HPC with Excel

Lee Chin Yau

(Microsoft Operations Pte Ltd)

HPC is commonly applied in the financial sector within the areas of Treasury, Risk Management, Credit Card Fraud Analysis, Data warehouse and others. Intensive computing work tends to take place in the financial sector as there is massive data to be processed for accuracy in trading, to minimise risks, and to check for any inconsistency in accounts. It also often used for marketing.

In these areas, users are often required to value their deals based on custom written codes (running on large SMPs or standalone Windows Desktop) or Microsoft Excel. Microsoft Excel is commonly used for its mathematically formulae capabilities and its ability to extend its functionality via a few methods:

- Internal VBA or sometimes called “Macros”
- Proprietary Plug-ins
- XLL or Excel linked libraries

There is little difference in the architecture between an in-house developed DLL or XLL against the proprietary ones. Excel is used as the front end in both cases, with the DLLs usually performing the different types of calculations functions, often called “Monte Carlo Simulation”

In a banking scenario, a trader will use an application or a spreadsheet to revalue his products, typically one spreadsheet per product (sometimes one spreadsheet can have multiple functions) to calculate sensitivities. WCCS is an important component that provides the infrastructure, sometimes called analytics backbone for such massive number crunching activities.

Typically, calculations include (1) interest rate derivate product (2) equities derivate & others.

![Diagram](image)

You might have realised by now that the calculations that spreadsheet perform are massive and often require quick turnaround and response. In such cases, we can leverage on high performance computing technologies or “Grid Computing”.

In the diagram below, you can see a sample architecture based on Microsoft Windows Compute Cluster which allows the workload above to scale out and off load its UDF or User Define Functions for speedy calculations.
Looking at the above, there are three major layers that we need to consider:

- **The client layer**, often the Microsoft Excel or in-house developed .NET based applications. Leveraging on CCPAPI.DLL, which is part of the Windows Compute Cluster, or leveraging on UDFs (described below), the user is provided with an interface to access the compute cluster and to schedule jobs or run them immediately.
- **The head node** performs the function of the centralised job scheduler for all time-based jobs.
- **Each compute node** is running Windows Compute Cluster Edition. The Compute node will perform different methods of workbook calculations (offload of UDFs, .NET or even a whole workbook). The compute nodes receive jobs from the head node, perform calculations based on a subset of the overall parameter list and then store the results within a share located on the head node for further processing, if needed.

So what can you off load? What is UDF?

UDF stands for "User-Defined Function" and is one of the commonly used methods to off load computation in a excel HPC farm or Grid. UDF also extends the calculation and data-import capability. It is often implemented as Excel Addin via COM, XLL or other methods and implements functions and calculations that Excel lacks.

In the sample below, we created a simple function called "AddXLL" which will add two numbers together.

- Create a XLL project with the sample code below.
- Test XLL
Open a new workbook in Excel. Select Tools, Add-Ins, Automation. ExcelAddIn.Functions should be listed - select it. OK.
- In a cell, type =AddXLL(1,1), Cell will display 2
- You can convert this XLL to a binary or EXE file. This can then be distributed to the cluster for computation

Sample code that can be freely found on Internet.

```csharp
using System;
using System.Runtime.InteropServices;
namespace ExcelAddIn
{
    [ClassInterface(ClassInterfaceType.AutoDual)]
    public class Functions
    {
        public Functions()
        {
        }
        public double AddXLL(double v1, double v2)
        {
            return v1 + v2;
        }
        [ComRegisterFunctionAttribute]
        public static void RegisterFunction(Type t)
        {
            Microsoft.Win32.Registry.ClassesRoot.CreateSubKey("CLSID\{" + t.GUID.ToString().ToUpper() + "\}" +\"Programmable");
        }
        [ComUnregisterFunctionAttribute]
        public static void UnregisterFunction(Type t)
        {
            Microsoft.Win32.Registry.ClassesRoot.DeleteSubKey("CLSID\{" + t.GUID.ToString().ToUpper() + "\}" +\"Programmable");
        }
    }
}
```