```
   Variable timerRefNum = StartMSTimer
   Variable elapsedTime

   Variable threadGroupID = ThreadGroupCreate(numThreads)

   // Start all threads
   int i
   for(i=0; i<numThreads; i+=1)
      ThreadStart threadGroupID, i, ThreadWorkerFunc(i, numWaves, numPoints)
   endfor

   // Wait for threads to finish or for abort to occur
   Variable threadGroupResult
   try
      do
         Variable threadGroupStatus = ThreadGroupWait(threadGroupID,100)
         DoUpdate // Needed for Printf output from worker to appear in history
      while(threadGroupStatus != 0)
   catch
      // If a user or programmed abort occurs, this executes and stops threads
      elapsedTime = StopMSTimer(timerRefNum) / 1E6       // In seconds
      Printf "Aborted after %g seconds\r", elapsedTime
      DoUpdate // Needed for Printf output from worker to appear in history

      // Signal running threads to stop
      threadGroupResult = ThreadGroupRelease(threadGroupID)
      return -1
   endtry

   // Here if threads ran to normal completion
   threadGroupResult = ThreadGroupRelease(threadGroupID)
   elapsedTime = StopMSTimer(timerRefNum) / 1E6    // In seconds
   Printf "ThreadCoordinatingFunction completed in %g seconds\r", elapsedTime
   return threadGroupResult
End
```

To try this example, execute this from the command line:

```
ThreadCoordinatingFunction(5, 1E3, 1E5)   // Completes in about 25 seconds
```

After about 25 seconds, it completes normally. If you click the Abort button in Igor's status bar, the code in the catch block runs. The ThreadGroupRelease call in the catch block signals running threads to quit. After a short period of time, all threads have quit and the coordinating function returns.

If Igor signals a thread to quit but it fails to do so, Igor forces the thread to quit.

Aborting Threads Gracefully

When an abort occurs, whether initiated by the user (see **User Abort Key Combinations** on page IV-51) or programmatically using the **AbortOnValue**, **AbortOnRTE** or **Abort** operations, Igor sets an internal per-thread abort flag. The abort flag causes Igor to short-circuit loops and subroutine calls so that the function to be aborted finishes quickly.

If a thread worker function needs to run cleanup code in the event of an abort then the it must include a try-catch-endtry block (see **try-catch-endtry Flow Control**). This section explains what you must do to enable the catch code to run without interference.

By default, ThreadGroupRelease attempts to stop running threads by marking them with an unclearable abort flag. "Unclearable" means that, if a thread worker function has a catch block, when the abort flag is set, the catch block is called but the abort is still in effect which interferes with the code in the catch block.