Using Premia and Nsp for Constructing a Risk Management Benchmark for Testing Parallel Architecture

Jean-Philippe Chancelier, Bernard Lapeyre
Université Paris-Est, CERMICS, Ecole des Ponts
Champs sur Marne, 77455 Marne la Vallée Cedex 2, France
Email: jpc@cermics.enpc.fr, bl@cermics.enpc.fr

Jérôme Lelong
Ecole Nationale Supérieure de Techniques Avancées
ParisTech
Unité de Mathématiques Appliquées
42 bd Victor 75015 Paris
Email: jerome.lelong@ensta.fr

Abstract

Financial institutions have massive computations to carry out overnight which are very demanding in terms of the consumed CPU. The challenge is to price many different products on a cluster-like architecture. We have used the Premia software to valuate the financial derivatives. In this work, we explain how Premia can be embedded into Nsp, a scientific software like Matlab, to provide a powerful tool to valuate a whole portfolio. Finally, we have integrated an MPI toolbox into Nsp to enable to use Premia to solve a bunch of pricing problems on a cluster. This unified framework can then be used to test different parallel architectures.

1. Introduction: The context of risk evaluation

Banking legislation (Bale II[1][4]) imposes to financial institutions some daily evaluation of the risk they are exposed to because of their market positions. The main investment banks own very large portfolios of contingent claims (several thousands of claims, 5000 being a realistic estimation).

For a given contingent claim and model parameters, the evaluation of the price (or other risk features as delta, gamma, vega, ...) requires a computation time which can greatly vary, from a few millisecond (as for a standard option in the Black and Scholes model) to dozens of minutes (as for American options price on a large number of underlying assets).

A model is specified by several parameters: volatility, interest rate, ... and, in the context of risk evaluation, it is necessary to price the contingent claims for various values of these model parameters to measure its sensibility to the parameters. As a consequence, a huge number of atomic computations (around $10^8$) is necessary to evaluate the risk of the whole portfolio. These computations must be done on a daily basis to provide an evaluation of the position of the bank to the risk control organism. They are so complex that the financial institutions often need to use very large computer clusters with up to several thousands of nodes.

2. Premia: a library for numerical computations in finance

Premia is devoted to the computation of prices and hedges for derivatives (see [7] for an introduction), which is a major issue for financial institutions. It is a research project dedicated to the development of algorithms and scientific documentation for option pricing, hedging and model calibration. It is developed in the framework of the MATHFI research team uniting scientists working in probability and finance from INRIA and École des Ponts.
This project keeps track of the most recent advances in the field of computational finance in a well-documented way. It focuses on the implementation of numerical analysis techniques for both probabilistic and deterministic numerical methods. An important feature of the Premia platform is the detailed documentation which provides extended references in option pricing. Besides being a single entry point for accessible overviews and basic implementations of various numerical methods, the aim of the Premia project is to be a powerful testing platform for comparing different numerical methods to each other.

Premia is developed in interaction with a consortium of financial institutions or departments presently composed of: Calyon, Natixis, Société Générale, Raiffeisen Zentralbank, Bank Austria. The members of the consortium support the development of Premia and help to determine the directions in which the project should evolve.

Premia is a fairly complete library with regards to what is currently used in advanced finance. For an exhaustive presentation see[8]. In its current public release, it contains finite difference algorithms, tree methods and Monte Carlo methods for pricing and hedging European and American options on equities in several models going from the standard Black-Scholes model to more complex models such as local and stochastic volatility models and even Lévy models. Sophisticated algorithms based on quantisation techniques or Malliavin calculus for European and American options are also implemented. More recently, various interest rate and credit risk models and derivatives have been added.

3. Tools

3.1. Nsp

Nsp is a Matlab-like Scientific Software Package developed under the GPL license. It is a high-level programming language which can be used as a scripting language which gives an easy access to efficient numerical routines. It can be used as an interactive computing environment or as a programming language. It supports imperative programming and features a dynamic type system and automatic memory management. It contains internally a class system with simple inheritance and interface implementation, this class system is visible at the Nsp programming level but not extendable at the Nsp level. When used as an interactive computing environment, it comes with online help facilities and an easy access to GUI facilities and graphics.

A large set of libraries are available and it is moreover easy to implement new functionalities. It requires to write some wrapper code also called interfaces to give glue code between the external library and Nsp internal data. The interface mechanism can be either static or dynamic. Using dynamic functionalities we are able to build toolboxes.

Nsp shares many paradigms with other Matlab-like Scientific Softwares as for example: Matlab, Octave, ScilabGtk[3][5] and also with scripting languages such as Python for instance.

Two typical toolboxes were used in this work. The first one is the Nsp Premia toolbox which gives access at Nsp level to the Premia financial library. The second one is a MPI interface, which gives at Nsp level access to mainly all MPI-2 functions.

3.2. MPI toolbox for Nsp

Having a direct access to MPI functions within a scripting language can be very useful for many aspects. The main advantage is that it gives an easy way to get familiar to the large set of MPI functions which can be tested interactively. It also hides the tedious work of packing and unpacking complex data since a scripting language contains high level data and the packing and unpacking of such data can be hidden to the user.

Similar toolboxes are available. As for example, Mpitb[6] is a toolbox developed initially by Javier Fernández Baldomero and Mancia Anguita which provides such a full MPI interface for the Matlab and Octave languages. The Nsp MPI toolbox follows the same philosophy and was implemented using the Nsp interface language. Note however that the Matlab version of the Mpitb toolbox is implemented through wrapper code which are called mexfiles and since a mexlib interface library is available in Nsp it was possible to make the Matlab toolbox work in Nsp with mainly no additional work. But, for maximum efficiency and flexibility the MPI function interfaces have been directly written using the Nsp interface API.

Now, we give some examples to highlight facilities that are given inside Nsp to access MPI primitives. It is possible to launch a master Nsp and then to spawn slaves Nsp, this is done by using the MPI_Comm_spawn primitive as shown on Fig. 1:

The code given in Fig. 1 will start a new Nsp which will execute the transmitted cmd to start interacting with the master through a merged communicator. Note that the interface between Nsp and MPI does no just consists in a set of functions but also on new Nsp object devoted to MPI. For example mpicomm_create creates a new Nsp communicator object which internally contains a MPI communicator. Since starting a set of Nsp slaves is a classic task, the previous given code can be written in a Nsp function NSP_spawn and it is then possible to start n slaves by the simple Nsp command

`NEWORLD=NSP_spawn(n);`

It is possible to transmit and receive almost all the Nsp objects using the MPI_Send_Obj and MPI_Recv_Obj Nsp functions. These two functions use the fact that almost
MPI_Init();
COMM = mpicomm_create('SELF');
INFO_NULL = mpiinfo_create('NULL');

cmd = "exec('src7.x/loader.sce');MPI_Init();";

args = ['"name","ns-child","-e"', cmd];
[children, errs] = MPI_Comm_spawn(nsp_exe, args, 1, INFO_NULL, 0, COMM);

cmd = cmd + "parent=MPI_Comm_get_parent();";

[NEWORLD] = MPI_Intercomm_merge(parent, 1);

nsp = mpi_comm_get_parent(NEWORLD);

B = l (3)
(1) = s (1x1)


Figure 1. Mpi primitives at Nsp level

all the Nsp objects can be serialised in a Serial object. The
two functions MPI_Send_Obj and MPI_Recv_Obj use
internal serialisation and packing to transparently transmit
Nsp Objects.

B = l (3)
(1) = s (1x1)

It gives us a very easy way to transmit a Premia problem
to a Nsp slave. Moreover it is easy to transmit jobs to Nsp
slaves as Nsp strings.

For standard objects such as non sparse matrices, cells,
lists and hash tables it is possible to use MPI_Send directly
or combined with the MPI_Pack function.

A=[%t, %f];
B={'foo', [1:4], 'bar', rand(100,100));
H=hash_create(A=A,B=B);
P=MPI_Pack(H,MCW),
MPI_Send(P,randk,TAG,MCW)

Receiving the transmitted packed data is also easy. A
mpibuf object can be created at Nsp level with a proper
size and be given to the MPI_Recv function for receiving
the transmitted packed data. A call to MPI_Unpack will
then recreate a Nsp object.

Moreover, it is possible to serialise objects at Nsp level
and transmit them. Note that in that case MPI_Recv_Obj
will unseals directly the Serial object received.

A large file called TUTORIAL.sce can be used to inter-
actively to learn MPI in general and also its Nsp interface.
This file is a simple Nsp adaptation of the excellent Mpitb
tutorial for Matlab [6].

3.3. Premia toolbox for Nsp

For long, the only way to use Premia was from the
command line. With the growing of Premia every year, the
need of real graphical user interface has become more and
more pressing. The idea of embedding the Premia library in a
Matlab-like Scientific Software has come up quite naturally.
Unlike a standalone graphical user interface, embedding
Premia into Matlab-like Scientific Software provides two
ways of accessing the library either through the scripting
language or using the graphical capabilities of the software
The possibility of accessing the Premia functions directly at the interpreter level makes it possible to make Premia interact with other toolboxes. Since the license of Premia gives right to freely distribute the version of Premia two year older that the current release, it was important that the scientific software used can be freely obtained and has extensive graphical feature. Nsp fulfilled all these conditions.

The inheritance system of Nsp enables to easily add new objects in the interpreter. This is how we introduced a new type named PremiaModel, through which the wide range of pricing problems described in Premia and their corresponding pricing methods are made available from Nsp. The results obtained in a given problem can be used in any post-treatment routines as any other standard data.

For practitioners, the daily valuation of a complex portfolio is a burning issue to which we tried to answer using MPI/Nsp/Premia. Given a bunch of pricing problems to be solved, which are implemented in Premia, how can we make the most of Nsp and the two previously described toolboxes? First, we needed a way to describe a pricing problem in a way that is understandable by Nsp so that it can create the correct instance of the PremiaModel class. We implemented the load and save methods for such an instance relying on the XDR library (eXternal Data Representation). This way, any PremiaModel object can be saved to a file in a format which is independent of the computer architecture; these files can be reloaded later by any Nsp process. Then, a bunch of pricing problems can be represented by a list of files created either from the scripting language or using the graphical interface. Let us give an example of how to create such a file. To save the pricing of an American call option in the one dimensional Heston model using a finite difference method, one can use the following instructions

```plaintext
P = premia_create()
P.set_asset[str="equity"]
P.set_model[str="Heston1dim"]
P.set_option[str="PutAmer"]
P.set_method[str="FD_Fem_Achdou"]
save('fic', P)
```

Creating an instance of the PremiaModel class and setting its parameters is very intuitive. The object saved in the file fic can be reloaded using the command load('fic').

To solve this list of problems, we could use a single Nsp process but as the problems are totally independent it is quite natural to try to solve them in parallel using the MPI toolbox presented in Section 3.2. The master process reads all the files and creates the corresponding instances of the PremiaModel class. Then, each instance is serialised and sent to a given remote node using MPI's communication functions.

### 4. Practical experiments

For our experiments, we consider a basket of American put options in dimension 3 with different maturities varying from 1 month to 5 years by steps of one month. We will use this basket of 60 options as a benchmark to test the Nsp/MP/Premia toolbox. The code described in Fig. 3 enables to solve our problem using a cluster in a way that the load of the cluster is well balanced between the different nodes. First, the master sends one job to each slave and as soon as a slave sends the answer back, it is assigned a new job. This mechanism goes on until all the jobs have been treated. Table 1 shows the speed-up observed for various number of nodes. When using several node, we have made sure that the master is located on a different machine from the slaves. This ensures that when comparing the speed-ups are taking into account the time spent exchanging data.

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>CPU time</th>
<th>speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>182</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>89.8</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>45.4</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>26.79</td>
<td>6.7</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>8.7</td>
</tr>
<tr>
<td>14</td>
<td>18.3</td>
<td>9.9</td>
</tr>
<tr>
<td>19</td>
<td>12.7</td>
<td>14.3</td>
</tr>
</tbody>
</table>

Table 1. Speed up

We have built a web site http://cermics.enpc.fr/~lelong/gcpmf which contains links to the tools described in the paper. It also contains the portfolio generator, some proposed portfolios benchmarks and test results. A test portfolio containing one instance of each possible Premia problems is also given. This web page will be regularly updated as we have more results from our experimentations on clusters. Our aim is to provide using Nsp syntax the descriptions of large portfolios of options which are representative of the computations carried on by banks to evaluate their risk level as they are required to do by the law.
if `~MPI_Initiated()` then   MPI_Init();end
MPI_COMM_WORLD=mpicomm_create('WORLD');
[rank] = MPI_Comm_rank (MPI_COMM_WORLD);
[size] = MPI_Comm_size (MPI_COMM_WORLD);
SLV = (rank <> 0)
MST = ~ SLV;
TAG=4;

if SLV
   while %t then
      Maturity=0;
      MPI_Recv (Maturity,-1,TAG,MPI_COMM_WORLD); // receives the vector
      if Maturity < 0 then break; end
      exec ('premia.sce');
      result= [rank,Maturity,L(1)(3)];
      MPI_Send(result,0,TAG,MPI_COMM_WORLD); // sends the results back
   end
else // Here at master
   Nt = 60;
   nb_per_node = floor (Nt / (size-1));
   Maturities = linspace(0.25,10,Nt);
   prm=grand(1,'perm',length(Maturities));
   M=Maturities(prm);
   for slv=1:size-1
      MPI_Send (M(slv),slv,TAG,MPI_COMM_WORLD);
   end
   M(1:size-1)=[];
   res=[];
   result=ones(1,3);
   while %t
      MPI_Recv(result,-1,TAG,MPI_COMM_WORLD);
      sl=result(1);
      res=[res;result];
      if ~isempty(M) then
         MPI_Send (M(1),sl,TAG,MPI_COMM_WORLD);
         M(1)=[];
      else
         break;
      end
   end
for slv=1:size-2 // we still have size-2 Revc to perform
   MPI_Recv(result,-1,TAG,MPI_COMM_WORLD);
   res=[res;result];
end
for slv=1:size-1 // tell the slaves to stop working
   MPI_Send([-1],slv,TAG,MPI_COMM_WORLD);
end
save('matu.bin',res,CPU=t);
end
MPI_Finalize(); // finalize slaves and master

Figure 3. Interactive dispatching of jobs
References


