x_n \le \xi_n | L \ge VaR_\alpha) \\ &= l_n \frac{P(x_n \le \xi_n, L \ge VaR_\alpha)}{P(L \ge VaR_\alpha)} \\ &= \frac{l_n PD_n}{\alpha} P(L \ge VaR_\alpha | x_n \le \xi_n) \end{aligned} \tag{3.19}$$

To solve the conditional probability $P(L \ge VaR_\alpha | x_n \le \xi_n)$, we can start from its characteristic function $\phi_{n,L}$,

$$\begin{aligned} \phi_{n,L}(\omega) &= \mathbb{E}\left[e^{i\omega L} | x_n \le \xi_n\right] \\ &= \frac{1}{PD_n} \mathbb{E}\left[\left(\prod_{j \ne n} \mathbb{E}\left[e^{i\omega l_n \cdot \mathbb{1}_{n \le \alpha_j(z_j)}} | Z = z\right]\right) \cdot \mathbb{E}\left[e^{i\omega l_n \cdot \mathbb{1}_{\epsilon_n \le \alpha_n(z_n)}} | Z = z\right]\right] \end{aligned} \tag{3.20}$$

where $\mathbb{E}\left[e^{i\omega l_n \cdot \mathbb{1}_{n \leq \alpha_j(z_j)}} | Z = z\right] = 1 + P(\alpha_j(z_j))(e^{i\omega l_j} - 1)$ and $\mathbb{E}\left[e^{i\omega l_n \cdot \mathbb{1}_{\epsilon_n \leq \alpha_n(z_n)}} | Z = z\right] = P(\alpha_n(z_n)) \cdot e^{i\omega l_n}$
From eqn (2.8) it follows that,

$$\begin{aligned} CVaR_\alpha &= l_n \cdot P(x_n \leq \xi_n | L = VaR_\alpha) \\ &\approx l_n \cdot PD_n \cdot \frac{P(VaR_\alpha - \varepsilon \leq L \leq VaR_\alpha + \varepsilon | x_n \leq \xi_n)}{P(VaR_\alpha - \varepsilon \leq L \leq VaR_\alpha + \varepsilon)} \end{aligned} \tag{3.21}$$

Above we replace the $L = VaR_\alpha$ by approximating within ε tolerance of VaR_α. COS method can be used to evaluate the probabilities in eqn. (3.21).

4 Preliminary Results

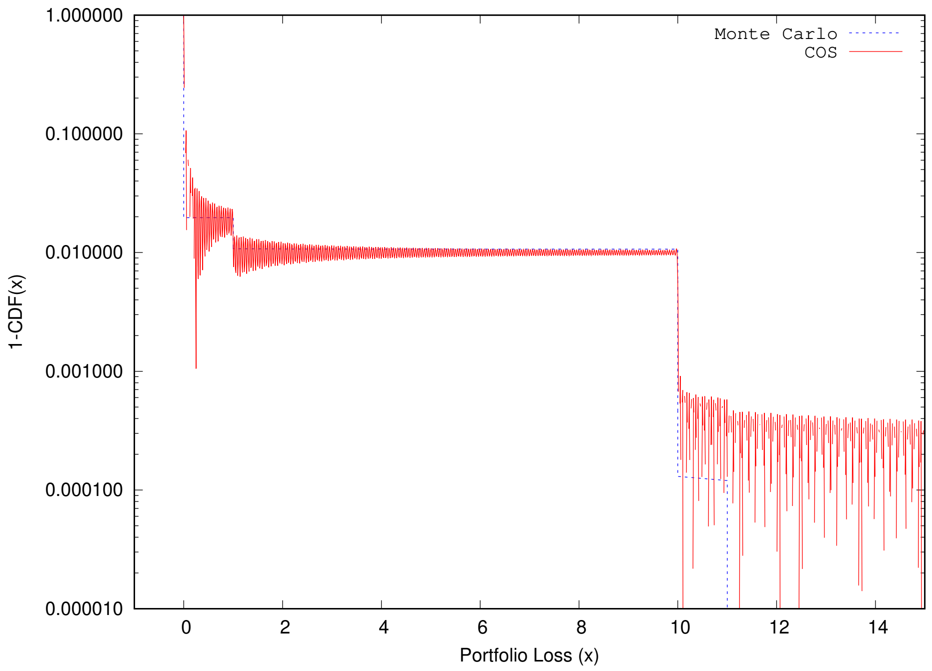
In this chapter we use the methodology described in the previous chapters to determine the loss distribution and the corresponding risk measure VaR. Other risk measures follow naturally from the computed distribution. Since our primary concern is finding an performance-optimized version of the COS method, we therefore ignore computing the risk measures in this literature report. These computations will eventually form a part in the final version of this thesis. More specifically, we see some preliminary results of performance (i.e accuracy and computational speed) of the COS method and benchmark it against conventional approaches like the Vasicek one-factor model and Monte Carlo simulations. Other risk measures follow naturally from the computed distribution, and therefore, we ignore computing them here. The reason is two fold. First, our intention is to gain fidelity on our implementation of the COS method. Next it allows us to benchmark performance of the present serial algorithms against an initial GPU-parallel version of the COS. We finally conclude the chapter by showing a few characteristic trends and identifying bottlenecks within this simplistic GPU implementation. These results will motivate our future study to find a computationally efficient implementation of the COS method. These results should also provide an impression of the 'embarrassingly parallel' nature of the algorithm, and thereby hinting at an opportunity to make significant performance upgrades over the current state of the art.

4.1. Test scenario

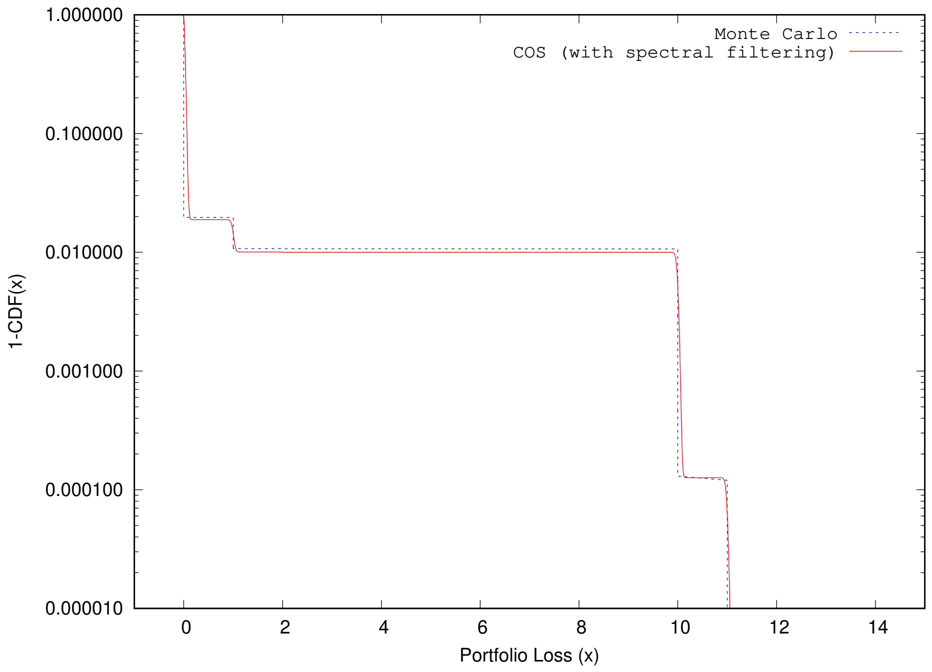
We first test the COS method with a small artificial portfolio with a name concentrated artificial portfolio with 10 obligors. This is achieved by setting probability of default and loss-at-default of the first obligor to be 10 times as large as of the other obligors. In details, we set the default probabilities of the obligors to be $PD_1 = 0.01$, $PD_n = 0.001$, $\forall n = 2, \dots, 10$. The loss at default of obligors was taken to be $l_1 = 10$, $l_n = 1$, $n = 2, \dots, 10$, respectively. We assume a one-factor Gaussian copula correlation between all obligors. Without loss of generality the factor loading $\beta_n = 0.2$ is set constant for all obligors. The piecewise constant CDF of the portfolio loss with name concentration is more likely to have a wide step at the location of VaR, particularly if the concentrated obligor defaults with a large probability at the event of VaR. In this case the loss CDF is a step function of few steps, and therefore we can notice the Gibbs phenomenon at the steps very clearly. Thus, the objective of this test case is to show that the filtered Fourier cosine series can recover the piecewise constant CDF. Figure 4.1 plots the recovered CDF of the portfolio loss from Monte carlo method with 100000 simulations compared with the result obtained from COS in both cases. As the plot indicates, the filtered COS approximation is accurate.

4.1.1. Verification with the Vasicek model

In 3.1.3 we described the model of Vasicek [35]. Here, we verify our recovered CDF from COS with the Large Homogeneous Pool (LHP) approximation from [35]. As described in earlier sections, the LHP approximation uses a one-factor Gaussian copula to represent the default correlation structure. The portfolio contains an infinite number of entities, which all have the same characteristics (e.g. PD_n, l_n, β_n). So, we check whether our piecewise constant CDF converges $a.s$ in the limiting case of number of obligors N to the computed Vasicek CDF with the same input data. Figure – shows the convergence.



(a) without filtering



(b) with exponential filter of order 2

Figure 4.1: Recovered CDF from Monte Carlo and COS methods for a test portfolio with 10 obligors.

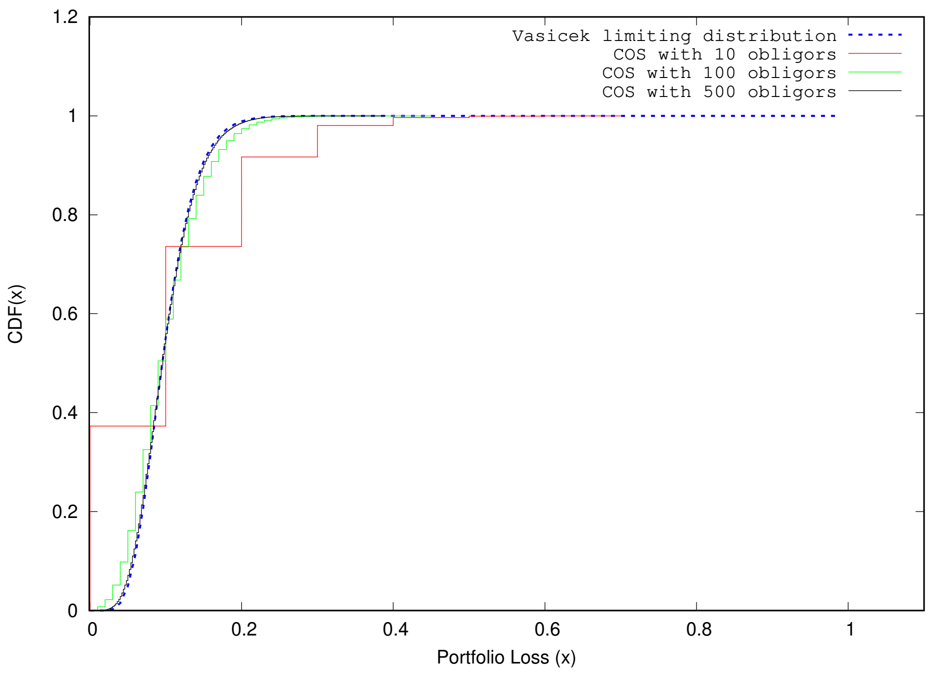


Figure 4.2: Comparison of Loss CDF for $PD = 0.1$, $\beta = 0.2$

4.2. S&P rated portfolio

We further give a numerical example with a large portfolio with 1000 obligors to mimic real-world situations. We first sample ratings of the obligors uniformly from the levels AAA, AA, A, BBB, BB, B and CCC, and the assign to each obligor the S&P[1] default probability by rating. Then the loss-at-default l n per obligor is uniformly sampled from [10, 1000]. Finally, a few name concentrations are created by means of multiplying the losses of some CCC obligors by a factor of 50 or 10. The reference values for calculating the relative errors are generated using the COS method, where we set the number of COS terms to be 800, the tolerance level of the integration range truncation error equal to be $1e-9$, the number of integration points to be 2^{20}. The benchmark MC results are obtained with 1 million simulations. As witnessed in the previous numerical example, the Figure 4.3 (a), confirms a good match of the loss distributions from the MC method and the COS method. As demonstrated in the Figure 4.3 (b), we see the errors of the COS method converge in a regular pattern for computed risk metric VaR with the increase in COS terms. The errors are computed with respect to the benchmark MC results.

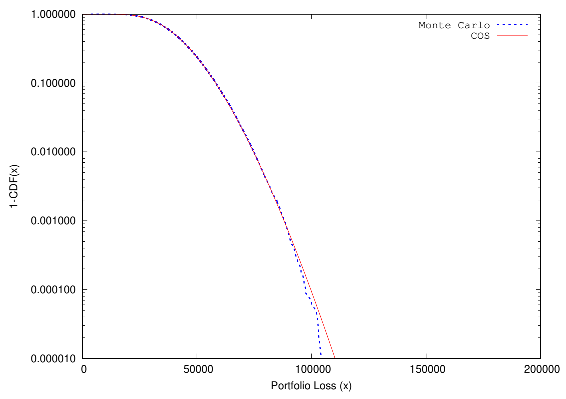
4.3. GPU runs

Serial runs on the CPU of the second numerical example from the previous section show that the computational time of the COS method is around one order of magnitude less than MC simulation. Profiling of the CPU implementation (runtimes tabulated in Table) reveals that the program spends approximately 99% of its entire runtime on calculating the Fourier cosine coefficients A_k (Ref. eqn (3.15)). This is expected since the integration of the characteristic function (Ref. (3.13))is computationally expensive. We show this task performed on a GPU using the CUDA platform [Ref. Section 2.4.3]. Earlier work in the direction of GPU acceleration of COS based option pricing have achieved impressive speed-ups on the GPU [36], where two approaches were followed, a parallelization over each vector element and a parallel vector summation. We follow a similar technique to the former. Thus, our first approach is to attempt to parallelize the computation of the Fourier cosine coefficients over threads of a GPU.[2]

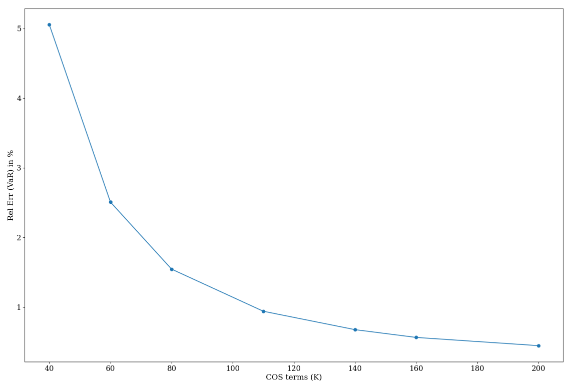
Figure 4.4 presents a comparison of the time consumed by our simple GPU implementation and also

[1]Standard and Poor's credit rating system url: https://www.spglobal.com/ratings/en/research/articles/200429-default-transition-and-recovery-2019-annual-global-corporate-default-and-rating-transition-study-11444862, Accessed: February 2, 2021

[2]The GPU we work on is a NVIDIA Maxwell architecture based TitanX card with 12GB of memory whereas the CPU test was done on the same computer on a dual 8-core 2.4GHz (Intel Haswell e5-2630-v3) with 64GB of memory. [3]



(a) recovered CDF vs portfolio loss



(b) Relative VaR error convergence

Figure 4.3: Results from an S&P rated portfolio with 1000 obligors

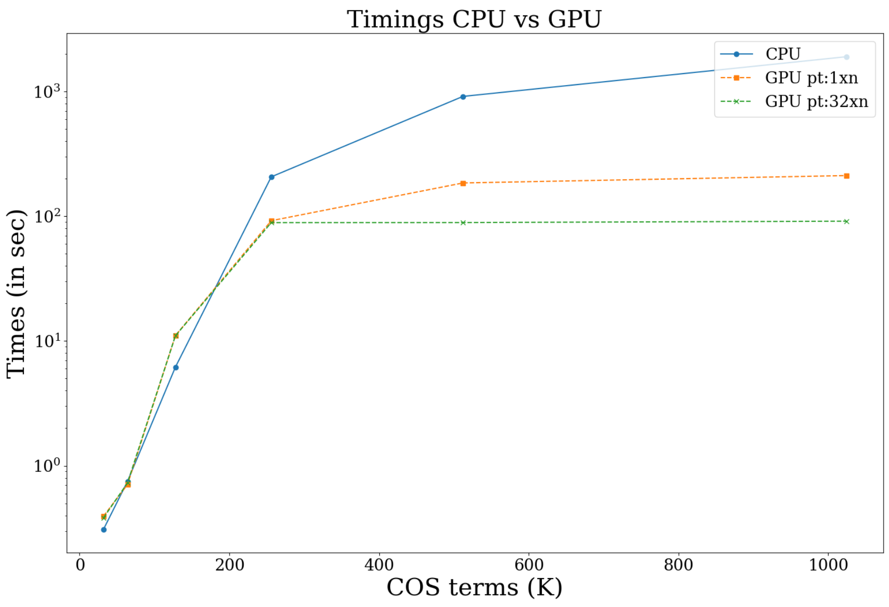


Figure 4.4: Runtime comparision between serial CPU and an initial GPU parallel versions of COS method with 1000 obligors

the CPU time. Clearly, the GPU version is faster than the CPU version only when the number of COS terms taken K is large. We follow 2 types of GPU thread divisions- one where all threads allocated over a single block (pt:1xn) and another where 32 threads are taken per block (pt:32xn). As observed in the plot, the latter performs slightly better with regards to computation time. Therefore, the division of work over constant 32 threads per block is a better alternative than using just a single block with all threads. In the particular example, the GPU implementation of the COS was an order of magnitude faster than the CPU version and almost 2 times as fast as the CPU implementation of the Monte Carlo method (see Table). However we note that it is not necessary to take such a large values of K in the COS method in practice, as with $K = 256$, the distribution already in the order of 10^{-6}. Therefore, the COS method with smaller values of K is not really advantageous on the GPU. Nevertheless, the implementation clearly shows the parallel nature of the COS method and identifies further areas of potential improvements, for example, when dealing with multi-factor models or parallelizing over other variable dimensions. Some approaches we outline in Section 5.1. The implementation of such strategies will require many more computations, hence the performance of the GPU version will be profound.

5 Concluding Remarks

This literature study provides an opportunity to gather the theoretical background necessary for understanding the problem of modeling portfolio credit risk measures using factor copula based COS method and its potential for data parallelism. The existing methods for modeling portfolio losses are outlined in Chapter 3. Traditional methods still used in this field use Monte Carlo approaches that are computationally slow and inefficient. The semi-analytical COS method is an attractive alternative that demonstrates superior computational speed as well as accuracy. However,a drawback of the method is the impact on computational speed when extending the higher dimensional spaces, such as with the multifactor copula models. However, due to the “embarrassingly parallel” nature of the algorithm, the COS method can be optimized further with the help of data parallelism concepts such as the GPU device. The future master thesis will try to address the identified problem. In particular, with the help of a few preliminary numerical tests in Chapter 4, the feasibility of tackling this problem was justified and further study aims to address the existing challenges. The work will be performed with the cooperation of ING’s Quantitative Risk Analysis division within the Structured Products and Quants team. Corresponding research questions, goals and approach stated in this chapter are guidelines for the remaining part of the master thesis.

5.1. Discussions

This section reflects on some points of the thesis. We combine this with some directions for further research.

5.1.1. Factor Copula models

Our current numerical experiments utilized a straightforward implementation of the single factor Gaussian copula correlation structure. However, many different model configurations are possible. First it would be interesting to compare the results from the factor model in our situation, for example with the Gaussian-t model (Ref. 3.1.2) applied by [29]. Next, in the DRC model there are different systematic factors we could have applied for modelling the capital charge. The ’sector’ and ’country’ factor are the two systematic factors the BCBS suggests. Since, we have only considered a single systematic factor so far, the model doesn’t mimic real life portfolios. Experiments with multifactor copulas allow for a more detailed specification of correlation structure between obligors.

5.1.2. Parallelization Strategies

As mentioned in 4.3, the preliminary GPU implementation of the COS method with smaller values of K is not really advantageous. The loss values already are within the required tolerance. However, there are subtasks within the algorithm which can be optimized. We first concentrate on our computation of the characteristic function ϕ_L, which is essentially a numerical integration routine (Ref. eqn. (3.13)). As we discussed briefly in Section 2.5.3, the numerical integration operation can be divided across threads and finally using reduction, a resulting summation can be computed. This requires to be able to use multiple thread blocks and each thread block reduces a portion of the array in a tree based approach. Here, we omit more implementation specific details here for brevity. The well known bound for time complexity of a parallel sum reduction [19] is known to be $\mathcal{O}(M/P + \log_2 P)$, where M in our situation becomes the number of quadrature points and P is the number of threads. Our next approach will be to vectorize the calculation of the characteristic function itself. Recall from eqn. (3.13) the computation as follow,

$$\phi_L(z;\omega) = \int_{\mathbb{R}} f_Z(z) \underbrace{\prod_{n=1}^{N} \left[1 + P(\epsilon_n \leq \alpha_n(z_n))(e^{i\omega l_n} - 1)\right]} dz.$$

The product term, as demarcated above can be divided over threads. Analogous to a parallel summation, the multiplication operator is also linear, hence each process can compute a single multiplicand independently. This approach has the advantage of scaling over the number of obligors N, which can significant performance upgrades in cases of large portfolios. Interesting results should be expected.

5.1.3. Principal Component Analysis

Finally, another direction for further research is the topic of dimensionality reduction. Principal component analysis (PCA) aims to reduce the dimensionality of highly correlated data by finding a small number of independent linear combinations that approximately captures all the characteristic behaviours of the original data. In the same vein as factor copula models although PCA is not a model in itself, it can be used as a way of constructing appropriate factors for a factor model. The key mathematical result behind the technique is the theorem of *spectral decomposition* of normal matrices. Although this is a popular approach largely discussed in the literature, our aim will be use the principal components computed, in our implementation of our efficient data-parallel version of the factor copula based COS method. We will contrast our approaches from the existing approaches with the PCA optimized COS implementation and detail our findings.

5.2. Research Questions

As a final note, we formulate our research questions based on the contents of this report. In the course of this MSc project, the aim is to address these main research questions.

- We have noted the computational complexities of the "workhorse" Monte Carlo methods and the COS method in computing large portfolios with high dimensional correlated structures. Preliminary results have shown a GPU parallelized version improve the baseline performance. But, a related question remains: can we find the optimal parallel implementation for this baseline model?
- As outlined in the previous subsections, the parallel optimization of the COS method can be achieved by different approaches. Which will be the most appropriate approach to reduce the time complexity or does the most efficient approach include a careful combination of them all? What could be understood about of performance as well as in the accuracy of the parallel version over the existing serial routine ?
- Our final aim is to compete with the existing models with the literature. Hence, an important question is: do the performance metrics improve upon the current industry standard models like the Monte Carlo method?

Bibliography

[1] Carlo Acerbi and Balazs Szekely. Back-testing expected shortfall. *Risk*, 27(11):76–81, 2014.

[2] Leif Andersen, Jakob Sidenius, and Susanta Basu. All your hedges in one basket. *RISK-LONDON-RISK MAGAZINE LIMITED-*, 16(11):67–72, 2003. URL `http://www.ressources-actuarielles.net/EXT/ISFA/1226.nsf/d512ad5b22d73cc1c1257052003f1aed/bf571acf7dbca8cbc12577b4001e3664`.

[3] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(05):54–63, may 2016. ISSN 1558-0814. doi: 10.1109/MC.2016.127.

[4] January BCBS. Minimum capital requirements for market risk. 2016.

[5] Michele Bonollo, Luca Di Persio, and Luca Prezioso. The default risk charge approach to regulatory risk measurement processes. *Dependence Modeling*, 6(1):309–330, 2018.

[6] P Bratley and BL Fox. Implementing sobols quasirandom sequence generator (algorithm 659). *ACM Transactions on Mathematical Software*, 29(1):49–57, 2003.

[7] Xavier Burtschell, Jonathan Gregory, and Jean-Paul Laurent. A comparative analysis of cdo pricing models under the factor copula framework. *The Journal of Derivatives*, 16(4):9–37, 2009.

[8] Gemma Colldeforns-Papiol, Luis Ortiz-Gracia, and Cornelis W Oosterlee. Quantifying credit portfolio losses under multi-factor models. *International Journal of Computer Mathematics*, 96(11): 2135–2156, 2019.

[9] Michel Crouhy, Dan Galai, and Robert Mark. A comparative analysis of current credit risk models. *Journal of Banking and Finance*, 24(1):59–117, 2000. ISSN 0378-4266. doi: https://doi.org/10.1016/S0378-4266(99)00053-9. URL `https://www.sciencedirect.com/science/article/pii/S0378426699000539`.

[10] Matthew F. Dixon, Thomas Bradley, Jike Chong, and Kurt Keutzer. Chapter 25 - monte carlo–based financial market value-at-risk estimation on gpus. In Wen mei W. Hwu, editor, *GPU Computing Gems Jade Edition*, Applications of GPU Computing Series, pages 337 – 353. Morgan Kaufmann, Boston, 2012. ISBN 978-0-12-385963-1. doi: https://doi.org/10.1016/B978-0-12-385963-1.00025-3. URL `http://www.sciencedirect.com/science/article/pii/B9780123859631000253`.

[11] FANG Fang and Kees Osterlee. The cos method: An efficient fourier method for pricing financial derivatives. *SIAM Journal of Scientific Computing*, 2008. doi: 10.1137/080718061. URL `https://ir.cwi.nl/pub/13283/13283A.pdf`.

[12] Anne Gelb and Sigal Gottlieb. The resolution of the gibbs phenomenon for fourier spectral methods. *Advances in The Gibbs Phenomenon. Sampling Publishing, Potsdam, New York*, 2007.

[13] Michael B Gordy. A risk-factor model foundation for ratings-based bank capital rules. *Journal of financial intermediation*, 12(3):199–232, 2003.

[14] Peter Grundke. Computational aspects of integrated market and credit portfolio models. *OR Spectrum*, 29(2):259–294, 2007.

[15] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, Amsterdam, 5 edition, 2012. ISBN 978-0-12-383872-8.

[16] Xinzheng Huang and Cornelis Oosterlee. Saddlepoint approximations for expectations and an application to cdo pricing. *SIAM J. Financial Math.*, 2:692–714, 01 2011. doi: 10.1137/100784084.

[17] Ken Jackson, Alex Kreinin, and Xiaofang Ma. Loss distribution evaluation for synthetic cdos. Working Paper, 2007.

[18] Mark Suresh Joshi. *More mathematical finance*. Pilot Whale Press, 2011.

[19] David B Kirk and W Hwu Wen-Mei. *Programming massively parallel processors: a hands-on approach*. Morgan kaufmann, 2016.

[20] Jean-Paul Laurent and Jon Gregory. Basket default swaps, cdos and factor copulas. 2005.

[21] Jean-Paul Laurent, Michael Sestier, and Stéphane Thomas. Trading book and credit risk: How fundamental is the basel review? *Journal of Banking & Finance*, 73:211–223, 2016.

[22] Pierre L'ecuyer, Richard Simard, E Jack Chen, and W David Kelton. An object-oriented random-number package with many long streams and substreams. *Operations research*, 50(6):1073–1075, 2002.

[23] David X Li. On default correlation: A copula function approach. *The Journal of Fixed Income*, 9 (4):43–54, 2000.

[24] Alexander McNeil, Rüdiger Frey, and P Embrechts. *Quantitative Risk Management: Concepts, Techniques, and Tools*, volume 101. 10 2005.

[25] Robert C Merton. On the pricing of corporate debt: The risk structure of interest rates. *The Journal of finance*, 29(2):449–470, 1974.

[26] Basle Committee on Banking Supervision. *Revisions to the Basel II Market Risk Framework: Updated as of 31 December 2010*. Bank for International Settlements, 2011.

[27] Simon J. Rees and Joseph Walkenhorst. Chapter 24 - large-scale credit risk loss simulation. In Wen mei W. Hwu, editor, *GPU Computing Gems Jade Edition*, Applications of GPU Computing Series, pages 323 – 335. Morgan Kaufmann, Boston, 2012. ISBN 978-0-12-385963-1. doi: https://doi.org/10.1016/B978-0-12-385963-1.00024-1. URL http://www.sciencedirect.com/science/article/pii/B9780123859631000241.

[28] John Shalf. The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A*, 378(2166):20190061, 2020.

[29] Xiaoyu Shen, Fang Fang, and Chujun Qiu. Using the cos method to calculate risk measures and risk contributions in multifactor copula models. 2020.

[30] S. Solomon, R.K. Thulasirman, and P. Thulasiraman. Option pricing on the gpu. In *2010 IEEE 12th International Conference on High Performance Computing and Communications (HPCC)*, pages 289–296, 2010. doi: 10.1109/HPCC.2010.54.

[31] BASLE COMMITEE ON BANKING SUPERVISION. Amendment to the capital accord to incorporate market risks. *Basle, Switzerland, jan*, 1996.

[32] Top500. Top500 list. Technical report.

[33] Anson H.T. Tse, David B. Thomas, K. H. Tsoi, and Wayne Luk. Efficient reconfigurable design for pricing asian options. *SIGARCH Comput. Archit. News*, 38(4):14–20, January 2011. ISSN 0163-5964. doi: 10.1145/1926367.1926371. URL https://doi-org.tudelft.idm.oclc.org/10.1145/1926367.1926371.

[34] Hervé Vandeven. Family of spectral filters for discontinuous problems. *Journal of Scientific Computing*, 6(2):159–192, 1991.

[35] Oldrich Alfons Vasicek. The distribution of loan portfolio value. *Citeseer*, 2002.

[36] Bowen Zhang and Cornelis W. Oosterlee. Acceleration of option pricing technique on graphics processing units. *Concurrency and Computation: Practice and Experience*, 26(9):1626–1639, 2014. doi: https://doi.org/10.1002/cpe.2825. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.2825.