Design of High Performance Financial Modeling Environment

F.O.Bunnin*, Y.Guo, Y.Ren, J.Darlington
{fob1,yg,yr,jd}@doc.ic.ac.uk

*Department of Computing, Imperial College of Science, Technology and Medicine, 180 Queen's Gate, London SW7 2BZ UK, *corresponding author*

Abstract

The aim of our system is to generate solution code from a high level specification of a financial instrument. Solution code calculates prices and hedge ratios which are partial derivatives of the price with respect to various parameters. The user manipulates a representation of the problem at the domain level, with the complexities of the computer implementation hidden. As many of the problems have no analytical solution, symbolic transformations manipulate the equations specifying the stochastic model and instrument into forms that can be solved numerically. Techniques such as finite differences, spectral methods and Monte Carlo simulation are provided in sequential and parallel versions. The mathematical correctness of the transformation steps is examinable as is the degree of error introduced by approximating transformations. We include examples for the Black-Scholes model and for the Hull White stochastic volatility model. A Linux cluster is used to price an Asian option using the Hull White SV model.

Key words: derivative securities, financial modeling, problem solving environments, high performance computing

1 Introduction

When viewing a problem at the computer level the greatest difficulty may be in translating domain level operations and concepts into corresponding sequences of operations on computer memory and such devices. This difficulty is a reason for the use of general purpose programming languages instead of assembly language for most programming tasks. General purpose programming languages, although abstracting computer operations to a procedural, algorithmic level with constructs for sequence, selection and iteration, do not

abstract to a level in which real world objects and operations are directly modelled.

This situation becomes more difficult when parallel machines are used. The operational level programming becomes much more complicated due to the complication of multiple data flow and multiple control flow. It becomes vital that a parallel application can be developed using higher level abstraction, i.e. abstraction at the problem level. By employing domain knowledge as well as optimisation technology with respect to the special computational structure inherent in the application, efficient parallel code can be generated.

These difficulties motivate the development of problem-solving environments: systems that enable end-users to solve computational problems within a specific domain. Two design objectives of PSEs stated in Houstis et al [15] are to "develop a PSE architecture that supports the plug-and-play paradigm" and to "exploit multilevel abstractions and the complex properties of science". PSEs have been developed for many scientific areas such as physical simulation, weather forecasting [6], PDE solving [1,16] and PDE solving on the web [24]. Many such systems, such as Parallel Ellpack[16], develop parallel code to solve PDEs. SciComp have developed a program synthesis system, SciFinance [21], that generates finite difference code to solve financial PDEs. We base our approach on the design objectives above, and build on SciComp's innovative work in the financial domain.

The majority of scientific PSEs focus on PDE modeling. In finance, however, the fundamental models are stochastic equations. This expands the features required of a financial PSE. We provide tools for the symbolic manipulation of stochastic processes, for the transformation of stochastic models into deterministic PDEs (which then can be solved using finite differences or by the pseudospectral method), and for the numerical solution of problems stated in stochastic form. The latter invariably involves the computation of high dimension integrals, for which parallel machines are ideal.

1.1 Financial Modeling and Parallel Programming

The financial domain is suitable for parallel programming for several reasons. Financial modeling involves sophisticated mathematical models whose solution requires intricate numerical computation. At present the realism of models used in practice is limited both by speed of execution on single processor machines and by the complexity of hand coding programs in low level languages. The numerical pricing of exotic derivatives under such models as stochastic volatility stock models and multi-factor term structure models is computationally time consuming yet amenable to parallelisation. Since these

are models of systems influenced by human behaviour, the systems change unpredictably as knowledge and empirical results about the systems affect human participants in the system. Therefore there is always a need to formulate new models approximating the changing financial system. Better models, that more accurately approximate the relationships between financial variables, are a basis for companies' competitive advantage. This advantage results in faster, more accurate derivative pricing and hedging, which increases sales revenue and market-making profitability, and uncovers more arbitrage opportunities, increasing proprietary trading revenue.

New instruments are continuously being devised by financial engineers, creating new ways for clients to decompose risk and providing high-margin products. Developing code for these new instruments and models is time consuming, expensive and error prone. The requirements of users and regulators are strict: the code must deliver real time prices and hedge ratios in a distributed system, and must give verifiable results for internal and external auditing. Since the models and instruments are often proprietary banks are unwilling to reduce costs by outsourcing to third parties who develop software for several clients. This is all in a competitive environment that demands rapid application development.

The complexity of writing code for parallel machines compounds these problems and argues for a software development system that allows modelers to focus on the domain level abstractions of the problem and produces code meeting the above requirements. The aims of our system are summarised as:

- Rapid application development for new types of options and new models.
- Real time performance in a distributed environment for computationally intensive calculations.
- Ease of system development, maintenance and extension.
- Examinable accuracy.
- Transparent risk and pricing technology

The scope of our system is defined by the inputs allowed and the validity of the transformations. The inputs must be Ito stochastic differential equations representing complete no-arbitrage markets. A market is complete when every lower bounded contingent claim can be replicated by an admissible portfolio of market securities. Thus multi-factor, stochastic interest rate and certain stochastic volatility stock and term structure models can be dealt with, but general jump-diffusion models are outside of the scope. The scope of instruments is those whose payoff can be expressed as a function of the values of the underlying asset and time. Thus barrier, binary, look-back, and Asian calls and puts all are within the scope.

The structure of our paper reflects the structure of the system. This is based

on the PSE design objectives cited above [15]: multilevel abstractions and the plug-and-play paradigm. Section 2 discusses domain level abstraction: the hierarchy of instrument types and its software representation. The next few sections detail how models are specified and transformed to result in code that implements a financial instrument under the assumptions of that particular model. Section 3 describes the stochastic models for underlying instrument behaviour and the symbolic transformations that give expressions for theoretical prices of derivative instruments. Section 4 moves on to algorithmic level abstractions and discusses numerical methods to solve the pricing expressions. Section 5 discusses parallelism and templates at the implementation level. In section 6 examples are given illustrating the approach. We demonstrate the development of pricing method code for two financial instruments: first a European call option under the prototypical Black Scholes model, and second a discretely sampled arithmetic average price European call option under a stochastic volatility model. Numerical results and speed-up obtained follow in the figures. We also discuss user input and the level of automation. Section 7 introduces the design of distributed system aspects and the software context of the environment's output.

2 Domain Level Abstraction: Financial Instruments

Financial concepts and instruments have a hierarchical structure independent of the models themselves. The hierarchy defines the ontology of financial instruments and expresses the semantic relationships between instruments. This domain level concept is represented in software as an abstract class hierarchy. For example, an option class is a subclass of a security class. The purpose of these classes are to define the operations that derived concrete classes will support and hence to define the interface for applications that utilise financial instrument objects, such as valuation and risk management applications (see also section 7).

Without the assumption of a model for the behaviour of the underlying instrument these classes necessarily have no implementation. At this level we have abstracted above details that determine unique theoretical prices. Thus abstracting to the instrument specification level leaves ambiguity in the meaning of a derivative's price, which is resolved for concrete classes by specifying a model. This mechanism allows coupling of the abstract class specification with a specific model, which results in a concrete class, and then future coupling of the abstract class with different models resulting in different implementations of the abstract class, that is in different concrete classes. Extension is achieved through the creation of new instruments by inheritance and evolution through the adoption of a new model for the same instrument, both with automatic code generation.

Support for the plug-and-play paradigm is provided by maintaining a separation of interface from implementation. This is conducive to integration with existing applications within complex computing environments. The commonality of interface, due to inheritance, permits polymorphism. Client applications are only concerned with the model free interface of an instrument, and do not need modification when models and implementation algorithms change and new classes modeling new instruments are created.

Within this hierarchy of abstract classes the task is then to generate concrete classes implementing instruments under specific models with efficient algorithms. This is for existing instruments in the hierarchy, and for newly created instruments that are represented by new subclasses in the hierarchy. As the interface can be inherited from the abstract superclass, this task is reduced to generating implementations of the class' methods, such as `price`, `delta`, and so on. The implementation of the `price` method is fundamental, as the approximation of hedge ratios such as `delta` is often trivial once `price` has been computed. The task of translating from interface and model specifications (user supplied) to a machine program is split into stages reflecting the levels of abstraction: the mathematical level, the algorithmic level and the implementational level.

3 Mathematical Level Abstraction: Stochastic Models

The highest unambiguous level of abstraction is the mathematical level. The mathematical formulation of the problem is unambiguous and well defined. Mathematical proofs ensure the existence and uniqueness of solutions, and determine the validity of and conditions for solution techniques. Thus for a concrete class both instrument and model specifications must be mathematical. Since the problem is mathematically specified, we can benefit by using a Computer Algebra system, such as Mathematica [35]. The main benefits of Mathematica for this level are the ease of writing symbolic transformations using rewrite rules, and its built in functions such as symbolic calculus.

The abstract classes described above include as attributes technical terms, such as "Call", "European style", "Barrier", that describe and specify financial instruments to users. Each technical term has a mathematical meaning and consequently the system applies the mathematical meaning when generating code for that instrument. For example the term "American style" applied to an option denotes that early exercise is allowed. Hence mathematically we have the constraint: `value >= exercisevalue`, where the exercise value depends on the type of option.

The user inputs model specifications as Ito stochastic differential equations

(SDE). Asset price behaviour is generally modeled by SDEs (see Duffie [10]). Important considerations regarding the type of model are usefulness in practice, computational tractability and theoretical realism. The linearity or non-linearity of the SDE, the number of stochastic variables involved, and whether the process has the Markov property, are the main mathematical classifications that influence the considerations above. A Markov process is one where the future behaviour of the process is not dependent on the past values of the process, apart from the current value. At the moment performance limitations restrict practitioners to one or two factor Markov models. The systematic way we generate code and the use of parallel machines facilitates the implementation of multi-factor and non-Markov models.

In order to construct expressions for the prices of derivative instruments the environment manipulates the SDEs using symbolic transformations. The price process of a non-dividend paying stock can be represented as the solution to:

$$S_t = S_0 + \int_0^t a(S, v)dv + \int_0^t b(S, v)dW_v,$$

where the second integral is an Ito integral, with respect to a standard Brownian motion, W. The first integrand is known as the drift of S, and the second as the diffusion, which is related to the volatility of the stock. SDEs are often written in symbolic differential form:

$$dS = a(S, t)dt + b(S, t)dW,$$

where the meaning is the same as the integral form above. Results from stochastic calculus are used to transform the form of the model into one that is more easily solved. Any instrument that derives its value from the current price of the above stock can be represented by a function of S and time: $u(S_t, t)$. To put this in a more explicit form Ito's lemma [22] is used:

$$du = (\frac{\partial u}{\partial t} + a(S, t)\frac{\partial u}{\partial S} + \frac{1}{2}b(S, t)^2\frac{\partial^2 u}{\partial S^2})dt + b(S, t)\frac{\partial u}{\partial S}dW.$$

Further transformations are made until the problem is stated in a tractable form. The transformations are coded in Mathematica as functions or rewrite rules. There follows an example demonstrating the formation of the Black-Scholes equation for the price of a derivative security using transformations of user input.

A deterministic partial differential equation (PDE) is formed after a series of transformations, the solution of which is the price of the derivative. Specific final and boundary conditions specify exactly what kind of derivative instrument is being priced. In this example the constant proportional drift and diffusion stock price model is specified by the user.

$$dS = SDE[S, \mu S, \sigma S]$$

The function SDE takes arguments of the process name, its drift and its diffusion. It returns a SDE.

$$dS = \mu S dt + \sigma S dW$$

This is passed as the second argument to a function that applies Ito's lemma to result in the SDE whose solution is the first argument.

$$du = ItoLemma[u[S,t], dS]$$

$$du = (\frac{\partial u}{\partial t} + \mu S \frac{\partial u}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2})dt + \sigma S \frac{\partial u}{\partial S} dW.$$

Hence, following the analysis demonstrated by Black and Scholes [18], we form an instantaneously risk free portfolio of -1 derivative securities and $\frac{\partial u}{\partial S}$ shares, and use the no-arbitrage condition to equate the return on this portfolio to the risk free rate, r. Thus we derive the Black-Scholes PDE. For more details on the analysis see Duffie [10]. The function FormPDE is written in Mathematica and uses the above analysis.

$$FormPDE[u[S,t], dS]$$

$$\frac{\partial u}{\partial t} + rS\frac{\partial u}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2} - ru = 0. \tag{1}$$

The final condition is provided by the payoff, $g(S,T)$, at expiry time, T, of the derivative security:

$$u[S,T] = g[S,T] \tag{2}$$

Alternatively we could have applied results related to the Girsanov theorem to the stochastic process of S to make the discounted price processes, $(S_t e^{-rt})$ and $(u_t e^{-rt})$ martingales with respect to an equivalent probability measure, Q

[10]. Then an explicit expression for u is the conditional expectation, under the martingale measure, of its known payoff function, g, at expiry time, T, discounted by the instantaneous risk free interest rate, r.

$$u[S_t, t] = E_t^Q[e^{-r(T-t)} g[S_T]] \tag{3}$$

In general a sequence of transformations are applied to the user supplied model and result in a tractable expression for the model. At each step there may be alternative transformations that could be applied that will result in different final expressions. However all the final expressions, at this stage, are mathematically equivalent. In the example given above the expression for u[S,t] involving a conditional expectation (3) and the u[S,t] as the solution to the partial differential equation (1,2) can be shown to be equivalent using the Feynman-Kac formula [28]. Thus there is no canonical form to which the problem representation can be reduced to, but each step aims to transform the problem into one whose solution is more readily computed, whilst maintaining the meaning of the problem.

4 Algorithmic Level Abstraction

The construction of an efficient algorithm from sequential and parallel subalgorithms is vital as commercial demands make performance a key issue. The inputs to the pricing equation can change dramatically in seconds, and for the price to be valid the solution must be available in real time. At this level transformations leading to efficient algorithms are selected either by the environment, based on rules, or by the modeler using domain knowledge.

For example our system provides a function `CoordinateTransform` that takes as arguments a PDE, a list of the coordinates used in the PDE, a list of the transformations to be applied to each old coordinate, and a list of the new coordinates. It returns a PDE in the new coordinates. When we model interest rates using the Cox Ingersoll Ross model the space of the short term interest rate r is $[0, \infty)$. Developed economies' rates are usually between 0% and 20%, though we must include higher rates for approximately correct boundary conditions. Algorithms using a uniform grid for the discretisation of r will expend more computation on economically unlikely values, i.e. $r > 0.2$, than the relevant values. However we can choose to apply a non linear transformation $x = 1/(\gamma r + 1)$, γ being a constant (see Duffie [10]). A uniform grid for the discretisation of x thus contains proportionally more values of x corresponding to lower values of r than higher values of r, depending on the value of γ. Interest rates in developing economies are more likely to reach higher levels, perhaps ranging up to 100%, and so the modeler chooses an appropri-

ate level for γ when applying the coordinate transform to the model of the specific economy's interest rates. Our function `CoordinateTransform` is thus parameterised by the appropriate PDE and transformations. It then returns a new PDE that is easily solved with an efficient uniform grid finite difference method or pseudospectral method. Additionally performance models of algorithms' implementations on target architectures could be used as a basis for automatic algorithm selection.

In selecting and applying an algorithm approximations of the problem are made and thus errors are introduced. The validation of numerical methods in a PSE framework is an important but hard problem [32]. We approach this problem by analysing the deterministic or probabilistic nature of the errors, checking consistency, convergence and stability criteria, and thereby quantifying the accuracy of the resulting code. In some cases this can be done automatically by the environment (see below in sections 4.1 and 4.2). The representations of the problems resulting from the previous stage will be of two types:

- PDEs with final conditions derived from instrument payoffs, boundary conditions derived from asymptotic behaviour of the payoffs, and possibly free boundary conditions derived from American style features.
- Stochastic processes in the form of SDEs, and the conditional expectations of functions of such processes.

4.1 Numerical methods for PDEs

The issues involved in selecting algorithms to solve the PDE representation are the type of PDE: elliptic, parabolic or hyperbolic, linear or non-linear, its order and dimensionality; the availability of explicit solutions; and the stability, convergence and efficiency of numerical methods. The majority of PDEs derived from the models examined above are linear second order parabolic equations, for reasons due to the Feynman-Kac formula, [28,25] with the number of dimensions determined by number and dimensionality of the SDEs of the underlying processes.

Our system provides rewrite rules that when applied to PDEs lead to problem representations that are passed to templates implementing finite difference and pseudospectral methods. The finite difference method's stability and convergence criteria and measures are well documented [2,36]. We follow Wilmott et al [34] by using it for many types of options, including strongly path-dependent ones with early exercise features. Explicit schemes are only conditionally stable. Theta method implicit schemes are unconditionally stable if the weighting for the fully implicit scheme is 1/2 or greater [2]. We make Crank-Nicolson

our default scheme due to its stability and its higher order of convergence than Euler schemes. Implicit schemes involve the solution of a matrix equation at each time step, and this can be done directly, for example using LU decomposition, or iteratively. The iterative approach, though slower, lends itself to solving American option problems, for example with PSOR (projected successive over-relaxation).

In our system for the finite difference method the partial derivatives of the PDE are replaced by differences using rewrite rules of the form:

$$\frac{\partial^2 V[x,t]}{\partial x^2} \to \frac{V[x+\delta x,t]+V[x-\delta x,t]-2V[x,y]}{(\delta x)^2}$$

Combinations of such rules corresponding to schemes are made into lists and given the appropriate names, such as `ExplicitEuler` and `ImplicitEuler`. `CrankNicolson` and `Theta` schemes are formed by combining the `ExplicitEuler` and `ImplicitEuler` schemes. Once the PDE has been discretised by a scheme, rules to transfer the equations to an indexed grid are applied, thus forming a system of linear equations.

We estimate the accuracy of the finite difference approximation by symbolically calculating the truncation error, using the Taylor expansion. Consider the PDE $LU(x,t)=0$, where L is a partial differential operator and $U(x,t)$ is the solution. The truncation error at the point $(i\delta x, j\delta t)$ is defined by $T_{i,j}(U)=F_{i,j}(U)$, where $F_{i,j}$ is the finite difference scheme used. Our rewrite rule `TaylorExpand` expresses $F_{i,j}(U)$ in terms of δx, δt, and partial derivatives of $U(i\delta x, j\delta t)$. This is used in our `TruncationError` function:

```
TruncationError[LU_,FU_]:=Expand[FU/.TaylorExpand]/.LU->0
```

The leading order of this truncation error is then retrieved by finding the first terms of the truncation error of the form $a_n(\delta x)^n$, $n=0,1,2,\ldots$, with coefficient a_n non-zero. For the scheme to be consistent the limit of the truncation error must be zero when $\delta t \to 0$ and $\delta x \to 0$.

Finite difference methods are simple and easy to understand. However they are generally low order and if higher order methods are required we provide the pseudospectral method. Since the space domain for underlying instruments is regular, unlike irregular 3D domains encountered in engineering, there is usually no reason to use finite element methods. Pseudospectral methods, however, are suitable for regular domains and offer high order of convergence for suitably differentiable solutions [11,12,29]. We have implemented rewrite rules and pseudospectral solver code using Chebyshev basis functions and Chebyshev collocation points. The method and our implementation is discussed in detail in section 6. The time dimension accuracy of the pseudospectral method

is calculated in a similar way to that described above for the finite difference method. However the space dimension accuracy is harder to quantify. For certain types of options special techniques are useful, such as moving mesh methods for moving barrier options [21]. We plan to implement adaptive techniques in the future.

In both the finite difference case and the pseudospectral case the equations are passed to an algorithmic template along with the choice of a solver, which may also be selected by the system. We have implemented a library of solvers including direct and iterative methods.

The complexity of numerical solution of PDEs increases exponentially with the dimensionality of the PDE. Thus high dimensional PDEs present computational difficulties. Formulation of the problem as an expectation of a stochastic process permits the use of Monte Carlo simulation, where the complexity increases only linearly with dimension [18].

4.2 Numerical methods for SDEs

Our problem is to solve a conditional expectation of a known function of the underlying instrument given a SDE for the continuous process of the underlying. Our system provides rewrite rules to approximate this problem in ways suitable for Monte Carlo or tree based methods. The rewrite rules leading to Monte Carlo simulation replace the expectation by a summation of weighted function values for sample values of the underlying. When the exact solution of the SDE satisfied by the underlying's process is known we compute it at the appropriate discretisation points for the calculation of the derivative instrument. We provide an analytic SDE solver that finds the exact solution for special cases of SDEs partially based on the work of Cyganowski [9]. Otherwise we provide rewrite rules to approximate the SDE by time discretisations based on the Ito-Taylor expansion [23]. We are computing the expectation of a function of the underlying's stochastic process, and hence are concerned with the distributional, rather than pathwise, properties of the process. Therefore the weak order of convergence for time discretisation schemes is of relevance. If the Euler discretisation is chosen, the weak order of convergence is 1.0 (see Kloeden [23] for details), which is noted in the specification file (see below in section 6.5).

For a discussion of Monte Carlo methods in finance see Boyle et al [3]. The main issues in Monte Carlo simulations are the pseudo-random number generator, the conversion from uniformly distributed random variables to normally distributed random variables, variance reduction techniques, the number of sample runs to use and calculation of confidence intervals. The accuracy of

the Monte Carlo method for calculating an expectation such as:

$$E[f(x)] = \int f(x)p(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} f(x_i),$$

where the x_i are independent samples from the density $p(.)$, and $E[.]$ denotes the mathematical expectation, can be shown to be $O(n^{-1/2})$, where n is the number of samples. By the central limit theorem the estimate $m = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$ is normally distributed with mean $\mu = E[f(x)]$ and standard error $SE = \sqrt{\sum (f(x) - m)^2/n}$. Therefore confidence limits for estimating μ are $m \pm Z_c SE$, where Z_c is dependent on the confidence level, e.g. $Z_{95\%} = 1.96$.

Tree based methods approximate the underlying process by a discrete time, discrete space multinomial process [7]. Our rewrite rules thus replace the underlying process by a time and space indexed process. This approximation is then passed to a template implementing a binomial lattice solution method. A major computational consideration is whether the tree recombines at each time step. When approximating a non-Markov process they do not recombine, and thus the number of nodes at each time step increases exponentially with the number of time steps, as opposed to the linear growth in nodes for recombining trees [31]. Presently the environment supports only a sequential recombining binomial lattice template for tree methods. Lattices are conceptually close to explicit finite difference methods, and similar stability conditions apply.

5 Implementation level

The algorithmic level transformations convert the declarative form of the problem to an imperative form, i.e. an algorithm detailing specific solution steps is chosen. This conversion is a necessary step to arrive at a machine program, and the choices made in reaching this step will determine, to a large part, the speed of the resulting program. At the mathematical level possible transformations such as reducing the dimensionality of the PDE, coordinate transforms to reduce the steepness of the solution or to eliminate space or time dependence of the coefficients, or simply the choice between a PDE formation or a conditional expectation formation, can affect dramatically the efficiency of the resulting algorithm. The choice of numerical method, whether to use an explicit or implicit finite difference scheme for example, also influences efficiency but to a lesser extent. The higher the level of abstraction at which choices are made, the larger effect the choice has on the speed of the resulting program. Thus at the implementation level we provide pre-built optimised implementations of numerical methods, implicitly assumed to exist by the choices made

previously. These implementations: templates or solvers, exploit and encapsulate low level parallel behaviour, hiding implementation from the modeler yet providing flexibility, efficiency and reuseability. We make full use of third party provided libraries, such as the Sun Performance library, for solvers, and implement templates using C++ and the Message Passing Interface.

Higher level algorithms, such as Monte Carlo simulation, theta method finite difference solvers, and pseudospectral solvers are coded as templates that are parameterised by transformed user input and lower level solvers. In the Monte Carlo simulation algorithm independent simulation runs can be executed on different processors. The objective is to produce independent random number sequences on each (virtual) processor. A major consideration is how to distribute generation of sequences or sequences themselves among processors. Centralised generation with distribution of the generated sequence leads to communication costs and the processor owning the generator is likely to become a communication bottleneck. Therefore distributing generators is to be preferred. Coddington [8] suggests using a lagged Fibonacci generator with processors having random and uncorrelated seed tables and different lag tables, or using a combined linear congruential generator with sequence splitting, that is each processor generates a contiguous subsequence of the sequence. Clearly the subsequences must be disjoint. Alternatively different types of generators may be employed at each processor. It is always essential to check results using different generators, as it is impossible to "prove" randomness. We ease testing by encapsulating generators as objects that can be passed as parameters to the Monte Carlo template.

The explicit finite difference algorithm is suitable for domain decomposition of the grid, for example block-striping by row with a halo for boundary communication. Implicit finite difference methods involve the solution to matrix equations at each time step that can be performed by direct or iterative methods. Parallelisation is achieved by cyclic partitioning of the matrix for direct methods and by using a red-black update algorithm on a "chessboard" partitioning of the grid for iterative ones.

For convenience and efficiency solvers are pre-built components that can be plugged into the algorithmic form of the problem. Thus, for example, a solver component using LU decomposition that has been optimised can be re-used in various algorithms, such as implicit finite difference algorithms and pseudospectral algorithms. For different architectures, such as a distributed memory parallel computer like the AP3000, functionally equivalent, machine optimised versions of the components may be used in order to exploit such resources. Commonality of interface between functionally equivalent components ensures that the surrounding algorithm need not be modified. Opportunities for parallelism arise in both the higher level algorithm templates, and the lower level subalgorithms. In addition to this internal parallelism, the concur-

rent computation of different instruments is an obvious source of parallelism.

Thus an implementation is produced by composing such components with a template defining the basic algorithm and with problem specific information such as linear system equations and final and boundary conditions. As noted previously, these implementations act as the methods of domain level classes. New implementations of the class are generated from a different sequence of transformations.

6 Examples

For concreteness we show two examples of using transformations to generate code. The first uses the pseudospectral method to price a call option under the Black-Scholes model. The second prices a discretely sampled arithmetic average price call option under the Hull and White stochastic volatility model. For this we use Monte Carlo simulation on up to 30 linux PCs. The speed-up from using more processors is shown in the results. We outline the numerical method, and then show the sequence of transformations to specifications that combines with the method to produce problem specific code. We stress that the code generated is used as the `price` method for a domain level class representing a financial instrument (see section 2). Thus the financial application developer uses these classes with the parallel behaviour of their implementation encapsulated in their methods.

6.1 Pseudospectral Chebyshev Method

Pseudospectral (PS) methods play an important role in the numerical solution of differential equations. PS solutions are approximations by global basis functions, whereas finite difference and finite element methods are local approximations. For problems with smooth solutions PS method can obtain higher order numerical solutions as the number of collocation points increases.

Suppose a function $u(t,x)$ is approximated by the Nth degree polynomial expressed as

$$u(t,x) \sim u_N(t,x) = \sum_{k=0}^{N} a_k(t) T_k(x)$$

where T_k is a Chebyshev polynomial of the first kind of degree k. Here the

Chebyshev polynomial of degree k is defined by

$$T_k(x) = \cos(k \cos^{-1} x).$$

The interpolation points in the interval $(-1, 1)$ are chosen to be the extrema

$$x_j = \cos \frac{\pi j}{N} \quad (j = 0, 1, \cdots, N)$$

of the Nth order Chebyshev polynomials $T_N(x)$.

It follows that

$$T_k(x_j) = \cos \frac{\pi jk}{N}.$$

The coefficients $\{a_k\}_{k=0}^N$ are chosen such that

$$\begin{aligned}
&\sum_{k=0}^{N} T_k(x_j) \frac{da_k}{dt} = \sum_{k=0}^{N} LT_k(x_j) a_k(t) \quad for \;\; j = 1, 2, \cdots, N-1 \qquad (4)\\
&\sum_{k=0}^{N} a_k(t) T_k(x_0) = u(t, x_0)\\
&\sum_{k=0}^{N} a_k(t) T_k(x_N) = u(t, x_N)
\end{aligned}$$

Where $-LT_k(S) = rS\frac{\partial T_k}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 T_k}{\partial S^2} - rT_k$ for the Black-Scholes equation.

The problem (4) is an initial value ODE system. The initial condition for $\{a_k\}_{k=0}^N$ are determined by $u_N(0, x) = u(0, x)$, that is

$$\sum_{k=0}^{N} a_k(0) T_k(x_j) = u(0, x_j), \quad for \;\; j = 0, 1, \cdots, N \qquad (5)$$

The first derivative of u is approximated by

$$\frac{du_N}{dx} = \sum_{k=0}^{N} b_k^{(1)}(t) T_k(x)$$

where by the properties of Chebyshev polynomials (see [29])

$$b^{(1)} = E^{(1)} a.$$

We apply the Implicit-Euler method and the Crank-Nicholson scheme to the ODEs (4). So that

$$\sum_{k=0}^{N} a_k^{i+1} T_k(x_0) = u(t, x_0) \tag{6}$$

$$\sum_{k=0}^{N} [T_k(x_j) - \delta t \theta L T_k(x_j)] a_k^{i+1} = \sum_{k=0}^{N} [T_k(x_j) + \delta t (1-\theta) L T_k(x_j)] a_k^{i} \tag{7}$$

$$\sum_{k=0}^{N} a_k^{i+1} T_k(x_N) = u(t, x_N) \tag{8}$$

for $j = 1, 2, \cdots, N-1$ and $i = 1, 2, \cdots, M$, here δt is the time stepsize and θ is the parameter of schemes, i.e. $\theta = 1, 1/2, 0$ for Implicit, Crank-Nicolson and Explicit schemes, respectively.

This algorithm can be written as an algorithmic template that takes as arguments the partial differential expression LT_k, the initial condition function $u(0, x)$, the boundary value functions $u(t, x_0)$ and $u(t, x_N)$, which are problem specific, and also parameters N for the number of basis functions to use in the approximation, and θ for the numerical scheme. In addition implicit scheme solutions of the ODEs require LU decomposition. A library call within the algorithm code is used so that parallel or sequential machine optimised versions can be used.

6.2 Application to Black-Scholes

Given a filtered probability space $(\Omega, F, (F_t), P)$ we assume the price of a non-dividend paying stock S is an F_t adapted process that satisfies:

$$dS = \mu S dt + \sigma S dW,$$

Where $(W_t, F_t; 0 \leq t \leq \infty)$ is a standard Brownian Motion; μ and σ are constants. Following the example in section (3.1) the Black-Scholes equation for a derivative security u with S as underlying is formed:

$$\frac{\partial u}{\partial t} + rS\frac{\partial u}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 u}{\partial S^2} - ru = 0. \tag{9}$$

$$for \;\; 0 \leq t \leq T, \;\; 0 < S < \infty.$$

The infinite space domain is truncated to $[a, b]$ and then for application of the Chebyshev Pseudospectral method the truncated space is transformed to

the computational domain $[-1,1]$. Let $S = \frac{b-a}{2}x + \frac{b+a}{2}$ and let $\tau = T - t$ to make the equation a forward one. Then $J = \frac{\partial S}{\partial x} = \frac{b-a}{2}$. Where $x \in [-1,1]$ and $S \in [a,b]$ and T is expiry time.

Therefore the Black-Scholes equation becomes

$$\frac{\partial U}{\partial \tau} = \frac{1}{2}\sigma^2 S^2/J^2 \frac{\partial^2 U}{\partial x^2} + rS/J\frac{\partial U}{\partial x} - rU := LU \tag{10}$$

where $U(\tau, x) := u(t, S)$.

The payoff of a European Call is $(S-K)^+$, where K is the strike price. The expiry time condition for (9) is thus $u(T,S) = (S-K)^+$. When $S \to 0, u(t,S) \to 0$, and when $S \to \infty, u(t,S) \to S - Ke^{-r(T-t)}$. The time dimension is discretised by a mesh $\pi : 0 = \tau_0 < \tau_1 < \ldots < \tau_M = T$ with equidistant time steps $\tau_{i+1} - \tau_i = \delta\tau$ for $i = 0$ to $M-1$. For each time step the solution is approximated by a linear combination of the first N+1 Chebyshev polynomials:

$$U(\tau_i, x) \sim U_N(\tau_i, x) = \sum_{j=0}^{N} a_j(\tau_i) T_j(x)$$

The coefficients $a_j(\tau_i)$ are found by setting the approximation U_N equal to the solution U at the collocation points $x_j, j = 1$ to $N-1$, and using the boundary conditions for x_0 and x_N. This is achieved by replacing U by U_N in (10), and performing the required differentiation. This results in a set of N+1 equations, N-1 ordinary differential equations and the two boundary conditions, in N+1 unknowns: $a_j(\tau_i)$ $j = 0$ to $m, i = 1$ to n. The solutions of the coefficients at τ_{i-1} are the initial conditions for the set of ordinary differential equations at τ_i. At τ_0 the coefficients are found directly from the initial condition. Crank-Nicolson is used to solve the ordinary differential equations at each time step. At T the solution in the form of a function of x is found from the calculated $a_j(T)$. Table 1 has results for when K=10, r=0.1, σ=0.4, T=1, N=100 and dt=0.01.

6.3 Asian option under Stochastic Volatility using Monte Carlo Simulation

In this example code to price discretely sampled arithmetic average price call options is generated. Under a risk-neutral measure the underlying stock follows the stochastic volatility model proposed by Hull and White [18]. Note that this market model is made complete by the inclusion of one European Option as a traded security.

$$dS = rSdt + \sqrt{V}SdW_S \tag{11}$$

Table 1
European Call Option.

Stock Price	Option Price (PS)	Option Price (True)	Relative Error
3	0.0020	0.0018	-0.1059
4	0.0187	0.0180	-0.0370
5	0.0817	0.0808	-0.0116
6	0.2307	0.2302	-0.0021
7	0.4954	0.4960	0.0012
8	0.8878	0.8897	0.0020
9	1.4036	1.4065	0.0020
10	2.0284	2.0318	0.0017
11	2.7436	2.7474	0.0014
12	3.5305	3.5347	0.0012
13	4.3724	4.3774	0.0011
14	5.2556	5.2619	0.0012
15	6.1689	6.1776	0.0014
16	7.1039	7.1163	0.0017
17	8.0539	8.0717	0.0022
18	9.0141	9.0393	0.0027
19	9.9810	10.0158	0.0035
20	10.9516	10.9987	0.0043

$$dV = a(b - V)dt + \xi V^{\alpha} S dW_V \tag{12}$$

The system creates the SDEs (11) and (12) from the user specifications:

$$dS = SDE[S, rS, \sqrt{V}S]$$

$$dS = \mu S dt + \sqrt{V} S dW$$

$$dV = SDE[V, a(b - V)\xi V^{\alpha} S]$$

$$dV = a(b - V)dt + \xi V^{\alpha} S dW.$$

Discretisation schemes can be derived from the Ito-Taylor expansion [23]. Applying the Milstein scheme to both equations we obtain:

$$Milstein[S, dS]$$

$$S + rS\Delta + S\sqrt{V}\Delta W + \frac{SV(\Delta W^2 - \Delta)}{2}$$

$$Milstein[V, dV]$$

$$V + a(b - V)\Delta + SV^{\alpha}\xi\Delta W + \frac{S^2V^{2\alpha-1}\alpha\xi^2(\Delta W^2 - \Delta)}{2}.$$

Thus we can simulate approximate sample paths of S using the above equations.

The payoff of an arithmetic average price call is:

$$\texttt{payoff} = Max[Sum[S_i, i, n]/n - K, 0]$$

where S_i are the discrete samples of S at times t_i and K is the fixed strike price. The sampling may be for any period: daily, weekly, monthly and so on. The price of the option is given by:

$$e^{-rT}E_t^Q[\texttt{payoff}] \tag{13}$$

where Q is the risk-neutral measure that produces equations (11) and (12) and T is the expiry time of the option. The dimensionality of this integral depends on the sampling frequency and the life to expiry of the option. For all non-trivial cases the Monte Carlo simulation method dominates non-probabilistic methods. Although there are many variance reduction techniques for Monte Carlo methods, for simplicity we will not discuss them here. Using the basic Monte Carlo method we simulate a large number of sample paths for S, and for each sample path calculate the payoff of the Asian option. The mean and standard deviation of the sample payoffs is calculated to give a confidence interval for the true price of the option, giving a measure of the accuracy of the code. As each sample path can be simulated independently, parallelisation is achieved by assigning to each processor a portion of the total number of sample paths. The only communication needed is to accumulate the individual results from the processors. As discussed above each processor must use an independent random number stream. We have used a linear congruential uniform random number generator with a different subsequence on each processor. The polar rejection method then transforms these into standard normally distributed numbers, using on average $4/\pi$ uniform random numbers for one normal random number. Table 2 has results for this example with

Table 2
Asian Call Option under Hull White SV model

Processors	Option Price	Confidence interval: 95%	Time/s	Rel. Speedup	Rel. Efficiency
1	10.0531	10.0431-10.0631	51.0420	1	1
5	10.0514	10.0414-10.0614	10.1706	5.0186	1.0037
10	10.0487	10.0387-10.0587	5.1143	9.9802	0.9980
15	10.0404	10.0304-10.0504	3.4276	14.8913	0.9928
20	10.0440	10.0340-10.0540	2.6498	19.2626	0.9631
25	10.0459	10.0359-10.0559	4.0975	12.4568	0.4983
30	10.0446	10.0346-10.0546	3.7055	13.7747	0.4592

monthly sampling, 1 million sample paths, K=10, r=0.1, a=0.3, b=0.4, α=1, ξ=1, V_0=0.4, S_0=20. The code was compiled with mpiCC and executed on a network of 30 Pentium PCs running Linux. The network cannot be used exclusively, and some processors had a high load when the results were obtained. This explains the drop in relative efficiency when more than 20 processors are used.

6.4 Transformations

In order to produce a concrete implementation the environment needs an instrument specification (either from an existing abstract class or from a newly created abstract class for a new type of instrument) and a model specification in the form of an SDE for the underlying's behaviour. If no abstract class for the instrument exists, the user gives the necessary information for one to be created: the payoff, whether early exercise is permitted, the underlying instrument, and the base class.

The steps below summarise the transformations used in generating the `price` method for a concrete Equity Call Option class.

(1) Set payoff and boundary conditions (from abstract Equity Call Option class):
`payoff=Max[S-K,0]`
`lowerboundary=0`
`upperboundary=S-K Exp[-r(T-t)]`
(2) Specify the model and form a partial differential equation from the stochastic equation.
`dS=SDE`$[S, \mu S, \sigma S]$
`pde1=FormPDE[u[S,t],dS]`
(3) Transform the physical space to the computational space, thus derive a

new equation.
`pde2=coordinatetransform[pde1,{S,t},` $\frac{b-a}{2}x + \frac{b+a}{2}, T - \tau, x, \tau$`]`

(4) Derive the partial differential operator L from the new equation and form the approximating expression LT_k.
`ltk=LTK[pde2]`

(5) Parameterise the template code with the transformed problem representation: that is LT_k, and the initial and boundary functions. This produces the `price` method for a concrete Equity Call Option class.
`PseudospectralChebyshev[ltk,payoff,lowerboundary,upperboundary]`

6.5 User Input

The transformation process is currently semi-automatic. All the transformation rules are written in Mathematica, and users may use Mathematica's command line and input palette interface to perform the transformations. However this requires a detailed knowledge of the system and the command line input style is not popular.

Therefore we also provide a form-like interface (see figure 1). The user enters the option specification: its name (for file name and class name purposes), its payoff, and upper and lower boundary conditions. The model is then specified by the drift and diffusion terms that define the stochastic process of the underlying instrument. All transformations are performed by clicking on a bank of related buttons. In the section forming the problem as a PDE the first two buttons take user input of the drift and diffusion terms of the SDE. The third button creates an SDE by applying the function SDE[] to the drift and diffusion arguments. The next button changes the probability measure to a risk-neutral measure by applying the Girsanov transformation. It is necessary for the expectation formation but is optional for the PDE formation. The next bank of buttons applies the FormPDE transformation to the SDE created and a function u of the underlying process, and coordinatetransform, which also takes user input of the change of variables to use. Alternatively in the section forming the problem as an expectation, after creating the SDE the user may choose the equivalent Martingale measure approach and click on buttons to apply the Girsanov transformation to the SDE and then to discretise using the Milstein scheme.

The user, therefore, decides on the form of the problem to solve and the solution method by their choice of which sections of the input form to fill in and which buttons to click. The output of the form is a file containing all the relevant information: the option specification, the solution method and its accuracy (such as $O((\delta t)^2)$), and equations derived from the model such as *ltk* above. A text processing program combines information from this file with

the appropriate pre-built components to generate the final code and thus the class implementation. Important parts of the specification file are included as comments in the final code.

In the future we aim to make the transformation process fully automatic, with default transformations and algorithms being determined by features of the user specifications, such as dimensionality of the SDEs and possibility of early exercise. The user will also be able to override the defaults based on their domain knowledge.

7 System Design

A bank is a distributed system. The information system in a bank is a distributed information system. Within a bank many different sections of the bank will be involved in any one transaction. For example the sales desk, the dealing desk, risk management and operations and back office staff will all be involved in selling an interest rate swap to a corporate customer. Each may be interested in various aspects of the transaction at various times from quoting, execution, end of day mark-to-marketing, until expiry or unwinding of the transaction. Therefore, it is the basic requirement for building up a financial application that the system can be organised as financial components, i.e. software representations of financial instruments. Such components can be embedded into the distributed information system of a financial institution.

In our development, the system, organised as software components, consists of the following objects:

- view objects: visual representation of data from financial instrument and transactions, i.e. tables of prices, list of hedge ratios.
- server objects (with methods automatically generated): parallel implementation of computational model for instrument; given market data computes prices and hedge ratios.
- persistent storage: state of instrument in the form of a grid of contingent prices as well as transactional information such as date and counterparty of transaction.

The view objects run on the client platform (PC or workstation) and are written in Java. Their function is to visually represent the component to the user. Data and information are represented visually in the form of tables, graphs, plots and other graphical representation. Java provides the ideal feature of platform independence. The server objects are instantiations of the financial instrument classes described above. They execute on a high performance parallel server. The distributed object computing model provided by the Java

RMI (Remote Method Invocation) mechanism forms an efficient and secure client/server environment where risk management service enabled by the high performance server can be provided to the users across the intranet of a financial house. The state of the server objects are saved to persistent object store. This state may be in the form of a finite difference grid, which is the solution to a financial equation. When revaluing the instrument, say when marking the transaction to market, the numerical code would not have to be rerun, but instead the value could be found by table look-ups and interpolation, the table being the state saved in the object store. Similarly approximations to hedge ratios are calculated this way. The system is under construction in the Imperial College Parallel Computing Centre.

8 Conclusion

We have discussed our design of a financial modeling environment that generates accurate and efficient code from high level model and instrument specifications. The systematic process of transforming the specification into executable code was demonstrated. The implementation issues and transformation techniques were detailed at each level of abstraction from the domain level to the implementation level. The system has been used to generate valuation and hedging code for European and American Vanilla, Barrier and Asian options using the pseudospectral method, explicit, fully implicit and Crank-Nicolson finite difference schemes, Monte Carlo simulation and binomial trees.

The generated codes act as the methods of concrete classes representing types of financial instrument. The stochastic models used have been relatively simple, and we plan to generate code using more sophisticated stochastic models in the future. Certain types of options introduce complications in their numerical solution, and thus we plan to add solvers and algorithm templates in order to exploit a wider range of numerical methods.

Acknowledgements

This project is funded by the ESRC award RO22250129. F.O. Bunnin is funded by the EPSRC award 97306304. We are grateful for the very useful comments of two anonymous referees.

References

[1] R.L. Akers, E. Kant, C.J. Randall, S. Steinberg, R.L. Young, SciNapse: A Problem-Solving Environment for Partial Differential Equations, *IEEE Computational Science and Engineering* vol. 4 no. 2 (1997)

[2] W.F. Ames, *Numerical Methods for Partial Differential Equations* (Academic Press Inc. 1992)

[3] P. Boyle, Broadie, Glasserman, Monte-Carlo Methods for Option Pricing, *Journal of Economic Dynamics and Control* vol. 21 (1997) 1267-1321

[4] R. Brent, *Random Number Generation on Supercomputers-Extended Abstract*, Europar98

[5] W. Briggs, *A Multigrid Tutorial* (SIAM 1987)

[6] G. Cats, R. van Engelen, L. Wolters, Tomorrow's Weather Forecast: Automatic Code Generation for Atmospheric Modeling, *IEEE Computational Science and Engineering* vol. 4 no. 2 (1997)

[7] J. Cox, S. Ross, M. Rubinstein, Option Pricing: A Simplified Approach, *Journal of Financial Economics*, vol 7 (October 1979) 229-263

[8] P.D. Coddington, Random Number Generators for Parallel Computers, (Syracuse University, 1997)

[9] S. Cyganowski, Solving Stochastic Differential Equations with Maple, http://net.indra.com/ sullivan/q253.html

[10] D. Duffie, *Dynamic Asset Pricing*, (Princeton University Press, 1996)

[11] B. Fornberg, *A Practical Guide to Pseudospectral Methods*, (Cambridge University Press, 1996)

[12] B. Fornberg and D. Sloan, A review of PS methods for solving PDEs, in *Acta Numerica* (1994) 203-267

[13] R. Geske, K. Shastri, Valuation by Approximation: A Comparison of Alternative Option Valuation Techniques, *Journal of Financial and Quantitative Analysis* vol. 20 (March 1985) 45-72

[14] E. Gallopoulos, E. Houstis, J. Rice, Computer as Thinker/Doer: Problem-Solving Environments for Computational Science, *IEEE Computational Science and Engineering* vol. 1 no. 2 (1994)

[15] E.N. Houstis, E. Gallopoulos, R. Bramley, J. Rice, Problem-Solving Environments for Computational Science, *IEEE Computational Science and Engineering* vol. 4 no. 2 (1997)

[16] E.N. Houstis, S.B. Kim, S. Markus, P. Wu, N.E. Houstis, A.C. Catlin, S. Weerawarana, http://www.cs.purdue.edu/homes/markus/research/pubs/pde-solvers/paper.html

[17] P. Hudak, Building Domain-Specific Embedded Languages, *ACM Computing Surveys* 28A (4) (December 1996)

[18] J. Hull, *Options, Futures, and Other Derivative Securities*, (Prentice-Hall, 1993)

[19] J. Hull, A. White, Valuing Derivative Securities using the Explicit Finite-Difference Method, *Journal of Financial and Quantitative Analysis* vol 25 (March 1990) 87-100

[20] E. Kant, Synthesis of Mathematical Modelling Software, *IEEE Software* 1993

[21] E. Kant, C. Randall, A. Chhabra, Using Program Synthesis to Price Derivatives, *Journal of Computational Finance* vol. 1 no. 2 (Winter 1997/98) 97-129

[22] I. Karatzas, S.E. Shreve, *Brownian Motion and Stochastic Calculus*, (Springer-Verlag, 1991)

[23] P.E. Kloeden, E. Platen, *Numerical Solution of Stochastic Differential Equations*, (Springer-Verlag, 1994)

[24] S. Markus, S. Weerawarana, E.N. Houstis, J.R. Rice, Scientific Computing via the Web: The Net Pellpack PSE Server, *IEEE Computational Science and Engineering* vol. 4 no. 2 (1997)

[25] M. Musiela, M. Rutkowski, *Martingale Methods in Financial Modeling*, (Springer, 1997)

[26] B. Nardi, *A Small Matter of Programming*, (MIT Press 1993)

[27] S.N. Neftci, *An Introduction to the Mathematics of Financial Derivatives*, (Academic Press, 1996)

[28] B. Oksendal, *Stochastic Differential Equations*, (Springer, 1998)

[29] D. Gottlieb, S. A. Orszag, *Spectral Methods for Partial Differential Equations*, (SIAM, Philadelphia, 1977)

[30] S. Paskov, J. Traub, Faster Valuation of Financial Derivatives, *Journal of Portfolio Management* 22 (Fall 1995)

[31] R. Rebonato, *Interest Rate Option Models*, (Wiley, 1996)

[32] J. Rice, PSE Challenges for the 21st Century, European Research Conference: Advanced Environments and Tools for High Performance Computing, San Feliu de Guixols, Spain June 1999

[33] M. Rubinstein, Exotic Options, Compilation of articles from *Risk* magazine

[34] P. Wilmott, J. Dewynne, S. Howison, *Option Pricing. Mathematical Models and Computation*, (Oxford Financial Press, 1993)

[35] S. Wolfram, *The Mathematica Book*, (Cambridge University Press, 1996)

[36] G. D. Smith, *Numerical Solution to Partial Differential Equations: Finite Difference Methods*, (Oxford University Press, 1985)

[37] R. Zvan, P. Forsyth, K. Vetzal, Robust Numerical Methods for PDE Models of Asian Options, *Journal of Computational Finance* vol. 1 no. 2 (Winter 1997/98) 39-78