```
1  // PThreads Example
2  // ECE4893/8893, Fall 2011
3  // George F. Riley, Georgia Tech, Fall 2011
4
5  #include <iostream>
6  #include <string>
7  #include <stdlib.h>
8  #include <pthread.h>
9  #include <sys/time.h>
10
11 #include "InputImage.h"
12 #include "Complex.h"
13
14 using namespace std;
15
16 // Global variable visible to all threads
17 Complex* ImageData;
18 int      ImageWidth;
19 int      ImageHeight;
20 // Global variable that threads use to count elements
21 int      elementCount = 0;
22
23 // Each thread needs to know how many threads there are.
24 // This is similar to MPI_Comm_size
25 int      nThreads = 4;
26
27 // The mutex and condition variables allow the main thread to
28 // know when all helper threads are completed.
29 pthread_mutex_t startCountMutex;
30 pthread_mutex_t exitMutex;
31 pthread_mutex_t elementCountMutex;
32 pthread_cond_t  exitCond;
33 int             startCount;
34
35 // Millisecond clock function
36 int GetMillisecondClock()
37 {
38   timeval tv;
39   gettimeofday(&tv, 0);
40   static bool first = true;
41   static int startSec = 0;
42   if (first)
43     {
44       startSec = tv.tv_sec;
45       first = false;
46     }
47   // Time in milliseconds
48   return (tv.tv_sec - startSec) * 1000 + tv.tv_usec / 1000;
49 }
50
51 // This is the starting point for each of our threads
52 void* CountThread(void* v)
53 {
54   unsigned long myId = (unsigned long)v; // The parameter is actually the thread number
55   unsigned long localCount = 0;
56   // We can assume evenly divisible here. Would be a bit more complicated
```

Program ThreadedCount.cc

```
57    // if not.
58    int rowsPerThread = ImageHeight / nThreads;
59    int startingRow = myId * rowsPerThread;
60    // Now count the number of elements in the image with a magnitude < 100.0
61    for (int r = 0; r < rowsPerThread; ++r)
62      {
63        int thisRow = startingRow + r;
64        for (int c = 0; c < ImageWidth; ++c)
65          {
66            Complex thisElement = ImageData[thisRow * ImageWidth + c];
67            if (thisElement.Mag().real < 100.0)
68              { // Count it
69                //pthread_mutex_lock(&elementCountMutex);
70                localCount++;
71                //pthread_mutex_unlock(&elementCountMutex);
72              }
73          }
74      }
75    pthread_mutex_lock(&elementCountMutex);
76    elementCount += localCount;
77    pthread_mutex_unlock(&elementCountMutex);
78    // This thread is done; decrement the active count and see if all
79    // have finished
80    pthread_mutex_lock(&startCountMutex);
81    startCount--;
82    if (startCount == 0)
83      { // Last to exit, notify main
84        pthread_mutex_unlock(&startCountMutex);
85        pthread_mutex_lock(&exitMutex);
86        pthread_cond_signal(&exitCond);
87        pthread_mutex_unlock(&exitMutex);
88      }
89    else
90      {
91        pthread_mutex_unlock(&startCountMutex);
92      }
93  }
94
95  int main(int argc, char** argv)
96  {
97    string fileName("Tower-Extra-Large.txt");
98    // See if number of thread specified on command lind
99    if (argc > 1) nThreads = atol(argv[1]);
100
101   // See if file name specified on command line
102   if (argc > 2) fileName = string(argv[2]);
103   InputImage image(fileName.c_str());
104   // We use a global pointer so all threads can see the
105   // same image data array as well as width/height
106   ImageData = image.GetImageData();
107   ImageWidth = image.GetWidth();
108   ImageHeight = image.GetHeight();
109
110   // All mutex and condition variables must be "initialized"
111   pthread_mutex_init(&exitMutex,0);
112   pthread_mutex_init(&startCountMutex,0);
```

Program ThreadedCount.cc (continued)

```
113     pthread_mutex_init(&elementCountMutex,0);
114     pthread_cond_init(&exitCond, 0);
115     // Main holds the exit mutex until waiting for exitCond condition
116     pthread_mutex_lock(&exitMutex);
117
118     // Get elapsed milliseconds (starting time after image loaded)
119     GetMillisecondClock();
120     startCount = nThreads;  // Total threads (to be) started
121     // Now start the threads
122     for (int i = 0; i < nThreads; ++i)
123       {
124         // Now create the thread
125         pthread_t pt; // pThread variable (output param from create)
126         // Third param is the thread starting function
127         // Fourth param is passed to the thread starting function
128         pthread_create(&pt, 0, CountThread, (void*)i);
129       }
130     // Main program now waits until all child threads completed
131     pthread_cond_wait(&exitCond, &exitMutex);
132     // At this point all thread have completed and global "count"
133     // is the number of image elements with magnitude < 100.0
134     cout << "Elapsed time (seconds) " << GetMillisecondClock() / 1000.0 << endl;
135     cout << "Count is " << elementCount << endl;
136   }
137
138
139
```

Program ThreadedCount.cc (continued)