The first result shows that the MultiThread keyword slowed the assignment down. This is because the assignment involved a small number of points and MultiThread has some overhead.

The remaining results illustrate that MultiThread can provide increased speed for assignments involving large waves.

In the last result, the speed improvement factor was greater than the number of processors. This is explained by the fact that, once running, a threadsafe expression has slightly less overhead than a normal expression.

If the right-hand expression involves calling user-defined functions, those functions must be threadsafe (see **ThreadSafe Functions** on page IV-106) and must also follow these rules:

1. Do not do anything to waves that are passed as parameters that might disturb memory. For example, do not change the number of points in the wave or change its data type or kill it or write to a text wave.
2. Do not write to a variable that is passed by reference.
3. Note that any waves or global variables created by the function will disappear when then wave assignment is finished.
4. Each thread has its own private data folder tree. You can not use WAVE, NVAR or SVAR to access objects in the main thread.

Failure to heed rule #1 will likely result in a crash.

Although it is legal to use the MultiThread mechanism in a threadsafe function that is already running in a preemptive thread via **ThreadStart**, it is not recommended and will likely result in a substantial loss of speed.

For an example using MultiThread, open the Mandelbrot demo experiment file by choosing "File→Example Experiments→Programming→MultiThreadMandelbrot".

Data Folder Reference MultiThread Example

Advanced programmers can use waves containing data folder references and wave references along with MultiThread to perform multithreaded calculations more involved than evaluating an arithmetic expression. Here we use **Free Data Folders** (see page IV-96) to facilitate multithreading.

In this example, we extract each of the planes of a 3D wave, perform a filtering operation on the planes, and then finally assemble the planes into an output 3D wave. The main function, Test, executes a multithreaded assignment statement where the expression includes a call to a subroutine named Worker.

Because MultiThread is used, multiple instances of Worker execute simultaneously on different cores. Each instance runs in its own thread, working on a different plane. Each instance returns one filtered plane in a wave named M_ImagePlane in a thread-specific free data folder. The use of free data folders allows each instance of Worker to work on its own M_ImagePlane wave without creating a name conflict.

When the multithreaded assignment is finished, the main function assembles an output 3D wave.

```
// Extracts a plane from the 3D input wave, filters it, and returns the
// filtered output as M_ImagePlane in a new free data folder
ThreadSafe Function/DF Worker(w3DIn, plane)
   WAVE w3DIn
   Variable plane

   DFREF dfSav= GetDataFolderDFR()

   // Create a free data folder to hold the extracted and filtered plane
   DFREF dfFree= NewFreeDataFolder()
   SetDataFolder dfFree

   // Extract the plane from the input wave into M_ImagePlane.
   // M_ImagePlane is created in the current data folder
```

```
   // which is a free data folder.
   ImageTransform/P=(plane) getPlane, w3DIn
   Wave M_ImagePlane             // Created by ImageTransform getPlane

   // Filter the plane
   WAVE wOut= M_ImagePlane
   MatrixFilter/N=21 gauss,wOut

   SetDataFolder dfSav

   // Return a reference to the free data folder containing M_ImagePlane
   return dfFree
End

Function Demo(numPlanes)
   Variable numPlanes

   // Create a 3D wave and fill it with data
   Make/O/N=(200,200,numPlanes) src3D= (p==(2*r))*(q==(2*r))

   // Create a wave to hold data folder references returned by Worker.
   // /DF specifies the data type of the wave as "data folder reference".
   Make/DF/N=(numPlanes) dfw

   Variable timerRefNum = StartMSTimer

   MultiThread dfw= Worker(src3D,p)

   Variable elapsedTime = StopMSTimer(timerRefNum) / 1E6

   Printf "Statement took %g seconds for %d planes\r", elapsedTime, numPlanes

   // At this point, dfw holds data folder references to 50 free
   // data folders created by Worker. Each free data folder holds the
   // extracted and filtered data for one plane of the source 3D wave.

   // Create an output wave named out3D by cloning the first filtered plane
   DFREF df= dfw[0]
   Duplicate/O df:M_ImagePlane, out3D

   // Concatenate the remaining filtered planes onto out3D
   Variable i
   for(i=1; i<numPlanes; i+=1)
      df= dfw[i]     // Get a reference to the next free data folder
      Concatenate {df:M_ImagePlane}, out3D
   endfor

   // Redimension out3D to contain the correct number of planes
   Redimension/N=(-1, -1, numPlanes) out3D

   // Assign values to the remaining planes of out3D
   Variable i
   for(i=1; i<numPlanes; i+=1)
      df= dfw[i]                          // Reference to next free data folder
      WAVE thisPlaneWave = df:M_ImagePlane// Reference to wave in data folder
      out3D[][][i] = thisPlaneWave[p][q]
   endfor

   // dfw holds references to the free data folders. By killing dfw,
   // we kill the last reference to the free data folders which causes
   // them to be automatically deleted. Because there are no remaining
```