```
    Variable ttime= stopMSTimer(-2)

    Variable t0= stopMSTimer(-2)
    MTFillWave(jack)
    Variable t1= stopMSTimer(-2)
    STFillWave(jack)
    Variable t2= stopMSTimer(-2)

    ttime= (stopMSTimer(-2) - ttime)*1e-6

    // Times are in microseconds
    printf "ST: %d, MT: %d; ",t2-t1,t1-t0
    printf "speed up factor: %.3g; total time= %.3gs\r",(t2-t1)/(t1-t0),ttime
End
```

The empty loop above is necessary because of periodic pauses in execution when Igor checks for user aborts. If a pause was pending, we want to get it out of the way beforehand to avoid it affecting the first timing test.

After starting Igor Pro, there is initially some extra overhead associated with creating new threads. Consequently, in the test results to follow, the first test is run twice.

Results for Mac Mini 1.66 GHz Core Duo, OS X 10.4.6:

```
•ThreadTest(100)
  ST: 223, MT: 1192; speed up factor: 0.187; total time= 0.00146s
•ThreadTest(100)
  ST: 211, MT: 884; speed up factor: 0.239; total time= 0.0011s
•ThreadTest(1000)
  ST: 1991, MT: 1821; speed up factor: 1.09; total time= 0.00381s
•ThreadTest(10000)
  ST: 19857, MT: 11921; speed up factor: 1.67; total time= 0.0318s
•ThreadTest(100000)
  ST: 199174, MT: 113701; speed up factor: 1.75; total time= 0.313s
•ThreadTest(1000000)
  ST: 2009948, MT: 1146113; speed up factor: 1.75; total time= 3.16s
```

As you can see, when there is sufficient work to be done, the speed up factor approaches the theoretical maximum of 2 for dual processors.

Now on the same computer but booting into Windows XP Pro:

```
•ThreadTest(100)
  ST: 245, MT: 523; speed up factor: 0.468; total time= 0.000776s
•ThreadTest(100)
  ST: 399, MT: 247; speed up factor: 1.61; total time= 0.000655s
•ThreadTest(1000)
  ST: 3526, MT: 1148; speed up factor: 3.07; total time= 0.00468s
•ThreadTest(10000)
  ST: 34830, MT: 10467; speed up factor: 3.33; total time= 0.0453s
•ThreadTest(100000)
  ST: 350253, MT: 99298; speed up factor: 3.53; total time= 0.45s
•ThreadTest(1000000)
  ST: 2837645, MT: 1057275; speed up factor: 2.68; total time= 3.89s
```

So, what is happening here? The speed-up factors for Windows XP are greater than for Mac OS X, but mostly because the ST version is much slower. We do not known why the ST version runs more slowly — the Benchmark 2.01 example experiment shows similar values for OS X vs. XP on this same computer.

Parallel Processing - Thread-at-a-Time Method

In the previous section, we dispatched a group of threads, waited for them to all finish, and then dispatched another group of threads. Using that technique, a slow thread in the group would cause all of the group's threads to wait.

In this section, we dispatch a thread anytime there is a free thread in the group. This technique requires Igor Pro 6.23 or later.

The only thing that changes from the preceding example is that the MTFillWave function is replaced with this MTFillWaveThreadAtATime function: