International Journal of Applied Mathematics
Volume 2 No. 4 2000, 441-457

HIGH PERFORMANCE MATHEMATICAL SOFTWARE IN FINANCE: A CASE STUDY

G. Laccetti[1], M. Lapegna[1], M. Marino[1], A. Murli[1], F. Perla[2]
[1] Center for Research on Parallel Computing and Supercomputers (CPS)
of the Italian National Research Council (CNR)
and University of Naples "Federico II", Naples, Italy
email: (laccetti, lapegna, marino, murli)@matna2.dma.unina.it

[2]University of Rome "La Sapienza", Rome, Italy
email: perla@nava1.uninav.it

Abstract: Finance and markets problems are a sort of *Grand Challange* to be faced nowadays by *High Performance Computing*. The development of a mathematical software building block (a parallel algorithm for high dimension integrals) to manage, for example, a particular class of financial derivatives, namely Collateralized Mortgage Obligations, is discussed in this paper. Numerical evidence of good performances of a couple of Quasi Monte Carlo methods on a set of functions "similar" to real "financial" functions is also showed.

AMS Subj. Classification: 90A09, 68N, 65D32
Key Words: high performance computing, finance problems, mathematical software, multidimensional quadrature.

1. High Performance Computing and Mathematical Software Libraries in Finance

Among the problems that actually are growing in complexity and importance in the real world life there are *finance and market problems* [4,28]. The uncertainties in financial markets in terms of interest rates, currency exchange rates, inflation and other economic factors, have a profound impact on the *risk*

Received: July 13 1999

and evaluation of financial instruments. Therefore, the simulation of innovative financial instruments, such as *mortgage backed-securities*, *collable bonds*, *collateralized mortgage obligations*, and other derivatives, is highly complex and computational intensive, mainly due to the huge number of variables and conditions to take into account for accurate forecasts [25,28]. Further, finance operators have to be able to react in a "useful time" to the changes of complex world financial market. So that, finance and market problems can be faced nowadays only by High Performance Computing [19]. This is confirmed by the existence of international research projects in this area, e.g. *HPC-Finance Project* [25], the wide activity of working groups [5], the amount of recent specialized journals, e.g. the *Journal of Computational Finance*, and conferences related to this topic, e.g. the *First Annual Conference on Computational and Quantitative Finance 98*, *HPCFIN Conference 1999*.

Mathematical models arising in the simulation of the new financial instruments usually involve *optimization models, binomial lattice models, evaluation of partial differential equations* and *multidimensional integrals* [1,4,28].

Anyway, the widespread and effective use of HPC resources is still inhibited by the lack of software suitable for the advanced computational environments [26]. While the state of art of low level software, the so-called *system software*, can be considered satisfactory, until now this is not true for *mathematical software* (the medium level software). Different applications are described by similar mathematical models, leading to common computational kernels; mathematical software provides solution to these kernels, and hence supplies *building blocks* for the development of application software.

Then, it is of paramount importance to develop building blocks for HPC environments able to solve the computational kernels of finance and market problems.

In this paper we report the experiences related to the development of a building block to manage a particular class of financial derivatives. In section 2 we describe a *collateralized mortgage obligation* (CMO), whose computational kernel is one of the above mentioned: a *multidimensional integral*. We choose this derivative since the involved integral has fairly high dimension and the evaluation of each integrand is very expensive. Therefore, this problem makes clear why financial firms have a strong interest to the availability of efficient mathematical software for approximate integrals in HPC environments. In section 3 we present the most promising methods and algorithms for the computation of multidimensional integrals involved in the considered finance problem. Finally, in section 4, we show the computational experiments performed on several problems in a HPC environment.

2. A Case Study: Collateralized Mortgage Obligations (CMO)

A CMO consists of a pool of bond classes, named *tranches*, which derives the cashflows from an underlying pool of mortgages. Investors purchase interest in the pool and receive pro-rated shares of the cashflows. The issuing institutions handle the transfer of funds and retain a service fee. The cashflows received from the mortgages consist of *interest* and *repayments of the principal* and are divided and distributed to each of the tranches according to a set of prespecified rules.
Since the cashflow of an individual mortgage loan is uncertain because of the potential of the prepayments, the same is true for CMO. Then, the technique of distributing the cashflows transfers the prepayment risk among the tranches. We stress that the interest rates are variable, so the cashflows vary with fluctuations in future interest rates, especially when lower rates induce people to prepay loans, perhaps by refinancing. The problem is to compute *the expected value* of the sum of *the present values of future total cashflows and/or future cashflows for each of the tranches.* For more details on the stucture of CMOs see [11].

Determining the expected present value of CMO is a complex process since the cashflows take into account both payment of principal and interest, and prepayment of mortgages (i.e. exercise of the underlying call option by some homeowners).

The first step in the *problem solving methodology* is the formulation of the mathematical model $\mathbf{M(P)}$ from the real problem $\mathbf{P}$; so, considering a security backed by mortgages of length M months and cashflows obtained monthly, the present value is then:

$$PV = \sum_{k=1}^{M} u_k m_k \quad , \tag{1}$$

where u_k = discount factor for month k,
m_k = cashflow for month k.

We want to compute the expected value $E[PV]$, in which E is the expectation over the random variables involved in the interest rate fluctuations.
The other variables in the problem are the following:

i_k = interest rate for month k,
w_k = fraction of remaining mortgages prepaying in month k,
r_k = fraction of remaining mortgages at month k,
c = monthly payment on the underlying pool of mortgages,

c_k = (remaining annuity at month k)/c,
ξ_k = a random variable with mean 0 and variance σ^2.

Some of these variables can be defined by [3]:

$$u_k = \prod_{j=0}^{k-1} (1 + i_j)^{-1}, \tag{2}$$

$$m_k = cr_k[(1 - w_k) + w_k c_k], \tag{3}$$

$$r_k = \prod_{j=1}^{k-1} (1 - w_j), \tag{4}$$

$$c_k = \prod_{j=0}^{M-k} (1 + i_0)^{-j}, \tag{5}$$

where i_0 is the current interest rate at the beginning of the mortgage.
Now we have to introduce models for the interest rate fluctuations and the prepayment rate. Following [3,24] we use

$$i_k = K_0 e^{\xi_k} i_{k-1} = K_0^k e^{\xi_1 + \ldots + \xi_k} i_0, \tag{6}$$

$$w_k = K_1 + K_2 arctan(K_3 i_k + K_4), \tag{7}$$

where $K_1,\ K_2,\ K_3,\ K_4$ are constants of the model, and $K_0 = e^{-\sigma^2/2}$ is chosen to normalize the log-normal distribution, i.e. $E[i_k] = i_0$.

Alternatively, other models for i_k, such as the CIR model [8], and for w_k, such as the PSA model [11], could be used.

Then, if we consider the total cashflow, the mathematical model for the expected value $E[PV]$ consists of a multidimensional integral over R^M with Gaussian weights:

$$y(\xi) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\xi^2/2\sigma^2}. \tag{8}$$

By change of variables, it is easy to see that:

$$E[PV] = \int_{[0,1]^M} PV(Y(x_1), \ldots, Y(x_M)) dx_1 \ldots dx_M\ , \tag{9}$$

where the considered mapping is $Y(x) = \xi$ with $Y'(x) = y(\xi)$. Therefore, the mathematical model $\mathbf{M(P)}$ for our problem is a multidimensional integral over the M-dimensional unit cube.

More general, if we want to compute the expected value for each of the tranches of the CMO, we have to consider that the montly cashflow m_k is divided and distributed to the tranches according to some prespecified rules of the CMO under consideration. Let $G_{k,T}(\xi_1, \ldots, \xi_M)$ be the portion of the cashflow m_k directed to the tranche T, then PV_T, the present value for the tranche T, is given by

$$PV_T = \sum_{k=1}^{M} u_k G_{k,T} \ , \tag{10}$$

and, by similar reasonings, we obtain that the expected value for the tranche T, namely $E[PV_T]$, is still represented by (9).

Once again, the mathematical model consists of multidimensional integrals over the M-dimensional unit cube. Therefore, a multidimensional integration routine is a "real" building block in finance problems solving.

A typical CMO consists of 30-year mortgages with monthly payment ($M = 360$), so the integrals to be evaluated are 360-dimensional. The function being integrated combines the assumptions about the way interest rates fluctuate and the way prepayments occur with the algebraic expressions of the discount factors; then this function is conceptually not complicated. The complications arise from the rules of the CMO that divide and distribute the cashflow among the tranches [6]. Therefore the form of the integrand function $G_{k,T}$ is very complex, but in general it is a $C^{(0)}$ function composed by min functions and smooth functions [24].

Then, the dimension of the integrals involved and the complex form of the integrand function make crucial, to the effective evaluation of the financial instrument, the choice of suitable methods and algorithms as well as the development of efficient mathematical software to compute the above integrals in a high performance computing environment. Next sections are devoted to this aim.

3. Methods, Algorithms and Software for Multidimensional Integration

It is well known that the most promising methods and algorithms to compute high dimensional integrals (say $s > 30$) with a not smooth integrand function are the Monte Carlo methods (see for example [10]).

Classical Monte Carlo (MC) methods refer to the problem in a probabilistic sense, and the solution is estimated by random variables (see for example [10]).

For our problem, first let:

$$I[f] = \int_\Omega f(t_1, ..., t_s)\, dt_1 \cdots dt_s =$$

$$= mis(\Omega) \int_\Omega \frac{f(t_1, ..., t_s)}{mis(\Omega)} dt_1 \cdots dt_s = mis(\Omega)\mu[f] \quad , \tag{11}$$

where $\mu[f]$ is the expected value of the integrand function f, interpreted as random variable, with respect to the probability density $mis(\Omega)^{-1}$. Then, given N independent s-dimensional random points $\mathbf{t_i} = (t_{i_1}, ..., t_{i_s})$ $i = 1, ..., N$ with probability density $mis(\Omega)^{-1}$, the MC estimate for the expected value $\mu[f]$ is:

$$\mu[f] \approx Q[f] = \frac{1}{N} \sum_{i=0}^{N-1} f(\mathbf{t_i}) \quad . \tag{12}$$

The strong law of large numbers guarantees the convergence to $I[f]$ (almost surely) of (12) when $N \to \infty$; further a probabilistic estimate of the error $R[f] = |\mu[f] - Q[f]|$ can be achieved by using the standard deviation of $Q[f]$:

$$R[f] \approx \sigma(Q[f]) = \frac{\sigma(f)}{\sqrt{N}} \quad . \tag{13}$$

That is the well known result establishing that the MC method for numerical integration has a convergence speed of $\mathcal{O}(N^{-1/2})$. In practice, all *random number generators* are actually based on deterministic algorithms, producing a periodic sequence of numbers that should look "apparently random". Therefore, such generators are referred as *pseudo random numbers generators.* In this sense, the most common employed generator is the *linear congruential generator*:

$$v_i = (av_{i-1} + c) \text{mod } M \quad , \tag{14}$$

where the parameters a, c and M are chosen to maximize the period of the sequence. The nodes $\mathbf{t_i}$ in (12) are therefore defined by:

$$\mathbf{t_i} = (v_{is+1}, ..., v_{is+s}) \quad . \tag{15}$$

The main feature of (13) is that the convergence speed does not depend on the dimension s and on the regularity of the integrand function $f(t_1, ..., t_s)$, and this is the reason because the MC method well agrees with the computational requirements of a CMO problem described in section 2 (large number of variables and $C^{(0)}$ integrand functions).

However it should be noted that the convergence speed of MC method is very poor, and rarely gives more than 1 or 2 significant digits of accuracy.

Recently, Quasi Monte Carlo (QMC) methods have been successfully used for very large values for the dimension s, especially in finance applications [3,6,16,23,24,27].

The basic idea of QMC methods is to replace in (12) the random nodes $\mathbf{t}_i$ with equidistributed ones, with the aim to achieve a convergence speed better than the $\mathcal{O}(N^{-1/2})$ rate of the classical MC method. These methods are deeply reviewed in [22], and a complete description is outside the aim of this work. In this paper we shortly describe two of these methods, based on the *Halton sequence* and the *Faure sequence*, respectively.

The starting point to describe the QMC methods is the *Koksma-Hlawka inequality* [15]:

Theorem 3.1: *Let f with bounded variation $V[f]$ in the sense of Hardy and Krause on the s-dimensional cube $\Omega = [0,1]^s$, then for any $\mathbf{t}_1, ..., \mathbf{t}_N \in \Omega$:*

$$R[f] = \left| \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{t}_i) - \int_{\Omega} f(t_1, ..., t_s)\, dt_1 \cdots dt_s \right| \leq V[f] D_N^*(\mathbf{t}_1, ..., \mathbf{t}_N) \quad , \quad (16)$$

where $D_N^*(\mathbf{t}_1, ..., \mathbf{t}_N)$ is the discrepancy *of the points set $\mathbf{t}_1, ..., \mathbf{t}_N$: a measure for the deviation from the equidistribution.*

The aim in the development of QMC methods is therefore to find points sets with small discrepancy. For this reason the QMC methods are also called *low discrepancy methods.*

The first sequence of nodes we describe is the *Halton sequence* [14]. To this aim we remember that for a given integer $p \geq 2$, any integer i can be uniquely represented as $i = \sum_{r=0}^{m} d_r(i) p^r$ where $d_r(i) < p$. Then we define the *radical inverse function* in base p as:

$$\phi_p(i) = \sum_{r=0}^{m} d_r(i) p^{-r-1} \quad . \quad (17)$$

With this function the digits of $\phi_p(i)$ are obtained by a symmetric reflection of the digits of i after the decimal point. Finally, let $p_1, ..., p_s$ be the first s prime numbers, the s-dimensional nodes $\mathbf{t}_i$ of the Halton sequence are defined as:

$$\mathbf{t}_i = (\phi_{p_1}(i), \phi_{p_2}(i), ..., \phi_{p_s}(i)) \in [0,1[\qquad i = 1, 2, ..., N \quad . \quad (18)$$

It is shown in [14] that a bound for the discrepancy of Halton sequences is:

$$D_N^*(\mathbf{t}_1, ..., \mathbf{t}_N) \leq C \frac{(log\ N)^s}{N} \quad , \quad (19)$$

where C is a constant independent of f increasing with s.

Using a single base p avoids the growth of C, as well as using s linear transformation of the digits of i before to define the i-th node with the radical inverse function in (17), avoids the correlation of the nodes.

In this way we obtain the $(t,s)-$ *sequences* introduced by Niederretier in [21]. The Faure sequences are a special case of the $(t,s)-$ *sequences* where p is the smaller prime number such that $p \geq s$.

More precisely, given a prime $p \geq s$ the representation of i in base p is $i = \sum_{r=0}^{\infty} d_r(i) p^r$ where $d_r(i) < p$ and $d_r(i) = 0$ for sufficently large r. Then, we define the i-th node of the Faure sequence as:

$$\mathbf{t}_i = \left(\sum_{r=1}^{\infty} y_{ir}^{(1)}, ..., \sum_{r=1}^{\infty} y_{ir}^{(s)} \right) \quad . \tag{20}$$

The $y_{ir}^{(k)}$ are defined by the linear transformations:

$$y_{ir}^{(k)} = \sum_{j=0}^{\infty} c_{rj}^{(k)} d_j(i-1) \mod p \quad , \tag{21}$$

and the coefficients $c_{rj}^{(k)}$ of the linear transformation are given by:

$$c_{rj}^{(k)} = \begin{cases} 0 & \text{for} \quad 0 \leq j < r-1 \\ \begin{pmatrix} j \\ r-1 \end{pmatrix} (k-1)^{j-r+1} & \text{for} \quad j \geq r-1 \quad . \end{cases}$$

In the case of the Faure sequences the bound for the discrepancy is the same as (19), but with the constant C decreasing with s.

4. Development of a HPC building block for multidimensional quadrature

The third step in the problem solving methodology deals with the development of an algorithm $\mathbf{A}(\mathbf{P})$, based on the numerical method $\mathbf{M_h}(\mathbf{P})$, suitable for a given computational environment. So, in this section, we describe the development methodology of a multidimensional quadrature algorithm and its testing procedure in a high performance computing environment.

4.1 Parallel implementation issues

Our target computing environment is a MIMD distributed memory computer consisting of P processors numbered from 0 to $P-1$ and connected by a communication network that allows to exchange messages among the processors.

The basic idea to solve a problem on a such computer is to split the problem in independent subproblems that can be solved simultaneously and to compute the global result from the partial ones. To obtain high values of efficiency, the workload required by the subproblems has to be balanced as well as the subproblems have to require few or none interactions. In general, such requirements are not easy to achieve.

To reach the goal of high efficency, we introduce parallelism in a rule of kind (12) distributing the N function evaluations among the P processors, that is:

$$Q[f] = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{t}_i) = \frac{1}{P}\sum_{j=0}^{P-1}\frac{1}{N/P}\sum_{i=0}^{N/P-1} f(\mathbf{t}_i^{(j)}) = \frac{1}{P}\sum_{j=0}^{P-1} Q^{(j)}[f] \quad , \tag{22}$$

where $Q^{(j)}[f]$ is the cubature rule "locally" computed by the processor j and $\mathbf{t}_i^{(j)} = \mathbf{t}_{j+iP}$ is the i-th node generated by the processor j. After $Q^{(j)}[f]$ has been (locally) computed, the P processors have to combine the partial results in order to obtain the final result $Q[f]$. We note that for the Halton sequence (18) and for the Faure sequence (20), the processor j can generate the node $\mathbf{t}_i^{(j)}$ without interaction with the other processors, whereas for the linear congruential generator (14), the processor j has to communicate with the processor $j-1$ for the computation of $\mathbf{t}_i^{(j)}$. In order to overcame this drawback we used the generator introduced in [18]. Other parallel generators are introduced in [2,9].

Following is the parallel algorithm, described using the *Single Program Multiple Data* programming model. In this model the same copy of the algorithm (the so called *node algorithm*), is executed by each processor on different data (see for example [12]):

```
locinit(j;P)
input(N;f)
locsum := 0
for i = 0, N/P − 1
    generate(t_i^(j))
    locsum := locsum + f(t_i^(j))
endfor
compglobal(Q[f])
Q[f] := Q[f]/N
```

Algorithm 1: node algorithm for a QMC method for multidimensional quadrature

In the previous algorithm the function *locinit* initializes the *id-processor* j and the number of processors P, *input* acquires the number of nodes N and the integrand function f, *generate* computes the nodes $\mathbf{t}_i^{(j)}$ according to the selected method, and *compglobal* computes the global sum $Q[f]$ in (12).

In order to analyze the computational cost of the algorithm, we note that the (12) is well adapted for a parallel computation, because each processor j has to generate the N/P nodes $\mathbf{t}_i^{(j)}$ independently from the other ones and the number of the integrand function evaluations is equally distributed among the processors.

Finally the computation of the global result is performed by a *cascade sum* algorithm in $\log_2 P$ steps. The computational cost of the node algorithm on each processor is then:

$$CC_P(N) = \frac{N}{P}(Fev + Sum) + \log_2 P(Comm + Sum)$$

where Fev, $Comm$ and Sum stand for function evaluations, communication steps and sums in the algorithm.

To evaluate the efficiency of the proposed parallel algorithm we tested it on a *cluster of 16 Pentium Pro processors* operated by *Center for Research on Parallel Computing and Supercomputers* (CPS) of the Italian Research National Council (CNR). The processors are connected by a 100 Megabit/sec *Ethernet switch*, and they implement the *Parallel Virtual Machine* (PVM) communication system. This computational environment is very attractive to solve financial problems since it can achieve sustained speeds of more than 1

Gigaflops (10^9 floating point operations/sec) on a real problem, with a very high price-performance ratio (about 17,000 US dollars for 1 Gigaflops) [7].

4.2 Numerical experiments

In our experiments, the first step was to test accuracy and reliablity of the algorithm. To do that we use test integrand functions derived from the *Genz package* [13], some of them having similar properties and/or behaviour of real "financial" functions. The functions have been tested using several dimensions and different values of N on the region $\Omega = [0,1]^s$.

Each function can be classified using the parameters α_i and β_i that determine the sharpness and the location of the difficulty, respectively. The parameters $\beta_i \in [0,1]$ are random values. The parameters $\alpha_i \in [0,1]$ are also random values, but they are scaled according to:

$$s^{e_j} \sum_{i=1}^{s} \alpha_i = h_j \quad , \quad j = 1, 2,$$

in order to control the difficulty of the integrand when s increases. The values of e_j and h_j are:

$$\underline{e} = (2.\ ,\ 2.)$$
$$\underline{h} = (200\ ,\ 300).$$

The functions we used are:

$$\begin{array}{lll} f^{(1)}(\mathbf{t}) = \exp(-\sum_{i=1}^{s} \alpha_i |t_i - \beta_i|) & & C^{(0)} \text{ function} \\ f^{(2)}(\mathbf{t}) = \begin{cases} 0 & \text{if } t_1 > \beta_1 \text{ or } t_2 > \beta_2 \\ \exp(\sum_{i=1}^{s} \alpha_i t_i) & \text{otherwise} \end{cases} & & \text{discontinuous function} \\ f^{(3)}(\mathbf{t}) = (\sum_{i=1}^{s} |4t_i - 2|)/s & & C^{(0)} \text{ function} \end{array} \tag{23}$$

The first set of experiments evaluates the reliability and the accuracy on just one processor, by comparing the results of the three methods.

Figures 1-3 report the error $R[f]$ as function of the number of nodes N for the three methods for the functions $f^{(j)}$ $j = 1, .., 3$ with $s = 10$ and $s = 80$.

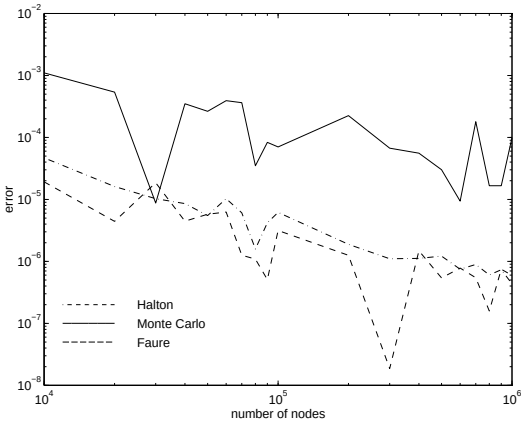

Figure 1a: error for the function $f^{(1)}$ in $s = 10$ dimensions

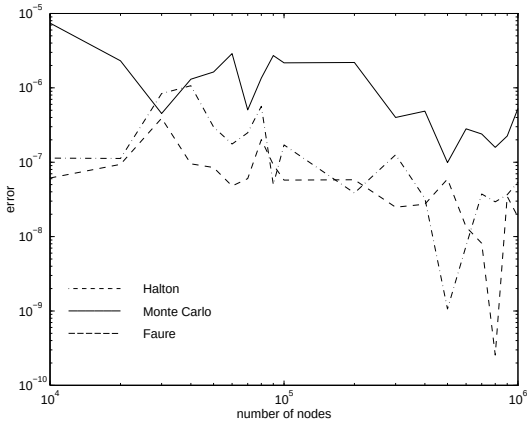

Figure 1b: error for the function $f^{(1)}$ in $s = 80$ dimensions

The plotted results show that, for all the three considered functions, for moderate number of variables ($s = 10$), the QMC methods outperform the classical MC method. In this case Halton and Faure sequences produce an *experimental* convergence speed $\mathcal{O}(N^{-\alpha})$ with $1/2 < \alpha < 1$, higher than the MC method one. For high dimensions ($s = 80$) the improvement of the convergence speed is less evident. However, the QMC methods are still competitive with respect to the MC method, and this is not explained by the discrepancy bound (19). For large value of s, this bound produce extremely large value, then (19) does not give a real description of the convergence speed of QMC methods. On the other hand, computational and theoretical results presently are attempting to partially explain all that, demonstrating that the QMC methods are very promising for the development of mathematical software for finance problems [3,17,23,27].

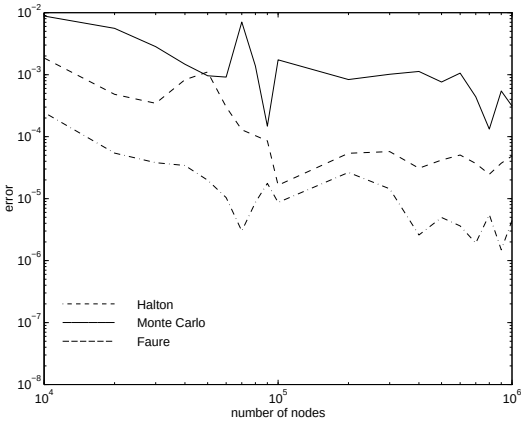

Figure 2a: error for the function $f^{(2)}$ in $s = 10$ dimensions

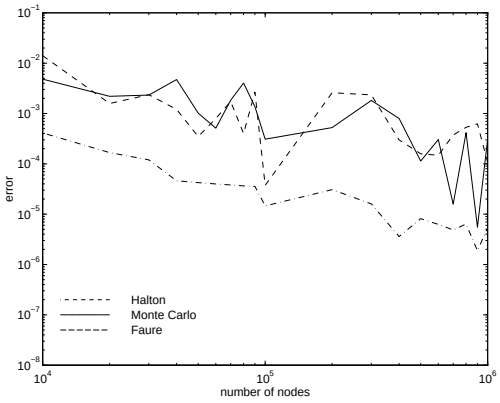

Figure 2b: error for the function $f^{(2)}$ in $s = 80$ dimensions

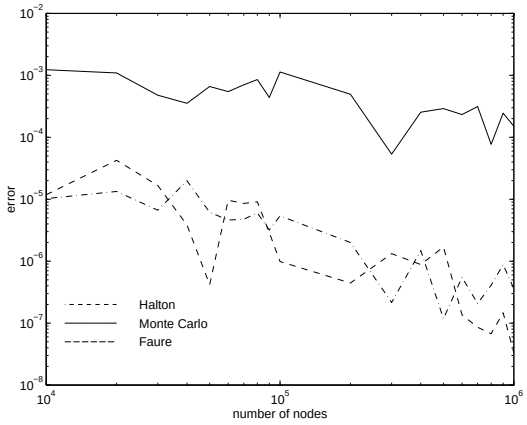

Figure 3a: error for the function $f^{(3)}$ in $s = 10$ dimensions

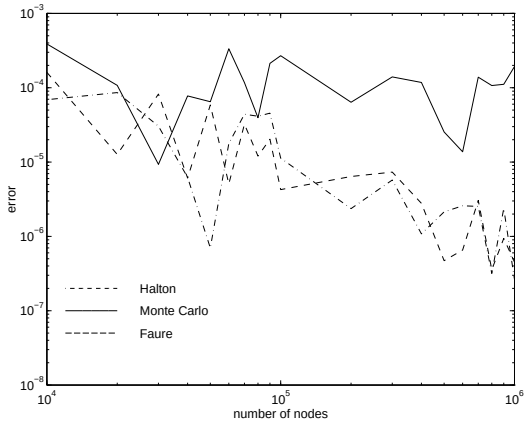

Figure 3b: error for the function $f^{(3)}$ in $s = 80$ dimensions

Further, we tested the three methods on a very simple CMO problem. This model consist of three sequential payment class [11]. So we emphasize that the aim of this test is to evaluate the performance of the algorithm and does not to validate the underlying financial model. Since to simulate a real financial problem, the functions $G_{k,T}$ and the constants K_i in (6) and (7) have to be specified and have to be related to real cases (in general proprietary infos managed by financial firms), we use the *PSA prepayment model* [11] for w_k and the simple interest rate model $i_k = (1 + \xi_k) i_{k-1}$.

In Figure 4, we report the market price for the first tranche of the CMO computed by the three considered methods, with different number of nodes N in (12).

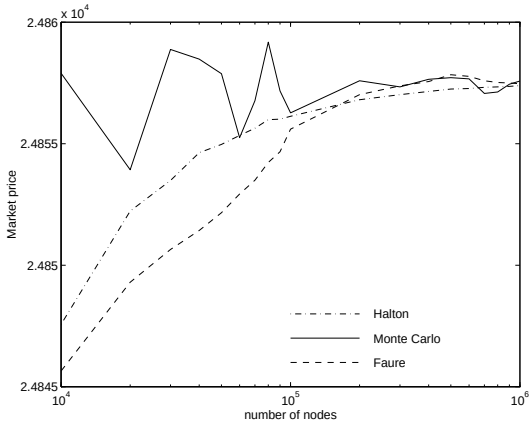

Figure 4: expected value for a simple CMO problem

The figure shows that all methods converge towards the same expected value, but with different behaviour: the convergence of the two QMC methods is smoother than the MC method one, because of the probabilistic nature of

the MC method.

The second set of experiments has been performed to evaluate the efficiency of the algorithm when the number of processes P increases. To evaluate the performance of the algorithm we use the classical parameter:

$$\text{SPEED-UP} \quad S_P = \frac{T_1(N)}{T_P(N)}$$

where $T_P(N)$ is the elapsed execution time on P processors using N integrand function evaluations. Figures 5-6 report speed-up values for the functions $f^{(1)}$ and $f^{(2)}$ varying the number of processors P and with $s = 10$ and $s = 80$.

The results of the experiments confirm the expectation of good performance of the analysed methods and show that QMC methods are well adapted for parallel computing. We note that the achieved speed-up values in all tests, are very close to ideal one $S_P = P$. This means that it is possible to evaluate (22) with P processors in about $1/P$-th of the time required for the same computation with 1 processor.

Finally, an idea of the gain obtained, in terms of execution time required to compute (22) on the described computational environment, is given considering that execution times for the QMC method based on the Halton sequence with $N = 10^5$ nodes are about 14 minutes on 1 processor and 1.83 minutes on 8 processors, with a speed-up $S_P \approx 7.6$.

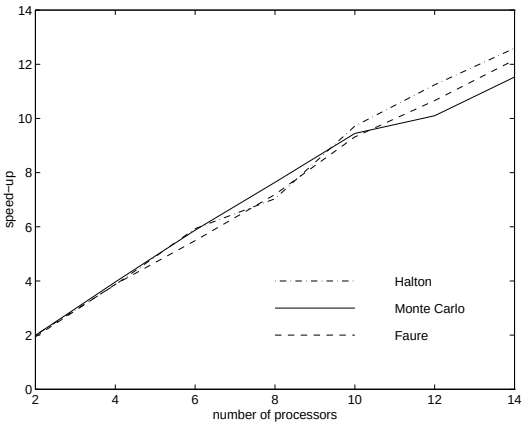

Figure 5a: speed-up for the function $f^{(1)}$ in $s = 10$ dimension

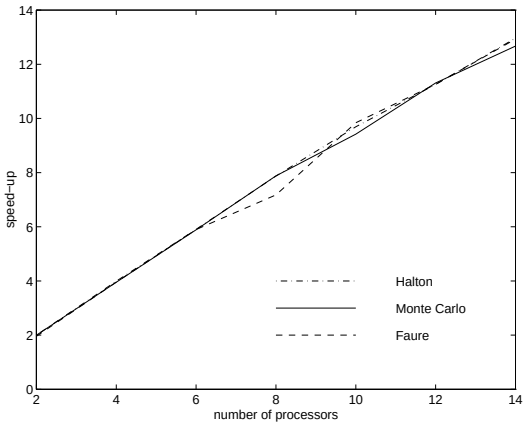

Figure 5b: speed-up for the function $f^{(1)}$ in $s = 80$ dimension

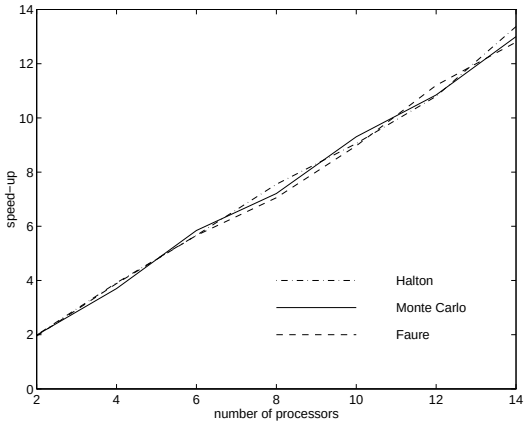

Figure 6a: speed-up for the function $f^{(2)}$ in $s = 10$ dimensions

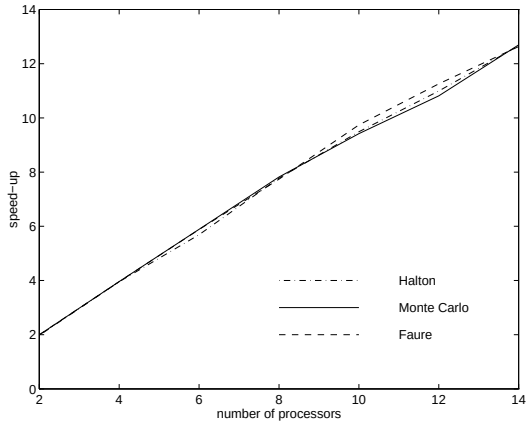

Figure 6b: speed-up for the function $f^{(2)}$ in $s = 80$ dimensions

5. Concluding remarks

Summarizing, all things above discussed seem to encourage to develop parallel building blocks based on QMC methods and parallel algorithms for multidimensional integration, to efficiently solve financial problems in HPC environments. Our tests show that the QMC methods based on the Halton and Faure sequences are more effective than the MC method. However, we note also that, as the dimension s increases, the QMC methods lose their effectiveness. Anyway, as already mentioned, many scientists are presently involved in the task of finding new sequences able to preserve the good convergence properties in high dimensions too, both from the theoretical and computational point of view. The main critical aspects remain, again, related to the general problem to realize efficient mathematical software in High Performance Computing environments.

Acknowledgement

This work was partially supported by contract *HPC-Finance* (no. 951139) of the European Commission.

References

[1] F. AitSahlia and P. Carr, American Options: a comparison of numerical methods, in: *Numerical methods in Finance*, eds. L.C.G. Rigers and D. Talay, Cambridge University Press (1997), 67-87.

[2] S. Aluru, G.M. Prabhu and J. Gustafson, A random number generator for parallel computers, *Parallel Computing*, **18** (1992) 839-847.

[3] R.E. Caflisch, W. Morokoff and A. Owen, Valuation of mortgage-backed securities using Brownian bridges to reduce effective dimension, *The Journal of Computational Finance*, **1** (1998), 27-46.

[4] J. Case, The mathematization of finance (Part I and Part II), *SIAM News*, **27** (1994).

[5] G. Cheng and K. Mills, Financial Modelling, NHSE Software Catalog, url: *http://www.npac.syr.edu/users/gcheng/finance/home.html.*

[6] B. Cipra, In Math We Trust, AMS, 1996.

[7] B. Cipra, In scientific computing, many hands make light work, *SIAM News*, **30** (1994).

[8] J.C. Cox, J.E. Ingersoll and S.A. Ross, A theory of the term structure of interest rates, *Econometrica*, **53** (1985).

[9] A. De Matteis and S. Pagnutti, Parallelization of random number generators and long range-correlations, *Num. Math.*, **53** (1988), 595-608.

[10] P. Davis and P. Rabinovitz, *Methods of Numerical Integration. 2 ed.*, Academic Press (1984).

[11] F. Fabozzi, *The Handbook of Mortgage Backed Securities. fourth ed.*, Irwin Prof. Publ. (1995).

[12] J. Fox et al., *Solving problems on concurrent processors. Vol. 1*, Prentice Hall (1984).

[13] A. Genz, Testing multidimensional integration routines, in: *Tools, Methods and Languages for Scientific and Engineering Computation*, eds. B. Ford, J. Rault and F. Thommaset, North-Holland (1984).

[14] J.H. Halton, On the efficiency of certain Quasi-random sequences of points in evaluating multidimensional integrals, *Num. Math.*, **2** (1960), 84-90.

[15] E. Hlawka, Discrepancy and uniform distribution of sequences, *Compositio Math*, **16** (1964), 83-91.

[16] B. Keister, Multidimensional quadrature algorithms, *Comp. in Phys.*, **10** (1996), 119-122.

[**17**] L. Kocis and W. Withen, Computational investigations of low-discrepancy sequences, *ACM Trans. on Math. Soft*, **23** (1997), 266-294.

[**18**] P. L'Ecuyer, Efficient and portable combined random number generators, *Comm. of the ACM*, **31** (1988), 742-774.

[**19**] NSF Blue Ribbon Panel on High Performance Computing, From desktop to Teraflop: Exploiting the U.S. Lead in High Performance Computing, NSF Report (1993).

[**20**] Numerical Algorithms Group, *The Parallel Nag Libraries. Release 2*, Oxford (1997).

[**21**] H. Niederreiter, Point set and sequences with small discrepancy, *Monatsh. Math.*, **104** (1987), 273-337.

[**22**] H. Niederreiter, *Random Number Generation and Quasi Monte Carlo Methods*, SIAM (1992).

[**23**] A. Papageorgiu and J.F. Traub, Faster Evaluation of Multidimensional Integrals, *Comp. in Phys.*, **11** (1997), 574-578.

[**24**] S.H. Paskov, New methodologies for valuing derivatives, Department of Computer Science, Columbia University, Tech. Rep. CUCS-29-96 (1996).

[**25**] Research Proposal, High performance computing for financial planning under uncertainty INCO-DC Project NO. 951139, Department of Public and Business Administration, University of Cyprus, Cyprus (1995).

[**26**] J.R. Rice and R.F. Boisvert, From scientific software libraries to problem-solving environments, *IEEE Computational Science and Engineering*, **3** (1996), 44-53.

[**27**] I.H. Sloan and H. Wozniakowsky, When are Quasi Monte Carlo algorithms efficient for high dimensional integrals?, *J. of Complexity*, **14** (1998), 1-33.

[**28**] S. Zenios, Massively Parallel Computation in Finance, *SIAM News*, **24** (1991).