Improving Performance in Excel 2007

Office 2007

Summary: Learn about the increased worksheet capacity in Microsoft Office Excel 2007 and techniques that you can use as you design and create worksheets to improve calculation performance. (40 printed pages)

Charles Williams, Decision Models Limited

October 2006

Applies to: Microsoft Office Excel 2007, Microsoft Office Excel 2003, Microsoft Excel 2002, Microsoft Excel 2000

Contents

Overview

The "Big Grid" of 1 million rows and 16,000 columns in Microsoft Office Excel 2007, along with many other limit increases, substantially increases the size of worksheets that you can build compared to earlier versions of Excel. A single worksheet in Excel 2007 can contain over 1,000 times as many cells as earlier versions.

In earlier versions of Excel, many people created slow-calculating worksheets, and larger worksheets usually calculate more slowly than smaller ones. With the Excel 2007 "Big Grid," performance really matters: Slow calculation makes it more difficult for users to concentrate, and it increases errors.

Excel 2007 introduces a number of features to help you handle this capacity increase, such as the ability to use more than one processor at a time for calculations. This can substantia reduce worksheet calculation time. However, the most important factor that influences Excel calculation speed is still the way your worksheet is designed and built.

You can modify most slow-calculating worksheets to calculate tens, hundreds, or even thousands of times faster. The objective of this article is to review how you can speed up calculation by making the best use of the new features in Excel 2007, and by identifying, measuring, and then improving the calculation bottlenecks in your worksheets.

> **Note**
>
> This article is based on experience with the Beta 2 version of Microsoft Office Excel 2007 and therefore may not accurately reflect the content, user-interface, or performance of the final version of Excel 2007.

The "Big Grid" and Increased Limits in Excel 2007

The Excel 2007 "Big Grid" increases the maximum number of rows per worksheet from 65,536 to over 1 million, and the number of columns from 256 (IV) to 16,384 (XFD).

If you work with large workbooks, you probably have found that the increased memory capacity of recent versions of Excel has meant that you hit some of the other Excel specificatic limits more frequently. Excel 2007 includes many changes to these limits to accompany the large increase in row and column capacity.

- **Memory** Usable memory for formulas and pivot caches is increased to 2 gigabytes (GB) from 1 GB in Microsoft Office Excel 2003, 128 megabytes (MB) in Microsoft Excel 200 and 64 MB in Microsoft Excel 2000.
- **Smart recalculation limits** The dependency limits that enable smart recalculation rather than full calculation are now limited only by available memory rather than 8,000 cell dependent on a single area and 64,000 areas having dependencies.
- **Array formulas** Full column references are now allowed, and the limit on array formulas referring to another worksheet is increased from 65,000 to available memory.
- **PivotTables** Maximum rows displayed in a PivotTable report is 1 million. Maximum columns displayed in a PivotTable report is 16,000. Maximum number of unique items wit a single Pivot field is 1 million. Maximum number of fields visible in the Fields list is 16,000.
- **Sorting** Levels increased from 3 to 64.
- **AutoFilter** Drop-down list length changed from 1,000 items to 10,000 items.
- **Maximum formula length** Increased from 1,000 to 8,000.
- **Formula nesting levels** Increased from 7 to 64.

- **Arguments in a function** Increased from 30 to 255.
- **Conditional formats per cell** Increased from 3 to available memory.
- **Unique cell styles in a workbook** Increased from 4,000 to 64,000.
- **Unique colors per workbook** Increased from 56 to 4.3 billion.
- **Characters in a cell that can be displayed and printed** Increased to 32,000.

You can find more information about these and other improvements in Excel 2007 in David Gainer's blog Microsoft Excel 2007.

Of course, some of these limit increases can also significantly affect calculation speed.

Why Is Calculation Speed Important?

Poor calculation speed affects productivity and increases user errors. Studies have shown that a user's productivity and ability to focus on a task deteriorate as response time lengthe

Excel has two main calculation modes, which let you control when calculation occurs:

- **Automatic calculation** Formulas are automatically recalculated whenever you make a change.
- **Manual calculation** Formulas are only recalculated when you request it (for example, by pressing F9).

For calculation times of less than about a tenth of a second, users feel that the system is responding instantaneously. They can use automatic calculation even when entering data.

Between a tenth of a second and one second, users can successfully keep a train of thought going, although they will notice the response time delay.

As calculation time increases, users must switch to manual calculation when entering data.

Between 1 and 10 seconds, users will probably switch to manual calculation. User errors and annoyance levels start to increase, especially for repetitive tasks, and it becomes difficult retain a train of thought.

For calculation times greater than 10 seconds, users become impatient and usually switch to other tasks while waiting. This can cause problems when the calculation is one of a sequ of tasks and the user loses track.

Understanding Calculation Methods in Excel

To improve the calculation performance in Excel, you must understand both the available calculation methods and how to control them.

Full Calculation and Recalculation Dependencies

The smart recalculation engine in Excel tries to minimize calculation time by continuously tracking both the precedents and dependencies for each formula (the cells referenced by th formula) and any changes that were made since the last calculation. Then, at the next recalculation, Excel recalculates only the following:

- Cells, formulas, values, or names that have changed or are flagged as needing recalculation.
- Cells dependent on other cells, formulas, names, or values that need recalculation.
- Volatile functions and conditional formats.

Excel continues calculating cells that depend on previously calculated cells even if the value of the previously calculated cell does not change when it is calculated.

Because you only change part of the input data or a few formulas between calculations in most cases, this smart recalculation usually takes only a fraction of the time that a full calculation of all the formulas would take.

In manual calculation mode, you can trigger this smart recalculation by pressing F9. You can force a full calculation of all the formulas by pressing CTRL+ALT+F9, or you can force a complete rebuild of the dependencies and a full calculation by pressing SHIFT+CTRL+ALT+F9.

Calculation Process

Excel formulas that reference other cells can be put before or after the referenced cells (forward referencing or backward referencing). This is because Excel does not calculate cells in fixed order, or by row or column. Instead, Excel dynamically determines the calculation sequence based on a list of all the formulas to calculate (the calculation chain) and the dependency information of each formula

Excel has three distinct phases in the overall calculation process:

1. Build the initial calculation chain and determine where to begin calculating. This phase occurs when the workbook is loaded into memory.
2. Track dependencies, flag cells as uncalculated, and update the calculation chain. This phase executes at each cell entry or change, even in manual calculation mode. Ordinarily executes so fast that you do not notice it.
3. Calculate all formulas. As a part of the calculation process, Excel reorders and restructures the calculation chain to optimize future recalculations.

The third phase executes at each calculation or recalculation. Excel tries to calculate each formula in the calculation chain in turn, but if a formula depends on one or more formulas tl have not yet been calculated, the formula is sent down the chain to be calculated again later. This means that a formula can be calculated more than once per recalculation. In Excel 2000, a separate calculation chain is maintained for each worksheet, and the worksheets are calculated in alphabetical name sequence. In Excel 2002 and later versions, there is a sing global calculation chain, which gives faster calculation speed for most workbooks.

For a more detailed description of the calculation process in Excel, see Recalculation in Microsoft Excel 2002.

The second time you calculate a workbook is often significantly faster than the first time. This occurs for several reasons:

- Excel usually recalculates only cells that have changed, and their dependents.
- Excel stores and reuses the most recent calculation sequence so that it can save most of the time used to determine the calculation sequence.
- With multiple core computers, Excel 2007 tries to optimize the way the calculations are spread across the cores based on the results of the previous calculation.

- In an Excel session, both Microsoft Windows and Excel cache recently used data and programs for faster access.

Calculating Workbooks, Worksheets, and Ranges

You can control what is calculated by using the different Excel calculation methods.

Calculate All Open Workbooks

Each recalculation and full calculation calculates all the workbooks that are currently open, and resolves any dependencies within and between workbooks and worksheets.

Calculate Selected Worksheets

You can also recalculate only the currently selected worksheets by using SHIFT+F9. This resolves only the intra-sheet dependencies for the calculated sheets.

Calculate a Range of Cells

Excel also allows for the calculation of a range of cells using the Microsoft Visual Basic for Applications (VBA) method **Range.Calculate**. The behavior of **Range.Calculate** has change considerably in the different versions of Excel:

- **Excel 2000** **Range.Calculate** calculates left to right and top to bottom, ignoring all dependencies.
- **Excel 2002 and Excel 2003** **Range.Calculate** resolves the dependencies within the range being calculated.
- **Excel 2007** This version has both **Range Calculate** methods. **Range.Calculate** works in the same way as it did in Excel 2002 and Excel 2003, but **Range.CalculateRowMajorOrder** works the same way as it did in Excel 2000. Because **CalculateRowMajorOrder** does not resolve any dependencies within the range that is being calculated, it is usually significantly faster. However, it should be used with care because it may not give the same results as **Range.Calculate**. For more information, see Excel 2007 Performance Improvements section of this article.
- **Range.Calculate** is one of the most useful tools in Excel for performance optimization because you can use it to time and compare the calculation speed of different formulas

Volatile Functions

A volatile function is always recalculated at each recalculation even if it does not appear to have any changed precedents. Using many volatile functions slows down each recalculatio but it makes no difference to a full calculation. You can make a user-defined function volatile by including **Application.Volatile** in the function code.

Some of the built-in functions in Excel are obviously volatile: **RAND()**, **NOW()**, **TODAY()**. Others are less obviously volatile: **OFFSET()**, **CELL()**, **INDIRECT()**, **INFO()**.

Some functions that have previously been documented as volatile are not in fact volatile: **INDEX()**, **ROWS()**, **COLUMNS()**, **AREAS()**.

Volatile Actions

Volatile actions are actions that trigger a recalculation. These include the following:

- Clicking a row or column divider when you are in automatic mode.
- Inserting or deleting rows, columns, or cells anywhere on a sheet.
- Adding, changing, or deleting defined names.
- Renaming worksheets or changing worksheet position when you are in automatic mode.
- Filtering, hiding, or un-hiding rows in Excel 2003 or Excel 2007.
- Opening a workbook when you are in automatic mode. If the workbook was last calculated by a different version of Excel, opening the workbook usually results in a full calculation.
- Saving a workbook in manual mode if the **Calculate before Save** option is selected.

Formula and Name Evaluation Circumstances

A formula or part of a formula is immediately evaluated (calculated), even in manual calculation mode, when you do one of the following:

- Enter or change the formula.
- Enter or change the formula by using the **Function Wizard**.
- Enter the formula as an argument in the **Function Wizard**.
- Select the formula in the formula bar and press F9 (press ESC to undo and revert to the formula), or click **Evaluate Formula**.

A formula is flagged as uncalculated whenever it refers to (depends on) a cell or formula that has one of these conditions:

- It has been entered.
- It has been changed.
- It is in an AutoFilter list and the criteria drop-down list has been activated.
- It is flagged as uncalculated.

A formula that is flagged as uncalculated is evaluated whenever the range, worksheet, workbook, or Excel instance that contains it is calculated or recalculated.

The circumstances that cause a defined name to be evaluated are not the same as those for a formula in a cell.

- A defined name is evaluated every time a formula that refers to it is evaluated, so that using a name in multiple formulas can cause the name to be evaluated multiple times.
- Names that are not referred to by any formula are not calculated even by a full calculation.

Data Tables

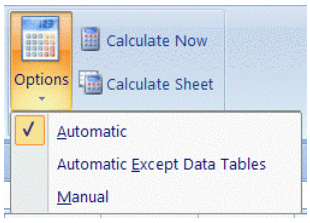
Excel data tables (on the **Data** tab, in the **Data Tools** group, click **What-If Analysis**, and then click **Data Table**) should not be confused with the table feature (on the **Home** tab, in th **Styles** group, click **Format as Table**, or, on the **Insert** tab, in the **Tables** group, click **Table**). Excel data tables do multiple recalculations of the workbook, each driven by the different values in the table. Excel first calculates the workbook normally. Then, for each pair of row and column values, it substitutes the values, recalculates, and stores the results in the data table.

Data tables give you a convenient way to calculate multiple variations and view and compare the results of the variations. You can use the **Automatic except Tables** calculation optic to stop Excel from automatically triggering the multiple calculations at each calculation, but still calculate all dependent formulas except tables.

Controlling Calculation Options

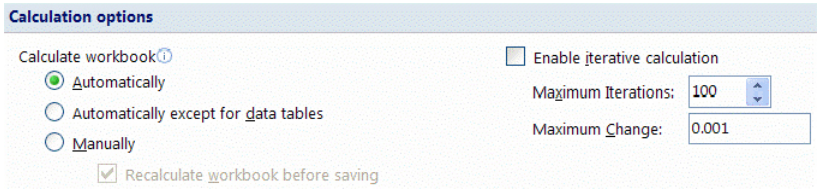
Excel has a range of options that enable you to control the way it calculates. You can change the most frequently used options in Excel 2007 by using the **Calculation** group on the **Ribbon Formulas** tab.

Figure 1. Calculation group on the Formulas tab



To see more Excel 2007 calculation options, click the **Microsoft Office Button**, click **Excel Options**, and then click the **Formulas** tab.

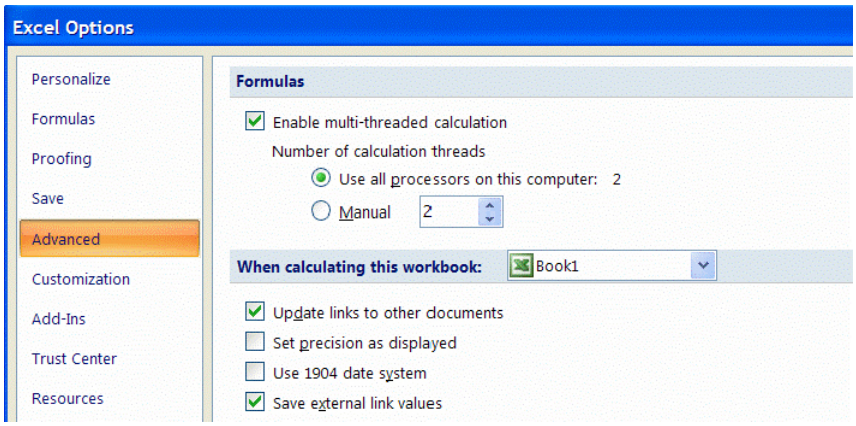
Figure 2. Calculation options on the Formula tab in Excel Options



Many calculation options (**Automatic**, **Automatic except tables**, **Manual**, **Calculate before save**) and the iteration settings (**Iteration on/off**, **Max iterations**, **Max change**) operate the application level rather than at the workbook level (they are the same for all open workbooks).

To find advanced calculation options, click the **Microsoft Office Button**, click **Excel Options**, and then click **Advanced**.

Figure 3. Advanced calculation options



In earlier versions of Excel, select **Options** on the **Tools** menu, and then click the **Calculation** tab, which contains all the calculation options.

When you start Excel, or when it is running with no workbooks open, the initial calculation mode and iteration settings are set from the first non-template, non-add-in workbook that you open. This means that the calculation settings in subsequently opened workbooks are ignored, although, of course, you can manually change the settings in Excel at any time. W you save a workbook, the current calculation settings are stored in the workbook.

Automatic Calculation

Automatic calculation mode means that Excel automatically recalculates all open workbooks at every change, and whenever you open a workbook. Usually when you open a workboo automatic mode and Excel recalculates, you do not see the recalculation because nothing has changed since the workbook was saved.

You might notice this calculation when you open a workbook in a later version of Excel than you used the last time the workbook was calculated (for example, Excel 2007 versus Exce 2003). Because the Excel calculation engines are different, Excel performs a full calculation when it opens a workbook that was saved using an earlier version of Excel.

Manual Calculation

Manual calculation mode means that Excel recalculates all open workbooks only when you request it by pressing F9 or CTRL+ALT+F9, or when you save a workbook. For workbooks take more than a fraction of a second to recalculate, you need to set calculation to manual mode to avoid an irritating delay whenever you make changes.

Excel tells you when a workbook in manual mode needs recalculation by displaying **Calculate** in the status bar. The status bar also displays **Calculate** if your workbook contains circu references and the iteration option is selected. For versions of Excel earlier than Excel 2007, **Calculate** appears if there are too many dependencies.

Iteration Settings

If you have intentional circular references in your workbook, the iteration settings enable you to control the maximum number of times the workbook is recalculated (iterations) and t convergence criteria (maximum change: when to stop). Usually you should clear the iteration box so that if you have accidental circular references, Excel will warn you and will not try solve them.

Making Workbooks Calculate Faster

This section shows you the steps and methods that you can use to make your workbooks calculate faster.

Processor Speed and Multiple Cores

For most versions of Excel, faster Intel or AMD processors will, of course, enable faster Excel calculation, and you can usually experience a modest performance improvement through buying a top-of-the-range processor rather than a low-price processor or a mid-price processor.

Excel 2007 offers new features to support multiprocessor systems. The multithreaded calculation engine in Excel 2007 enables Excel to make excellent use of multiprocessor systems, you can expect significant performance gains with most workbooks. For more information, see the Excel 2007 Performance Improvements section of this article.

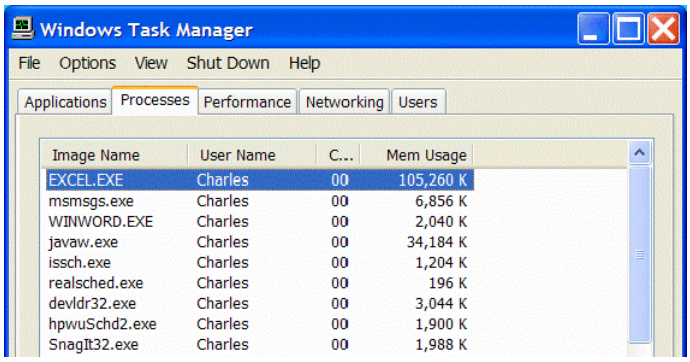
RAM

Paging to your virtual-memory swap file is very slow. You must have enough physical RAM for the operating system, Excel, and your workbooks. If you have more than very occasion hard disk activity during calculation, you need more RAM.

As already mentioned, recent versions of Excel can make effective use of large amounts of memory, and Excel 2007 can handle a single workbook or a combination of workbooks usi up to 2 GB of workbook memory.

A rough guideline for efficient calculation is to have enough RAM to hold the largest set of workbooks you need to have open at the same time, plus 256 MB or 512 MB for Excel anc operating system, plus additional RAM for any other running applications. You can check the amount of memory that Excel is using in Windows Task Manager.

Figure 4: Windows XP Task Manager showing Excel memory usage



Measuring Calculation Time

To make workbooks calculate faster, you must be able to accurately measure calculation time. You need a timer that is faster and more accurate than the VBA **Time** function. The **MICROTIMER()** function shown in the following example uses Windows API calls to the system high-resolution timer. It can measure time intervals down to small numbers of microseconds. Note that because Windows is a multitasking operating system, and because the second time you calculate something it may be faster than the first time, the times yc get usually do not repeat exactly. To achieve the best accuracy, you should time calculation tasks several times and average the results.

For information about how the Visual Basic Editor can significantly affect VBA user-defined function performance, see the Faster VBA User-Defined Functions section in this article.

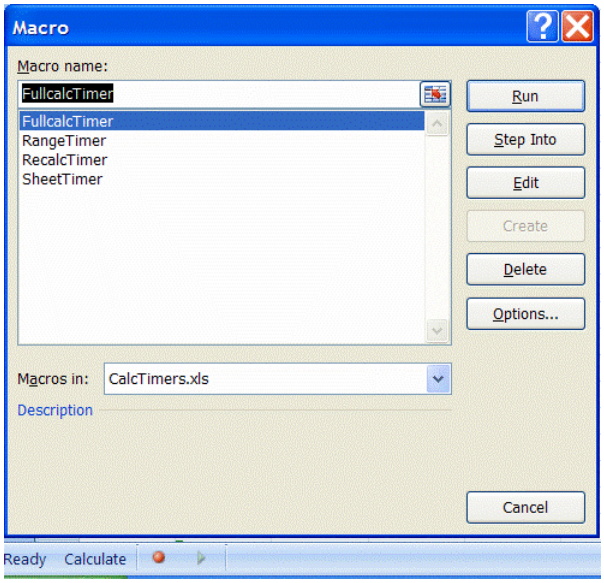
VB

```
Private Declare Function getFrequency Lib "kernel32" _
Alias "QueryPerformanceFrequency" (cyFrequency As Currency) As Long
Private Declare Function getTickCount Lib "kernel32" _
Alias "QueryPerformanceCounter" (cyTickCount As Currency) As Long
'
Function MicroTimer() As Double
'

' Returns seconds.
'
    Dim cyTicks1 As Currency
    Static cyFrequency As Currency
    '
    MicroTimer = 0

' Get frequency.
    If cyFrequency = 0 Then getFrequency cyFrequency

' Get ticks.
    getTickCount cyTicks1

' Seconds
    If cyFrequency Then MicroTimer = cyTicks1 / cyFrequency
End Function
```

To measure calculation time, you must call the appropriate calculation method. These subroutines give you calculation time for a range, recalculation time for a sheet or all open workbooks, or full calculation time for all open workbooks.

You should copy all these subroutines and functions into a standard VBA module. To open the VBA editor, press ALT+F11. On the **Insert** menu, select **Module**, and then copy the co into the module.

Figure 5. The Excel 2007 Macro window showing the calculation timers



When you want to run the subroutines in Excel 2007, press ALT+F8 or click **Play Macro**. Select the subroutine you want, and then click **Run**.

VB

```
Sub RangeTimer()
    DoCalcTimer 1
End Sub
Sub SheetTimer()
    DoCalcTimer 2
End Sub
Sub RecalcTimer()
    DoCalcTimer 3
End Sub
Sub FullcalcTimer()
    DoCalcTimer 4
End Sub

Sub DoCalcTimer(jMethod As Long)
    Dim dTime As Double
    Dim dOvhd As Double
    Dim oRng As Range
    Dim oCell As Range
    Dim oArrRange As Range
    Dim sCalcType As String
    Dim lCalcSave As Long
    Dim bIterSave As Boolean
    '
    On Error GoTo Errhandl

' Initialize
    dTime = MicroTimer

    ' Save calculation settings.
    lCalcSave = Application.Calculation
    bIterSave = Application.Iteration
    If Application.Calculation <> xlCalculationManual Then
        Application.Calculation = xlCalculationManual
    End If
    Select Case jMethod
    Case 1

        ' Switch off iteration.

        If Application.Iteration <> False Then
            Application.Iteration = False
        End if

        ' Max is used range.

        If Selection.Count > 1000 Then
            Set oRng = Intersect(Selection, Selection.Parent.UsedRange)
        Else
            Set oRng = Selection
        End If

        ' Include array cells outside selection.

        For Each oCell In oRng
            If oCell.HasArray Then
                If oArrRange Is Nothing Then
                    Set oArrRange = oCell.CurrentArray
                End If
                If Intersect(oCell, oArrRange) Is Nothing Then
                    Set oArrRange = oCell.CurrentArray
                    Set oRng = Union(oRng, oArrRange)
                End If
            End If
```

```
        Next oCell

        sCalcType = "Calculate " & CStr(oRng.Count) & _
            " Cell(s) in Selected Range: "
    Case 2
        sCalcType = "Recalculate Sheet " & ActiveSheet.Name & ": "
    Case 3
        sCalcType = "Recalculate open workbooks: "
    Case 4
        sCalcType = "Full Calculate open workbooks: "
    End Select

' Get start time.
    dTime = MicroTimer
    Select Case jMethod
    Case 1
        If Val(Application.Version) >= 12 Then
            oRng.CalculateRowMajorOrder
        Else
            oRng.Calculate
        End If
    Case 2
        ActiveSheet.Calculate
    Case 3
        Application.Calculate
    Case 4
        Application.CalculateFull
    End Select

' Calc duration.
    dTime = MicroTimer - dTime
    On Error GoTo 0

    dTime = Round(dTime, 5)
    MsgBox sCalcType & " " & CStr(dTime) & " Seconds", _
        vbOKOnly + vbInformation, "CalcTimer"

Finish:

    ' Restore calculation settings.
    If Application.Calculation <> lCalcSave Then
         Application.Calculation = lCalcSave
    End If
    If Application.Iteration <> bIterSave Then
         Application.Calculation = bIterSave
    End If
    Exit Sub
Errhandl:
    On Error GoTo 0
    MsgBox "Unable to Calculate " & sCalcType, _
        vbOKOnly + vbCritical, "CalcTimer"
    GoTo Finish
End Sub
```

Finding and Prioritizing Calculation Bottlenecks

Most slow-calculating workbooks have only a few problem areas or bottlenecks that consume most of the calculation time. If you do not already know where they are, use the drill-d approach outlined in this section to find them. If you do know where they are, you must measure the calculation time that is used by each bottleneck so that you can prioritize your v to remove them.

Drill-Down Approach to Finding Bottlenecks

The drill-down approach starts by timing the calculation of the workbook, then the calculation of each worksheet, then blocks of formulas on slow-calculating sheets. You should do each step in order and take notes of all the calculation times.

To find bottlenecks using the drill-down approach

1. Ensure that you have only one workbook open and no other tasks are running.
2. Set calculation to manual.
3. Make a backup copy of the workbook.
4. Open the workbook that contains the Calculation Timers macros, or add them to the workbook.
5. Check the used range by pressing CTRL+END on each worksheet in turn.

   This shows where the last used cell is. If this is beyond where you expect it to be, consider deleting the excess columns and rows and saving the workbook. For more informati see the Minimizing the Used Range section of this article.
6. Run the **FullCalcTimer** macro.

   The time to calculate all the formulas in the workbook is usually the worst-case time.
7. Run the **RecalcTimer** macro.

   A recalculation immediately after a full calculation usually gives you the best-case time.
8. Calculate workbook volatility as the ratio of recalculation time to full calculation time.

   This measures the extent to which volatile formulas and the evaluation of the calculation chain are bottlenecks.
9. Run the **SheetTimer** macro on each worksheet in turn.

Because you just recalculated the workbook, this gives you the recalculate time for each worksheet. This should enable you to determine which ones are the problem worksh

10. Run the **RangeTimer** macro on selected blocks of formulas.
    a. For each problem worksheet, divide the columns or rows into a small number of blocks.
    b. Select each block in turn, and then run the **RangeTimer** macro on the block.
    c. If necessary, drill down further by subdividing each block into a smaller number of blocks.
11. Prioritize the bottlenecks.

Speeding up Calculations and Reducing Bottlenecks

It is not the number of formulas or the size of a workbook that consumes the calculation time. It is the number of cell references and calculation operations, and the efficiency of the functions being used.

Because most worksheets are constructed by copying formulas that contain a mixture of absolute and relative references, they usually contain a large number of formulas that contai repeated or duplicated calculations and references.

Avoid complex mega-formulas and array formulas. In general, it is better to have more rows and columns and fewer complex calculations. This gives both the smart recalculation and multithreaded calculation in Excel 2007 a better chance to optimize the calculations. It is also easier to understand and debug. The following are a few golden rules to help you speec workbook calculations.

First Golden Rule: Remove Duplicated, Repeated, and Unnecessary Calculations

Look for duplicated, repeated, and unnecessary calculations, and figure out approximately how many cell references and calculations are required for Excel to calculate the result for bottleneck. Then think how you might obtain the same result with fewer references and calculations.

Usually this involves one or more of the following steps:

- Reduce the number of references in each formula.
- Move the repeated calculations to one or more helper cells, and then reference the helper cells from the original formulas.
- Use additional rows and columns to calculate and store intermediate results once, so that you can reuse them in other formulas.

Second Golden Rule: Use the Most Efficient Function Possible

When you find a bottleneck that involves a function or array formulas, determine if there is a more efficient way to achieve the same result. For example:

- Lookups on sorted data can be tens or hundreds of times more efficient than lookups on unsorted data.
- VBA user-defined functions are usually slower than the built-in functions in Excel (although carefully written VBA functions can be very fast).
- Minimize the number of used cells in functions like **SUM** and **SUMIF**. Calculation time is proportional to the number of used cells (unused cells are ignored).
- Consider replacing slow array formulas with user-defined functions.

Third Golden Rule: Make Good Use of Smart Recalculation

The better use you make of smart recalculation in Excel, the less processing has to be done each time Excel recalculates, so:

- Avoid volatile functions like **INDIRECT** and **OFFSET** where you can, unless they are significantly more efficient than the alternatives. (Well-designed use of **OFFSET** is often ver fast.)
- Minimize the size of the ranges that you are using in array formulas and functions.
- Break array formulas and mega-formulas out into separate helper columns and rows.

Fourth Golden Rule: Time and Test Each Change

Some of the changes that you make might surprise you, either by not giving the answer that you thought they would, or by calculating more slowly than you expected. Therefore, yo should time and test each change, as follows:

1. Time the formula that you want to change by using the **RangeTimer** macro.
2. Make the change.
3. Time the changed formula by using the **RangeTimer** macro.
4. Check that the changed formula still gives the correct answer.

Golden Rule Examples

The following sections provide examples of how to use the golden rules to speed up calculation.

Period-to-Date Sums

Suppose you need to calculate the period-to-date SUMs of a column that contains 2000 numbers. Assume that column A contains the numbers, and that column B and column C sh contain the period-to-date totals.

You could write the formula using **SUM**, which is a very efficient function.

```
B1=SUM($A$1:$A1)
B2=SUM($A$1:$A2)
```

Figure 6. Example of period-to-date SUM formulas

	A	B	C
1	1	=SUM($A$1:$A1)	=A1
2	2	=SUM($A$1:$A2)	=C1+A2
3	3	=SUM($A$1:$A3)	=C2+A3
4	4	=SUM($A$1:$A4)	=C3+A4
5	5	=SUM($A$1:$A5)	=C4+A5
6	6	=SUM($A$1:$A6)	=C5+A6
7	7	=SUM($A$1:$A7)	=C6+A7
8	8	=SUM($A$1:$A8)	=C7+A8

Then copy the formula down to B2000.

How many cell references are added up by **SUM** in total? B1 refers to one cell, and B2000 refers to 2000 cells. The average is 1000 references per cell, so the total number of referenc 2 million. Selecting the 2000 formulas and using the **RangeTimer** macro shows you that the 2000 formulas in column B calculate in 80 milliseconds. But most of these calculations ar duplicated many times: **SUM** adds A1 to A2 in each formula from B2:B2000.

You can eliminate this duplication if you write the formulas as follows:

```
C1=A1
C2=C1+A1
```

Then copy this formula down to C2000.

Now how many cell references are added up in total? Each formula, except the first one, uses two cell references, and therefore the total is 1999*2+1=3999. This is a factor of 500 few cell references.

RangeTimer indicates that the 2000 formulas in column C calculate in 3.7 milliseconds compared to the 80 milliseconds for column B. This change has a performance improvement factor of only 80/3.7=22 instead of 500 because there is a small overhead per formula.

Error Handling

Suppose you have a calculation-intensive formula where you want the result to be shown as zero if there is an error (this frequently occurs with exact match lookups). You can write t in several different ways:

- You can write it as a single formula, which is slow:

```
B1=IF(ISERROR(time expensive formula),0,time expensive formula)
```

- You can write it as two formulas, which is fast:

```
A1=time expensive formula
B1=IF(ISERROR(A1),0,A1)
```

- In Excel 2007, you can use the **IFERROR** function, which is fast and simple, and it is a single formula:

```
B1=IFERROR(time expensive formula,0)
```

Dynamic Count Unique

Figure 7. Example list of data for Count Unique

	A
1	**ctry-ctry**
2	AT-BE
3	AT-BE
35	AT-CH
36	AT-CH
37	AT-CH
38	AT-CZ
50	AT-DE
51	AT-DE

Suppose you have a list of 11,000 rows of data in column A, which frequently changes, and you need a formula that dynamically calculates the number of unique items in the list, ignoring blanks. Here are a few possible solutions.

Array Formulas

You can do it with the following array formula (use CTRL+SHIFT+ENTER):

```
{=SUM(IF(LEN(A2:A11000)>0,1/COUNTIF(A2:A11000,A2:A11000)))}
```

RangeTimer indicates that this takes 13.8 seconds.

SUMPRODUCT

SUMPRODUCT usually calculates faster than an equivalent array formula:

```
=SUMPRODUCT((A2:A11000<>"")/COUNTIF(A2:A11000,A2:A11000&""))
```

This formula takes 10.0 seconds. This gives an improvement factor of 13.8/10.0=1.38, which is better, but not good enough.

User-Defined Functions

The following example shows a VBA user-defined function that uses the fact that the index to a collection must be unique. For an explanation of some of the techniques used, see th section about user-defined functions in the Using Functions Efficiently section of this article.

VB

```
Public Function COUNTU(theRange As Range) As Variant
    Dim colUniques As New Collection
    Dim vArr As Variant
    Dim vCell As Variant
    Dim vLcell As Variant
    Dim oRng As Range

    Set oRng = Intersect(theRange, theRange.Parent.UsedRange)
    vArr = oRng
    On Error Resume Next
    For Each vCell In vArr
    If vCell <> vLcell Then
        If Len(CStr(vCell)) > 0 Then
             colUniques.Add vCell, CStr(vCell)
        End If
    End If
    vLcell = vCell
    Next vCell

    COUNTU = colUniques.Count
End Function
```

This formula, `=COUNTU(A2:A11000)`, takes only 0.061 seconds, giving an improvement factor of 13.8/0.061=226.

Adding a Column of Formulas

If you look at the previous sample of the data, you can see that it is sorted (Excel takes 0.5 seconds to sort the 11,000 rows). You can exploit this by adding a column of formulas that checks if the data in this row is the same as the data in the previous row. If it is different, the formula returns 1, otherwise it returns 0.

Add this formula to cell B2:

```
=IF(AND(A2<>"",A2<>A1),1,0)
```

Then copy the formula down. Then add a formula to add up column B:

```
=SUM(B2:B11000)
```

A full calculation of all these formulas takes 0.027 seconds, giving an improvement factor of 13.8/0.027=511.

Excel 2007 Performance Improvements

The following sections discuss some of the new features in Excel 2007 that you can use to improve calculation performance.

Multithreaded Calculation

Excel 2007 can now split calculation across multiple processors or cores. When Excel 2007 loads a workbook, it determines from the operating system how many processors are avail and then creates a separate calculation thread for each processor. These threads can then run in parallel. The beauty of this system is that it scales extremely well with the number of processors. Many new computer systems already contain two processors, and both Intel and AMD roadmaps show systems that have four or eight processors will become widely available in the not-too-distant future.

Most workbooks show a significant improvement in calculation speed on a system with multiple cores. The degree of improvement depends on how many independent calculation t the workbook contains. If you make a workbook that contains one continuous chain of formulas, it will not show any multithreaded calculation (MTC) performance gain, whereas a workbook that contains several independent chains of formulas will show gains close to the number of processors available.

A test on a range of workbooks with between 840K and 23K formulas using Excel 2007 on a Dual-Core system showed improvement factors from using MTC ranging from 1.9 to no improvement, with the larger workbooks tending to show the most improvement.

In collaboration with Intel Corporation, Microsoft conducted testing on a suite of user-created spreadsheets. Comparisons were done between Excel 2007 and Excel 2003. (A prerelea version of Excel 2007 was used, but little to no difference is expected with the final release version.)

Results showed calculation times ranging from no improvement to greater than theoretical (2x/4x) improvement on both the Dual-Core and Quad-Processor systems. Typical (media improvement for a system with an Intel Dual-Core Pentium 4 at 3.0 GHz with 1 GB of RAM compared to the same file calculating in Excel 2003 were 48 percent, or a 1.92x speedup. Typical (median) speedup for a system with an Intel Quad-Core Xeon at 3.0 GHz with 4 GB of RAM were 76 percent, or a 4.17x speedup. Similar speed improvements were observed other processors and platforms. Improvements beyond theoretical speedup (due to multithreading) are attributed to other performance enhancements in Excel 2007, such as enhancements to the speed of function execution.

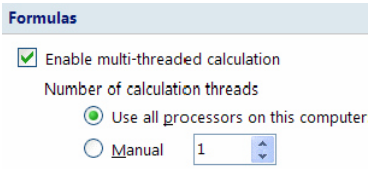
Some Excel features do not use multithreaded calculation, for example:

- Data table calculation (but structured references to tables do use MTC).
- User-defined functions (but XLL functions can be multithread-enabled).
- XLM functions.
- INDIRECT, CELL functions that use either the **format2** or **address** options.
- GETPIVOTDATA and other functions referring to PivotTables or cubes.
- **Range.Calculate** and **Range.CalculateRowMajorOrder**.
- Cells in circular reference loops.

The first time that Excel calculates a workbook on a computer that has multiple processors, you incur some overhead while Excel examines dependencies. Therefore, you can see the maximum performance increase on the second and subsequent calculations (although there is still usually improvement on the first calculation versus running the same task on the s speed of computer with a single processor).

You also incur this overhead the first time you calculate a workbook on a multiple-processor computer that has a larger number of processors than the computer on which you last saved the workbook. If you turn off MTC, or run Excel 2007 on a system that has a single processor, there is no performance gain or loss from the MTC feature. You can use MTC in E 2007 even in compatibility mode, and the information that is stored by the calculation can be reused even after the workbook is calculated and saved by using an earlier version of E

Figure 8. Controlling the number of calculation threads in Excel 2007



You can manually specify the number of threads to run simultaneously. This number can be more than the number of processors on the computer. This is useful if, for example, you h XLL user-defined functions dependent on long-running external calls to a database server. If the database server can process multiple requests in parallel, you can make very effective use of multithreading even on a single-processor system.

To control MTC options, click the **Microsoft Office Button**, select **Excel Options**, select **Advanced**, and then select **Formulas**.

Increased Memory Capacity and Limits

Earlier versions of Excel had several limits on the number of dependencies that were tracked for the smart recalculation feature. When you exceeded these limits, Excel always did a fu calculation, and the status bar displayed **Calculate**.

Excel 2007 removes these limits. Subject to the overall 2-GB Windows memory limit, large Excel workbooks can always use smart recalculation, which usually calculates significantly fa than a full calculation.

Workbook.ForceFullCalculation

You can set the new workbook property, **Workbook.ForceFullCalculation** (unavailable in Beta 2), programmatically by using the Excel object model. When this property is set to **Tru** dependencies are not loaded at open, and every calculation of the workbook is a full calculation.

If you have a workbook that has so many complex dependencies that loading the dependencies at workbook open takes a long time or recalculation takes longer than full calculatio you can use this property to force Excel to skip loading the dependencies and always use full calculation.

SUMIFS, COUNTIFS, and AVERAGEIFS

Excel 2007 has three new functions that you can use to **SUM**, **COUNT**, or **AVERAGE** using multiple criteria. In earlier versions of Excel, you had to use slow-calculating, hard-to-understand array formulas or **SUMPRODUCT** to use multiple criteria. The new functions are easy to use and fast to calculate.

```
SUMIFS(sum_range, criteria_range1, criteria1 [,criteria_range2, _
    criteria2…])
COUNTIFS(criteria_range1, criteria1 [,criteria_range2, criteria2…])
AVERAGEIFS(average_range, criteria_range1, criteria1 _
    [,criteria_range2, criteria2…])
```

These functions handle full column references ($A:$A) very efficiently by using special handling for the empty cells. The criteria that evaluates text cells can use the wildcard character (any set of characters) and **?** (any single character). Because these functions are so much faster to calculate than equivalent array formulas, you should use them to replace your array formulas wherever possible.

IFERROR

The following **IFERROR** function simplifies and speeds up error checking:

```
IFERROR(Formula, value_if_error)
```

In earlier versions of Excel, it was common to see formulas that trapped errors by duplicating the formulas:

```
=IF(ISERROR(VLOOKUP("Charles",$A$1:$C$10000,3,False),"NotFound", _ VLOOKUP("Charles",$A$1:$C$10000,3,False))
```

Using this formula, if no error occurs in the **VLOOKUP**, Excel executes it twice. In Excel 2007, you can easily avoid this duplication of calculation time by using **IFERROR**:

```
=IFERROR(VLOOKUP("Charles",$A$1:$C$10000,3,False),"NotFound")
```

And it is easier to understand and maintain!

Named Tables and Structured Referencing

Excel 2007 introduces named tables (called *Microsoft Office Excel tables* in Excel 2007 Beta 2) to define a block of formulas and data. You can more easily reference named tables and their columns in formulas by using structured references like **Sales[2004]** to refer to the 2004 column in the Sales table, rather than ordinary Excel references such as **C2:C50**.

A major advantage of using this technique is that references to the table automatically adjust as you add data to the rows and columns of the table. Using structured referencing is m efficient than using dynamic ranges because it does not involve volatile functions like **OFFSET** coupled with counting functions like **COUNTA**. Another advantage is that you can have more than one named table on a worksheet and use AutoFilter on each named table.

If you currently use array formulas, try to use structured references wherever possible to minimize the number of cells that are calculated in the array formula.

User-Defined Functions

All user-defined functions in Excel 2007, regardless of the language and add-in method used to develop them, can now support the increased limits of Excel 2007, including the num of function arguments and the "Big Grid." Fully supporting the "Big Grid" may require changes to your user-defined functions wherever your code assumes a maximum of 256 colum or 65,536 rows, plus changes to XLLs may be required to support the new Excel C API.

The only type of user-defined functions that are able to take advantage of multithreaded calculations are XLLs. By updating your XLL functions for multithreaded recalculation, you ca enable your XLL to be run simultaneously on different threads. All other user-defined functions (VBA, Automation add-ins, XLM functions, and XLLs not updated to work on multiple threads) are always run on the main thread, and so execute only one at a time regardless of how many processors or threads are used.

Excel 2007 has an updated C API to provide support for the following features:

- The "Big Grid"
- Multithreaded calculation
- More function arguments

If you want to take advantage of these new capabilities, you must update your XLL functions. For more information, see Developing Add-ins (XLLs) in Excel 2007. If you do not update your add-in functions, they will continue to work but they will not be able to take advantage of the new capabilities of Excel 2007.

Range Calculate

Excel 2007 has two **Range** calculation methods. There is no standard user interface for running these calculation methods: you must call them by using VBA or some other programm language. These methods are useful when you want to calculate only a small block of cells while leaving all other formulas unchanged.

Range.Calculate

Range.Calculate calculates the range one row at a time, left to right and top to bottom, and then resolves all dependencies within the range. This is the same method that Excel 200 and Excel 2003 use, except that it has been enhanced to handle iterative calculations in manual mode.

Range.CalculateRowMajorOrder

Range.CalculateRowMajorOrder calculates the range one row at a time, left to right and top to bottom, but it completely ignores all dependencies. This is the same method as Microsoft Excel 97 and Excel 2000 use. Because **CalculateRowMajorOrder** does not try to resolve dependencies, it is usually significantly faster than **Range.Calculate**.

If you can ensure that any dependencies within a block of formulas always refer backward to cells to the left or above, the **Range.CalculateRowMajorOrder** can be the fastest calculation method in Excel on a single processor system.

Range.CalculateRowMajorOrder is one of the most useful tools in Excel for performance optimization because you can use it to time and compare the calculation speed of differen formulas while ignoring dependency effects.

Excel Services

Excel Services is a new server technology that is included in Microsoft Office SharePoint Server 2007. You can use Excel Services to offload a time-consuming calculation from a deskt to a more powerful (and expensive) server. Using multithreaded calculation on an eight-core server could produce major performance gains. For more information about Excel Servic see Learn Excel Services.

Tips for Optimizing Bottlenecks

This section covers specific tips for optimizing many of the most frequently occurring bottlenecks.

Forward Referencing and Backward Referencing

To increase clarity and avoid errors, design your formulas so that they do not refer forward (to the right or below) to other formulas or cells. Forward referencing usually does not affe calculation performance, except in extreme cases for the first calculation of a workbook, where it might take longer to establish a sensible calculation sequence if there are many formulas that need to have their calculation deferred.

Links Between Workbooks

Avoid inter-workbook links wherever possible: they are slow, easily broken, and not always easy to find and fix.

Using fewer larger workbooks is usually (but not always) better than using many smaller workbooks. Some exceptions to this might be when you have a lot of front-end calculations are so rarely recalculated that it makes sense to put them in a separate workbook, or when you do not have enough RAM.

Try to use simple direct cell references that work on closed workbooks. By doing this, you can avoid recalculating *all* your linked workbooks whenever you recalculate *any* workbook. Also, you can see the values Excel has read from the closed workbook, which is frequently important for debugging and auditing the workbook.

If you cannot avoid using linked workbooks, try to have them all open rather than closed, and open the workbooks that are linked to before you open the workbooks that are linked from.

Links Between Worksheets

Using many worksheets can make your workbook easier to use, but generally it is slower to calculate references to other worksheets than references within worksheets.

In Excel 97 and Excel 2000, worksheets and workbooks are calculated in alphabetical name sequence with individual calculation chains. With these versions, it is very important to nar the worksheets in a sequence that matches the flow of calculations between worksheets.

Minimizing the Used Range

To save memory and reduce file size, Excel tries to store information about the area only on a worksheet that has been used. This is called the *used range*. Sometimes various editing formatting operations extend the used range significantly beyond the range that you would currently consider used. This can cause performance bottlenecks and file-size bottlenecks

You can check the visible used range on a worksheet by using CTRL+END. Where this is excessive, you should consider deleting all the rows and columns below and to the right of yo real last used cell and then saving the workbook. Be sure to make a backup copy first. If you have formulas with ranges that extend into or refer to the deleted area, these ranges will reduced in size or changed to **#N/A**.

Allowing for Extra Data

When you frequently add extra rows or columns of data to your worksheets, you need to find a way of having your Excel formulas automatically refer to the new data area, instead of trying to find and change your formulas every time.

You can do this by using a large range in your formulas that extends well beyond your current data boundaries. However, this can cause very inefficient calculation under some circumstances, and it is difficult to maintain because deleting rows and columns can shrink the range without you noticing.

Excel 2007 Structured Table References

In Excel 2007, you can use structured table references, which automatically expand and contract as the size of the referenced table increases or decreases. This solution has several advantages:

- There are fewer performance disadvantages than the alternatives of whole column referencing and dynamic ranges.
- It is easy to have multiple tables of data on a single worksheet.
- Formulas that are embedded in the table also expand and contract with the data.

Whole Column and Row References

An alternative approach is to use a whole column reference, for example **$A:$A**. This reference returns all the rows in Column A, so you can add as much data as you want, and the reference will always include it.

This solution has both advantages and disadvantages:

- Many Excel built-in functions (**SUM**, **SUMIF**) calculate whole column references efficiently because they automatically recognize the last used row in the column. However, arra calculation functions like **SUMPRODUCT** either cannot handle whole column references or calculate all the cells in the column.
- User-defined functions do not automatically recognize the last-used row in the column and therefore tend to calculate whole column references very inefficiently. However, it easy to program user-defined functions so that they recognize the last-used row.
- It is difficult to use whole column references when you have multiple tables of data on a single worksheet.
- Array formulas in versions before Excel 2007 cannot handle whole-column references. In Excel 2007, array formulas can handle whole-column references, but this forces calculation for all the cells in the column, including empty cells. This can be extremely slow to calculate, especially for 1 million rows.

Dynamic Ranges

By using the **OFFSET** and **COUNTA** functions in the definition of a named range, you can make the area that the named range refers to dynamically expand and contract. For exampl create a defined name as follows:

```
=OFFSET(Sheet1!$A$1,0,0,COUNTA(Sheet1!$A:$A),1)
```

When you use the dynamic range name in a formula, it automatically expands to include new entries.

There is a performance hit because **OFFSET** is a volatile function and therefore is always recalculated, and because the **COUNTA** function inside the **OFFSET** has to examine a large number of rows. You can minimize this performance hit by storing the **COUNTA** part of the formula in a separate cell, and then referring to the cell in the dynamic range:

```
Counts!z1=COUNTA(Sheet1!$A:$A)
DynamicRange=OFFSET(Sheet1!$A$1,0,0,Counts!$Z$1,1)
```

You can also use functions such as **INDIRECT** to construct dynamic ranges. Dynamic ranges have both advantages and disadvantages:

- Dynamic ranges work very well to limit the number of calculations done by array formulas.
- Using multiple dynamic ranges with a single column requires special-purpose counting functions.
- Using a large number of dynamic ranges can be a performance hit.

Lookups

Lookups are frequently significant calculation bottlenecks. Fortunately, there are many ways of improving lookup calculation time. If you use the exact match option, the calculation ti for the function is proportional to the number of cells scanned before a match is found. For lookups over large ranges, this time can be very significant.

Lookup time using the approximate match options of **VLOOKUP**, **HLOOKUP**, and **MATCH** on sorted data is fast and is not significantly increased by the length of the range you are looking up. (Characteristics are the same as binary search.)

Lookup Options

Ensure that you understand the matchtype and range-lookup options in **MATCH**, **VLOOKUP**, and **HLOOKUP**:

```
MATCH(lookup value, lookup array, matchtype)
```

- **Matchtype=1** returns the largest match less than or equal to the lookup value if the lookup array is sorted ascending (approximate match). This is the default option.
- **Matchtype=0** requests an exact match and assumes that the data is not sorted.
- **Matchtype=-1** returns the smallest match greater than or equal to the lookup value if the lookup array is sorted descending (approximate match).

```
VLOOKUP(lookup value, table array, col index num, range-lookup)
HLOOKUP(lookup value, table array, row index num, range-lookup)
```

- **Range-lookup=TRUE** returns the largest match less than or equal to the lookup value (approximate match). This is the default option. Table array must be sorted ascending.
- **Range-lookup=FALSE** requests an exact match and assumes the data is not sorted.

Avoid doing lookups on unsorted data wherever possible because it is slow. If your data is sorted but you want to do an exact match, see the Sorted Data with Missing Values sectior this article.

VLOOKUP vs. INDEX and MATCH or OFFSET

Try using the **INDEX** and **MATCH** functions instead of **VLOOKUP**. **VLOOKUP** is slightly faster (approximately 5 percent faster), simpler, and uses less memory than a combination of **MATCH** and **INDEX**, or **OFFSET**. However, the additional flexibility that **MATCH** and **INDEX** offer often enables you to significantly save time. For example, you can store the result o exact **MATCH** in a cell and reuse it in several **INDEX** statements.

The **INDEX** function is very fast and is a non-volatile function, which speeds up recalculation. The **OFFSET** function is also very fast. But it is a volatile function, and it sometimes significantly increases the time taken to process the calculation chain.

It is easy to convert **VLOOKUP** to **INDEX** and **MATCH**. The following two statements return the same answer:

```
VLOOKUP(A1, Data!$A$2:$F$1000,3,False)

INDEX(Data!$A$2:$F$1000,MATCH(A1,$A$1:$A$1000,0),3)
```

Speeding Up Lookups

Because exact match lookups are so slow, it is well worth looking for ways to speed them up. For example:

- Use one worksheet. It is faster to keep lookups and data on the same sheet.
- Whenever you can, **SORT** the data first (**SORT** is very fast), and use approximate match.
- When you must use an exact match lookup, restrict the range of cells to be scanned to a minimum. Use dynamic range names rather than referring to a very large number of r
  or columns. Sometimes you can pre-calculate a lower-range limit and upper-range limit for the lookup.

Sorted Data with Missing Values

Two approximate matches are significantly faster than one exact match for a lookup over more than a few rows. (The breakeven point is about 10 to 20 rows.)

If you can sort your data but still cannot use approximate match because you cannot be sure that the value you are looking up exists in the lookup range, you can use this formula:

```
IF(VLOOKUP(lookup_val ,lookup_array,1,True)=lookup_val, _
    VLOOKUP(lookup_val, lookup_array, colnum, True), "notexist")
```

The first part of the formula works by doing an approximate lookup on the lookup column itself:

```
VLOOKUP(lookup_val ,lookup_array,1,True)
```

If the answer from the lookup column is the same as the lookup value:

```
IF(VLOOKUP(lookup_val ,lookup_array,1,True)=lookup_val,
```

You have found an exact match, so you can do the approximate lookup again, but this time, return the answer from the column you want:

```
VLOOKUP(lookup_val, lookup_array, colnum, True)
```

If the answer from the lookup column did not match the lookup value, it is a missing value, and it returns "notexist".

Note that if you look up a value smaller than the smallest value in the list, you receive an error. You can handle this error by using **IFERROR**, or by adding a very small test value to th
list.

Unsorted Data with Missing Values

If you have to use exact match lookup on unsorted data, and you cannot be sure whether the lookup value exists, you often have to handle the #N/A that is returned if no match is
found. In Excel 2007, you can use the **IFERROR** function, which is both simple and fast:

```
IF IFERROR(VLOOKUP(lookupval, table, 2 FALSE),0)
```

In earlier versions, a simple but slow way is to use an **IF** function that contains two lookups:

```
IF(ISNA(VLOOKUP(lookupval,table,2,FALSE)),0,_
    VLOOKUP(lookupval,table,2,FALSE))
```

You can avoid the double exact lookup if you use exact **MATCH** once, store the result in a cell, and then test the result before doing an **INDEX**:

```
In A1 =MATCH(lookupvalue,lookuparray,0)
In B1 =IF(ISNA(A1),0,INDEX(tablearray,A1,colnum))
```

If you cannot use two cells, use **COUNTIF**. It is generally faster than an exact match lookup:

```
IF (COUNTIF(lookuparray,lookupvalue)=0, 0, _
    VLOOKUP(lookupval, table, 2 FALSE))
```

Exact Match Lookups on Multiple Columns

You can often reuse a stored exact **MATCH** many times. For example, if you are doing exact lookups on multiple result columns, you can save a lot of time by using one **MATCH** and
many **INDEX** statements rather than many **VLOOKUP** statements.

Add an extra column for the **MATCH** to store the result (stored_row), and for each result column use the following:

```
INDEX(Lookup_Range,stored_row,column_number)
```

Alternatively, you can use **VLOOKUP** in an array formula:

```
{VLOOKUP(lookupvalue,{4,2},FALSE)}
```

Looking Up a Set of Contiguous Rows or Columns

You can also return many cells from one lookup operation. If you want to look up several contiguous columns, you can use the **INDEX** function in an array formula to return multiple columns at once (use 0 as the column number). You can also use the **INDEX** function to return multiple rows at one time:

```
{INDEX($A$1:$J$1000,stored_row,0)}
```

This returns column A to column J from the stored row created by a previous **MATCH** statement.

Looking Up a Rectangular Block of Cells

You can use the **MATCH** and **OFFSET** functions to return a rectangular block of cells.

Two-Dimensional Lookup

You can efficiently do a two-dimensional table lookup using separate lookups on the rows and columns of a table by using an **INDEX** function with two embedded **MATCH** function one for the row and one for the column.

Multiple-Index Lookup

In large worksheets, you frequently need to look up using multiple indexes, such as looking up product volumes in a country. The simple way to do this is to concatenate the indexes perform the lookup by using concatenated lookup values. This is inefficient for two reasons:

- Concatenating strings is a calculation-intensive operation.
- The lookup will cover a large range.

It is often more efficient to calculate a subset range for the lookup: for example, by finding the first and last row for the country, and then looking up the product within that range.

Three-Dimensional Lookup

If you need to look up the table to use in addition to the row and the column, you can use the following techniques, focusing on how to make Excel look up or choose the table.

If each table you want to look up (the third dimension) is stored as a set of named structured tables, range names, or as a table of text strings that represent ranges, you might be ab use the **INDIRECT** or **CHOOSE** functions.

Using **CHOOSE** and range names can be a very efficient method. **CHOOSE** is not volatile, but it is best-suited to a relatively small number of tables:

```
INDEX(CHOOSE(TableLookup_Value,TableName1,TableName2,TableName3), _
MATCH(RowLookup_Value,$A$2:$A$1000),MATCH(colLookup_value,$B$1:$Z$1))
```

The previous example dynamically uses **TableLookup_Value** to choose which range name (TableName1, TableName2, ...) to use for the lookup table.

```
INDEX(INDIRECT("Sheet" & TableLookup_Value & "!$B$2:$Z$1000"), _ MATCH(RowLookup_Value,$A$2:$A$1000),MATCH(colLookup_value,$B$1:$Z$1))
```

This example uses the **INDIRECT** function and **TableLookup_Value** to dynamically create the sheet name to use for the lookup table. This method has the advantage of being simple and able to handle a large number of tables. But because **INDIRECT** is a volatile function, the lookup is calculated at every calculation even if no data has changed.

You could also use the **VLOOKUP** function to find the name of the sheet or the text string to use for the table, and then use the **INDIRECT** function to convert the resulting text into range:

```
INDEX(INDIRECT(VLOOKUP(TableLookup_Value,TableOfTAbles,1)),MATCH(RowLookup_Value,$A$2:$A$1000),MATCH(colLookup_value,$B$1:$Z$1))
```

Another technique is to aggregate all your tables into one giant table that has an additional column that identifies the individual tables. You can then use the techniques for multiple index lookup shown in the previous examples.

Wildcard Lookup

The **MATCH**, **VLOOKUP**, and **HLOOKUP** functions allow you to use the wildcard characters **?** (any single character) and ***** (no character or any number of characters) on alphabetical exact matches. Sometimes you can use this method to avoid multiple matches.

Array Formulas and SUMPRODUCT

Array formulas and the **SUMPRODUCT** function are very powerful, but you must handle them with care. A single array formula might require a very large number of calculations.

The key to optimizing the calculation speed of array formulas is to ensure that the number of cells and expressions that are evaluated in the array formula is as small as possible. Remember that an array formula is a bit like a volatile formula: If any one of the cells that it references has changed, is volatile, or has been recalculated, the array formula calculates the cells in the formula and evaluates all the virtual cells it needs to do the calculation.

To optimize the calculation speed of array formulas:

- Take expressions and range references out of the array formulas into separate helper columns and rows. This makes much better use of the smart recalculation process in Exce
- Do not reference complete rows, or more rows and columns than you need. Array formulas are forced to calculate all the cell references in the formula even if the cells are em or unused. With 1 million rows available in Excel 2007, an array formula that references a whole column is extremely slow to calculate.
- Use Excel 2007 Structured References where you can to keep the number of cells that are evaluated by the array formula to a minimum.
- If you do not have Excel 2007, use dynamic range names where possible. Although they are volatile, it is worthwhile because they minimize the size of the ranges.

- Be careful with array formulas that reference both a row and a column: this forces the calculation of a rectangular range.
- Use **SUMPRODUCT** if possible: it is slightly faster than the equivalent array formula.

Array Formulas SUM with Multiple Conditions

In Excel 2007, you should always use the **SUMIFS**, **COUNTIFS**, and **AVERAGEIFS** functions instead of array formulas wherever you can because they are much faster to calculate.

In versions before Excel 2007, array formulas are often used to calculate a sum with multiple conditions. This is relatively easy to do, especially if you use the **Conditional Sum Wizar** Excel, but it is often very slow. Usually there are much faster ways of getting the same result. If you have only a few multiple-condition SUMs, you may be able to use the **DSUM** funct which is much faster than the equivalent array formula.

If you must use array formulas, some good methods of speeding them up are as follows:

- Use Dynamic Range Names or Excel 2007 Structured Table References to minimize the number of cells.
- Split out the multiple conditions into a column of helper formulas that return **True** or **False** for each row, and then reference the helper column in a **SUMIF** or array formulas. might not appear to reduce the number of calculations for a single array formula; but in fact, most of the time, it enables the smart recalculation process to recalculate only th formulas in the helper column that need to be recalculated.
- Consider concatenating together all the conditions into a single condition, and then using **SUMIF**.
- If the data can be sorted, a good technique is to count groups of rows and limit the array formulas to looking at the subset groups.

Using SUMPRODUCT for Multiple-Condition Array Formulas

In Excel 2007, you should always use the **SUMIFS**, **COUNTIFS**, and **AVERAGEIFS** functions instead of **SUMPRODUCT** formulas wherever possible.

In earlier versions, there are a few advantages to using **SUMPRODUCT** instead of **SUM** array formulas:

- **SUMPRODUCT** does not have to be array-entered by using CTRL+SHIFT+ENTER.
- **SUMPRODUCT** is usually slightly faster (5 to 10 percent).

You can use **SUMPRODUCT** for multiple-condition array formulas as follows:

```
SUMPRODUCT(--(Condition1),--(Condition2),RangetoSum)
```

In this example, `Condition1` and `Condition2` are conditional expressions such as `$A$1:$A$10000<=$Z4`. Because conditional expressions return **True** or **False** instead of numbers, t must be coerced to numbers inside the **SUMPRODUCT** function. You can do this by using two minus signs (**--**), or by adding 0 (**+0**), or by multiplying by 1 (***1**). Using **--** is very slight faster than **+0** or ***1**.

Note that the size and shape of the ranges or arrays that are used in the conditional expressions and range to sum must be the same, and they cannot contain entire columns.

You can also directly multiply the terms inside **SUMPRODUCT** rather than separate them by commas:

```
SUMPRODUCT((Condition1)*(Condition2)*RangetoSum)
```

But this is usually slightly slower than using the comma syntax, and it gives an error if the range to sum contains a text value. However, it is slightly more flexible in that the range to s may have, for example, multiple columns when the conditions have only one column.

Using SUMPRODUCT to Multiply and Add Ranges and Arrays.

In cases like weighted average calculations, where you need to multiply a range of numbers by another range of numbers and sum the results, using the comma syntax for **SUMPRODUCT** can be 20 to 25 percent faster than an array-entered **SUM**:

```
{=SUM($D$2:$D$10301*$E$2:$E$10301)}
=SUMPRODUCT($D$2:$D$10301*$E$2:$E$10301)
=SUMPRODUCT($D$2:$D$10301,$E$2:$E$10301)
```

These three formulas all produce the same result, but the third formula, which uses the comma syntax for **SUMPRODUCT**, takes only about 77 percent of the time to calculate that th other two formulas need.

Array and Function Calculation Bottlenecks

The calculation engine in Excel is optimized to exploit array formulas and functions that reference ranges. However, some unusual arrangements of these formulas and functions can sometimes, but not always, cause significantly increased calculation time.

If you find a calculation bottleneck that involves array formulas and range functions, you should look for the following:

- Partially overlapping references.
- Array formulas and range functions that reference part of a block of cells that are calculated in another array formula or range function. This situation can frequently occur in t series analysis.
- One set of formulas referencing by row, and a second set of formulas referencing the first set by column.
- A large set of single-row array formulas covering a block of columns, with **SUM** functions at the foot of each column.

Using Functions Efficiently

Functions significantly extend the power of Excel, but the manner in which you use them can often affect calculation time.

Functions That Handle Ranges

For functions like **SUM**, **SUMIF**, and **SUMIFS** that handle ranges, the calculation time is proportional to the number of used cells you are summing or counting. Unused cells are not examined, so whole column references are relatively efficient, but it is better to ensure you do not include more used cells than you need. Use tables, or calculate subset ranges or dynamic ranges.

Volatile Functions

Volatile functions can slow recalculation because they increase the number of formulas that must be recalculated at each calculation.

You can often reduce the number of volatile functions by using **INDEX** instead of **OFFSET**, and **CHOOSE** instead of **INDIRECT**. But **OFFSET** is a fast function and can often be used in creative ways that give very fast calculation.

User-Defined Functions

User-defined functions that are programmed in C or C++ and that use the C API (XLL add-in functions) generally perform faster than user-defined functions that are developed using VBA or Automation (XLA or Automation add-ins). For more information, see Developing Add-ins (XLLs) in Excel 2007.

XLM functions can also be fast, because they use the same tightly coupled API as C XLL add-in functions. The performance of VBA user-defined functions is very sensitive to how you program and call them.

Faster VBA User-Defined Functions

It is usually faster to use the Excel formula calculations and worksheet functions than to use VBA user-defined functions. This is because there is a small overhead for each user-defined function call and significant overhead transferring information from Excel to the user-defined function. But well-designed and called user-defined functions can be much faster than complex array formulas.

Ensure that you have put all the references to worksheet cells in the user-defined function input parameters instead of in the body of the user-defined function, so that you can avoid adding **Application.Volatile** unnecessarily.

If you must have a large number of formulas that use user-defined functions, ensure that you are in manual calculation mode, and that the calculation is initiated from VBA. VBA user defined functions calculate much more slowly if the calculation is *not* called from VBA, for example, in automatic mode or when you press F9 in manual mode. This is particularly true when the Visual Basic Editor (ALT+F11) is open or has been opened in the current Excel session.

You can trap F9 and redirect it to a VBA calculation subroutine as follows. Add this subroutine to the Thisworkbook module.

VB

```
Private Sub Workbook_Open()
    Application.OnKey "{F9}", "Recalc"
End Sub
```

Add this subroutine to a standard module.

VB

```
Sub Recalc()
    Application.Calculate
    MsgBox "hello"
End Sub
```

User-defined functions in Automation add-ins (Excel 2002 and later versions) do not incur the Visual Basic Editor overhead because they do not use the integrated editor. Other performance characteristics of Visual Basic 6 user-defined functions in Automation add-ins are similar to VBA functions.

If your user-defined function processes each cell in a range, declare the input as a range, assign it to a variant that contains an array, and loop on that. If you want to handle whole column references efficiently, you must make a subset of the input range, dividing it at its intersection with the used range, as in this example:

VB

```
Public Function DemoUDF(theInputRange as Range)
    Dim vArr as Variant
    Dim vCell as Variant
    Dim oRange as Range
    Set oRange=Union(theInputRange, theRange.Parent.UsedRange)
    vArr=oRange
    For Each vCell in vArr
        If IsNumeric(vCell) then DemoUDF=DemoUDF+vCell
    Next vCell
End Function
```

If your user-defined function is using worksheet functions or Excel object model methods to process a range, it is generally more efficient to keep the range as an object variable that transfer all the data from Excel to the user-defined function.

VB

```
Function uLOOKUP(lookup_value As Variant, lookup_array As Range, _
                 col_num As Variant, sorted As Variant, _
                 NotFound As Variant)
    Dim vAnsa As Variant
    vAnsa = Application.VLookup(lookup_value, lookup_array, _
                                col_num, sorted)
    If Not IsError(vAnsa) Then
        uLOOKUP = vAnsa
    Else
        uLOOKUP = NotFound
    End If
End Function
```

If your user-defined function is called early in the calculation chain, it can be passed uncalculated arguments. Inside a user-defined function, you can detect uncalculated cells by using the following test for empty cells that contain a formula.

VB

```
If ISEMPTY(Cell.Value) AND Len(Cell.formula)>0 then
```

There is a time overhead for each call to a user-defined function and for each transfer of data from Excel to VBA. Sometimes one multi-cell array formula user-defined function can help you minimize these overheads by combining multiple function calls into a single function with a multi-cell input range that returns a range of answers.

Faster VBA Macros

This article is primarily focused on Excel calculation speed rather than VBA execution speed; but here are some basic tips for creating faster VBA macros:

- Turn off screen updating and calculation while you are executing your macro.
- Get data from a range into a variant containing a two-dimensional array rather than looping cell by cell.
- Reference Excel objects such as **Range** objects directly, without selecting or activating them.
- Return results by assigning an array directly to a **Range**.
- Turn on screen updating and calculation when your macro has finished.

Here is an example of a macro that sums the numbers in each column (of course it would be much faster in practice to use the Excel **SUM** function to do the same thing).

VB

```
Option Explicit
Option Base 1
Sub myMacro()
' Sum each column of input data and store the result.

    Dim vData As Variant
    Dim dOutput() As Double
    Dim j As Long
    Dim k As Long

    Application.ScreenUpdating = False
    Application.Calculation = xlCalculationManual

    ' Assign the range to a variant containing an array.
    vData = Worksheets("InputData").Range("B2:K10000")

    ' Set up the result array.
    ReDim dOutput(1, UBound(vData, 2))

    ' Sum the numbers.
    For k = 1 To UBound(vData, 2)
        For j = 1 To UBound(vData, 1)
            If IsNumeric(vData(j, k)) Then
                dOutput(1, k) = dOutput(1, k) + vData(j, k)
            End If
        Next j
    Next k

    ' Return results by assigning the result array to the range.
    Worksheets("Results").Range("B2"). _
    Resize(1, UBound(dOutput, 2)) = dOutput

    Application.Calculation = xlCalculationManual
    Application.ScreenUpdating = True
End Sub
```

SUM and SUMIF

The Excel **SUM** and **SUMIF** functions are frequently used over a large number of cells. Calculation time for these functions is proportionate to the number of cells covered, so try to minimize the range of cells that the functions are referencing.

Wildcard SUMIF and COUNTIF

You can use the wildcard characters **?** (any single character) and ***** (no character or any number of characters) as part of the **SUMIF** and **COUNTIF** criteria on alphabetical ranges.

Period-to-Date and Cumulative SUMs

There are two methods of doing period-to-date or cumulative SUMs. Suppose the numbers that you want to cumulatively **SUM** are in column A, and you want column B to contain t cumulative sum; you can do either of the following:

- You can create a formula in column B such as `=SUM($A$1:$A2)` and drag it down as far as you need. The beginning cell of the SUM is anchored in A1, but because the finishir cell has a relative row reference, it automatically increases for each row.
- You can create a formula such as `=$A1` in cell B1 and `=$B1+$A2` in B2 and drag it down as far as you need. This calculates the cumulative cell by adding this row's number to th previous cumulative **SUM**.

For 1,000 rows, the first method makes Excel do about 500,000 calculations, but the second method makes Excel do only about 2,000 calculations.

Subset Summing

When you have multiple sorted indexes to a table (for example, Site within Area) you can often save significant calculation time by dynamically calculating the address of a subset rar of rows (or columns) to use in the **SUM** or **SUMIF** function:

1. Count the number of rows for each subset block.
2. Add the counts cumulatively for each block to determine its start row.
3. Use **OFFSET** with the start row and count to return a subset range to the **SUM** or **SUMIF** that covers only the subset block of rows.

Subtotals

Use the **SUBTOTAL** function to **SUM** filtered lists. The **SUBTOTAL** function is useful because, unlike **SUM**, it ignores the following:

- Hidden rows that result from filtering a list. In Excel 2003 and Excel 2007, you can also make **SUBTOTAL** ignore all hidden rows, not just filtered rows.
- Other **SUBTOTAL** functions.

DFunctions

The DFunctions **DSUM**, **DCOUNT**, **DAVERAGE**, and so on are significantly faster than equivalent array formulas. The disadvantage of the DFunctions is that the criteria must be in a separate range, which makes them impractical to use and maintain in many circumstances. In Excel 2007, you should use **SUMIFS**, **COUNTIFS**, and **AVERAGEIFS** functions instead of DFunctions.

PivotTables

PivotTables provide a very efficient way to summarize large amounts of data.

Totals as Final Results

If you need to produce totals and subtotals as part of the final results of your workbook, try using PivotTables.

Totals as Intermediate Results

PivotTables are a great way to produce summary reports, but try to avoid creating formulas that use PivotTable results as intermediate totals and subtotals in your calculation chain unless you can ensure the following conditions:

- The PivotTable has been refreshed correctly during the calculation.
- The PivotTable has not been changed so that the information is still visible.

If you still want to use PivotTables as intermediate results, use the **GETPIVOTDATA** function.

Circular References with Iteration

Calculating circular references with iterations is slow because multiple calculations are needed. Frequently you can "unroll" the circular references so that iterative calculation is no lor needed. For example, in cash flow and interest calculations, try to calculate the cash flow before interest, then calculate the interest, and then calculate the cash flow including the interest.

Excel calculates circular references sheet by sheet without considering dependencies. Therefore, you usually get very slow calculation if your circular references span more than one worksheet. Try to move the circular calculations onto a single worksheet or optimize the worksheet calculation sequence to avoid unnecessary calculations.

Before the iterative calculations start, Excel must recalculate the workbook to identify all the circular references and their dependents. This process is equivalent to two or three iterati of the calculation.

After the circular references and their dependents are identified, each iteration requires Excel to calculate not only all the cells in the circular reference, but also any cells that depend the cells in the circular reference chain, together with volatile cells and their dependents. So if you have a complex calculation that depends on cells in the circular reference, it can be faster to isolate this into a separate closed workbook and open it for recalculation after the circular calculation has converged.

And of course, it is important to minimize both the number of cells in the circular calculation and the calculation time that is taken by these cells.

Conditional Formats and Data Validation

Conditional formats and data validation are great, but using a lot of them can significantly slow down calculation. Every conditional format formula is evaluated at each calculation ar also whenever the display of the cell that contains the conditional format is refreshed. The Excel object model has a **Worksheet.EnableFormatConditionsCalculation** property so th you can enable or disable the calculation of conditional formats.

Defined Names

Defined names are one of the most powerful features in Excel, but they do take additional calculation time. Using names that refer to other worksheets adds an additional level of complexity to the calculation process. Also, you should try to avoid nested names (names that refer to other names).

Because names are calculated every time a formula that refers to them is calculated, you should avoid putting calculation-intensive formulas or functions in defined names. In these cases, it can be significantly faster to put your calculation-intensive formula or function in a spare cell somewhere and refer to that cell instead, either directly or by using a name.

Formulas That Are Used Only Occasionally

Many workbooks contain a significant number of formulas and lookups that are concerned with getting the input data into the appropriate shape for the calculations, or are being us as defensive measures against changes in the size or shape of the data. When you have blocks of formulas that are used only occasionally, you can copy and paste special values to temporarily eliminate the formulas, or you can put them in a separate, rarely opened workbook. Because worksheet errors are often caused by not noticing that formulas have been converted to values, the separate workbook method may be preferable.

Workbook Opening, Closing, Saving, and Size

You may find that opening, closing, and saving workbooks is much slower than calculating them. Sometimes this is just because you have a large workbook, but there can also be otl reasons.

Excel 2007 File Formats Performance and Size

Excel 2007 contains a wide variety of file formats compared to earlier versions. Ignoring the Macro, Template, Add-in, PDF, and XPS file format variations, there are three main format XLS, XLSB, and XLSX.

XLS Format

The XLS format is the same format as earlier versions. When you use this format, you are restricted to 256 columns and 65,536 rows. When you save an Excel 2007 workbook in XLS format, Excel runs a compatibility check. File size is almost the same as earlier versions (some additional information may be stored), and performance is slightly slower than earlier versions.

XLSB Format

XLSB is the Excel 2007 binary format. It is structured as a compressed folder that contains a large number of binary files. It is much more compact than the XLS format, but the amour compression very much depends on the contents of the workbook. For example, ten workbooks show a size reduction factor ranging from two to eight with an average reduction fac of four. In Excel 2007, opening and saving performance is only slightly slower than the XLS format.

XLSX Format

XLSX is the Excel 2007 XML format, which is a compressed folder that contains a large number of XML files (if you change the file name extension to .zip, you can open the compress folder and examine its contents). Typically, the XLSX format creates larger files than the XLSB format (1.5 times larger on average), but they are still significantly smaller than the XLS f You should expect opening and saving times to be slightly longer than for XLSB files.

Slow Open and Close

If one or more of your workbooks open and close more slowly than is reasonable, it might be caused by one of the following issues.

Temporary Files

Temporary files can accumulate in your \Windows\Temp directory (in Microsoft Windows 95, Microsoft Windows 98, and Microsoft Windows ME), or your \Documents and Settings\ Name\Local Settings\Temp directory (in Microsoft Windows 2000 and Microsoft Windows XP). Excel creates these files for the workbook, and in particular, for controls that are used l open workbooks. Software installation programs also create temporary files. If Excel stops responding for any reason, you might need to delete these files.

Too many temporary files can cause problems, so you should occasionally clean them out. However, if you have installed software that requires that you restart your computer and y have not yet done so, you should restart before deleting the temporary files.

An easy way to open your temp directory is from the Windows **Start** menu: Click **Start**, and then click **Run**. In the text box, type **%temp%**, and then click **OK**.

Tracking Changes in a Shared Workbook

Tracking changes in a shared workbook causes your workbook file-size to increase rapidly.

Fragmented Swap File

Make sure that your Windows swap file is located on a disk that has a lot of space and that you defragment the disk periodically.

Workbook with Password-Protected Structure

A workbook that has its structure protected with a password (on the **Tools** menu, point to **Protection**, and then click **Protect Workbook** and enter the optional password) opens an closes much slower than one that is protected without the optional password.

Used Range Problems

Oversized used ranges can cause slow opening and increased file size, especially if they are caused by hidden rows or columns that have non-standard height or width. For more information about used range problems, see Minimizing the Used Range in the Tips for Optimizing Bottlenecks section of this article.

Large Number of Controls on Worksheets

A large number of controls (check boxes, hyperlinks, and so on) on worksheets can slow down opening a workbook because of the number of temporary files that are used. This mig also cause problems opening or saving a workbook on a WAN (or even a LAN). If you have this problem, you should consider redesigning your workbook.

Large Number of Links to Other Workbooks

If possible, open the workbooks that you are linking to before you open the workbook that contains the links. Often it is faster to open a workbook than to read the links from a clos workbook.

Virus Scanner Settings

Some virus scanner settings can cause problems or slowness with opening, closing, or saving, especially on a server. If you think that this might be the problem, try temporarily switc the virus scanner off.

Slow Calculation Causing Slow Open and Save

Under some circumstances, Excel recalculates your workbook when it opens or saves it. If the calculation time for your workbook is long and is causing a problem, ensure that you ha calculation set to **manual**, and consider turning off the **calculate before save** option (on the **Tools** menu select **Options**, and then select **Calculation**).

Toolbar Files (.xlb)

Check the size of your toolbar file. A typical toolbar file is between 10 KB and 20 KB. You can find your XLB files by searching for *.xlb using Windows search. Each user has a unique X file. Adding, changing, or customizing toolbars increases the size of your toolbar.xlb file. Deleting the file removes all your toolbar customizations (renaming it "toolbar.OLD" is safer). new XLB file is created the next time you open Excel.

Conclusion

Excel 2007 enables you to effectively manage much larger worksheets, and it provides significant improvements in calculation speed. When you create large worksheets, it is easy to build them in a way that causes them to calculate slowly. Slow-calculating worksheets increase errors because users find it difficult to maintain concentration while calculation is occurring.

By using a straightforward set of techniques, you can speed up most slow-calculating worksheets by a factor of 10 or 100. You can also apply these techniques as you design and cre worksheets to ensure that they calculate quickly.

About the Author

Charles Williams founded Decision Models in 1996 to provide advanced consultancy, decision support solutions, and tools based on Microsoft Excel and relational databases. Charles the author of FastExcel, the widely used Excel performance profiler and performance toolset, and co-author of Name Manager, the popular utility for managing defined names. For m information about Excel calculation performance and methods, memory usage, and VBA user-defined functions, visit the Decision Models Web site.

This technical article was produced in partnership with A23 Consulting.

Additional Resources

To learn more about Excel 2007, see the following resources:

- Excel Developer Portal
- Blog: Microsoft Excel 2007