Umeå University
Department of Computing Science

Parallel Programming 7.5 p
5DV152

Exercises, Chapter/Topic 5

Submitted	2017-03-02
Revision date	2016-03-28
Author:	Lorenz Gerber (`dv15lgr@cs.umu.se` `lozger03@student.umu.se`)
Instructor:	Lars Karlsson / Mikael Ränner

Contents

1 Introduction

This is the resubmission of exercise A5.1 from chapter 5 [1, p.267].

2 Histogram A5.1

As mentioned by the supervisor, in the initial submission a global variable was accessed from the individual threads in a parallel for pragma. To solve this problem several solutions were considered. The two obvious ones were:

1. critical section
   This is the simplest solution, however it probably also has the worst performance as it makes a large fraction of the parallel for loop serial.

2. reduction
   This would be the most elegant version. But reduction of an array structure is only supported in the newest openMP versions which are not yet supported by the compilers installed on the HPC2N system.

After some research, a third compromise solutions was found. It should perform better than locking the whole vector with a critical section but still worse than using local variables that get reduced with collective communication in the end of the loop: using separate locks for the different indices of the array. The source code can be found in the appendix A.

A C Source Code of Exercise A5.1

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

void Usage(char prog_name[]);

void Get_args(
      char*    argv[]        /* in  */,
      int*     bin_count_p   /* out */,
      float*   min_meas_p    /* out */,
      float*   max_meas_p    /* out */,
      int*     data_count_p  /* out */);

void Gen_data(
      float   min_meas    /* in  */,
      float   max_meas    /* in  */,
      float   data[]      /* out */,
      int     data_count  /* in  */);

void Gen_bins(
      float min_meas      /* in  */,
      float max_meas      /* in  */,
      float bin_maxes[]   /* out */,
      int   bin_counts[]  /* out */,
      int   bin_count     /* in  */);

int Which_bin(
      float    data         /* in */,
      float    bin_maxes[]  /* in */,
      int      bin_count    /* in */,
      float    min_meas     /* in */);

void Print_histo(
      float    bin_maxes[]   /* in */,
      int      bin_counts[]  /* in */,
      int      bin_count     /* in */,
      float    min_meas      /* in */);

int main(int argc, char* argv[]) {
   int bin_count, i, bin;
   float min_meas, max_meas;
   float* bin_maxes;
   int* bin_counts;
   int data_count;
   float* data;
   int thread_count;


   /* Check and get command line args */
   if (argc != 6) Usage(argv[0]);
   Get_args(argv, &bin_count, &min_meas, &max_meas, &data_count);
   thread_count = strtol(argv[5], NULL, 10);

   /* Allocate arrays needed */
   bin_maxes = malloc(bin_count*sizeof(float));
   bin_counts = malloc(bin_count*sizeof(int));
   data = malloc(data_count*sizeof(float));
   omp_lock_t lock[bin_count];

   for (int i=0; i<bin_count;i++)
     omp_init_lock(&(lock[i]));

   /* Generate the data */
   Gen_data(min_meas, max_meas, data, data_count);

   /* Create bins for storing counts */
   Gen_bins(min_meas, max_meas, bin_maxes, bin_counts, bin_count);

   /* Count number of values in each bin */
#  pragma omp parallel for num_threads(thread_count) shared(lock, bin_counts) private (bi
  for (i = 0; i < data_count; i++) {
    bin = Which_bin(data[i], bin_maxes, bin_count, min_meas);

    omp_set_lock(&(lock[bin]));
    bin_counts[bin]++;
    omp_unset_lock(&(lock[bin]));

   }

#  ifdef DEBUG
   printf("bin_counts = ");
   for (i = 0; i < bin_count; i++)
      printf("%d ", bin_counts[i]);
   printf("\n");
#  endif

   /* Print the histogram */
   Print_histo(bin_maxes, bin_counts, bin_count, min_meas);

   for(int i=0; i < bin_count; i++)
     omp_destroy_lock(&(lock[i]));

   free(data);
   free(bin_maxes);
   free(bin_counts);
   return 0;

}  /* main */


/*---------------------------------------------------------------------
 * Function:  Usage
 * Purpose:   Print a message showing how to run program and quit
 * In arg:    prog_name:  the name of the program from the command line
 */
void Usage(char prog_name[] /* in */) {
   fprintf(stderr, "usage: %s ", prog_name);
   fprintf(stderr, "<bin_count> <min_meas> <max_meas> <data_count> <threads>\n");
   exit(0);
}  /* Usage */


/*---------------------------------------------------------------------
 * Function:  Get_args
 * Purpose:   Get the command line arguments
 * In arg:    argv:  strings from command line
 * Out args:  bin_count_p:   number of bins
 *            min_meas_p:    minimum measurement
 *            max_meas_p:    maximum measurement
 *            data_count_p:  number of measurements
 */
void Get_args(
      char*    argv[]        /* in  */,
      int*     bin_count_p   /* out */,
      float*   min_meas_p    /* out */,
      float*   max_meas_p    /* out */,
      int*     data_count_p  /* out */) {

   *bin_count_p = strtol(argv[1], NULL, 10);
   *min_meas_p = strtof(argv[2], NULL);
   *max_meas_p = strtof(argv[3], NULL);
   *data_count_p = strtol(argv[4], NULL, 10);

#  ifdef DEBUG
   printf("bin_count = %d\n", *bin_count_p);
   printf("min_meas = %f, max_meas = %f\n", *min_meas_p, *max_meas_p);
   printf("data_count = %d\n", *data_count_p);
#  endif
}  /* Get_args */


/*----------------------------------------------------------------------
 * Function:  Gen_data
 * Purpose:   Generate random floats in the range min_meas <= x < max_meas
 * In args:   min_meas:    the minimum possible value for the data
 *            max_meas:    the maximum possible value for the data
 *            data_count:  the number of measurements
 * Out arg:   data:        the actual measurements
 */
void Gen_data(
        float   min_meas    /* in  */,
        float   max_meas    /* in  */,
        float   data[]      /* out */,
        int     data_count  /* in  */) {
   int i;

   srandom(0);
   for (i = 0; i < data_count; i++)
      data[i] = min_meas + (max_meas - min_meas)*random()/((double) RAND_MAX);

#  ifdef DEBUG
   printf("data = ");
   for (i = 0; i < data_count; i++)
      printf("%4.3f ", data[i]);
   printf("\n");
#  endif
}  /* Gen_data */


/*---------------------------------------------------------------------
 * Function:  Gen_bins
 * Purpose:   Compute max value for each bin, and store 0 as the
 *            number of values in each bin
 * In args:   min_meas:   the minimum possible measurement
 *            max_meas:   the maximum possible measurement
 *            bin_count:  the number of bins
 * Out args:  bin_maxes:  the maximum possible value for each bin
 *            bin_counts: the number of data values in each bin
 */
void Gen_bins(
      float min_meas      /* in  */,
      float max_meas      /* in  */,
      float bin_maxes[]   /* out */,
      int   bin_counts[]  /* out */,
      int   bin_count     /* in  */) {
   float bin_width;
   int   i;

   bin_width = (max_meas - min_meas)/bin_count;

   for (i = 0; i < bin_count; i++) {
      bin_maxes[i] = min_meas + (i+1)*bin_width;
      bin_counts[i] = 0;
   }

#  ifdef DEBUG
   printf("bin_maxes = ");
   for (i = 0; i < bin_count; i++)
      printf("%4.3f ", bin_maxes[i]);
   printf("\n");
#  endif
}  /* Gen_bins */


/*---------------------------------------------------------------------
 * Function:  Which_bin
 * Purpose:   Use binary search to determine which bin a measurement
 *            belongs to
 * In args:   data:       the current measurement
 *            bin_maxes:  list of max bin values
 *            bin_count:  number of bins
 *            min_meas:   the minimum possible measurement
 * Return:    the number of the bin to which data belongs
 * Notes:
```

```
 * 1.  The bin to which data belongs satisfies
 *
 *            bin_maxes[i-1] <= data < bin_maxes[i]
 *
 *     where, bin_maxes[-1] = min_meas
 * 2.  If the search fails, the function prints a message and exits
 */
int Which_bin(
      float   data          /* in */,
      float   bin_maxes[]   /* in */,
      int     bin_count     /* in */,
      float   min_meas      /* in */) {
   int bottom = 0, top =  bin_count-1;
   int mid;
   float bin_max, bin_min;

   while (bottom <= top) {
      mid = (bottom + top)/2;
      bin_max = bin_maxes[mid];
      bin_min = (mid == 0) ? min_meas: bin_maxes[mid-1];
      if (data >= bin_max)
         bottom = mid+1;
      else if (data < bin_min)
         top = mid-1;
      else
         return mid;
   }

   /* Whoops! */
   fprintf(stderr, "Data = %f doesn’t belong to a bin!\n", data);
   fprintf(stderr, "Quitting\n");
   exit(-1);
}  /* Which_bin */


/*----------------------------------------------------------------------
 * Function:  Print_histo
 * Purpose:   Print a histogram.  The number of elements in each
 *            bin is shown by an array of X’s.
 * In args:   bin_maxes:   the max value for each bin
 *            bin_counts:  the number of elements in each bin
 *            bin_count:   the number of bins
 *            min_meas:    the minimum possible measurment
 */
void Print_histo(
        float  bin_maxes[]   /* in */,
        int    bin_counts[]  /* in */,
        int    bin_count     /* in */,
        float  min_meas      /* in */) {
   int i, j;
   float bin_max, bin_min;

   for (i = 0; i < bin_count; i++) {
      bin_max = bin_maxes[i];
      bin_min = (i == 0) ? min_meas: bin_maxes[i-1];
      printf("%.3f-%.3f:\t", bin_min, bin_max);
      for (j = 0; j < bin_counts[i]; j++)
         printf("X");
      printf("\n");
   }
}  /* Print_histo */
```

References

[1] P.S. Pacheco. *An Introduction to Parallel Programming*. Morgan Kaufman, 2011.