Direct simulation of price particles for option pricing using Monte-Carlo

Ruppa K. Thulasiram[†*], **Parimala Thulasiram**[†], **and K.P.J. Reddy**[‡]
[†]Department of Computer Science,
University of Manitoba, Winnipeg, MB R3T 2N2, Canada
[‡]Department of Aerospace Engineering, Indian Institute of Science,
Bangalore, India 560012

Abstract

Absence of closed form solutions for many financial models has given rise to numerical and simulation techniques in the recent past. In the current study direct simulation, which is one of the popular approaches in aerospace engineering is used for studying applications in finance, especially option pricing.

Monte Carlo (MC) is one of the popular simulation approaches for approximating the value of the quantity under question. However, the slow convergence rate, $O(N^{-1/2})$ for N number of samples of the MC method has motivated research in Quasi Monte-Carlo (QMC) techniques. QMC methods use low discrepancy (LD) sequences that provide faster, more accurate results than MC methods. In this paper, we focus on the parallelization of the QMC method on a heterogeneous network of workstations (HNOWs) for option pricing. HNOWs are machines with different processing capabilities and have distinct execution time for the same task. It is, therefore, vital to allocate and schedule the tasks depending on the performance and resources of these machines. We present an adaptive, distributed QMC algorithm for option pricing, taking into account the performances of both processors and communications. The algorithm will distribute data and computations based on the architectural features of the available processors at run time. We implement the algorithm using mpC, an extension of ANSI C language for parallel computation on heterogeneous networks. We compare and analyze the performance results with different parallel implementations. The results of our algorithm demonstrate a good performance on heterogenous parallel platforms.

1. INTRODUCTION

We introduce in this section some basic motivation for molecular simulation in finance. Our aim is in the design and development of a Quasi Monte-Carlo algorithm to price options.

1.1 Need for Simulation

The main idea of microscopic simulation (MS) methodology is to study complex systems by representing each of the microscopic elements individually on a computer and simulating the behavior of the entire system, keeping track of all the elements and their interactions in each time period without making any simplifying assumptions. Throughout the simulation, global, or ”macroscopic”, variables that are of interest such as temperature and pressure can be recorded, and their dynamics can be investigated. Such complex systems generally do not yield to analytical treatment.

The main advantage of the microscopic simulation is that, unlike analytical methods, it does not force one to make simplifying assumptions for the sake of tractability. Thus, virtually any system with heterogeneous elements and complicated interactions can be investigated. Complexity arises from interaction and disorder, from the cooperation and competition of the basic units. Financial markets certainly are complex systems, judged both their multiscale structure and interaction of sub structures. Millions of investors frequent many different markets organized by exchanges for stocks, bonds, commodities etc. Change in investment decisions (by an individual investors (micro)) affects the prices of the traded assets, and these price changes influence decisions in turn (at macro level, for example

*author for correspondence:tulsi@cs.umanitoba.ca change in the interest rate etc). The scales of activity in a given system varies widely in finance, like any other scientific domains.

When attempting to draw parallel between physics and financial markets, an important source of concern is the complexity of human behavior which is at the origin of the individual trades. However, nowadays a significant fraction on many markets is performed by computer programs (for simpler decisions); complex situations could be handled by autonomic computing and no longer by human operators. Still, human behavior is largely an unexplained phenomenon to account for. Furthermore, if we make abstraction of the trading volume, an operator only has the possibility to buy or to sell, or to stay out of the market.

Microscopic simulation with the knowledge of statistical physics has a great potential as a research tool in finance and in economics. In this study, we try to quantify a mechanism to link the microscopic behavior of high temperature gases and investor behavior. There is no doubt that financial markets, in which a multitude of heterogeneous quasi-rational investors operate, are very complex systems. MS allows the modeling and investigation of such systems without unrealistic simplifying assumptions. The unrealistic assumptions can be relaxed one by one, and the effect of each simplifying assumption on the results can be investigated. One additional component of challenge in this study is handling the multiscale of phenomena from microscopic (individual investor behavior) to macroscopic (market behavior that in turn affects individual investor's behavior) phenomenon. With these observations of similarities between microscopic behavior of a physical system, it becomes easier to apply the computational tools and techniques from high temperature physics to financial systems.

1.2 Simulation in Finance

The Black-Scholes model of option pricing is a continuous time model similar to the Navier-Stokes equations. The solution describe the option value in terms of the macroscopic quantities, viz. asset price, strike price, volatility, etc. a collection of asset prices. The option pricing system can then be quantified based on the particle properties like investment capital, investment growth, greediness of the investor etc. All the macroscopic quantities are related to the microscopic quantities through summations or averages. The continuum assumption is valid when the characteristic length (the length scale which is of interest) namely the period of the option contract is much larger than the mean free path, which is the frequency of interaction among individual investors or frequency at which individual assets are traded.

This can be quantified by saying that the continuum assumption is valid only in the low Knudsen number regimes. The market (fluid)) in this state is said to be in equilibrium. This means that over the length scale of interest (option periods contract, for example, three months or longer), several particle collisions (overall exchanges between investors) occur. This causes the market to reach an equilibrium condition. Such a situation implies that the statistical fluctuations in the macroscopic quantities are miniscule, and can be neglected. However, this assumption of continuous market does not hold under extreme conditions (like very low pressures at higher altitudes for physical systems or financial product that are of no interests to investors). Such situations correspond to a high mean free path of the molecules in the physical system leading to less collisions; and infrequent interaction among the investors means the market may not reach an equilibrium condition. Such extremely infrequent integrations may be of interest to long dated options such as bond options.

Accurate modeling of such pricing problem is possible by treating it at a microscopic level as an aggregation of particles based on the more fundamental laws of physics. This is a computationally very challenging task especially since the number of particles (investors) to handle is very very large. Still, Direct Simulation is a more computationally feasible approach for analyzing such problems. Bird [3] proposed a research method to study gas flows in upper atmosphere, where the air almost cease to be a continuum media. The scheme adopts a statistical/physical, rather than mathematical, description of the phenomenon under study. Hence, the name Direct Simulation as it can be used to directly simulate the financial system rather than modeling it mathematically. For example, microscopic structure in physics is equivalent to the random walk taken by price particles of various assets in the market. There are many particle models in physics and details of these models can be found in Bird [4]. The methodology extracts meaningful data from the simulations by sampling a large number of repetitive random walks. This is the essence of a Monte Carlo methodology.

The ability of miscroscopic simulation to capture individual phenomenon within a market is an added advantage over other approaches such as binomial lattice or fast Fourier transform. Moreover, Monte-Carlo simulation is easy for parallel implementation than the above technique. MC simulation can use as many processors as there are in the system, one for each random walk.

The slow convergence rate, for N number of samples of the MC method has motivated research in Quasi Monte-Carlo (QMC) techniques. QMC methods use low discrepancy (LD) sequences that provide faster, more accurate results than MC methods. In this paper, we focus on the parallelization of the QMC method on a heterogeneous network of workstations (HNOWs) for option pricing.

1.3 Monte-Carlo Technique

Monte Carlo is a widely used microscopic simulation method in science, as well as to simulate asset prices as random walk [7] with the help of these asset prices, pricing of financial options were carried out. This is highly desirable since it is very difficult to get closed from (analytical) solution for many financial models. Since Boyle [7] introduced the MC simulation in pricing option in 1977, the literature in this area has grown very rapidly. For example, Hull and White [17] employed MC method in stochastic volatility application and obtained more accurate results than using Black-Scholes model [6]; the latter often overprices options about ten percent and the error will be exaggerated as the time to maturity increase. Schwartz and Torous [29] use MC method to simulate the stochastic process of prepayment behavior of mortgage holders and the results matched closely to that actually observed. Fu [12] gives introductory details concerning the use of Monte Carlo simulation techniques for options pricing. Even though the prevailing belief that American-style options cannot be valued efficiently in a simulation model, Tilley [32], Grant et al. [15], as well as Broadie and Glasserman [9] and some others, have proposed MC methods for American-style options and obtained acceptable results. Examples about valuing exotic options can be found in [19]. The literature on MC methods in valuing options keeps growing. More examples can be found in [8, 14]. The traditional MC methods have been shown to be a powerful and flexible tool in computational finance [27].

While the traditional MC methods are widely applied in option pricing, their disadvantages are well known. In particular, for some complex problems which require a large number of replications to obtain precise results, a traditional MC method using pseudo-random numbers can be quite slow because its convergence rate is only $O(N^{-1/2})$ where N is the number of samples. Different variance reduction techniques have been developed for increasing the efficiency of the traditional MC simulation, such as control variates, antithetic variates, stratified sampling, Latin hypercube sampling, moment matching methods, and importance sampling. For detail about these techniques, please refer to [14]. Another technique for speeding up the MC methods and obtaining more accurate result is to use low discrepancy (LD) sequences instead of random sequences. The use of LD sequences in MC method leads to what is known as quasi-Monte Carlo (QMC) method. The error bounds in QMC methods are the order of $(\log N)^d \cdot N^{-1}$ where d is the problem dimension and N is the number of simulations.

Birge [5] reported how quasi-Monte Carlo sequences can be used in option pricing in 1994 and demonstrated improved estimates through both analytical and empirical evidence. In 1995, Paskov and Traub [26] performed tests about two low-discrepancy algorithms (Sobol and Halton) and two randomized algorithms (classical Monte Carlo and Monte Carlo combined with antithetic variables) on Collateralized Mortgage Obligation (CMO). They obtained more accurate approximations with QMC methods than with traditional MC methods and concluded that for the CMO the Sobol sequence is superior to other algorithms. Acworth et al. [1] compared some traditional MC methods and QMC sequences in option pricing and drew the similar conclusion. Boyle et al. [8] also found that QMC outperforms traditional MC and Sobol sequence outperforms other sequences. Galanti and Jung [13] used both pseudo-random sequences and LD sequences (Sobol, Halton and Faure) with MC simulations to value some complex options and demonstrated that LD sequences are a viable alternative to random sequences and the Sobol sequence exhibits better convergence properties than others.

Due to the replicative nature, QMC simulation often consumes large amount of computing time. Solution on a sequential computer will require hours and may be even days depending on the size of the problem [27]. In financial markets, there is a high premium on rapid solution. Any rapid solution in information processing can be translated into potential gains. Therefore, parallel computing is an ideal choice since it provides a solution for large computational problems in a reasonable computational time using more than one processing units. QMC simulations are well suited to parallel computing since it is an embarrassingly parallel problem (no communication between processors [31]). We can employ many processors to simulate various random walks, then average these values to produce a final answer. In this scenario, the whole simulation time is minimized. There are three parallel techniques of using random sequence in literature, namely, *Leapfrog, Blocking,* and *Parameterization*. In recent times, the above three schemes have been proposed for parallelizing LD sequences (see, for example [10, 23, 25, 28]). Srinivasan [31] compared the effectiveness of these three strategies in pricing financial derivatives and concluded that *blocking* is the most promising method if there are a large number of processors running at unequal speeds. However, the disadvantages of *blocking* scheme are well pronounced. First, if a processor consumes more LD elements than it was assigned, then the subsequences could overlap. Second, if a processor consumes less LD elements than it was assigned, then some elements will be wasted. The final result will be the same as the sequential computation that use the same LD sequence with some "gaps". Third, if processors run at unequal speeds, the fastest processor will finish its task first and wait for the slowest processor at synchronization point; then the overall computation time will be determined by the time elapsed on the slowest processor. It is wasteful to do this since the most powerful processors will idle most of the time. The parallel computing cannot take full advantage of the potential computing power. Hence, a good parallel MC algorithm should distribute computations based on the actual performance of processors at the moment of the execution of the program. The more powerful a processor, the more tasks it will be assigned. That is, data, computations, and communications should be distributed unevenly among processors to provide the best execution performance. In this research, we present such an adaptive QMC algorithm for European-style option pricing, taking into account performances of both processors and communications. The algorithm is implemented in mpC, which is a relatively new programming language for implementing parallel computing on heterogeneous networks.

The rest of this paper is organized as follows. In the following section we provide a basic introduction to the world of options and option pricing. In section 3, we present a theoretical background to both Monte-carlo and Quasi Monte- Carlo methods followed by a review of use of these methods for option pricing in section 4. In section 5, we present a partition algorithm for QMC.We describe our parallel algorithm in section 6 and compare the experimental results to its carefully written MPI counterparts in section 7 followed by conclusion in section 8.

2. BACKGROUND

An option is an agreement between two parties to buy or sell an asset at a certain time in the future for a certain price. There are two types of options:

- *Call Option*: A call option [18] is a contract that gives the right to its holder (i.e. buyer) without creating an obligation, to buy a prespecified underlying asset at a predetermined price. Usually this right is created for a specific time period, e.g., six months, or more.
- *Put Option*: A put option [18] is a contract that gives its holder the right without creating the obligation, to sell a prespecified underlying asset at a predetermined price.

If the option can be exercised only at its expiration (i.e. the underlying asset can be bought/sold only at the end of the life of the option) the option is referred to as a European style Call/Put Option (Or European Call/Put). If it can be exercised any date before its maturity, the option is referred to as an American style Call/Put option (or American Call/Put). We use the following notation: K is the strike price; T is the life time of (expiration date) of the option; S_t is the stock price at time t; r is the interest rate; μ is the drift rate of the underlying asset (a measure of the average rate of growth of the asset price); σ is the volatility of the stock; C denotes the option value. Here is an example to illustrate the concept of option pricing. Suppose an investor enters into a call option contract to buy a stock at price K after six months. After six months, suppose the stock price is S_T. If $S_T > K$ then he can exercise his option by buying the stock at price K and by immediately selling in the market to make a profit of $S_T - K$. On the other hand, If $S_T \leq K$ he is not obligated to be buy the stock. Hence, we see a call option to buy the stock at time T at price K will get payoff $(S_T - K)^+$, where $(S_T - K)^+ \equiv \max(0; S_T - K)$.

3. MONTE CARLO AND QUASI-MONTE CARLO METHODS

In general, Monte Carlo (MC) and Quasi-Monte Carlo (QMC) methods are applied to estimate the integral of function $f(x)$ over the $[0, 1]^d$ unit hypercube where d is the dimension of the hypercube, the solution space.

$$I = \int_{[0,1]^d} f(x)dx \tag{1}$$

In MC methods, I is estimated by evaluating $f(x)$ at N independent points randomly chosen from a uniform random distribution over $[0, 1]^d$ and then evaluating the average

$$\hat{I} = \frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{2}$$

From the law of large numbers, $\hat{I} \to I$ as $N \to \infty$. The standard deviation is

$$\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(f(x_i) - I)^2}. \tag{3}$$

Therefore, the error of MC methods is proportional to $\frac{1}{N-1}$.

QMC methods compute the above integral based on low-discrepancy (LD) sequences. The elements in a LD sequence are "uniformly" chosen from $[0, 1]^d$ rather than "randomly". The discrepancy is a measure to evaluate the uniformity of points over $[0, 1]^d$. Let fqng be a sequence in $[0, 1]^d$, the discrepancy D_N^* of q_n is defined as follows, using Niederreiter's notation [24].

$$D_N^*(q_n) = \sup_{B\in[0,1)^d} |\frac{A(B, q_n)}{N} - v_d(B)| \tag{4}$$

where B is a subcube of $[0, 1]^d$ containing the origin, $A(B,\ q_n)$ is the number of points in q_n that fall into B, and $v_d(B)$ is the d-dimensional Lebesgue measure of B. The elements of q_n is said uniformly distributed if its discrepancy $D_N^* \to 0$ as $N \to \infty$. From the theory of uniform distribution sequences [?], the estimate of the integral using a uniformly distributed sequence $\{q_n\}$ is $\hat{I} = \frac{1}{N}\sum_{n=1}^{N} f(q_n)$, as $N \to \infty$ then $\hat{I} \to I$. The integration error bound is given by the Koksman-Hlawka inequality:

$$|I - \frac{1}{N}\sum_{n=1}^{N} f(q_n)| \leq V(f)D_N^*(q_n) \tag{5}$$

where $V(f)$ is the variation of the function in the sense of Hardy and Krause (please see [20]), which is assumed to be finite. The inequality suggests a smaller error can be obtained by using sequences with smaller discrepancy. The discrepancy of many uniformly distributed sequences satisfies $O((log\ N)^d/N)$. These sequences are called lowdiscrepancy (LD) sequences [24]. Inequality (5) shows that the estimates using a LD sequence satisfy the deterministic error bound $O((log\ N)^d/N)$.

4. MONTE CARLO SIMULATIONS FOR OPTION PRICING

Under the risk-neutral measure, the price of a fairly valued European call option is the expectation of the payoff $E[e^{-rT}(S_T - K)^+]$. In order to compute the expectation, Black and Scholes [6] modeled the stochastic process generating the price of a non-dividend-paying stock as geometric Brownian motion:

$$dS_t = \mu S_t dt + \sigma S_t dW_t, \tag{6}$$

where W is a standard Wiener Process, also known as Brownian motion. Under the risk-neutral measure, the drift μ is set to $\mu = r$.

To simulate the path followed by S, suppose the life of the option has been divided into n short intervals of length Δt $(\Delta t = T/n)$, the updating of the stock price at $t + \Delta t$ from t is [18]:

$$S_{t+\Delta t} - S_t = rS_t\Delta t + \sigma S_t Z\sqrt{\Delta t}, \tag{7}$$

where Z is a standard random variable, i.e., $Z \sim (0; 1)$. This enables the value of $S_{\Delta t}$ to be calculated from initial value S_0 at time Δt, the value at time $2\Delta t$ to be calculated from $S_{\Delta t}$, and so on. Hence, a completed path for S has been constructed.

In practice, in order to avoid discretization errors, it is usual to simulate ln S rather than S. From Itô's lemma, the process followed by $\ln\ S$ of (7) is given by [18]:

$$d\ln S = (r - \frac{\sigma^2}{2})dt + \sigma dz \tag{8}$$

so that

$$\ln S_{t+\Delta t} - \ln S_t = (r - \frac{\sigma^2}{2})dt + \sigma Z\sqrt{\Delta t} \tag{9}$$

or equivalently:

$$S_{t+\Delta t} = S_t \exp[(r - \sigma^2/2)\Delta t + \sigma\sqrt{\Delta t}Z]. \tag{10}$$

Substituting independent samples $Z_1; \ . \ . \ . \ ;Z_n$ from the normal distribution into (10) yields independent samples $S_T^{(i)}$, $i = 1; \ . \ . \ . \ ; \ n$, of the stock price at expiry time T. Hence, the option value is given by

$$C = \frac{1}{n}\sum_{i=1}^{n} C_i = \frac{1}{n}\sum_{i=1}^{n} e^{-rT}\max\{S_T^{(i)} - K, 0.0\}. \tag{11}$$

The QMC simulations follow the same steps as the MC simulations, except that the pseudo-random numbers are replaced by LD sequences. The basic LD sequences known in literature are Halton [16], Sobol [30] and Faure [11]. Niederreiter [24] proposed a general principles of generating LD sequences. In finance, several examples [1, 8, 13, 26] have shown that the Sobol sequence is superior to others. For example, Galanti and Jung [13] observed that "the Sobol sequence outperforms the Faure sequence, and the Faure marginally outperforms the Halton sequence. At 15,000 simulations, the random sequence exhibits an error of 0:07%; the Halton and Faure sequences have errors of 0:1%; and the Sobol sequence has an error of 0:03%. These errors decrease as the number of simulations increases". In this research, we use Sobol sequence in our experiments. The generator used for generating the Sobol sequence comes from GNU Scientific Library (http://www.gnu.org/software/gsl/).

5. PARALLELIZATION METHOD

An ideal parallel QMC algorithm should distribute computation tasks to processors proportional to processors' actual computing powers. To achieve this, the algorithm must collect information of the entire computing space and compute relative performances of actual processors in the run time; otherwise, the load of processors will be unbalanced, resulting in overall poor performance. Traditional parallel programming tool cannot implement such parallel algorithm except mpC. As a relatively new parallel programming tool, mpC offers a very convenient way to obtain the statistical information of the computing space and the power of each processor. Information about mpC is provided in section 6. Using mpC, a programmer can explicitly specify the uneven distribution of computations across parallel processors. In addition, mpC system has its own mapping algorithms to ensure each process to perform computations at the speed proportional to the volume of computation it performs. Hence, these two way mappings lead to a more balanced and faster parallel program.

Having known the power of each processors, the volumes of computation assigned to each processor can be computed by performing the following partition algorithm (algorithm 1).

Suppose each simulation consumes q elements of the given LD sequence, and processor i has t_i tasks, then the whole number of elements will be consumed by processor i is $B = t_i \times q$. Note that B is not necessarily N/P where N is the number of points and P is the number of processors. Hence, the LD sequence is partitioned into uneven blocks. This partition of LD sequence is somewhat like the general *blocking* scheme in MPI, however, it is superior to general *blocking* scheme, in which the LD sequence is partitioned into equal size or the burden is on the programmer to determine B. In the literature, B is usually chosen to be greater than N/P to avoid overlapping of subsequences. In general, one does not sure the exact number of points of a sequence which a processor will consume. Because

Algorithm 1 Partition

```
1:  Given $N$ is the total tasks, $p$ is the number of processors, *power*, is the *i*-th processor's power.
    then, the tasks assigned to the *i*-th processor is:
        $task_i = [N \times \frac{power_i}{\sum_{i=0}^{p} power_i}]$    (12)
2:  After step1, if there are tasks left, then assign it to host processor.
```

in QMC simulations, there are no "safe" stopping rules [14]. Without experimentation, it is difficulty to know the number of points of a LD sequence needed to achieve a desired accuracy. Unlike the traditional MC methods, one can use a standard error estimated from a number of simulations to determine if a desire precision is reached. Hence, the LD numbers must be generated more than needed. So the discrepancy of the contiguous points in a block might be different from the discrepancy of the points which is enough for a sequential run. Therefore, there is a chance that the result produced from the *blocking* scheme of QMC is not the same as that of a sequential computing [31]. With the help of mpC, we can use the same number of points as that used in sequential program, and there is no overlapping issue in sub-sequences and that the sequential run and the parallel run results match.

6. PARALLEL QMC ALGORITHM WITH MPC

We implemented our algorithm using mpC, which is an extension of the ANSI C language. The mpC programming tool is specially designed for writing high performance parallel computation programs on common networks of heterogeneous computers. The current mpC programming environment contains a compiler, run-time support system (RTSS), libraries and a command-line user interface. For detail information about mpC language, programming environment and samples, please refer to [2, 21, 22] or online website available at http://www.ispras.ru/~mpc/.

The mpC offers a mechanism where other parallel programming languages don't have, through which a programmer can describe (dynamically) a virtual network topology for his application. In run time, the mpC environment will map the virtual network to real executing network based on information about performances of processors and links of the real network. A *network*, a basic notation in mpC, can be taken as a user-defined data object in general programming language. Allocating *network* objects and discarding them is performed in similar way as allocating data objects and discarding them.

Suppose we have m processors doing QMC simulations. One processor (host processor) distributes tasks to the other processors and collects their results. In QMC simulations, each simulation is independent, there is no communications between processors except with the host processor. Hence, we see QMC simulations has a *star* topology where the host processor is the central node and the other processors (nodes) connected directly to the central node. The computing and communication are based on this topology. We define a *star* topology for our MC simulations. The following *network* declaration describes such kind of topology.

```
/*line 1 */    nettype Star(m, p[m]) {
/*line 2 */      coord I = m;
/*line 3 */      node {
/*line 4 */        I ≥  0: p[I];
/*line 5 */        };
/*line 6 */      link {
/*line 7 */        I > 0: [0] ↔ [I − 1];
/*line 8 */        };
/*line 9 */      parent [0];
/*line 10 */     };
```

The header (Line 1) introduces parameters of the topology *Star*, namely, the integer parameter m and the vector parameter p consisting of m integers. Vector p is used to store the relative performances of the m processors. Line 2 introduces a coordinate declaration declaring the coordinate system to which virtual processors are related. The integer coordinate variable I ranges from 0 to $m - 1$. Lines 3–5 are node declaration. Line 4 stands for the predicate for all $I \geq 0$, the virtual processor, whose relative performance is specified by the value of $p[I]$, is related to the point with coordinate $[I]$, and so on. Lines 6–8 are link declaration, which specify links between virtual processors. Line 7 stands for the predicate for $I > 0$ and $I < m$ there exists undirected links connecting virtual processors with coordinates [0] and [I-1]. Line 9 is a parent declaration. It specifies that the parent has coordinate [0].

After *network* is created in mpC program, it executes the rest of computations and communications. A call to library function *MPC_Processors_static_info* returns the number of actual processors and their relative performances. Based on relative performances of actual processors, algorithm *Partition* computes the number of simulations should be computed by every actual processor. Then the subsequences for each processor can be determined using the method mentioned in section 5. Further, the steps to follow are: **a)** broadcast option's arguments; **b)** scatter subsequences, **c)** perform sequential computing on each processor, and finally **d)** the host processor gather the results from each processor and produce the final result. The pseudo-code of this procedure is presented in algorithm 2.

Table 1. Relative performance of 7 heterogeneous workstations

processor	p1	p2	p3	p4	p5	p6	p7
power	883	930	879	999	959	870	899

Algorithm 2 Parallel Quasi-Monte Carlo Algorithm

```
1:  Initialize the parameters such as $S, r, K, T, \sigma$,
2:  Compute relative performances of actual processors
3:  Partition tasks (call partition algorithm)
4:  Assign elements of LD sequence to processors according to their tasks (Scatter blocks)
5:  Broadcast options' parameters
6:  Execute the sequential algorithm on each processor
7:  Gather the results of each processors
8:  Produce the final result on host processor.
```

7. EXPERIMENT RESULTS

We run 1,000,000 simulations on seven distributed memory machines running SunOS 5.8. Their relative performance of the seven processors were detected during the creation of the virtual parallel machines (table 1). Timing is obtained via the mpC wall clock function, *MPC_Wtime()*. By using Algorithm 1, the volume of computations of each processor is computed. The computing time is 3.3054 seconds. The sequential runtime of the same computing is 20.13 seconds. Table 2 lists the computing time of different combinations of simulations and processors. In the table, 1P denotes one processor, 2Ps denotes two processors, and so on. Note that the running time of the mpC program substantially depends on the network load.

To get a better estimation of our mpC program, we developed two versions of the MPI programs: 1) static distribution tasks among processors (general blocking (BK) scheme); 2) a manager-workers (MW) approach which simulates load balancing scheme to some extent.

Using general blocking (BK) scheme, the tasks (N) and the LD sequence are equally distributed among the m processors. In this experiment, usually the number of processors must be a factor of the number of simulations (i.e. N/m is an integer); otherwise, the result will be different from a sequential algorithm's result using the same arguments. Since our focus here is to compare the performance of different algorithms, we do not discuss the computing results.

When designing a manager-workers algorithm, the manager in the algorithm acts as a dispatcher; it dispatches tasks and gather results; it doesn't take any tasks to do. However, from experiments, if the manager takes tasks like workers, the algorithm will be more efficient than the general algorithm. This is because the manager processor consumes part of whole tasks, the workers will be assigned fewer tasks; the communications decrease; hence the whole computing time decrease. So in this study, the manager algorithm is a little different from usual manager algorithm: manager assigns tasks to itself.

In the manager-worker scheme, if the manager assigns one task per request from worker, the communication will be an overhead for large simulations, results in very poor performance (see Table 3).

Table 3 gives the execution time for 1,000,000 simulations on 7 processor on three different implementations: mpC, blocking scheme (BK) with MPI and manager-worker scheme (MW) with MPI. The table also indicates the number of tasks distributed to each processor in each of the three

Table 2. Time to do QMC simulations (Sec) (x=100000)

	number of processors						
sim.	**1P**	**2P**	**3P**	**4P**	**5P**	**6P**	**7P**
1x	0.02	0.13	0.24	0.35	0.44	0.60	0.43
2x	2.24	2.06	1.95	1.76	1.32	1.04	0.76
3x	5.26	3.18	2.05	1.94	1.87	1.40	1.21
4x	7.11	5.41	3.25	2.44	2.10	1.78	1.34
5x	9.48	6.71	4.30	2.71	2.34	2.19	1.68
6x	11.89	8.10	5.92	3.84	2.64	2.45	1.89
7x	14.22	8.12	6.51	4.11	2.99	2.91	2.28
8x	17.25	9.60	7.29	4.97	3.07	3.11	3.07
9x	19.11	10.25	8.02	5.62	3.31	3.29	3.16
10x	20.13	11.60	8.49	5.71	4.82	3.53	3.31

schemes. In BK scheme, since the task distribution is static, each processor receives 142857 tasks. In the MW scheme, each processor is assigned one task per request. It's interesting to see in the table that though processor 4 has better computing performance than processor 1, processor 1 is assigned more number of tasks in total. This is done by the scheduler. The MW scheme in MPI does not take the performance of the processors into consideration. Finally, with mpC, we notice that the processor are given tasks according to their performance. Processor 4 get the most number of tasks, 155631, since it is the fastest; while processor 6 gets 135535 number of tasks since it is the slowest. The computing time also shows the mpC implementation is the most efficient while the MW scheme is very inefficient since the workload is unbalanced and communications dominate the whole computing (sending requests and results to manager, receiving tasks from manager).

Table 3. Time to do 1,000,000 simulations using three schemes

Processors'	**Processor's**		**MPI**	
ID	**performance**	**mpC**	**BK**	**MW**
1	883	137563	142857	500000
2	930	144882	142857	133597
3	879	136937	142857	81659
4	999	155631	142857	77377
5	959	149400	142857	74096
6	870	135535	142857	67771
7	899	140052	142857	65500
time (Sec.)		3.31	4.67	95.48

In the MW program, instead of assigning one task per request, we tried to assign arbitrary number of tasks to each request, thereby simulating mpC to some extent. Let pt be the number of tasks assigned to a processor per request. When $pt = 1$, the computing time is 95.48 seconds. We experimented from $pt = 100$ till $pt = 30000$, with 100 tasks in each increment. Figure 1 illustrates the test results.

From the test, we found in some cases the MW performs better than mpC program and in some cases does not. For example, when $pt = 500$, the runtime is 2:70764 seconds; and when $pt = 4300$, the run time is 3:81360 seconds. We cannot find any trend about the value of pt that will give the best performance. Hence, when designing MW algorithm, it would be tedious to find a suitable value for pt which is user-defined. In addition, the MW scheme is not portable when some conditions are changed. For example, if the number of processors is changed or the workload of some processors is changed, the pt value must also be changed manually. Unlike mpC program, the number of tasks will be automatically changed based on the processors' performance when the number of processors is changed.

8. CONCLUSION

In this paper, we presented a distributed QMC method for option pricing that is adaptable to the heterogenous network of workstations. We used the Sobol LD sequence for the QMC Technique and implemented the algorithm in mpC. The algorithm distributes the data depending on the architectural

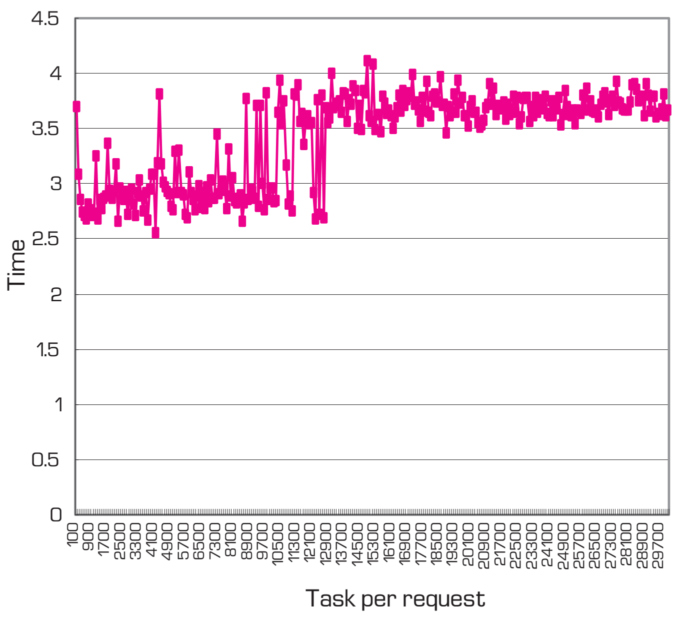

Figure 1. Manager-workers scheme with different amount tasks per request

features of the machines, and takes into account the actual performances of both processors and communication links. In comparison to other implementation, especially with MPI, the speedup exhibited by our algorithm presented in this paper are promising. An outstanding feature of using mpC is that a programmer can specify the topology of the application under study and mpC system can map the topology to real network system based on processors' processing speeds and network bandwidths in run time.

ACKNOWLEDGEMENT

The authors are thankful to Profs. John Anderson and Rasit Eskicioglu for their consent with the use of heterogeneous machines. Thanks are also due to Mr. Gilbert Detillieux for his help all through the installation of mpC. The first two authors acknowledge the partial financial support from Natural Sciences and Engineering Research Council (NSERC) Canada through Discovery Grants and the Research Grants Program (URGP) of the University of Manitoba.

REFERENCES

[1] P. Acworth, M. Broadie, and P. Glasserman. A comparison of some Monte Carlo and quasi Monte Carlo methods for option pricing. In H. Niederreiter, P. Hellekalek, G. Larcher, and P. Zinterhof, editors, *Monte Carlo and Quasi-Monte Carlo Methods 1996*, pages 1–18. Springer-Verlag, Berlin, 1998.

[2] D. Arapov, V. Ivannikov, A. Kalinov, A. Lastovetsky, I. Ledovskih, and T. Lewis. A parallel language for modular distributed programming. In *Proceedings of the 2nd Aizu International Symposium on Parallel Algorithms/Architectures Synthesis*, pages 248– 255, Aizu, Japan, March 1997. IEEE Computer Society.

[3] G. Bird. Approach to translational equilbrium in a rigid sphere gas. *Physics of Fluids*, 6:1518, 1963.

[4] G. Bird. *Molecular Gas Dynamics and the Direct Simulation of Gas Flows*. Claredon, Oxford, 1994.

[5] J. R. Birge. Quasi-Monte Carlo approaches to option pricing. Technical report 94–19, Department of Industrial and Operations Engineering, University of Michigan, 1994.

[6] F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of Political Economy*, 81:637–654, 1973.

[7] P. Boyle. Options: A Monte Carlo approach. *Journal of Financial Economics*, 4:323–338, 1977.

[8] P. Boyle, M. Broadie, and P. Glasserman. Monte Carlo methods for security pricing. *Journal of Economic Dynamics and Control*, 21:1267–1321, 1997.

[9] M. Broadie and P. Glasserman. Pricing American-Style Securities Using Simulation. Technical Report 96–12, Columbia - Graduate, School of Business, 1996. available at http://ideas.repec.org/p/fth/colubu/96-12.html.

[10] B. C. Bromley. Quasirandom Number Generators for Parallel Monte Carlo Algorithms. *Journal of Parallel and Distributed Computing*, 38(0132):101–104, 1996.

[11] H. Faure. Discrepance de suites associees a un systeme de numeration (en dimension s). *Acta Arithmetica*, 41:337–351, 1982.

[12] M. C. Fu. Pricing of financial derivatives via simulation. In C. Alexopoulos, K. Kang, W. Lilegdon, and D. Goldsman, editors, *In Proceedings of the 1995 Winter Wimulation conference*, pages 126–132, Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, 1995.

[13] S. Galanti and A. Jung. Low-discrepancy sequences: Monte Carlo simulation of option prices. *Journal of Derivatives*, 5(1):63–83, 1997.

[14] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, New York, 2004.

[15] D. Grant, G. Vora, and D.Weeks. Path-Dependent Options: Extending the Monte Carlo Simulation Approach. *Management Science*, 43(11):1589–1602, Nov 1997.

[16] J. H.Halton. On the efficiency of certain quasirandom sequences of points in evaluating multidimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.

[17] J. Hull and A. White. The Pricing of Options on Assets with Stochastic Volatilities. *Journal of Finance*, 42:281–300, 1987.

[18] J. C. Hull. *Options, Futures, and Other Derivatives*. Prentice Hall, Upper Saddle River, NJ, 7th edition, 2008.

[19] A. G. Z. Kemna and A. C. F. Vorst. A Pricing Method for Options Based on Average Asset Values. *Journal of Banking and Finance*, 14:113–129, 1990.

[20] L. Kuipers and H. Niederreiter. *Uniform Distribution of Sequences*. John Wiley & Sons, New York, 1974.

[21] A. Lastovetsky. Adaptive parallel computing on heterogeneous networks with mpC. *Parallel Computing*, 28:1369–1407, 2002.

[22] A. Lastovetsky. *Parallel Computing on Heterogeneous Networks*. John Wiley & Sons, 2003.

[23] J. X. Li and G. L. Mullen. Parallel computing of a quasi-monte carlo algorithm for valuing derivatives. *Parallel Computing*, 26(5):641–653, 2000.

[24] H. Niederreiter. *Random Number Generation and Quasi–Monte Carlo Methods*, volume 63 of *CBMS-NSF Regional Conference Series in Appl. Math*. SIAM, Philadelphia, PA, 1992.

[25] G. Okten and A. Srinivasan. Parallel quasi-Monte Carlo methods on a heterogeneous cluster. In H. N. et al., editor, *Proceedings of Fourth International Conference on Monte Carlo and Quasi-Monte Carlo*, pages 406–421, Hong Kong, 2000.

[26] S. H. Paskov and J. F. Traub. Faster valuation of financial derivatives. *Journal of Portfolio Management*, 22(1):113–120, Fall 1995.

[27] S. Rakhmayil, I. Shiller, and R. K. Thulasiram. Cost of Option Pricing Errors Associated with Incorrect Estimates of the Underlying Assets Volatility: Parallel Monte Carlo Simulation. *IMACS J. Mathematics and Computers in Simulation*, (under review).

[28] W. Schmid and A. Uhl. Techniques of parallel quasi-Monte Carlo integration with digital sequences and associated problems. *Mathematics and computers in simulation*, 55:249–257, 2000.

[29] E. S. Schwartz andW. N. Torous. Prepayment and the Valuation of Mortgage-Backed Securities. *Journal of Finance*, 44(2):375–392, 1989.

[30] I. M. Sobol. On the distribution of points in a cube and the approximate evaluation of integers. *U.S.S.R. Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

[31] A. Srinivasan. Parallel and distributed computing issues in pricing financial derivatives through quasi monte carlo. In *Proc. (CDRoM) 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, Fort Lauderdale, FL, USA, 2002. IEEE Computer Society.

[32] J. A. Tilley. Valuing American Options in a Path Simulation Model. *Transactions of the Society of Actuaries*, 45:83–104, 1993.