1. Why is “chunky” more suited to parallelization?

   By rearranging the workload into chunks allows the threads to work on a larger working set, causing less overhead.

2. Where do we need to put synchronization (mutex/barrier) and why (use ‘chunky’)?

   Barrier, because this is where the threads wait for all the threads to finish, where we can chunk the data for the next step

3. How could we make this program perform even faster?

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NUM_THREADS 4
#define ARRAY_SIZE 2000000
#define STRING_SIZE 16
#define ALPHABET_SIZE 26
char char_array[ARRAY_SIZE][STRING_SIZE];
int char_counts[ALPHABET_SIZE]; // count of individual characters
char getRandomChar()
{
   int randNum = 0;
   char randChar = ' ';
   randNum = ALPHABET_SIZE * (rand() / (RAND_MAX + 1.0)); // pick number 0 < # < 25
   randNum = randNum + 97; // scale to 'a'
   randChar = (char) randNum;
   // printf("%c", randChar);
   return randChar;
}
void init_arrays()
{
   int i, j;
   for ( i = 0; i < ARRAY_SIZE; i++) {
      for ( j = 0; j < STRING_SIZE; j++ ) {
         char_array[i][j] = getRandomChar();
      }
   }
   for ( i = 0; i < ALPHABET_SIZE; i++ ) {
      char_counts[i] = 0;
   }
}
void count_array(int myID) // void count_array(void *threadid)
{
   char theChar;
   int i, j, charLoc;
   int local_char_count[ALPHABET_SIZE];
   int startPos = myID * (ARRAY_SIZE / NUM_THREADS);
   int endPos = startPos + (ARRAY_SIZE / NUM_THREADS);
   printf("myID = %d startPos = %d endPos = %d \n", myID, startPos, endPos);
   // init local count array
   for ( i = 0; i < ALPHABET_SIZE; i++ ) {
      local_char_count[i] = 0;
   }
   // count up our section of the global array
   for ( i = startPos; i < endPos; i++) {
      for ( j = 0; j < STRING_SIZE; j++ ) {
         theChar = char_array[i][j];
         charLoc = ((int) theChar) - 97;
         local_char_count[charLoc]++;
      }
   }
   // sum up the partial counts into the global arrays
   for ( i = 0; i < ALPHABET_SIZE; i++ ) {
      char_counts[i] += local_char_count[i];
   }
}
void print_results()
{
   int i,j, total = 0;
   // then print out the totals
   for ( i = 0; i < ALPHABET_SIZE; i++ ) {
      total += char_counts[i];
      printf(" %c %d\n", (char) (i + 97), char_counts[i]);
   }
   printf("\nTotal characters: %d\n", total);
}
main() {
   int i;
   init_arrays();
   for (i = 0; i < NUM_THREADS; i++ ) {
      count_array(i);
   }
   print_results();
}

  pthread_t threads[NUM_THREADS];                  // probably wrong but owell

  int rc, t;

  for(t=0;t<NUM_THREADS;t++)

     { printf("Creating thread %d\n", t);

      rc = pthread_create(&threads[t], NULL, count_array, (void *)t);

      if (rc)

        { printf("ERROR; return code from pthread_create() is %d\n", rc);

          exit(-1); } }

  pthread_exit(NULL);

  print_results();
```