```c
#include <pthread.h>
#include <stdio.h>
#define SHARED 1
#define MAXSIZE 2000   /* maximum matrix size */
#define MAXWORKERS 4   /* maximum number of workers */

pthread_mutex_t barrier;  /* lock for the barrier */
pthread_cond_t go;        /* condition variable */
int numWorkers;           /* number of worker threads */
int numArrived = 0;       /* number who have arrived */

/* a reusable counter barrier */
void Barrier() {
  pthread_mutex_lock(&barrier);
  numArrived++;
  if (numArrived < numWorkers)
    pthread_cond_wait(&go, &barrier);
  else {
    numArrived = 0;  /* last worker awakens others */
    pthread_cond_broadcast(&go);
  }
  pthread_mutex_unlock(&barrier);
}

void *Worker(void *);
int size, stripSize;   /* size == stripSize*numWorkers */
int sums[MAXWORKERS];  /* sums computed by each worker */
int matrix[MAXSIZE][MAXSIZE];

/* read command line, initialize, and create threads */
int main(int argc, char *argv[]) {
  int i, j;
  pthread_attr_t attr;
  pthread_t workerid[MAXWORKERS];

  /* set global thread attributes */
  pthread_attr_init(&attr);
  pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);

  /* initialize mutex and condition variable */
  pthread_mutex_init(&barrier, NULL);
  pthread_cond_init(&go, NULL);

  /* read command line */
  size = atoi(argv[1]);
  numWorkers = atoi(argv[2]);
  stripSize = size/numWorkers;

  /* initialize the matrix */
  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
      matrix[i][j] = 1;

  /* create the workers, then exit main thread */
  for (i = 0; i < numWorkers; i++)
    pthread_create(&workerid[i], &attr,
                   Worker, (void *) i);
  pthread_exit(NULL);
}

/* Each worker sums the values in one strip.
   After a barrier, worker(0) prints the total */
void *Worker(void *arg) {
  int myid = (int) arg;
  int total, i, j, first, last;

  /* determine first and last rows of my strip */
  first = myid*stripSize;
  last = first + stripSize - 1;

  /* sum values in my strip */
  total = 0;
  for (i = first; i <= last; i++)
    for (j = 0; j < size; j++)
      total += matrix[i][j];
  sums[myid] = total;
  Barrier();
  if (myid == 0) {  /* worker 0 computes the total */
    total = 0;
    for (i = 0; i < numWorkers; i++)
      total += sums[i];
    printf("the total is %d\n", total);
  }
}
```

Figure 5.18 Parallel matrix summation using Pthreads.