Mathematical modeling of complex problems is far more mainstream than it once was, and Excel has played a major role in making that happen. From linear programming to statistical models, Excel is used to implement many different modeling techniques. Built-in functions and custom worksheets significantly enhance Excel's flexibility and suitability as the modeling tool of choice within existing customer-based applications.

Performance remains an important limiting factor in the expansion of Excel based modeling techniques. To address this issue Microsoft introduced HPC Server's extensions(http://technet.microsoft.com/en-us/library/ff877824%28WS.10%29.aspx). These extensions enable code written in Excel to run on clusters, overcoming the limitations of current Excel-based models.

The goal of this paper is to explore the steps needed to port an Excel-based model to run effectively on clusters using HPC, and to demonstrate the benefits of such porting.

1. Conversion of Excel code to HPC

Standard steps to convert Excel code to HPC

To take advantage of HPC cluster capabilities, an Excel workbook originally written to be run on a standard desktop must be modified, or additional modules created. Microsoft documentation addresses 3 major approaches to consider when converting Excel code to run on an HPC cluster (http://www.microsoft.com/downloads/en/confirmation.aspx?familyid=A48AC6FE-7EA0-4314-97C7-D6875BC895C5&displaylang=en).

Those approaches can be summarized as follows:

1. In the first approach, Excel workbook should be deployed on each node of a cluster. Each instance of Excel will then run the same calculations, but with a different data set. A special module should be written to combine the results of all the calculations. This implementation allows the Excel code to run as written, while using additional code to distribute data to the different instances of Excel and subsequently combine the results.

2. The second approach requires rewriting individual cells or macros that cause performance degradation as functions of the Excel Link library. Rewritten functions can then be deployed on an HPC cluster and run in parallel when called from the original Excel model.
3. The third approach uses Excel as a cluster SOA client that consumes cluster-deployed resources, in order to allow calculations to be deployed on a Window HPC cluster. This is similar to the second approach, but allows Excel to call any resource deployed on HPC.

Each approach has its advantages and drawbacks. Application developers should consider the requirements of the specific Excel application before deciding which approach or combination of approaches to use.

Example: Conversion of CDO break-even default rate model to HPC

The Collateralized Debt Obligation (CDO) is one of the complex financial products that have become popular in recent years. Several approaches are used to rate CDO products. One of the most common is the methodology used by the Standard & Poor rating agency (S&P). An important element of the S&P approach is the assessment of the break-even default rate of a specific CDO. The break-even default rate is the maximum percentage of defaults a CDO can sustain and still pay the ultimate principal and all due interest
(http://books.google.com/books?id=b9glyZxCcicC&printsec=frontcover#v=onepage&q&f=false).

In this example, an Excel-based model to simulating break-even default rates for CDO products was used to illustrate the performance gains that can be achieved by modifying the model to run on an HPC cluster. Analysis of the code showed that implementation was done as one complex macro called from the individual cells. Each cell contained a different set of parameters simulating various scenarios in terms of portfolio performance, interest rate environment, and time horizon. This approach led to a strictly sequential calculation, making this model an ideal candidate for demonstrating the performance improvements that can be achieved from parallelism.

Since the macro that performs the actual simulation is complex, it was decided that the first of the approaches discussed above should be used when porting to the HPC environment. As a result, the actual simulation code remained untouched. At the same time, special routines required by Microsoft specification to handle data partitioning and merging of results were created. Appendix A presents the original code and workflow, while Appendix B shows the code that performs the same calculations but is ported to the HPC. Appendix B also contains a flowchart of the resulting Excel-based model working in the HPC environment.

2. Results

Figure 1 below shows the results of executing a CDO break-even default rate model in an HPC environment. The Y-axis represents speed of execution relative to the original sequential code, while the X-axis represents the number of cores used in the experiment.

The experiment was run for models containing 44, 92, 156, 348, 540 and 1020 modeling cells. Increasing the number of cells in the model improves the ability of the model to predict break-even default rates in a different interest rate environments and expanded time horizons.

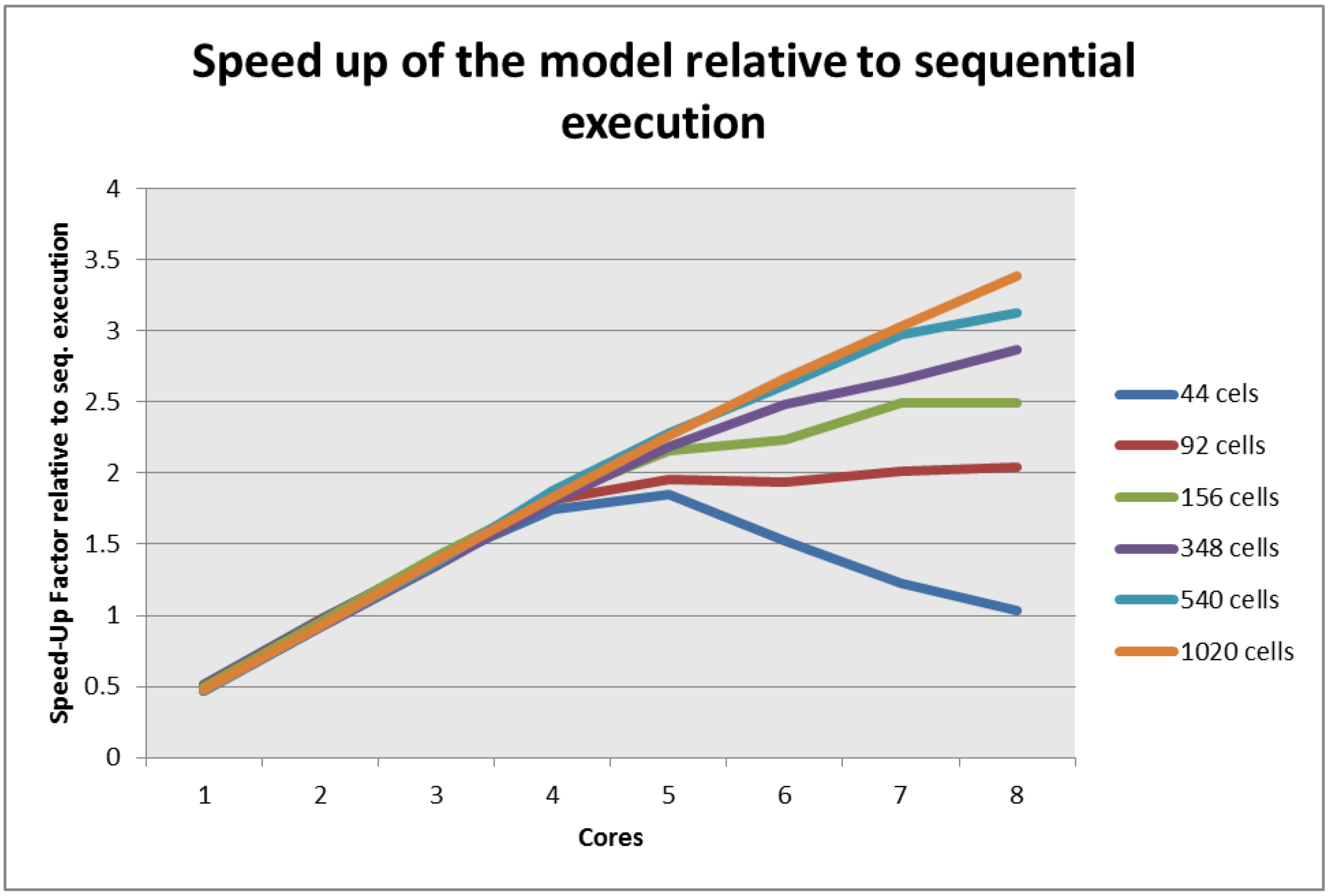


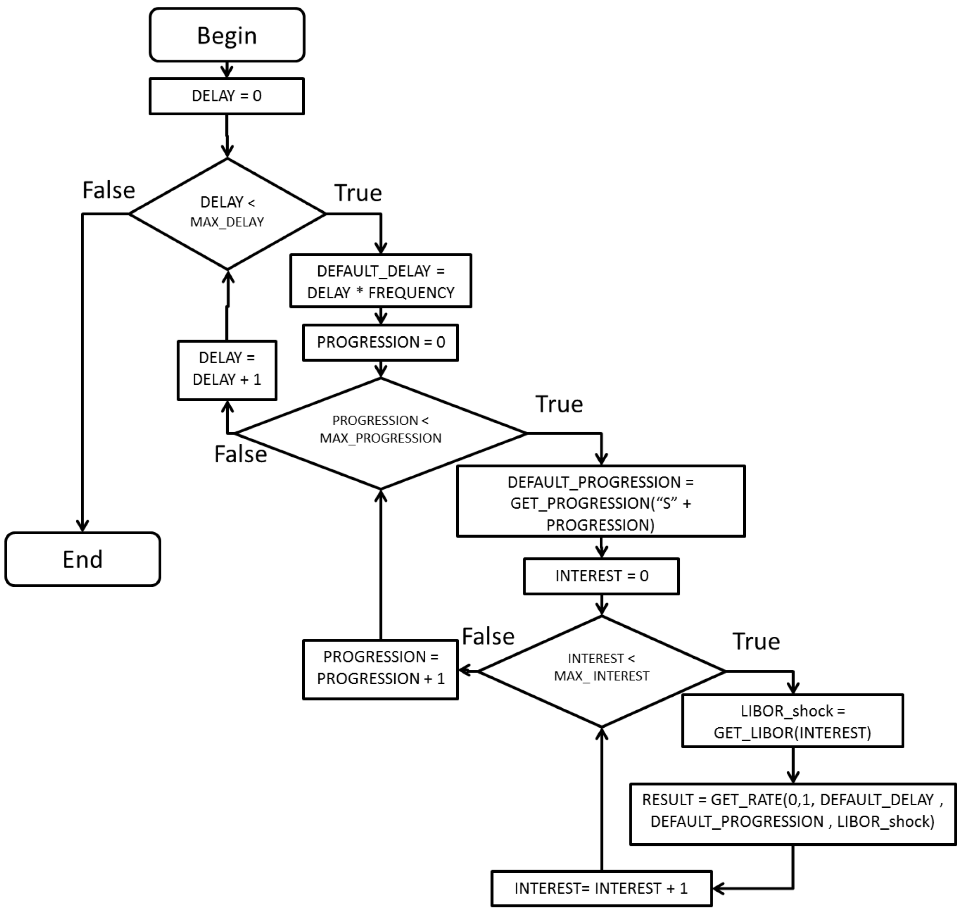
Figure 1

Data presented in Figure 1 illustrate the well-known paradigm of high performance computing: i.e., parallel computations start to pay off only once the degree of parallelism is significant and data is massive. In this case, parallel performance does not exceed the sequential sample until number of cores is more than 3, and significant improvement in performance is achieved only at the upper levels of data volume. In particular, our original data set of 44 cells shows degradation of performance when the number of cores exceeds 4. It shows that at this level of computation, the overhead associated with parallel computations overcomes the gains delivered by multithreaded execution.

At the same time, this article demonstrates that with reasonably small effort, an Excel-based model can be ported to the HPC environment and achieve a significant improvement in performance. It is important to note that the approach used in this article allows for porting without altering a single line of the code that performs the actual simulation. Consequently, the resulting package can run on HPC with very little data-result verification efforts.

3. Appendix A (Serial Code Sample and Work flow)

```
Sub SP_tables()

        <Initializing constants and other resources>

        Dim DELAY, PROGRESSION, INTEREST As Integer

        <Loop over  DELAY>
                <Loop over  PROGRESSION>
                        <Loop over  INTEREST>

                                Dim res As Double
                                < Calculating value of res using DELAY, PROGRESSION, INTEREST>

                                <Getting the cell reference for given DELAY, PROGRESSION, INTEREST>
                                <Writing the result value res to the cell>

                        <End Loop over  INTEREST>
                <End Loop over  PROGRESSION>
        <End Loop over  DELAY>

        <Finalization of the resources>

End Sub
```


Begin
DELAY = 0
DELAY < MAX_DELAY
False
True
DEFAULT_DELAY = DELAY * FREQUENCY
PROGRESSION = 0
DELAY = DELAY + 1
PROGRESSION < MAX_PROGRESSION
True
False
DEFAULT_PROGRESSION = GET_PROGRESSION("S" + PROGRESSION)
End
INTEREST = 0
False
True
PROGRESSION = PROGRESSION + 1
INTEREST < MAX_ INTEREST
LIBOR_shock = GET_LIBOR(INTEREST)
RESULT = GET_RATE(0,1, DEFAULT_DELAY , DEFAULT_PROGRESSION , LIBOR_shock)
INTEREST= INTEREST + 1

4. Appendix B (HPC Macro based code sample and work flow)

```
Public Function HPC_GetVersion()

        HPC_GetVersion = "1.0"

End Function
```

```
Public Function HPC_Initialize()

        <Initializing global constants and other resources which will be used in HPC_Partition (not applicable for
HPC_Execute)>

End Function
```

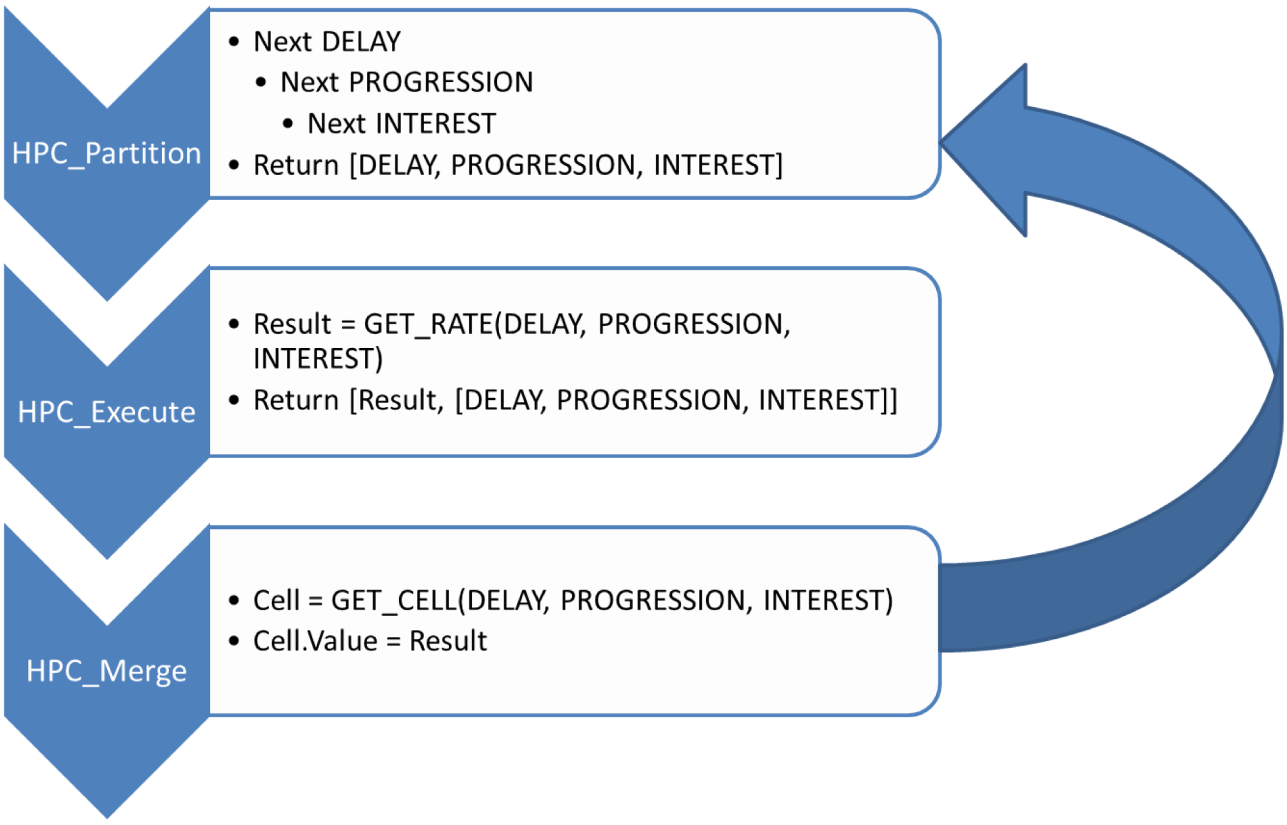
```
Public Function HPC_Partition() As Variant

        <For each parameter (DELAY, PROGRESSION, INTEREST) generate next value>

        Dim data(1 To 3) As Variant
        data(1) = INTEREST
        data(2) = PROGRESSION
        data(3) = DELAY

        HPC_Partition = data

End Function
```

```
Public Function HPC_Execute(data As Variant) As Variant

        Dim DELAY, PROGRESSION, INTEREST As Integer

        INTEREST = data(0)
        PROGRESSION = data(1)
        DELAY = data(2)

        Dim res As Double

        <Calculating value of res using DELAY, PROGRESSION, INTEREST>

        Dim out(1 To 4) As Variant

        out(1) = res
        out(2) = INTEREST
        out(3) = PROGRESSION
        out(4) = DELAY

        HPC_Execute = out

End Function
```

```
Public Function HPC_Merge(data As Variant)

        Dim res As Double
        Dim DELAY, PROGRESSION, INTERESTAs Integer

        res = data(0)
        INTEREST= data(1)
        PROGRESSION = data(2)
        DELAY = data(3)

        <Getting the cell reference for given DELAY, PROGRESSION, INTEREST>
        <Writing the result value res to the cell>

End Function
```

```
Public Function HPC_Finalize()

        <Finalization of the resources>

End Function
```

```
Public Function HPC_ExecutionError(errorMessage As String, errorContents As String)

        MsgBoxerrorMessage&vbCrLf&vbCrLf&errorContents

End Function
```


HPC_Partition
Next DELAY
Next PROGRESSION
Next INTEREST
Return [DELAY, PROGRESSION, INTEREST]
HPC_Execute
Result = GET_RATE(DELAY, PROGRESSION, INTEREST)
Return [Result, [DELAY, PROGRESSION, INTEREST]]
HPC_Merge
Cell = GET_CELL(DELAY, PROGRESSION, INTEREST)
Cell.Value = Result