Author(s) Kanniainen, Juho; Piché, Robert; Mikkonen, Tommi

Title Use of distributed computing in derivative pricing

Citation Kanniainen, Juho; Piché, Robert; Mikkonen, Tommi 2009. Use of distributed computing in derivative pricing. International Journal of Electronic Finance vol. 3, num. 3, 270-283.

Year 2009

DOI http://dx.doi.org/10.1504/IJEF.2009.027850

Version Post-print

URN http://URN.fi/URN:NBN:fi:tty-201405221200

Copyright International Journal of Electronic Finance 3/2009, http://dx.doi.org/10.1504/IJEF.2009.027850 © Inderscience Enterprises Ltd.

Use of Distributed Computing in Derivative Pricing[*]

Juho Kanniainen[†], Robert Piché[‡], and Tommi Mikkonen[§]
Tampere University of Technology, Tampere, Finland

April 4, 2008

Abstract

Monte Carlo option pricing algorithms are well suited to distributed computing because simulations can be run on different computational units with no need for communication between these tasks. In this paper we investigate and compare the use of two distributed computing environments for such computation: a PC grid that exploits the spare computing capacity of up to 470 computing cores in 300 office and teaching lab PCs scattered on a university campus, and a scientific computing cluster of 120 computing cores in 32 rack-mounted servers. We outline the process of adapting a Monte Carlo algorithm for computing prices for a set of 100 arithmetic Asian options with stochastic volatility to run on these environments, and investigate the performance for different distributing strategies.[1] The paper closes with a discussion of the opportunities and challenges of distributed computing in computational finance.

Keywords: distributed computing; cluster computing; grid computing; Monte Carlo methods; Asian option pricing;
JEL Classification Numbers: C15; C63; G13

1 Introduction

Monte Carlo simulation is well established as an effective and easy to use tool for pricing complicated path-dependent europan-style contracts, and can even easily accommodate models with stochastic volatility. For example, Monte Carlo methods can be used to price arithmetic-average asian options with stochastic volatility, which do not have closed-form solutions (Fouque and Han 2003).[2] Monte Carlo methods can be succinctly coded in any high-level computer language, and a single price of a vanilla European option can typically be computed to reasonable accuracy on a modern desktop PC in at most a few minutes.

[*]The draft is accepted for presentation at 5th Int. Conference on Computational Management Science, 26–28 March 2008, Imperial College London. All comments are welcomed.

[†]Institute of Industrial Management, `juho.kanniainen@tut.fi`

[‡]Institute of Mathematics, `robert.piche@tut.fi`

[§]Institute of Software Systems `tommi.mikkonen@tut.fi`

[1]Asian option pricing is a preliminary benchmarking problem. Other problems are also considered for the final version of this paper, such as exotic cliquets, CDO's, and Bermundan options.

[2] Partial differential equation approaches can also be used, see (Vecer 2001, Vecer 2002).

Often, however, the computation needed to simulate the price and Greeks of an exotic multi-underlying derivative to reasonable levels of accuracy can run on a single high-performance computer even for hours. Consider, for example, the simulation of prices and greeks of path-dependent exotic cliquets written on a portfolio of 30 selected stock. A financial institute may have hundreds of such computationally-intensive derivative contracts whose prices and Greeks must be computed daily or even hourly. Moreover, it may be required to compute a set of prices or volatilities corresponding to a set of parameters. For example, the hedging of co-movements in the option's parameters can require the computation of a price surface, and computing a single price surface on a 10×10 grid to reasonable level of accuracy may require several hours. Another example of a time-consuming task is model calibration, which requires computing a large set of the implicit volatilities of vanilla options. Also the implied volatilities of exotic options can be requested (see Ewald, Yang, and Xiao 2006). For stochastic models that do not have Heston-type (1993) closed form solutions, doing the calibration with a Monte Carlo method is a very computationally-intensive task. If more accuracy is desired, the computation becomes even more protracted. According to the standard rule of thumb for Monte Carlo methods, each additional correct digit in an option price requires a hundred-fold increase in computing time. Variance-reduction techniques can reduce the number of Monte Carlo simulations needed to achieve a given accuracy up to a certain point, but not necessarily enough.

Any Monte Carlo computation can naturally be divided into independent subtasks. Time-consuming simulations can be run on different computational units ("cores"), with no need for communication between the subtasks, and the final result is given by a simple averaging operation. Price or volatility computations corresponding to different parameters can also be run independently. Computations that can so easily be distributed are called "embarrassingly parallel" by algorithm scientists. By running the subtasks simultaneously on a large number of cores, results can be obtained much faster: ideally, the speedup is nearly linear (i.e. proportional to the number of cores).

Although the concept of distributed computing has been around as long as computing, general-purpose low-cost systems have only recently started to become generally available. In contrast to a supercomputer, which nowadays means a custom-built array of identical cores linked by high-speed data connections, distributed computing is typically based on a heterogeneous network of stand-alone computers linked by conventional data connections. A famous example is the SETI@home public computing project[3], which uses spare capacity ("cycle scavenging") of three million cores around the world to analyse radio telescope data.

The applications of distributed computing to finance have been an active area of research. In addition to equity derivative pricing, the following financial computations can be solved using Monte Carlo methods, and hence easily distributed: Portfolio management, value-at-risk, interest rate derivatives, swaps and swaptions, forex derivatives, commondity derivatives and other new products (weather derivatives etc), assets and liability modeling, multi-asset options and cliquets, and calibration (see, for example Chatagny and Chopard 2000, Zanghirati, Cocco, Paruolo, and Taddei 2000, Godart 2000,

[3]`http://setiathome.berkeley.edu/`

Glasserman 2003, Tezuka, Murata, Tanaka, and Yumae 2005).

In this paper, we investigate and compare the use of two distributed computing environments for arithmetic Asian option price calculations: a cluster of dedicated high-performance computers intended for scientific and technical computing, and a "PC grid" that exploits the spare computing capacity of an organization's desktop PCs. We are interested in questions such as:

- how can algorithms coded in Matlab (a popular interactive scientific computing software system) be run in a distributed environment?
- what computing speeds are achievable?
- can a PC grid compute be competitive with a scientific computing cluster?

The paper is organised as follows. The option pricing model (an arithmetic-average Asian call option with stochastic volatility) and the Monte Carlo algorithm that we use as a benchmark problem are presented in section 2. Sections 3 and 4 describe the distributed computing environments, the process of adapting the code to run on them, and some timing results. We close with a discussion of the opportunities and challenges of distributed computing for computational finance.

2 Benchmark problem

2.1 Pricing Model

We price arithmetic Asian calls according to Heston's (1993) model. The price of the underlying asset under the martingale measure is assumed to follow the stochastic differential equation

$$d\ln S(t) = \left(r - \frac{1}{2}v(t)\right)dt + \sqrt{v(t)}dW(t), \quad S(0) = S_0 > 0, \tag{1}$$

where r is the constant instantaneous risk-free interest rate, v the stochastic return variance, and W a standard Brownian motion. The return variance is assumed to be a mean-reverting process

$$dv(t) = \kappa\left(\bar{v} - v(t)\right)dt + \psi\sqrt{v(t)}dW_v(t), \quad v(0) = v_0 > 0, \tag{2}$$

where $\bar{v}$ is a reference variance, κ the speed of reversion, ψ the volatility of volatility, and dW_v is a standard Brownian motion with $dW dW_v = \rho dt$, $\rho \in [-1, 1]$.

The price of a fixed strike Asian call option is

$$e^{-r(T-t)}\mathbb{E}_t\left[\bar{S} - E\right]^+, \tag{3}$$

where $t_0 < t$ is the writing day of the option, $T > t$ the maturity date of the option, E the strike price, $\mathbb{E}_t$ the conditional expectation at time t under the martingale measure, and

$$\bar{S} = \frac{1}{N}\sum_{i=1}^{N} S(t_i), \tag{4}$$

where $t_0 \le t_1 < t_2 < \ldots < t_N \le T$, is the arithmetic average price of the underlying stock.

2.2 Monte Carlo algorithm

Eq. (1) is discretized by the stochastic Euler scheme as

$$\ln S_{i+1} = \ln S_i + \left(r - \frac{1}{2}v_i\right)\Delta t + \varepsilon_i\sqrt{v_i\Delta t}, \tag{5}$$

where $\{\varepsilon_i\}$ is a sequence of independent standard normal random variables. The volatility process (2) is discretized by the Milstein scheme and an absorbing condition is applied at the zero-volatility boundary:

$$v_{i+1} = \left[v_i + \kappa(\bar{v} - v_i)\Delta t + \xi_i\psi\sqrt{v_i\Delta t} + \frac{1}{4}\psi^2\Delta t\left({\xi_i}^2 - 1\right)\right]^+ \tag{6}$$

Here $\xi_i = \rho\varepsilon_i + \sqrt{1-\rho^2}\varepsilon_i^*$, where $\{\varepsilon_i^*\}$ is a sequence of independent standard normal random variables that are uncorrelated to the $\{\varepsilon_i\}$.

To price the option we simulate M paths of stock price and volatility using (5) and (6) with $\Delta t = (T - t_0)/N$ and $\bar{v} = v_0$. We then take the mean of the discounted payoffs computed from (3). The Monte Carlo price at $t = t_0$ is then

$$\frac{e^{-r(T-t)}}{M}\sum_{k=1}^{M}\left[\frac{1}{N}\sum_{i=1}^{N}S(t_i)^{(k)} - E\right]^+ .$$

Matlab code that implements this algorithm is presented in the Appendix.

2.3 Benchmark Problem

As a benchmark problem we calculate the prices of 100 arithmetic Asian call options with the same maturities. Such a task is relevant if a bank has 100 Asian contracts to price (with same maturities) or if the bank want to compute a set of prices or volatilities corresponding to a set of parameters for hedging or calibration purposes. We calculate a price surface of equally spaced points on $(S_0, v_0) \in [45, 47] \times [0.35, 0.40]$ with the fixed parameter values $r = 0.06, E = 70, \rho = -0.5, \kappa = 0.1, \psi = 0.5, T = 1, N = 365$.

With $M = 10^4$ simulations for each point, the Monte Carlo error is still significant (Figure 1), and we need to take $M = 10^6$ to obtain a visually smooth surface. On a single high performance desktop computer the computation of this surface requires at least 7 hours.

3 Results

In this section we introduce our PC grid and Linux cluster, and present the computational results with them. Table 1 compares PC grid and our cluster, which are more or less typical of such systems.

3.1 Distribution on a PC grid

The PC grid at the Tampere University of Technology makes use of the spare computing capacity of PCs in researchers' offices and eight computer labs at different locations on the campus, altogether up to 470 cores. The grid includes a range of hardware (Intel Celeron 2.6 GHz, Core2 1.86 GHz, Pentium4 2.8 GHz

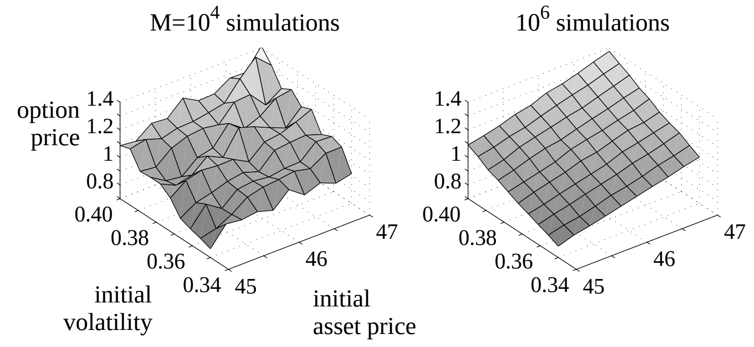

Figure 1: Asian call option price surface on a 10 by 10 grid.

and 3 GHz, and others), memory (between 0.5 and 2 GB), and operating systems (Windows XP, Linux and also Mac OS in near future). Data transfer is by 100 MB ethernet connections. The PC grid is built by using Java based grid middleware from Techila Technologies, which supports all platforms capable to run Java, for example, Windows, MAC OS, Linux etc. Even when the PC's are in "heavy" interactive use, the middleware is typically able to use 95–98% percent of the capacity without the interactive user noticing anything.

When performing computation with the PC grid, the following approach is used. The Matlab compiler is used to produce a stand-alone application that can be run on computers without Matlab licences. A programmer identifies computation-intensive hotspots in the application and adds code to define how to generate jobs to perform the associated computation. When a hotspot is encountered during the execution of the application, the associated set of jobs is uploaded to the grid server. The server in turn distributes the jobs to the computational units participating in the grid. The distribution can be parameterized, which allows one to take into account the performance and the current load of participating cores, as well as running the same application with different parameters with virtually no extra effort. If necessary, the Matlab runtime libraries (97 MB/226 MB extracted) are automatically installed on each computer on the grid. When all jobs have been processed, the grid server sends the results of the computation to the application, which can then continue its execution.

The computation task, the set of the prices of 100 Asian options, was divided into 400 jobs — four sets of 250000 simulations for each price. The Techila middleware shared these jobs out among computers on the grid, achieving nearly linear speedup, that is, results were produced at a rate that was nearly proportional to the number of cores in use (Figure 2). The overall running time for the problem was 24 minutes with 20 cores, and 4 min 40 sec with 120 cores.

The timings shown in Figure 3 used all 470 cores and different numbers of jobs. The best time to complete the task was 2 min 28 sec, achieved when the task is divided into 1000 jobs. For fewer jobs, load balancing cannot be performed well, while for larger numbers of jobs, delays associated with data transfer, scheduling, and start-up of the computation begin to dominate. Figure 3 also demonstrates that it is better to overestimate than underestimate the

Table 1: A comparison between PC grid and Akaatti cluster.

	PC grid	Akaatti cluster
gridification	application modified to contain support for distributed computation	application modified to enable running in a cluster with different sets of parameters; initialization scripts
operating system	different nodes have different OS (Windows XP, Windows Vista, etc.)	all nodes run Linux
administration	nodes belong to different organisational units; the grid can be administered separately by a single unit	single system administration unit
node usage	PCs are intended for interactive use, their computational capacity is idle most of the time	cluster nodes are intended for (typically long-duration) batch jobs
equipment purchase and maintenance	PCs upgrades and maintainance is done by the PC owners, at no cost to the grid; grid software licences needed	cluster costs include equipment upgrades, maintenance (electricity, air conditioning, machine room rental, etc.) and system administration salaries
data transfer rate	0.1 GB/s	1 GB/s
RAM / core	0.5–2 GB	4–8 GB
commercial software usage (Matlab)	not directly, but standalone executable files created by the Matlab compiler can be run	Matlab available on all nodes
scheduling software	Techila (proprietary)	SGE (open source)

number of jobs.

3.2 Distribution on a cluster

The Tampere University of Technology's scientific computing cluster "Akaatti" consists of 32 complete computers (”nodes”) mounted on a rack in an air-conditioned machine room. The system has 2.2 GHz and 2.6 GHz dual-core AMD Opteron processors and 2.33 GHz Intel quad-core Xeon processors, altogether 120 cores. In our tests we could use only 80 cores, the rest being in use by other researchers. Each node has 4GB to 8GB memory, gigabit ether-

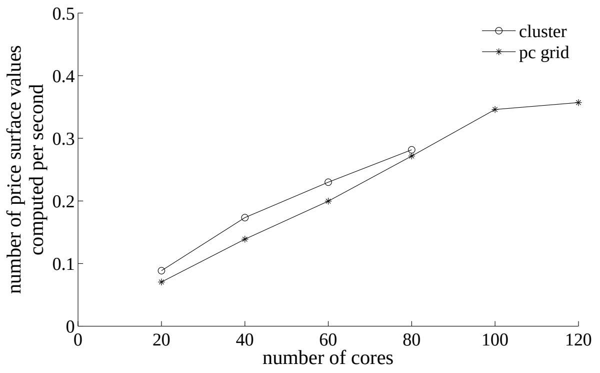

Figure 2: Timings for computing task divided into 400 jobs

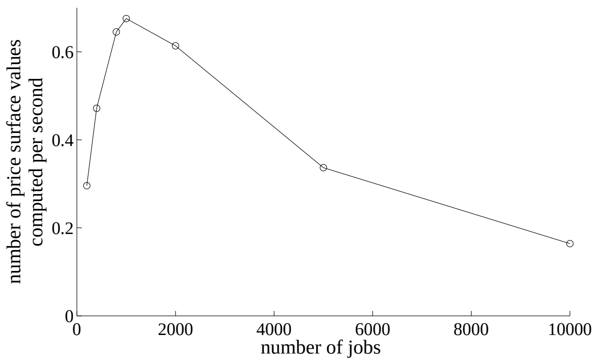

Figure 3: Timings for computing task distributed among 470 cores in PC grid

net, and runs open-source NPACI Rocks Linux version 4.1. Most of the nodes are accessible only in batch mode; batch jobs are scheduled by the open-source software Sun Grid Engine (SGE) version 6.0u8.

The first step in porting the benchmark problem to Akaatti was to compile the `asian` Matlab function along with a Matlab function that reads parameters from a file and writes results to a file. Timings are for codes produced with the same compiler version as the PC grid.

The price surface computation task was divided into 400 jobs – four sets of 250000 simulations for each price point. A Matlab script generates the 400 files containing input parameters for each job. The SGE "job array" feature allows all the jobs to be submitted using a single 10-line job control file. The scheduler keeps all the jobs in a wait queue, assigning them to cores as resources become available. After all the jobs have run, the 400 output files are combined by another Matlab script.

The computations were completed on the cluster at best only 25% faster than on the PC grid with the same number of cores.The difference in computing times was smaller with larger numbers of cores, becoming nearly equal with 80 cores. The speedup was nearly linear (Figure 2).

4 Discussion

4.1 Challenges

As mentioned earlier, there are several financial problems that require intensive computing, and are potential applications for distributed computing. However, we have identified critical points that must all be fulfilled to enable distributed computing to become a seriously taken tool in finance sector as well as in any field of industry.

From option analyst's point of view:

- **Usability**. The distributed computing system should be easy to use, meaning that the system must be seamlessly integratable to mathematical tools like Matlab. How or where the computation is performed is redundant information for the user. The most interested point is that the correct results are gained as easy and as fast as possible.
- **Gridification of the mathematical problem** (i.e. how the computation is modified to enable distributed computing). It should be possible to gridify the code without great efforts and deep software engineering skills.
- **Availability and reliability**. The distributed computing solution must be robust, i.e. it must be always available 24/7/365, the system must operate trustworthily, and the system has to be able to recover from abnormal situations.

From the point of view of IT management and system administration:

- **Data and infrastructure security**. Fine grained security must be involved in the distributed computing solution. In cluster computing systems this is not in as big relevance as in PC grid system since clusters are dedicated for computing and not in other use.
- **Administration and management ability**. Even without distributed computing systems the IT departments have hands full of work. There are not many IT departments that have in-depth knowledge and experience of distributed computing systems. That is why the whole administration and management of the distributed computing system must be simple and easy.

4.2 Don't neglect other ways to speed up the computation

Although distributed computing can, for some tasks, produce impressive speedups, so can intelligent modelling, algorithm development, and coding. For example:

- variance-reduction techniques can reduce the number of Monte Carlo simulations needed to achieve a given accuracy. (Glasserman 2003, Ch 4)
- In the `asian` code, the two lines `x=zeros(N,1); C=zeros(M,1);` are not strictly necessary, but by pre-allocating memory they speed up the code at least ten-fold.

- Coding the algorithms in fortran or C instead of Matlab could give faster programs, because of better compilers. Our Matlab code ran about three times slower than our hand-coded fortran90 implementation of the same algorithm. However, the Release 2007 Matlab compiler produced code that ran on the cluster only about 20% slower than the fortran implementation, so this performance gap is closing fast.
- With this particular benchmarking problem, the surface could be fitted by regression, greatly reducing the required number of price values.

5 Conclusions

In this paper, we have studied two different ways to run Asian option algorithm coded in Matlab in a distributed environment. One was based on a PC grid infrastructure where the actual application was reworked to utilize distributed computational facilities internally invisibly to the user, and the other on composing the program in the form that can be given as the input to a cluster, where the user defines the necessary input files. An additional difference is that the PC grid was using the surplus processor time of computers allocated to other tasks, whereas the cluster had computers that were dedicated to scientific computation.

Based on our experiments, the cluster was 25% faster than the PC grid when the number of cores was 40 or less, mainly due to differences in hardwares. Both systems try to utilize the most powerful cores.[4] Around 80 cores, the computing times become nearly equal. Thus, a PC grid can be considered somewhat competitive with a scientific computing cluster, at least when the number of nodes increases. Both the cluster and PC grid achieved nearly linear speedup, that is, the number of option prices computed in a given time was nearly linearly proportional to the number of cores used, i.e. a grid of 120 cores was nearly 120 times faster than a single PC.

Clusters are designed to accomodate a variety of computational problems, and because of their larger memories, unified disk systems, and faster data transfer rates are clearly superior to PC grids for problems that are data intensive or that require extensive communication between parallel tasks. However, for computation-intensive tasks that require little or no communication between parallel tasks, the PC grid is an attractive alternative because of its low cost, availability of extensive computational resources (many organisations have dozens or hundreds of PCs, nearly all of whose computational capacity is idle), and other factors.

We found that dividing a large computing task into more subtasks than the number of available cores can improve the overall performance by ensuring more equal distribution of the computations. The important question regarding the optimal granularity of computational tasks and its relation to available computation infrastructure remains future work. In the example used in this paper, setting the number of jobs to twice the number of cores gave the best results. However, this is not necessarily a general result, and believe that more work should be invested in researching this issue.

[4]Cluster has Intel quad core Xeon-processors (roughly 40 cores) which are more powerful than the most powerful processors in PC grid.

We conclude that distributed computing is a feasible and attractive option for Monte Carlo option pricing computations. The dedicated cluster and the PC grid both offer comparable performance, scalability, and ability to operate heterogenous environments (i.e. computers of different performance level). The choice of system will in the end depend on management issues such as overall cost of ownership, open-source vs. commercial software, data security, and compatibility with the organization's existing computing resources.

References

Chatagny, R., and B. Chopard "A Parallel Model for the Foreign Exchange Market." *Parallel Computing*, 26 (2000), 60–622.

Ewald, C.-O., Z. Yang, and Y. Xiao "Implied Volatility from Asian Options Via Monte Carlo Methods." *Working paper*, 17 (2006).

Fouque, J. P., and C. H. Han "Pricing Asian Options with Stochastic Volatility." *Quantitative Finance*, 3 (2003), 353–362.

Glasserman, P. *Monte Carlo Methods in Financial Engineering*, Springer (2003).

Godart, C. "Parallel Implementation of a Two-Factor Cheyette-Beta Model Calibration." *Parallel Computing*, 26 (2000), 569–586.

Heston, S. L. "A Closed-Form Solution for Options with Stochastic Volatility, with Applications to Bond and Currency Options." *Review of Financial Studies*, 6 (1993), 327–343.

Tezuka, S., H. Murata, S. Tanaka, and S. Yumae "Monte Carlo grid for financial risk management." *Future Generation Computer Systems*, 21 (2005), 811–821.

Vecer, J. "A New PDE Approach for Pricing Arithmetic Average Asian Options." *Journal of Computational Finance*, 4 (2001), 105–113.

Vecer, J. "Unified Pricing of Asian Options." *Risk*, June (2002), 113–116.

Zanghirati, G., F. Cocco, G. Paruolo, and F. Taddei "A Cray T3E Implementation of a Parallel Stochastic Dynamics Assets and Liabilities Management Model." *Parallel Computing*, 26 (2000), 539–567.

Acknowledgement

The extensive computations were made possible by the facilities of the Tampere University of Technology's Institute of Advanced Computing (IAC), and by the PC grid middleware provided by Techila Technologies Ltd.

Appendix

The following Matlab code computes the value of an asian option using the Monte Carlo algorithm described in section 2.

```
function price=asian(S0,v0,r,rho,kappa,psi,E,T,M,N)
Dt=1/N;
x=zeros(N,1);
C=zeros(M,1);
for m=1:M
    w=randn(N,1);
    y=randn(N,1);
    z=rho*w+sqrt(1-rho^2)*y;
    v=v0;
    x(1)=0;
    for j=1:N-1
         x(j+1)=x(j)+(r-0.5*v)*Dt+sqrt(v*Dt)*w(j);
         v=max(0,v+kappa*(v0-v)*Dt + psi*sqrt(v*Dt)*z(j) ...
                +0.25*psi^2*Dt*(z(j)^2-1));
    end
    S=S0*exp(x);
    C(m)=max(sum(S)/N-E,0);
end
price=exp(-r*T)*sum(C)/M;
```