Measuring Value at Risk(VaR) of a Portfolio of Interest Rate Derivatives with Monte-Carlo Simulation and Parallel Computing

SMA5505 Applied Parallel Computing

Gunjan Kathuria
Shakun Mahajan
Toni

21st May 2002

Contents

List of Figures

1 Introduction

1.1 Motivation

The use of formal mathematical models and optimization in finance became common practice in the 1980s and 1990s. Distributed and parallel computing is playing an increasingly important role in trading, financial engineering, and all aspects of investment and commercial banking, due to a convergence of the following factors:

- Increased volatility due to globalization of financial markets
- Global distribution of data sources
- Increase in complexity of derivatives and other risk management vehicles
- Increased volume of raw data and need to process large databases
- Increased volume on the retail side of the spectrum mainly due to on-line technologies (Internet and WWW)

This can be verified by the fact that more financial institutions invest in high-performance supercomputers. Based on the list of Top 500 Supercomputer Sites [1] as illustrated in figure 1. As per November 10, 2001, a supercomputer manufactured by IBM that ranks at 25th, is installed at Charles Schwab USA [2]. This supercomputer consists of 768 processors of SP Power3 375 MHz. It can achieve a maximum LINPACK [3] benchmark performance of 795 GFlops, while its theoretical peak performance is 1152 GFlops.In addition, the only supercomputer listed in the Top 500 Supercomputer Sites that is installed in Singapore is owned by Citibank Singapore [4]. This supercomputer is manufactured by Hewlett-Packard. It consists of 64 processors of SuperDome/HyperPlex. It can achieve a maximum LINPACK benchmark performance of 99.20 GFlops, while its theoretical peak performance is 141 GFlops.

From the point of view of parallel computing field, this data is encouraging. There is a strong trend leading to pervasive deployment of parallel computing and architectures to support market demands in financial sector. For our project, we derive our motivation from this trend, especially for financial engineering sector. In particular, we study the potential application of parallel computing concepts in financial risk management related software.

1.2 Scope

Financial risk management is used to establish a stable financial environment for businesses. It is a key to the success of a business. Financial risk management ranges widely, for example:

- Interest Rate Management
- Equity Risk Management

Home About Current List Archive Database In Focus Contact

TOP500 Sublist for November 2001 (Revised Edition as of Nov 10, 2001)

List	Rank	Manufacturer	Computer	R_{max}(GFlops)	Installation Site	Country	Year	Processors
11/2001	461	Hewlett-Packard	SuperDome/HyperPlex	99.20	Cisco	Netherlands	2001	64
11/2001	462	Hewlett-Packard	SuperDome/HyperPlex	99.20	Cisco	USA	2001	64
11/2001	463	Hewlett-Packard	SuperDome/HyperPlex	99.20	Citibank	Singapore	2001	64
11/2001	464	Hewlett-Packard	SuperDome/HyperPlex	99.20	Comline AG	Germany	2001	64
11/2001	465	Hewlett-Packard	SuperDome/HyperPlex	99.20	DeTeCSM	Germany	2001	64
11/2001	466	Hewlett-Packard	SuperDome/HyperPlex	99.20	Financial Institute	Japan	2001	64
11/2001	467	Hewlett-Packard	SuperDome/HyperPlex	99.20	ITS Information Tech	Italy	2001	64
11/2001	468	Hewlett-Packard	SuperDome/HyperPlex	99.20	Infineon	Germany	2001	64
11/2001	469	Hewlett-Packard	SuperDome/HyperPlex	99.20	Infopoint/Renault	France	2001	64
11/2001	470	Hewlett-Packard	SuperDome/HyperPlex	99.20	Infostrada SPA	Italy	2001	64

Figure 1: Top 500 Supercomputer Sublist

- Energy Price Management
- Foreign Exchange
- Etc.

In addition, each of above mentioned division covers a wide variety of financial products. For example, interest rate management has several choices of financial products, namely:

- Interest Rate Swaps
- Interest Rate Swaptions (i.e. Swap Options)
- Interest Rate Caps
- Interest Rate Floors.

In this project, we focus on interest rate management. Specifically, we focus on *interest rate swaps* risk management. The reason for focusing on interest rate swaps is that it is one of the largest components of the global derivatives market.

1.3 Objective

1. To use Monte Carlo simulation and Quasi Monte Carlo simulation to determine the Value at Risk (VaR) of interest rate derivative portfolios, i.e. Interest Rate Swaps for US dollar (USD).

2. To implement and compare the performance of:

    - Serial Monte Carlo simulation
    - Serial Quasi-Monte Carlo simulation using Faure Sequences
    - Parallel Monte Carlo simulation and
    - Parallel Quasi-Monte Carlo simulation using Faure Sequences.

The main objective of this project is to achieve a significant speed up in parallel implementation of the simulation. The reason for this need of speed up is explained in a later part in this section: Advantages of Parallelizing VaR Computation.

1.4 Financial Background

The compulsory pre-requisite of this project is a clear understanding of the financial concepts involved. We have spent a significant amount of our time in this project to understand those concepts. In this section, a summary of two most important concepts involved in this project is presented, namely: Interest Rate Swaps and Value at Risk. A glossary of financial terms is provided at the end of the report.

1.4.1 Interest Rate Swaps

Overview
An *Interest Rate Swap* is an agreement between two parties who exchange interest payments, based on a notional principal amount, over an agreed period of time 2. The term *Swaps* is used to refer to Interest Rate Swaps in the subsequent part of this report. In addition, the term *cash flows* is used to refer to periodic interest payments stated in the definition above. The agreed notional principal amount will be referred as *notional principal.* Additionally, the period of time in which the agreement holds is called *maturity.*

A Swap means that a party (e.g. John) and the Bank have agreed to exchange the net difference between two different interest rates (commonly *fixed* versus *floating* rate). This exchange is based upon the amount John requires at the frequency he requires to match his needs; for example, half-yearly interest. The net difference between the two interest rates offsets the cashflows on the underlying borrowing. There is no exchange of principal. This process is illustrated in Figure 2.

An Example
As an example, suppose there is a hypothetical company called MicroHard with a relatively poor credit rating that borrows most of its funds with short maturities [6]. MicroHard may want to change its exposure to interest rates to more correctly reflect the long-term nature of the projects it is funding. Or, MicroHard may believe that long-term interest rates are going to rise, causing it to

Figure 2: Interest Rates Swaps Process

seek protection against the impact of higher interest rates on its balance sheet.

One solution is for MicroHard to enter into an interest rate swap. Upon entering into an interest swap, MicroHard will receive payments tied to the floating rate index. Then MicroHard uses those payments for borrowing in the short maturities (with payment dates corresponding to the dates MicroHard must reset its short-term borrowing). In addition, MicroHard would pay a fixed rate index, all on the same notional amount as its total outstanding borrowings.

With the swap, the managers of MicroHard have closed out the company's exposure to changes in short term rates and they have taken on an exposure to long term rates that more closely corresponds to MicroHard's long term assets.

Applications
Swaps are used to assist in the management of interest rates and cash flows. A Swap can be used by borrowers who have a desire to alter their interest rate or cash flow profile to suit their particular needs.

Variable interest rate borrowers would use Swaps to hedge their interest costs in a market where variable interest rates are expected to rise. In contrast, fixed rate borrowers would use a Swap to take advantage of lower interest rates in a market where variable interest rates are expected to fall.

Further, borrowers may use Swaps to alter the frequency of their cash flows. For example, annual interest payments can be converted into quarterly payments or quarterly payments into semi annual cash flows. This could be beneficial when underlying cash flows have varied.

In addition, differences in the credit quality between entities borrowing money motivate the interest rate swap market. Specifically, some agents may have a better borrowing profile in the short maturities than they do in the long maturities. Other agents (with more creditworthy status) have a comparative advantage raising money in the longer maturities.

A counter-party's creditworthiness is an assessment of their ability to repay money lent to them over time. If a company has a good credit rating, they are more likely to be able to pay back a loan over time than a company with a poor credit rating. This effect is magnified with time. By making it easier for less creditworthy agents to borrow in the short term than in the long term, lenders make sure that they are less exposed to this risk.

Therefore, we would expect that in fixed-floating interest rate swaps, the entity paying fixed and receiving floating is usually the less creditworthy of the two counter-parties.

The interest rate swap gives the less creditworthy entity a way of borrowing fixed rate funds for a longer term at a cheaper rate than they could raise such funds in the capital markets, by taking advantage of the entity's relative advantage in raising funds in the shorter maturity buckets.

1.4.2 Value at Risk

Overview

Early in the 1990s, the CEO of JP Morgan, Dennis Weatherstone, wanted to know the bank's risk at the end of the trading day. He asked his staff to devise a risk measure that would yield one number that could be communicated to him at 4:15 pm each trading day and would give him an accurate view of the bank's risk. He wanted to have a sense of the risk of bad outcomes that would create problems for the bank. The risk measure that Weatherstone eventually received from his staff was an estimate of the bank's trading portfolio *Value at Risk.*

Value at Risk (VaR) is an attempt to provide a single number for senior management that summarizes the total risk in a portfolio of financial assets. The VaR calculation is aimed at making a statement of the form:

We are c percent certain not to lose more than V dollars in the next m days.

The VaR is therefore an estimate **V** dollars, with a given degree of confidence **c** percent, of how much one can lose from one's portfolio over a given time horizon m days. The VaR is usually calculated assuming normal market circumstances, meaning that extreme market conditions such as market crashes are not considered. In addition, it is attributable to changes in the market price of financial instruments.

An Example:

What does that definition mean in English? Suppose that we are investment managers with portfolio positions in interest rate swaps. We need an assessment of what we can expect the worst case to be for the position overnight with a 95% degree of confidence. The VaR number gives us this measurement. For example, the portfolio manager might have $ 100 million under management and an overnight-95% confidence interval VaR of 4 million. This means that 19 times out of 20 his/her biggest loss should be less than 4 million. Hopefully, he is making money instead of losing money.

Advantages of Parallelizing VaR Computation
In order to be effective, VaR analysis has to be highly accurate. For this reason, the input data sets have to be as complete as possible. This means that the input data sets have to cover a wide range of historical data plus the most current data, e.g. data from the last 10 years daily interest rate up to today's interest rate.

Another important factor to achieve higher accuracy is by performing more simulation. However, the trade-off is that it will take much longer time for the process to complete.

In most cases, VaR analysis has to be completed by 4:15 pm of each trading day. This gives a short period of time for VaR analysis to finish processing all the data, including data that is only available on that day itself. Therefore, VaR analysis, calculation, and simulation have to be done as fast as possible.

The solution to both requirements, i.e. high accuracy and fast processing time, is to use parallel computing in VaR analysis, calculation, and simulation. Parallel computing enables more processing power to be combined and work together to achieve a faster, more accurate, and more efficient VaR analysis.

1.5 Research and Related Work

The approaches to VaR can be essentially divided into two groups. The first is based on local valuation of the financial instruments. The best example from this group is the **variance-covariance** method. This method relies on the assumption of normality of the market risk factors and delta-equivalent presentation of the portfolio. Due to these assumptions it is possible to show that there is an explicit formula for the quantile and, therefore, for VaR. From a computational point of view, this method consists of relatively simple computations that are based on matrix algebra. For example, under this method options are specified by their delta-equivalent representation, i.e., the change of the option is given by the product of delta of the option and the change of the underlying instrument.

One disadvantage of this method is that it uses a large matrix involving literally thousands of points which are then compared against each other to provide variance-covariance. Another disadvantage is that at-the-money options may exhibit high convexity that is not taken into account by the variance-covariance method.

The second group is based on the simulation approach that performs full valuation of the financial instruments. The best examples are **historical simulation** and **Monte Carlo simulation** methods. The historical simulation method is similar to the Monte Carlo method except that historical changes in prices are used rather than simulated changes based on a stochastic model. Although both methods solve the convexity problem of the variance-covariance approach mentioned earlier, the Monte Carlo method is more flexible and has certain advantages over the historical simulation method. Some of the criticism of the historical simulation method is due to:

- The assumption that the past represents the immediate future, i.e., the distribution is assumed to be stationary.
- The high sensitivity of the results with respect to the length of the time horizon.
- The problem with collecting consistent historical data.

1.6 Report Organization

This report is organized into 4 major sections. Section 1 of this report presents an introduction to the project, including motivation, scope and objective of this project. In addition, a summary of financial concepts is presented to familiarize readers with the problem domain.

Section 2 explains implementation details of this project. This includes description of problem definition and overview of the system. Subsequently, an explanation of standard Monte Carlo and Quasi Monte Carlo algorithm used in this project is presented. Furthermore, parallel implementation of both algorithms is explained in this section.

Section 3 shows and discusses results achieved in this project. A comparative analysis of various implementations is presented in details. In addition, the details of the system specification and input data used are given in this section.

Finally, Section 4 presents a conclusion from this project. In addition, it describes some possible future work for this project. A glossary of financial terms used in this report is attached for the reader's benefit.

2 Implementation Details

2.1 System Overview

In section 1.3, VaR was explained conceptually. In this section, we provide a more precise problem definition and an overview of the system. The problem definition can be reiterated as:

I have a portfolio of p assets, I am c% certain not to lose more than V dollars in the next m days.

The important terms are defined as:

- **p:** Interest Rate Swaps: Notional Principal & Maturity for each asset
- **c:** Confidence Level (usually 99%)
- **V:** Value (Money) at Risk
- **m:** Horizon in number of days.

In addition, the input to the program must specify the *historical data* on Interest Rates, *Par Swap Yield Rates*, *LIBOR Rates*, the number of simulation paths N, and the δT i.e. the granularity. The financial terms are explained in the glossary while the use of the input in the simulation is presented in the following section. Figure 3 is an overview of the system. In the next section, we delve into the subtleties.

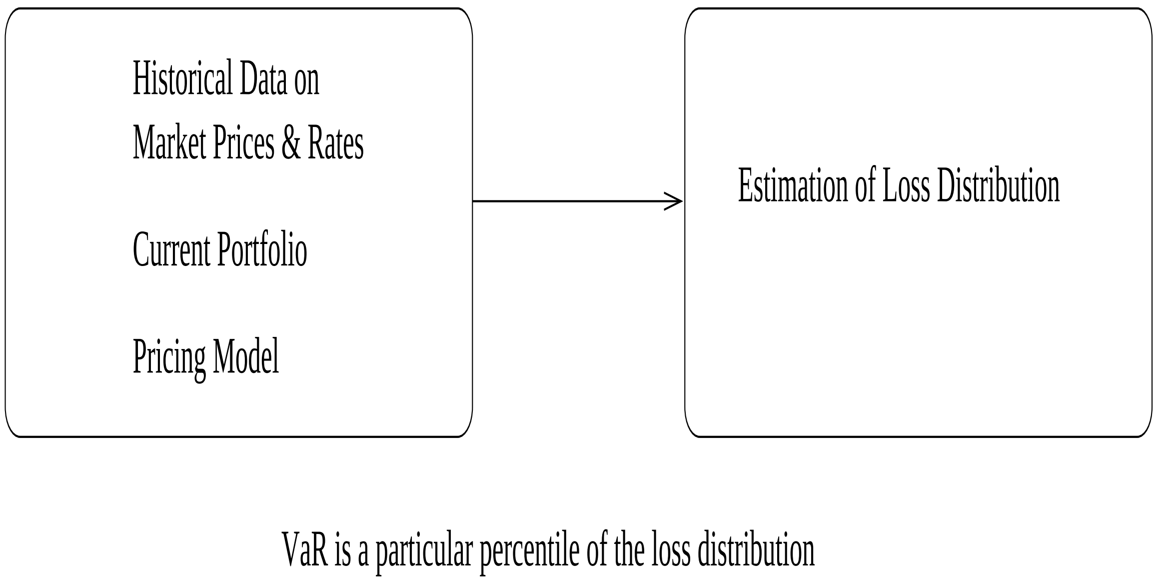

Figure 3: System Overview

2.2 Standard Monte Carlo Algorithm

Monte Carlo simulation is the random generation of values for uncertain variables over and over to simulate a model. In the context of VaR, the uncertain variable is *Interest Rates*.
The approach of Monte Carlo VaR is to randomly generate scenarios based upon some assumed joint probability distribution for the risk factors. Historical market data is used to infer statistical characteristics such as volatilities

and correlations for the assumed distribution.

For example, for a given portfolio we might assume that each risk factor is log normally distributed with a mean equal to today's value for that factor. Recent market data would then be analyzed to infer volatility for each risk factor as well as a correlation for each pair of risk factors.

Once an assumed joint distribution is specified, standard techniques for generating correlated random numbers are used to select a set of scenarios for the interest rates. In this way, the selected scenarios are literally drawn from the assumed distribution. They reflect the statistical characteristics-volatilities and correlations-inferred from the historical data.

The fundamental algorithm for determining the algorithm proceeds in the following main steps:

- *Step 1: Preprocessing* Preprocessing entails a series of functions performed prior to the monte carlo simulation. The more notable preprocessing procedures include:
  - The **Variance-Covariance Matrix** C is inferred from historical data. The historical information is in most cases annual and the canonical technique of *linear interpolation* is used to derive more information from the available data. If Interest rate available for 6 months, 1 yr, 1.5yrs, 2 yrs, 2.5yrs and 3yrs, then C is a 6x6 matrix.
  - **Discount Factors** for all the desired maturities are derived from the par swap yield rates using *bootstrapping techniques* and linear interpolation. Discount Factor for a time period gives the present value today of 1 dollar to be received at the end of that time period.
  - **Cholesky Decomposition** of the matrix C is performed to get matrix P such that:

$$C = P^T P$$

- *Step 2: Monte Carlo Simulation* Monte Carlo simulation works by computing the Interest Rates for the different time periods in the future over many sample paths. The pseudocode in figure 4 contains an algorithm for computing interest rates at n points in the future following geometric brownian motion.

- *Step 3: Valuation of Portfolio* The valuation of the portfolio takes place via the following two steps:
  - *Calculation of cash flows:* The cash flows for the different assets in the portfolio are calculated.

    Cash Flow for Asset 1 at ith installment = (Notational Principal of Asset 1) x (Corresponding Interest Rate for ith installment on the mth day(horizon) derived from the simulation)

```
Compute C from historical Data
Perform Cholesky Decomposition of C to get P
Initialize Interest rates for the different maturities to today's rates
Compute Discount Factors
For N iterations
  For m
    For i = 6 months, 1 year etc. (maturities)
      New Interest Rate [i] = Estimate from (Old Interest Rate [i], delta T, sqrt(Variance-Covariance Entry,
                              P from Cholesky Decomposition, , Random Number

      End For
  End For
  Estimate Discounted Cash Flows for all assets in the portfolio
  Sum the Discounted Cash Flows to obtain the portfolio value
End For
Sort N Portfolio Values
Estimate the VaR corresponding to the confidence level
```

Figure 4: Monte Carlo Simulation pseudocode for VaR Estimation

The simulations give the floating rate. The fixed rate is derived from the input. For instance, let us consider a portfolio of two assets: A1 and A2. A1 matures in 1 year while A2 matures in 6 months. After 6 months from today, using the above formula, say that I lose x dollars on A1 and gain y dollars on A2. After 12 months from today, A2 would have matured and say I gain z dollars on A1. Then the net cash flow for 6 months is $-x+y$ while the net cash flow for 12 months is z.

– *Discounting the Net Cash Flow to obtain the Net Present Value* In the preprocessing step, the discount factors were obtained from the par swap yield rates. The discount factors are used to discount the the corresponding net cash flows to obtain the net present values. The sum of the net present values is the value of the portfolio.

$$portfolio = \sum NetPresentValue = (NetCashFlow)x(DiscountFactor)$$

- *Step 4: VaR Estimation* The portfolio valuations for the various Monte Carlo simulations are sorted to estimate the loss distribution.The desired percentile (confidence level c) of the distribution provides an estimate of VaR

This approach has four sources of specification error:

1. The standard distribution we assume for risk factors may imperfectly reflect the "true" distribution q.

2. Because we estimate the Variance-Covariance matrix from a limited set of historical data, those estimates entail a sampling error

3. Because market conditions are non-stationary (vary over time), the historical data upon which we base volatility and correlation estimates may imperfectly reflect today's market conditions.

4. Imperfections in the random number generator we use for selecting scenarios may introduce a bias.

We can reduce first source of error by basing volatility and correlation estimates on a lot of historical data. We can reduce the second by basing those estimates on only the most recent data. We cannot reduce both the errors.

In addition to specification error, Monte Carlo VaR also entails convergence error. There is, however, no limit to the number of scenarios that can be used with Monte Carlo VaR. Accordingly, this error can be made as small as desired.

2.3 Quasi Monte Carlo using Faure Sequence

> "It is suggested that, instead of clinging to vague concepts of randomness, it might be better to aim at working with sequences making no pretence of random origin, but so devised as to give the best possible guarantee of accuracy in computations. Such methods of computation could be described as quasi-Monte Carlo." -Zaremba

Monte Carlo is a well-established technique used widely in other fields. However, the ordinary random sampling approach used in Monte Carlo simulation is relatively inefficient. So, the recent introduction of quasi-random number sequences in finance applications, coupled with enhanced computing power, has boosted the use of Monte Carlo simulation.

In recent years there has been a renewed interest in Monte Carlo methods. This is due to the flexibility of the method in handling increasingly complex financial instruments and the advent of the powerful workstations has significantly reduced the execution time required for the simulation. For example, Monte Carlo methods are normally more efficient than competing procedures if there are several state variables or other complexities. The disadvantages of Monte Carlo methods are that the error term is probabilistic and that it can be computationally burdensome to achieve a high level of accuracy. To reduce the computational burden of standard (or crude) Monte Carlo methods a variety of variance reduction methods have been proposed.

The quasi-Monte Carlo method was pioneered in computational physics where large-scale Monte Carlo calculations are common. The basic idea is to use numbers that are deterministic rather than random. These deterministic sequences can be used in place of the usual random (or pseudo random) numbers to obtain faster convergence with known error bounds. The use of quasi-Monte Carlo methods leads to more efficient numerical procedures with the added benefit of deterministic error bounds.

The basic idea of using quasi-Monte Carlo is to use a set of points that are carefully selected in a deterministic fashion. The term quasi-random is misleading, as these numbers have no random component. A particular class of quasi-random sequences is *Faure sequence.*

Faure sequences [8] in dimension $s >= 2$ are (0; s) sequences that can be defined for any prime base $b >= s$. We usually fix the base b as the smallest prime $b >= s$ for this value yields the lowest bound on the discrepancy [9]. We define the Pascal matrix C by:
$C = c_{uv}$
$where:$
$c_{uv} = \begin{cases} \frac{v}{u!(v-u)!}, u <= v \\ 0, u > v \end{cases}$ for all integers u,v > 0
For every integer k >= 1, the k-th power of C matrix is the matrix:

$$C^k = (c_{uv}{}^{(k)})\ where\ c_{uv}{}^{(k)} = k^{v-u} c_{uv}$$

for all integers u,v >= 0

We also have $C^0 = I$, the identity matrix. We define Faure (0,m,s)-net in base b in the following way:

For n = 0, 1,, b^m - 1, we consider the b-ary expansion:

$$n = \sum_{r=0}^{\infty} x_r(n) b^r \text{ with all } x_r(n) \in Z_b$$

For all 0≤n< b^m ,0≤i< S, and 0≤jm, we put

$$y_j^{(i)}(n) = (C^i x(n)(modb))_j = \sum_{r=j}^{m-1} c_{jr}^{(i)} x_r(n)(modb)) \in Z_b$$

$$y_j^{(i)}(n) = \sum_{j=0}^{m-1} y_j^{(i)}(n) b^{-j-1} \text{ for } 0 \leq n < b^m \text{ and } 0 \leq i < s$$

We define the Faure (0,m,s)-net y = y(0),..., y(b^m - 1) in base b where:

$$y(n) = y^{(0)(n),\ldots,y^{s-1}(n))} \in I^s \text{ for n = 0, 1, } ldots\text{, b}^m\text{-1.}$$

We know that we can shuffle a Faure sequence y in base b with b-ary Gray code G without affecting its uniformity properties. We define the shuffled Faure (0,m,s)-net

$$y^{-} = y^{-}(0), \ldots, y^{-}(b^m - 1)$$

in base b by

$$y^{-} = y(G(n)) \text{ for n=0,1,..., } b^m \text{ - 1.}$$

In a Faure sequence, we have $y\bar{}(0) = y(0) = 0$ If we denote by $q_n \in 0, \ldots, m-1$, the index of the first digit different from $b-1$ in the b-ary expansion of n, then for all 0≤n< b^m ,0≤i< S, and 0≤jm, we have

$$y_j^{-(i)}(n+1) = (C^i x(G(n+1))(mod b))_j = y_j^{-(i)}(n) + c_{jq_n}^{(i)}(mod b) \in Z_b$$

in base b by

$$y^{-(i)}(n+1) = \sum_{j=0}^{m-1} y_j^{-(i)}(n+1)b^{-j-1}$$

This method requires $O(ms)$ time to compute successively each point of $y\bar{}$. Hence, we have the same Faure-sequence, though the order of the elements in the sequence varies due to the shuffle. The following routines available in the public domain [9] have been interfaced with the software. The implementation details are described below:

- The initialization function **GrayFaureInit(dim, skip)**. This routine is called once to specify the parameters of the Faure sequence and to intialize the generator. It returns the value of the base b defined as the smallest prime greater or equal to the dimension s, or 0 if the parameters **dim** and **skip** do not satisfy the following conditions. The dimension s is specified by the integer parameter dim that must be at least 2. The parameter skip is an integer that must satisfy $0 \leq \quad skip < \lfloor log(LONG_M AX)/log(b) \rfloor$ where **LONG_MAX** is the maximum value of the machine's LONG (as defined in limits.h). The parameter **skip** allows deleting the first b skip points of the sequence.

- The generation function **GrayFaureNext**. Each call to this routine returns a pointer to an (s+1)dimensional LONG vector that defines the next point of the sequence, or the NULL pointer if the index of that point can not be represented suitably (i.e. the index is greater than $b^{\lfloor log(LONG\_MAX)/log(b) \rfloor}$.If we denote by (v[0]; : : : ; v[s]) the vector returned by this function, the next point of the Faure sequence is (v[0]/v[s] ,. . ., v[s-1]/v[s]) $\in I^s$. Computation is basically done by table lookup in integer arithmetic (avoiding any multiplication, division, exponentiation, and modulo operation). This function is called once in each iteration.

2.4 Simulation using Distributed Memory

MPI is the international standard for message passing between processes in high performance distributed and parallel computing applications. It is a basic technology that allows programmers to write parallel programs for a wide range of distributed and parallel high performance computing systems, from plug-in PC products, through shared memory server clusters, up to the very high end massively parallel processing systems.

The basic issue in cluster and distributed memory parallel computing is communication. Each parallel program is composed of a number of processes running on different processors that, by sharing the workload, can complete a complex calculation more quickly than one processor is able. Message passing is the technique used for communication between processes in parallel programs.

Message passing has been used on a large number of high performance parallel systems, and although there have been many variations of message passing the basic concept is well understood. In the last decade, significant applications have been cast in the message-passing paradigm. We have tried to extract this fact to our advantage for an efficient and accurate calculation of the VaR of a portfolio.

We have implemented the parallel version of the serial code. This version is suited for distributed memory and uses the MPI interface to achieve node level parallelism. The idea behind this is the large number of iterations in (quasi) Monte Carlo is independent of each other. Therefore these can be run in parallel as well on separate processors and can be combined to achieve the final answer. Hence, we have put this idea to work so that independent processors compute their own results independent of each other and then pass on to a specified processor for consolidation to get the final output as in figure 5. Processors divide work equally amongst themselves equally. Each of them sends the result back to processor 0 for consolidation and to get the final answer. It essentially becomes a master-slave configuration wherein the slave nodes report back to the master after doing the assigned task. This approach allows us to run more number of iterations and hence, achieve higher accuracy in our results.

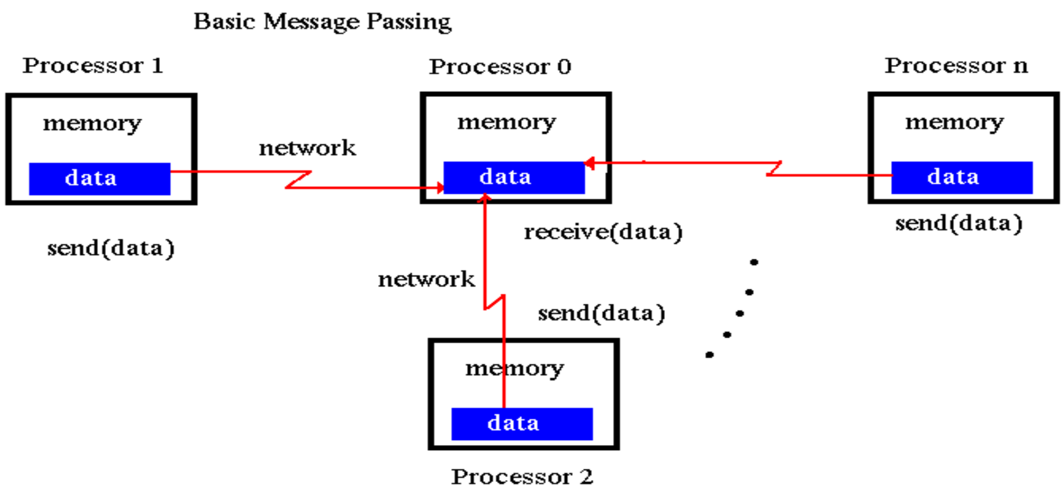

Figure 5: Message Passing Interface

2.5 Simulation using Shared Memory

It has long been recognized that the two necessary elements for achieving parallel scalability in application performance are scalable hardware and scalable software. Both these elements have existed for some time in their lowest common denominator form. Specifically, scalable hardware in the context of physically distributed memories connected through a scalable interconnection network such as a multistage interconnect, k-ary n-cube, or fat tree, have been commercially available since the 1980's. When such systems are built without any provision for cache coherence they are essentially *zeroth order* scalable architectures. In other words, such systems provide only a scalable interconnection network, and the burden of scalability then falls on the software. As a result scalable software for such systems will exist, at some level, only in a message-passing model. Message passing is the native model for these architectures, and higher-level models can only be built on top of message passing.

In the shared memory model every processor has direct access to the memory of every other processor in the system. This means a processor can directly load or store any shared address. The programmer also can declare certain pieces of memory to be private to the processor. This provides a very simple yet powerful model to the programmer for expressing and managing parallelism in an application.

At its most elemental level, OpenMP is a set of compiler directives and callable runtime library routines that extend Fortran (and separately, C and C++) to express shared memory parallelism. The base language is left unspecified and vendors may implement OpenMP in any Fortran compiler. Naturally, Fortan 90 and Fortran 95 require the OpenMP implementation to include additional semantics over Fortran 77 in order to support pointers and allocatables.

Although simple and effective, loop level parallelism is usually limited in its scalability because it typically leaves some constant fraction of sequential work in the program that by Amdahl's law can quickly overtake the gains from parallel execution. It is important however to distinguish between the type of parallelism (e.g. loop level versus coarse grained) and the programming model. The type of parallelism exposed in a program is dependent on the algorithm and data structures employed and not on the programming model. Therefore given a parallel algorithm and a scalable shared-memory architecture, a shared memory implementation scales as well as a message passing implementation.

We implemented the shared memory version of the code using OpenMP. The main objective while implementing a shared memory version was to parallelize the main loop, which is the most computationally expensive and is the heart of the program. We have attempted to maximumize loop-level parallelism. We were able to successfully run and test the shared memory version of our code. The results obtained and discussion of those results is presented in section 3.

3 Results and Discussion

In this section we elucidate on the results of the experiments conducted to examine the performance of the software and present a critique of the same.

3.1 System Specification

We have used the Beowulf cluster at MIT for obtaining our results. The system specifications are as follows:

- beowulf.lcs.mit.edu
- Nodes: 9 (1 frontend + 8 compute)
- Processor: 2 x Athlon MP 1800+ (1.53GHz) on each node
- Memory: 1 GB PC2100 DDR EEC Registered on each node
- Disks: 2 x IBM 120GXP 40GB on frontend, 1 on compute nodes.
- Network: Nodes are connected by 3 fast ethernet switches (100 Mbps), joined by channel bonding, yielding a total of 300Mbps bandwidth
- Peak performance: 3 FPU, 1.53GHz processors, with peak of 4.59GFlops. 4.59 x 18 giving 82.62 GFlops peak for the beowulf. (This is the peak for SINGLE precision floating point calculations).
- Cost: 15000 US Dollars

For further information, please visit the link http://beowulf.lcs.mit.edu/18.337/beowulf.html. This link contains detailed explanation on the usage of beowulf cluster.

3.2 Usage Instructions

The standard Monte Carlo and the quasi Monte Carlo versions have been incorporated into a single file. Please execute with the *-faure* option to run the quasi algorithm for sequential, distributed and shared versions. For further details please visit the following link: http://beowulf.lcs.mit.edu/18.337/beowulf.html

3.2.1 Sequential Version

File: serial.c
Compile: gcc -lm -o $< outputfile >$ serial.c
Execute: $< outputfile >$ [-faure]

3.2.2 MPI Version

File: distributed.c
Compile: mpicc -lm -o $< outputfile >$ distributed.c
Execute: mpirun n1-8 -np X $< outputfile >$ [-faure]
where X is the number of processes you want to spawn

3.2.3 OpenMP Version

File: shared.c
Compile: source /usr/bin/iccvars.sh
icc -O -openmp -o $< outputfile >$ shared.c
Execute: ./ $< outputfile >$ $< noofthreads >$ [-faure]

3.3 Experimental Data Sources

The two main sources of data for experimentation are the following:

- The information for 15 years for par swap yields is taken from Federal Reserve Statistical Release: http://www.federalreserve.gov/releases/H15/data.htm. This data helped us to calculate the discount factors which was required to discount the cash flows in future to obtain their value in present time.
- The data for LIBOR is obtained from IndiaInfoline.com: http://www.indiainfoline.com/bisc/intn/ch07.html. These rates are also used in the *bootStrapping* procedure.

3.4 Output for Varying Input Parameters

The input file consists of the following parameters:

- 4 assets (all interest rate swaps)
- Maturities of the 4 assets: 1, 3, 3 and 5 years
- Notional Principal amounts: 10, 20, 20 and 50 million dollars
- Confidence level: 99%
- Horizon: 10 days
- δT (granularity): 1 day
- Iterations: 100,000
- Par Swap Yields for the past 15 years
- LIBOR (London Inter-Bank Offer Rate) for 1, 2, 3, 4, 5, 6 and 12 months

We executed the sequential version of the code by varying few of the input parameters that affect the computation time most.

Firstly we varied the number of days of the horizon from 10 days to 100 days. We found that the computation time varied linearly with horizon. This behaviour was expected because every *predicted* value of the interest rate for any day depends on the interest rates for all previous days. Hence, an increase/decrease in the number of days of horizon would cause a similar change in the computation done by the program.

We notice that the time taken for the standard Monte Carlo version is always greater than the quasi-Monte Carlo version. The rationale behind this accounts for the difference in methodologies of the different versions. The standard Monte Carlo computes a random number for every execution of the **inner loop** in figure 4 corresponding to n. On the other hand, in a quasi-Monte Carlo version, the faure sequence is generated only once in the preprocessing step and shuffled once for every execution of the **outer loop** corresponding to N. Thus, the one time overhead in the quasi version coupled with the shuffling amounts to less computation compared to the generation of a random number for each execution of the inner loop in the standard Monte Carlo version. This effect will be more striking in a real-world application that employs larger input sets. Figure 6 represents the graph obtained for varying m. Secondly, we varied

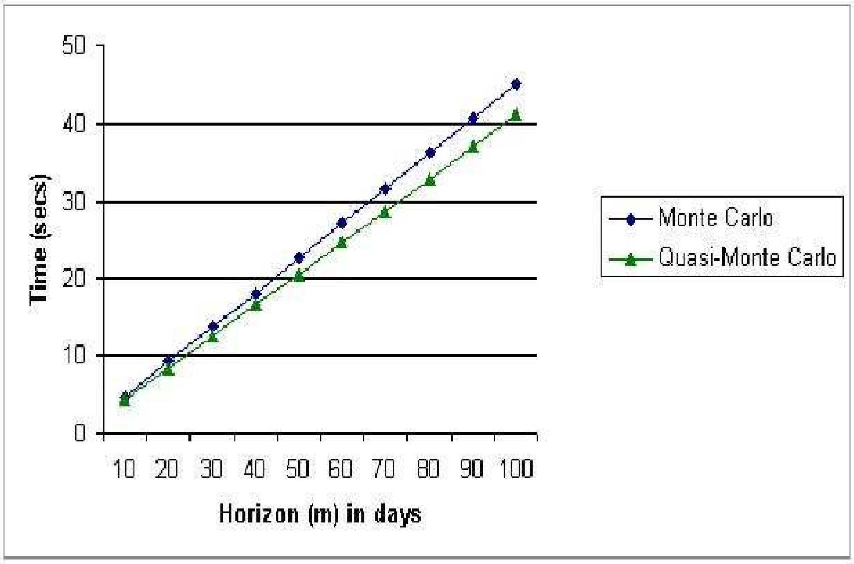

Figure 6: Horizon(m) in days Vs Time

the number of iterations but kept the number of days in horizon constant. We see that the time taken has a linear relationship with the number of iterations. Another observation is that here also the quasi version performs better than the standard Monte Carlo version. The reasoning for this behavior is the same as explained before. Figure 7 represents the graph obtained when we vary the number of iterations. Figure 8 shows the corresponding VaR values. We observe that the VaR values tend to remain within expected error bounds.

Thirdly, we varied the confidence level to get corresponding values of VaR keeping the other factors unchanged. Figure 9 shows the variation of VaR with changing confidence levels. We observe that the VaR values for standard Monte Carlo series are comparable to those of the quasi-Monte Carlo. Hence, quasi-Monte Carlo methods provide a way to improve the accuracy and reliability of Monte Carlo simulation. It has resulted in better convergence and known error

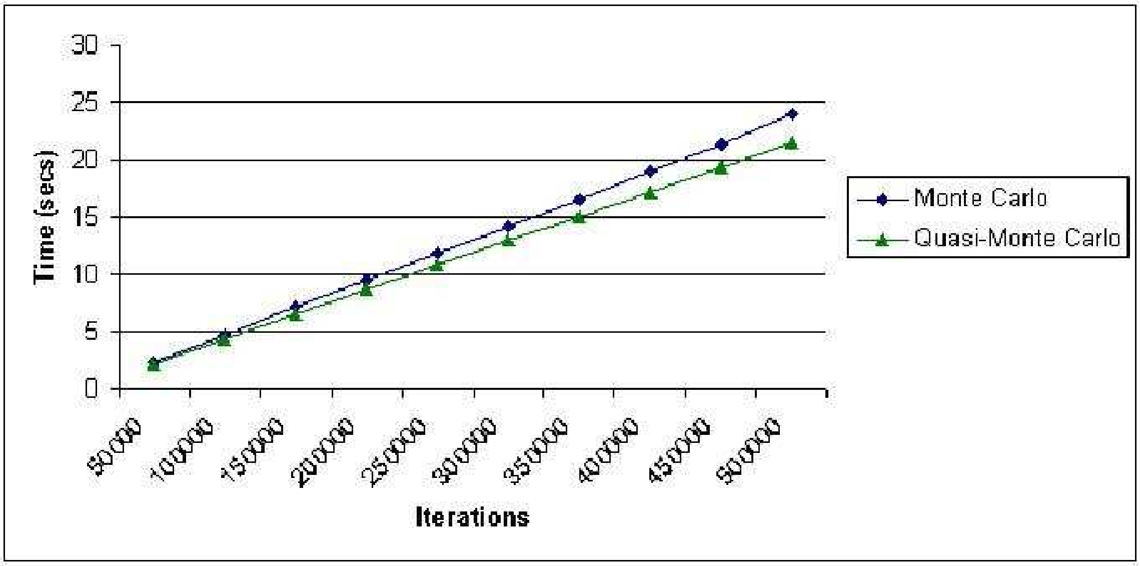

Figure 7: No.of Iterations Vs Time

bounds. This can be explained on the basis that in quasi-Monte Carlo, there is less randomness, thus it models the problem a lot more realistically. Hence, we see that the models follow each other very closely, and the Monte Carlo version should ideally converge to get values obtained by the quasi-version.

3.5 Comparative Analysis

3.5.1 Serial Vs Parallel

This section focuses on the results obtained by parallelizing the code. The Monte Carlo and its quasi versions were run in parallel. We varied the number of processors for the code and the results obtained are illustrated in figure 10. We observe that as we increase the number of processors, the time taken to compute the VaR reduces. This is well explained by the fact that with increase in number of processors, the per-processor computation reduces to a fraction of what it was previously. However, the code now encounters the extra cost of communication between them. As all processors send their results to processor 0, there might be a point when the number of processors will increase but the time taken to run the code will also increase. This will occur because the computation on each processor will be less taxing than the communication cost. This scenario may or may not occur when we look at the real-world applications that involve complex portfolios requiring intensive number crunching operations.

The speedup of the versions with respect to the number of processors is shown in figure 11. This is derived from figure 10. However, it helps us gain a

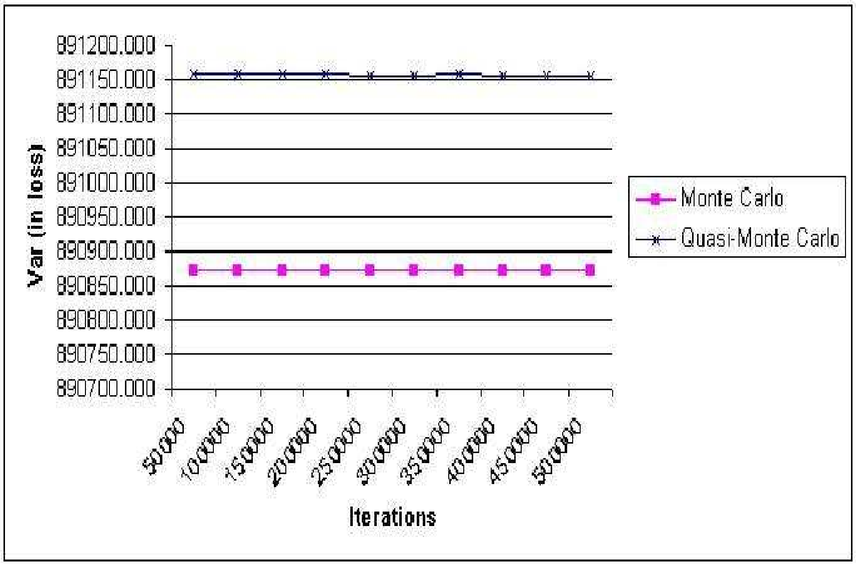

Figure 8: No.of Iterations Vs VaR

clearer picture of the speedup we achieve by increasing the number of processors.

3.5.2 Shared Memory Vs Distributed Memory

We implemented the shared memory version of our code for pure experimental reasons, but found interesting results. Figure 12 illustrates the results obtained by varying the number of threads in the shared memory model. When we set the number of threads to 1, we do not exploit any parallelism in the program. However, in the Monte Carlo version, we observe a sharp increase in the time taken when the number of threads is made 2. This time taken reduces as we further increase the number of threads. Even with 10 threads the time taken is a lot higher than taken for 1 thread. This can be explained by realizing that in the Monte Carlo version the threads have no need to share memory while exploiting loop-level parallelism. Hence, the compiler-initiated parallelism slows down the execution time rather than boosting it. In the quasi version of the code, the increase in execution time when the threads are increased to 2 is not sharp. With further increase in the number of threads, the time decreases and becomes less than the time to execute the program with one thread. This is possible because, the Faure sequence generated is shared between the different threads. However, the speedup is still not encouraging, leading to the conclusion that this kind of computation is not suited for shared memory architecture.Figure 13 illustrates the speed up in the different versions.

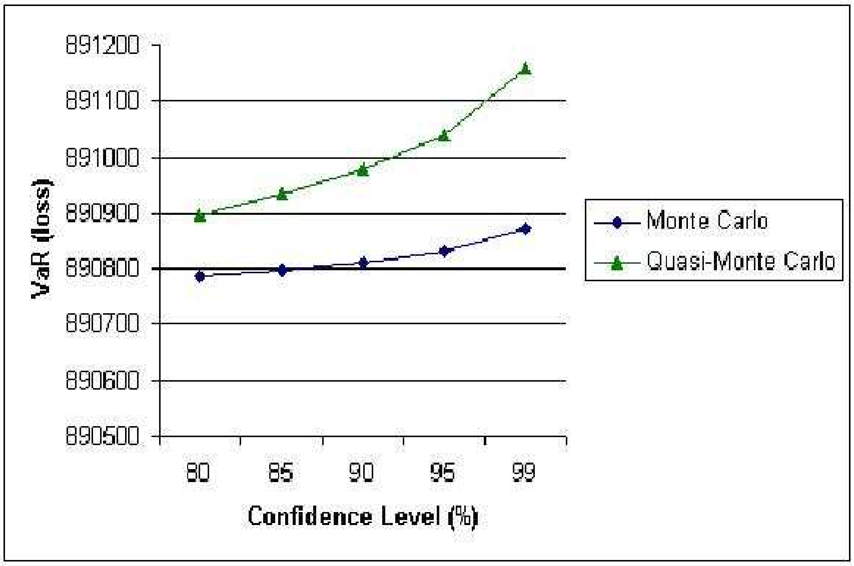

Figure 9: Confidence Level(c) Vs VaR

4 Conclusion and Future Work

The project has been an exciting maiden venture into the realm of finance. Requiring a detailed understanding of relevant financial engineering concepts, the project has been a challenging experience in rapidly scaling the learning curve. It can confidently be said that the following objectives that have been accomplished:

- The standard Monte Carlo has been implemented. The implementation entailed the following algorithms:
  - Computation of Variance-Covariance Matrix using the historical data on Interest Rates.
  - Cholesky Decomposition of the Variance-Covariance Matrix
  - Bootstrapping techniques for Discount Factor Evaluation
  - Valuation of portfolio to get Net Present Value
- Deployment of Faure Sequence to get a quasi variation.
- Shuffling of Faure Sequence using Gray Code to introduce randomness.
- Validation of Results with varying Input sets.
- An analysis of results substantiates that significant **Speed-Up** has been achieved in the parallel version.

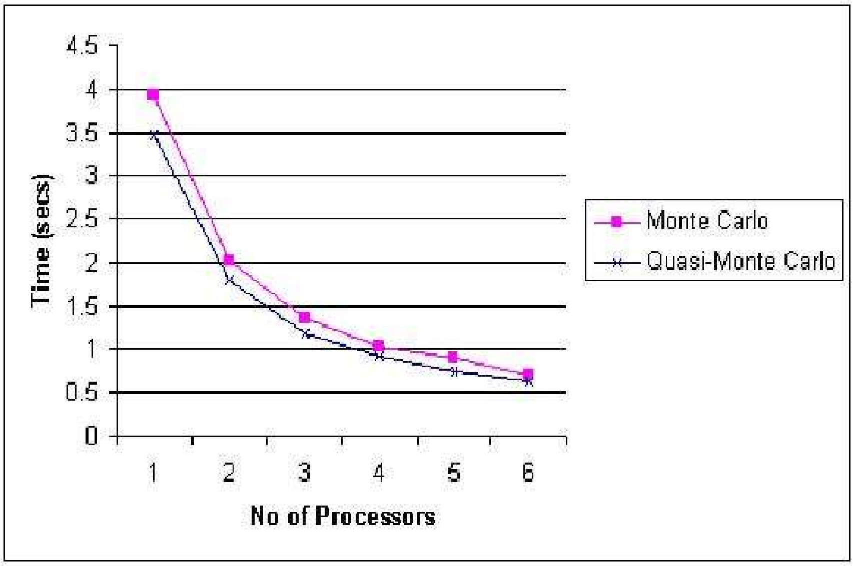

Figure 10: No.of Processors Vs Time

In conclusion, various versions of the Monte Carlo technique for computation of **Value At Risk**, including standard, quasi-random using faure sequence, quasi-random shuffled using gray code, shared memory and distributed memory have been successfully implemented and tested during the course of the project.

As a part of future work, the scope of the portfolio can be extended to entail other interest deivatives including swaptions, caps and floors.

Acknowledgement

We are very fortunate in having the support and guidance of Associate Professor Ng Kah Hwa, Centre for Financial Engineering Director, and Master of Financial Engineering. Program Director.

Glossary

- **Bootstrapping :**
  Is the technique to initiate a sample or process. It can use a piece of data to generate or infer other data.

- **Cap:**
  An upper limit on the interest rate on a floating-rate note (FRN) or an adjustable-rate mortgage (ARM).

- **Cash Cow :**
  Is a security, investment or a project that generates or throws off lots of funds due to contacts or contracts. Sometimes, the basis for this asset is an

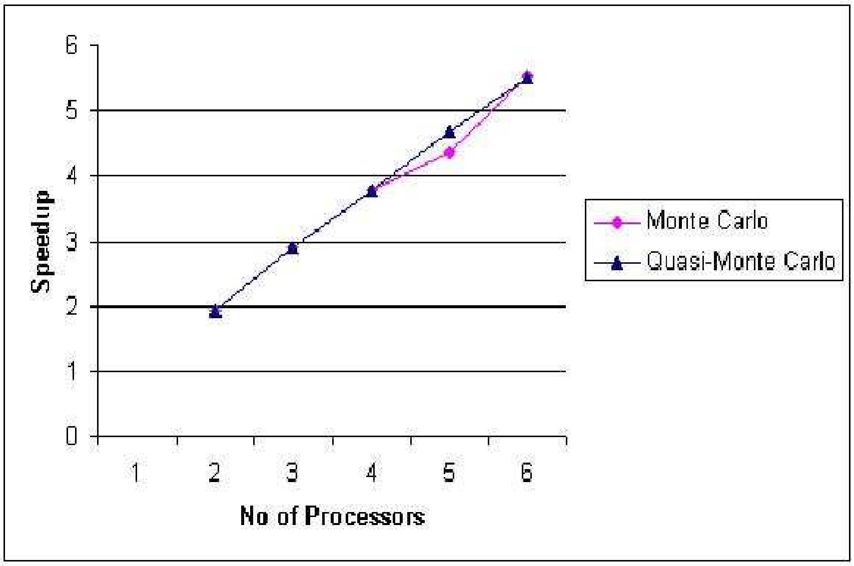

Figure 11: No.of Processors Vs SpeedUp

excellent customer, a monopolistic market position, or special advantage afforded by patents, licenses, or other economic properties. This cash flow can be used for many purposes. Typically, this situation constitutes the fundamental franchise of a business.

- **Derivative instrument:**
A type of financial instrument that derives its value from the value of other financial instruments.

- **Discount Rate:**
Is the interest rate used for adjusting for the time value of money for Net Present Value, Option Pricing or other Market Models.

- **Floating Rate:**
Refers to the condition whereby exchange rates are relatively free to change. It can also refer to an interest rate which changes relatively quickly or frequently.

- **Floor:**
The price at which a stop order is activated (when the price drops low enough to activate such an order).

- **Interest Rate:**
The monthly effective interest rate.

- **Interest Rate Swap:**
A binding agreement between counterparties to exchange periodic interest payments on some predetermined dollar principal, which is called the

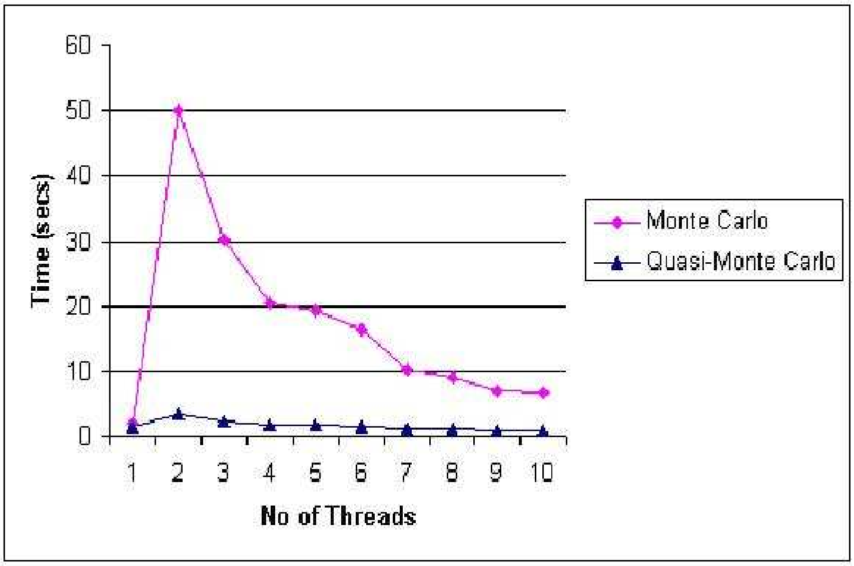

Figure 12: No.of Threads Vs Time

notional principal amount. For example, one party will pay fixed and receive variable.

- **LIBOR:**
  London Interbank Offered Rate.

- **Maturity Date :**
  The date when the issuer returns the final face value of a bond to the buyer.

- **Monte Carlo Simulation:**
  A technique of simulation that uses many randomly or "pseudo-randomly" generated scenarios.

- **Monte Carlo Value at Risk :**
  A technique for estimating value at risk

- **Par yield curve :**
  The yield curve of bonds selling at par, or face, value.

- **Portfolio:**
  A collection of investments, real and/or financial.

- **Return:**
  The change in the value of a portfolio over an evaluation period, including any distributions made from the portfolio during that period.

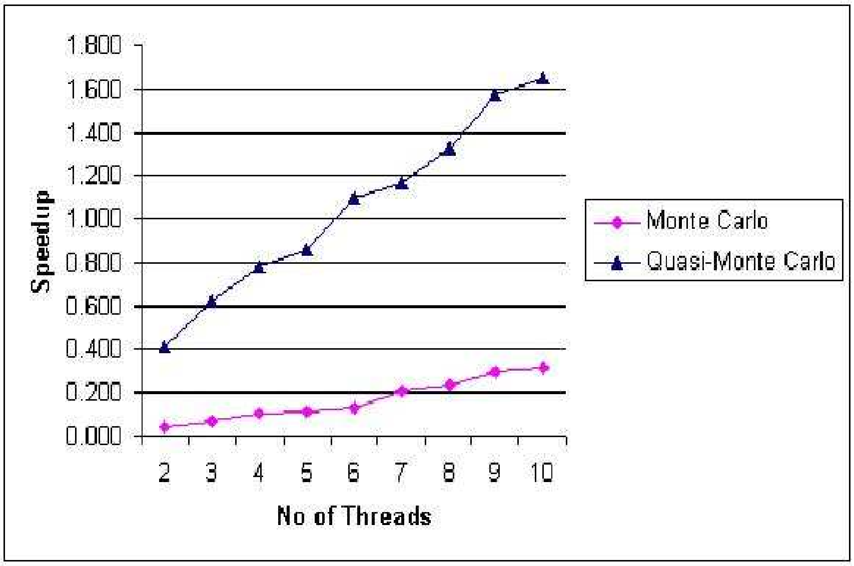

Figure 13: No.of Threads Vs SpeedUp

- **Risk:**
  Often defined as the standard deviation of the return on total investment. Degree of uncertainty of return on an asset.

- **Risk Management:**
  The process of identifying and evaluating risks and selecting and managing techniques to adapt to risk exposures.

- **Swap:**
  An arrangement in which two entities lend to each other on different terms, e.g., in different currencies, and/or at different interest rates, fixed or floating.

- **Swaption:**
  Options on interest rate swaps. The buyer of a swaption has the right to enter into an interest rate swap agreement by some specified date in the future. The swaption agreement will specify whether the buyer of the swaption will be a fixed-rate receiver or a fixed-rate payer. The writer of the swaption becomes the counterparty to the swap if the buyer exercises.

- **Value-at-Risk:**
  The probability of portfolio losses exceeding some specified proportion based on a statistical analysis of historical market price trends, correlations, and volatilities.

- **Volatility:**
  A measure of risk based on the standard deviation of the asset return.

There are volatility indexes. Such as a scale of 1-9; a higher rating means higher risk.

References

[1] http://www.top500.org/ : TOP500 Supercomputer Sites

[2] http://www.charlesschwab.com/ : Charles Schwab USA

[3] http://www.netlib.org/benchmark/ : LINPACK Benchmark Programs and Reports

[4] http://www.citibank.com.sg/ : Citibank Singapore

[5] http://www.banksa.com.au/banksa/treasury/ : BankSA Treasury

[6] http://www.finpipe.com/: Financial Pipeline

[7] http://www.cs.bsu.edu/homepages/gokten/USCMathFinance.pdf

[8] Faure, H. 1982. Discr'epance de suites associ'ees 'a un syst'eme de num'eration (en dimension s). Acta Arith. 41, 337–351.

[9] THIEMARD, ERIC. Economic generation of Low-Discrepancy Sequences with a b-ary Gray code.

[10] Stavros A. Zenois *Parallel Monte Carlo simulation of mortgage-backed securities*. Chapter 14 Financial Optimization, Cambridge Press.

[11] Corwin Joy, Phelim P. Boyle and Ken Seng Tan. *Quasi-Monte carlo Methods in Numerical Finance*. Management Science/Vol. 42, No.6, June 1996.

[12] David R. Smith.*A Simple Method for Pricing Interest Rate Swaptions*. Financial Analysts Journal/May-June 1991.

[13] Peter A. Abken.*An Empirical Evaluation of Value at Risk by Scenario Simulation* Office of the Comptroller of the Currency, E&PA Working Paper 2000-3, March 2000.

[14] John C. Hull.*Options, Futures, and Other Derivatives*. 3rd Edition Prentice Hall