SuperQuant Financial Benchmark Suite for Performance Analysis of Grid Middlewares

Abhijeet Gaikwad[2], Viet_Dung Doan[1], Mireille Bossy[1], Françoise Baude[1], and Frédéric Abergel[2]

[1] INRIA Sophia Antipolis – Université de Nice – CNRS - I3S
`First.Last@sophia.inria.fr`
[2] Laboratoire de Mathématiques Appliquées aux Systèmes – Ecole Centrale de Paris `First.Last@ecp.fr`

Abstract : Pricing and hedging of higher order derivatives such as multidimensional (up to 100 underlying assets) European and first generation exotic options represent mathematically complex and computationally intensive problems. Grid computing promises to give the capability to handle such intense computations. With several Grid middleware solutions available for gridifying traditional applications, it is cumbersome to select an ideal candidate, to develop financial applications, that can cope up with time critical computational demand for complex pricing requests. In this paper we present SuperQuant Financial Benchmark Suite to evaluate and quantify the overhead imposed by a Grid middleware on throughput of the system and turnaround times for computation. This approach is a step towards producing a middleware independent, reproducible, comparable, self-sufficient and fair performance analysis of Grid middlewares. The result of such performance analysis can be used by middleware vendors to find the bottlenecks and problems in their design and implementation of the system and by financial application developers to verify implementation of their financial algorithms. In this paper we explain the motivation and the details of the proposed benchmark suite. As a proof of concept, we utilize the benchmarks in an International Grid Programming contest and demonstrate the result of initial experiments.

1 Introduction

Over the past few decades financial engineering has become a critical discipline and have gained strategic reputation for its own. Financial mathematicians keep coming up with novel and complex financial products and numerical computing techniques which often increase volume of data or computational time while posing critical time constraints for transactional processing. Generally, Monte Carlo (MC) simulations based methods are utilized to overcome typical problems like "curse of dimensionality" (e.g. integration over high dimensional space) [4]. Despite the ease of numerics, MC simulations come at the cost of tremendous computational demand in addition to slower convergence rates. However, advances in computer architectures like multi–core, many–cores, General Purpose Graphics Processing Units (GPGPUs) and their macro forms like clusters and federated Grids have made such MC simulations a handy tool for financial engineers. Financial institution are using Grid computing to perform more time critical computations for competitive advantage. With this unprecedented computational capacity, running overnight batch processes for risk management or middle-office functions to re-evaluate whole product of portfolios have almost become a passé.

Grid middleware is what makes Grid computing work and easier to work with. It provided abstractions for core functionalities like authentication across large number of resources, authorization, resource matchmaking, data transfer, monitoring and fault–tolerance mechanisms in order to account for failure of resources etc. Any robust financial service operation cannot be achieved without paying a great attention to such issues. Current Grid middleware had its beginning in the Condor Project[3] and the Globus Alliance[4]. Recently, we have seen an upsurge of academic and commercial middleware providers such as gLite[5], ProActive/GCM Parallel Suite[6], Alchemi .NET Grid computing framework[7], and KAAPI/TakTuk[8] etc. Now the question is which middleware to choose for gridifying financial applications? An obvious way is to devise a set of benchmarks and put different implementations through their paces. The middleware that results in the fastest computation could be declared as a winner. For this, one would need a standard well defined benchmark which would represent a wide set of financial algorithms, for instance MC based methods, and could also generate enough load on the middleware in test.

Benchmarks provide a commonly accepted basis of performance evaluation of software components. Performance analysis and benchmarking, however, is relatively young area in Grid computing compared to benchmarks designed for evaluating computer architecture. Traditionally, performance of parallel computer systems has been evaluated by strategically creating benchmark induced load on the system. Typically, such benchmarks comprise of codes, workloads that may represent varied computation and are developed with different programming paradigms. Some examples are STREAM[9], LINPACK[10] and MPI

[3] http://www.cs.wisc.edu/condor/
[4] http://www.globus.org/
[5] http://glite.web.cern.ch/glite/
[6] http://proactive.inria.fr/
[7] http://sourceforge.net/projects/alchemi/
[8] http://kaapi.gforge.inria.fr/
[9] http://www.gridstream.org/
[10] http://www.netlib.org/benchmark/

Benchmarks[11], SPEC[12] and most popular NAS Parallel benchmark[13]. A key issue, however, is whether these benchmarks can be used “as is” for the Grid settings. The adoption of these benchmarks may raise several fundamental questions about their applicability, and ways of interpreting the results. Inherently, Grid is a complex integration of several functionally diverse components which may hinder evaluation of any individual components like middleware. Furthermore, in order to have fair evaluation, thus any benchmark would have to account for heterogeneity of resources, presence of virtual organizations and their diverse resource access policies, dynamicity due to inherent shared nature of the Grid. Such issues in turn have led to broader implications upon methodologies used behind evaluating middlewares as discussed in [1, 11]. In our work, however, for the sake of simplicity we assume the benchmark are run on Grid nodes in isolation. Thus, we primarily focus on quantifying performance of financial applications, achievable scalability, ease of deployment across large number of heterogeneous resources and their efficient utilization.

The goal of our work presented in this paper is to design and develop SuperQuant Financial Benchmark Suite, a tool for researchers that wish to investigate various aspects of usage of Grid middlewares using well-understood benchmark kernels. The availability of such kernels can enable the characterization of factors that affect application performance, the quantitative evaluation of different middleware solutions, scalability of financial algorithms ...

The rest of this paper is organized as follows: in Section 2 we discuss the motivation behind SuperQuant Financial Benchmark Suite and propose guidelines for designing such benchmark. In Section 3 we describe the components of the benchmark suite. Section 4 presents the preliminary benchmark usage in a Grid Programming Contest. We conclude in Section 5.

2 SuperQuant Financial Benchmark suite

In order to produce verifiable, reproducible and objectively comparable results, any middleware benchmark must follow the general rules of scientific experimentation. Such tools must provide a way of conducting reproducible experiments to evaluate performance metrics objectively, and to interpret benchmark results in a desirable context. The financial application developer should be able to generate metrics that quantify the performance capacity of Grid middleware through measurements of deployability, scalability, and computational capacity etc. Such metrics can provide a basis for performance tuning of application or the middleware. Alternatively, the middleware providers could utilize such benchmarks to make necessary problem specific software design changes. Hence, in order to formalize efforts to design and evaluate any

[11] http://hcl.ucd.ie/project/mpiblib
[12] http://www.spec.org/mpi2007/press/release.html
[13] http://www.nas.nasa.gov/Resources/Software/npb.html

Grid middleware, in this paper we present SuperQuant financial benchmark suite.

Some other considerations for the development of this benchmarks are described below and significantly follow the design guidelines of NAS benchmarks suite [2],

- Benchmarks must be conceptually simple and easy to understand for both financial and Grid computing community.
- Benchmarks must be "generic" and should not favor any specific middleware. Many middlewares provide different high level programming constructs such as tailored APIs or inbuilt functionalities like provision for parallel random number generators etc.
- The correctness of results and performance figures must be easily verifiable. This requirement implies that both input and output data sets must be limited and well defined. Since we target financial applications, we also need to consider real world trading and computation scenarios and data involved therewith. The problem has to be specified in sufficient detail and the required output has to be brief yet detailed enough to certify that the problem has been solved correctly.
- The problem size and runtime requirements must be easily adjustable to accommodate new middlewares or systems with different functionalities. The problem size should be large enough to generate considerable amount of computation and communication. In the kernel presented in this paper, we primarily focus on the computational load while future benchmark kernels may impose communication as well as data volume loads.
- The benchmarks must be readily redistributable.

The financial engineer implementing the benchmarks with a given Grid middleware is expected to solve the problem in the most appropriate way for the given computing infrastructure. The choice of APIs, algorithms, parallel random number generators, benchmark processing strategies, resource allocation is left open to the discretion of this engineer. The languages used for programming financial systems are mostly C,C++ and Java. Most of the Grid middlewares are available in these languages and the application developers are free to utilize language constructs that, they think give the best performance possible or any other requirements imposed by the business decisions, on the particular infrastructure available at their organization.

3 Components of SuperQuant Financial Benchmark Suite

Our benchmark suite consists of three major components, 1) an embarrassingly parallel kernel, 2) input/output data and Grid metric descriptors, and 3) output evaluator. Each of these components are briefly described in the following sections.

3.1 Embarrassingly Parallel Kernel

We have devised a relatively "simple" kernel which consists of a batch of high dimensional vanilla and barrier options. The objective is to compute price and Greeks of maximum number of options with acceptable accuracy and within definite time interval using MC based methods. The algorithm, pseudocodes and an exemplary parallel version of MC based pricing method is provided along with the benchmark suite.

The kernel is based on simple computationally intensive financial problems, pricing and hedging of high dimensional European options, as described below. The definitions of financial terms in this section can be found in common textbooks [9, 12], although reader may find the following information self–explanatory.

European Option Pricing

The Black–Scholes (BS) model describes the evaluation of a basket of assets price through a system of stochastic differential equations (SDEs) [10],

$$dS_t^i = S_t^i(r - \delta_i)dt + S_t^i\sigma_i dB_t^i,\ i = 1, \dots, d,\ \text{where} \tag{1}$$

- $S = \{S^1, \dots, S^d\}$ is a basket of d assets.
- r is the constant interest rate for every maturity date and at any time.
- $\delta = \{\delta_1, \dots, \delta_d\}$ are the constant dividend rates.
- $B = \{B^1, \dots, B^d\}$ is a correlated d-dimensional Brownian motion (BM).
- $\sigma = \{\sigma_1, \dots, \sigma_d\}$ is a constant volatility vector.

A European option is a contract which can be exercised only at a fixed future date T with a fixed price K. A call (or put) option gives option holder right (not the obligation) to buy (or sell) underlying asset at the date T. At T, exercised option contract will pay to the option holder a position payoff $\Phi(f(S_T))$ which depends only on the underlying asset price at the maturity date S_T (for Vanilla option) or $\Phi(f(S_t), t \in [0, T])$ which depends on the entire underlying asset trajectories price S_t (for Barrier option). The definition of $f(\cdot)$ is given by the option's payoff type (Arithmetic Average, Maximum, or Minimum) [9, 12]. According to the Arbitrage Pricing Theory [10], the fair price V for the option contract is given by the following expression: $V(S_0, 0) = \mathbb{E}\big[e^{-rT}\Phi(f(S_T))\big]$. The expectation value is calculated by computing the mean with MC simulation based methods [8] and the parallel approach for which can be found in [5].

European Greeks Hedging

The Greeks represent sensitivities of option price with respect to parameters like time remained to maturity, volatility, or interest rate. Usually Greeks are higher order derivatives that are computed using finite difference methods [9, 8]. Since Greeks, are not observed in the real time market but, are information that needs to be computed, their accurate values are important, and much more compute intensive. The detail explanation of Greeks such as Delta (Δ), Gamma (Γ), Rho (ρ) and Theta (θ) can be found in [9].

The core benchmark kernel consists of a batch of 1000 well calibrated *TestCase*s. Each *TestCase* is a high–dimensional European option with up to 100 underlying assets with necessary attributes like spot prices, payoffs types, time to maturity, volatility, and other market parameters. In order to constitute an option, the underlying assets are chosen from a pool of companies listed in the equity S&P500 index[14], while volatility of each asset and its dividend rate are taken from CBOE[15]. The composition of the batch is as follows,

- 500 *TestCase*s of 10–dimensional European options with 2 years time to maturity
- 240 *TestCase*s of 30–dimensional European options with 9 months time to maturity
- 240 *TestCase*s of 50–dimensional European options with 6 months time to maturity
- 20 *TestCase*s of 100–dimensional European options with 3 months time to maturity

Thus, the objective of the benchmark is pricing and hedging of maximum number of *TestCase*s by implementing the algorithms using a given Grid middleware.

3.2 Input/Output Data and Grid Metrics Format

To facilitate processing, exchanging and archiving of input data, output data and Grid related metrics, we define relevant XML data descriptors. The *TestCases* required by the kernel and the "reference" results are also included in the benchmark suite.

- Input ***AssetPool :*** represents the database of 250 assets required to construct a basket (collection) option of assets
- Input ***CorrelationMatrix :*** defines a correlation matrix of the assets in *AssetPool*. The provided matrix is positive–definite with diagonal values 1 and correlation coefficients in the interval of $[-1, 1]$.
- Input ***TestCases :*** defines a set of *TestCase*s, input parameters, needed by the pricing and hedging algorithm discussed above. Each *TestCase* includes parameters such an option, which is a subset of *AssetPool*, a submatrix of *CorrelationMatrix*, type of payoff, type of option, barrier value if needed, interest rate, maturity date and etc.

[14] http://www2.standardandpoors.com
[15] http://www.cboe.com/

- Output ***Results :*** defines a set of *Result* which consists of *Price* and *Greeks* of individual *TestCase* and time *Metrics* required to compute each output values.
- Output Grid ***Metrics :*** defines the total time required for the entire computation.

3.3 Output Evaluator

The output evaluator is a tool to compare the results computed by different implementations of the benchmark kernel *TestCases* with "reference" results provided in the suite.

Evaluation Criteria

In order to measure the precision, the results are estimated with a confidence interval of 95% [8]. We decide the tolerable error in computing the results is 10^{-3}. Since the accuracy of the computed results relies on the spot prices of the underlying assets, we consider relative errors with respect to the "reference results". These reference results are computed with sufficiently large number of MC simulations (more than 10^6 simulations) [8], in order to achieve lower confidence interval. The **Output Evaluator** employs a point based scheme to grade the results and also provides a detail analysis of points gained per *TestCase*. For further description on the evaluation criteria, see [7].

"Reference" Results Validation

The "reference" results provided in the benchmark suite are not analytical results and are computed using MC based methods. Pricing or hedging of high–dimensional European options is not possible with a standard analytical BS formula [6]. This intriguing question of correctness of the "reference" results also diverted us to investigate methods to validate the results computed by simulation.

We observed that in some specific cases we can analytically reduce a basket of assets into a one–dimensional "pseudo" asset. The option price on this "pseudo" asset can be computed by using the BS formula. This way we can compare simulated and analytical results. Further details of the reduction techniques are given in our technical report [7]. To highlight the usefulness of this approach, we provide below a numerical example.
Numerical Example : Consider a call/put Geometric Average (GA) option of 100 independent assets with prices modeled by SDEs (1). The parameters are given as $S_0^i = 100, i = 1, \ldots, 100$, $K = 100$, $r = 0.0$, $\delta_i = 0.0$, $\sigma = 0.2$ and $T = 1$ year. The basket option is simulated by using 10^6 MC simulations. The "pseudo" asset is $\Sigma_t = \prod_{i=1}^{d} {S_t^i}^{\frac{1}{d}}$ and it is the solution of the one–dimensional SDE: $d\Sigma_t/\Sigma_t = \left(\tilde{\mu} dt + \tilde{\sigma} dZ_t\right)$ where $\tilde{\mu} = r + \frac{\sigma^2}{2d} - \frac{\sigma^2}{2}, \tilde{\sigma} = \frac{\sigma}{\sqrt{d}}$

and Z_t is a Brownian motion. The parameters of Σ are given as $\Sigma_0 = 100, \tilde{\mu} = 0.0198, \tilde{\sigma} = 0.02$. We are interested in comparing the estimated option price V of d assets with the analytical "pseudo" one $\tilde{V}$ on Σ. We denote the absolute error $\Delta V = |V - \tilde{V}|$, then the relative error is computed as follow $\eta = \frac{\Delta V}{\tilde{V}}$. In Table 1 we present the numerical results. The first column represents the estimated option prices and their 95% confidence interval. The second column gives the analytical option prices. The last two columns show the absolute and relative errors. As it can be observed, the errors are very small. We can

Table 1. Call/Put price of a GA of 100 assets option and of the "pseudo" one

Call Price V (95% CI)	"Pseudo" Call Price $\tilde{V}$	$\Delta V(10^{-4})$	η (%)
0.16815 (0.00104)	0.16777	3.8	0.22
Put Price V (95% CI)	"Pseudo" Put Price $\tilde{V}$	$\Delta V(10^{-4})$	η (%)
2.12868 (0.00331)	2.12855	1.2	0.01

reduce the errors in case of call option pricing by increasing the number of MC simulations.

4 Proof of Concept : The V Grid Plugtest and Contest

As a proof of concept, we used the SuperQuant Benchmark Suite for the **2008 SuperQuant Monte Carlo Challenge** organized as a part of **V GRID Plugtest**[16] at INRIA-Sophia Antipolis. The details of the contest and the benchmark input data can be found on the Challenge website[17]. Each participant was given an exclusive one hour access for evaluating the benchmark on two academic Grids, Grid'5000[18] and InTrigger[19], which combined consisted around 5000 computational cores geographically distributed across France and Japan.

4.1 Challenge Results

Figure 1 presents the final results of the Challenge. The participants primarily used two middlewares, ProActive, an open source Java based Grid middleware and KAAPI/TAKTUK, which coupled KAAPI , a Parallel Programming Kernel and TAKTUK, a middleware for adaptive deployment. As we can see in Figure 1, the KAAPI/TAKTUK team was successful in computing the maximum number of *TestCases* and was also able to deploy application on a significantly large number of nodes. The other teams used ProActive to implement

[16] http://www.etsi.org/plugtests/GRID2008/About_GRID.htm
[17] http://www-sop.inria.fr/oasis/plugtests2008/ProActiveMonteCarloPricingContest.html
[18] https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home
[19] https://www.intrigger.jp/wiki/index.php/InTrigger

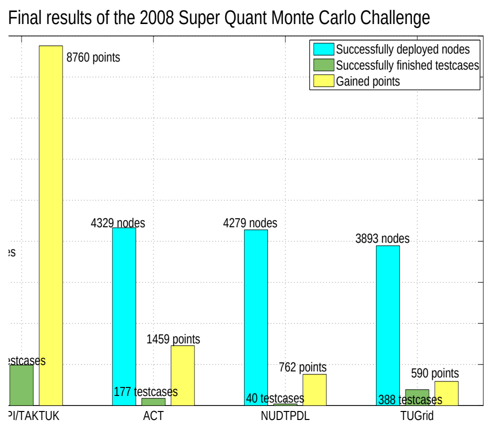

Fig. 1. Final results of the 2008 SuperQuant Monte Carlo challenge

the benchmark kernel. Both middlewares implement Grid Component Models (GCM), recently standardized by the ETSI technical committee GRID[20] for deploying the application over large number Grid nodes [3]. The infrastructure descriptors and application descriptors required by GCM were bundled with the benchmark suite. From Figure 1, we can observe that the benchmarks were not only useful to quantitatively compare two middleware solutions, but also gave the opportunity to evaluate different benchmark implementations using the same middleware. Such comparison is useful not only to middleware providers but also to Grid application developers.

5 Conclusion and Perspectives

In this paper we have presented SuperQuant Financial Benchmark Suite for performance evaluation and analysis of Grid middlewares in the financial engineering context. We described the preliminary guidelines for designing the benchmark. We also described the benchmark constituents along with a brief overview of the embarrassingly parallel benchmark kernel. As a proof of concept, we also utilized this benchmark in a Grid Programming Contest. Although this is a preliminary proposal for this benchmark, the specification of more complex kernels that can induce inter-cluster communication, high speed I/O requirements, or data processing, is necessary for truly understanding the overhead imposed by Grid middlewares in financial applications.

[20] http://www.etsi.org/WebSite/Technologies/GRID.aspx

Acknowledgments

We thank the organizers of the 2008 SuperQuant Monte Carlo Challenge for their useful support during the contest. We also thank the participants for providing permissions to use the contest results to carry out this work. Finally, this research is supported by the French "ANR-CIGC GCPMF" research project and Grid5000 has been funded by ACI-GRID.

References

1. P. Alexius, B.M. Elahi, F. Hedman, P. Mucci, G. Netzer, and Z.A. Shah. A Black-Box Approach to Performance Analysis of Grid Middleware. *LNCS*, 2008.
2. D. Baileym, J. Barton, T. Lasinski, and H. Simon. The NAS Parallel Benchmarks. *Technical Report RNR-91-002 Revision 2, NAS Systems Division, NASA Ames Research Center*, August 1991.
3. F. Baude, D. Caromel, C. Dalmasso, M. Danelutto, V. Getov, L. Henrio, and C. Pérez. GCM: A grid extension to fractal for autonomous distributed components. *Annals of Telecommunications*, 64(1):5–24, 2009.
4. R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.
5. S. Bezzine, V. Galtier, S. Vialle, F. Baude, M. Bossy, V.D. Doan, and L. Henrio. A Fault Tolerant and Multi-Paradigm Grid Architecture for Time Constrained Problems. Application to Option Pricing in Finance. *2nd IEEE International Conference on e-Science and Grid Computing*, Netherlands, December 2006.
6. F. Black and M. Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637, 1973.
7. V.D. Doan, A. Gaikwad, M. Bossy, F. Baude, and F. Abergel. A financial engineering benchmark for performance analysis of grid middlewares. Technical Report 0365, INRIA, 2009.
8. P. Glasserman. *Monte Carlo Methods in Financial Engineering.* Springer, 2004.
9. J. Hull. *Options, futures, and other derivatives.* Prentice Hall Upper Saddle River, NJ, 2000.
10. D. Lamberton and B. Lapeyre. *Introduction to stochastic calculus applied to finance.* CRC Press, 1996.
11. G. Tsouloupas and MD Dikaiakos. Gridbench: A tool for benchmarking grids. In *Grid Computing, 2003. Proceedings. Fourth International Workshop on*, pages 60–67, 2003.
12. P. Wilmott et al. *Derivatives: The Theory and Practice of Financial Engineering.* J. Wiley, 1998.