Scalable Computing: CA #3

Due on Monday, May 6th, 2013

Dr. John Burns

Alvaro Pereda

Contents

Q. 1

Part 1:

We wish to find the maximum, minimum and mean value in an array of 500 random integers between 0 and 1000. Write a C/C++ program to achieve this. You are NOT permitted to use any libraries for this - you must implement all aspects of the program yourself. If you use any libraries you will score 0 for this section. Use the timing methodology from the labs to demonstrate the wall-clock performance of this solution.

Listing 1 shows the algorithm implementation in C. The problem has been divided in three functions:

- The *main* function, that schedules the calls to the other functions, sets the timers up and displays the results.
- The *build* function, that with calls to the mathematical pseudo random numbers generation *rand()* initializes the integer array.
- The *stats* function, that collects the max value, min and average.

In order to collect the results, a struct has been implemented as can be seen in Listing 2. The loop printing the results has been commented out.
The results are as follows:
Start
Avg = 47

Max = 99

Min = 0
Static: Elapsed: 0.000082 seconds

Listing 1: No parallel implementation in C.

```c
/*
 * statistics-threads.c
 *
 *  Created on: Apr 29, 2013
 *      Author: alvaroperedasancho
 */
#include "statistics-threads.h"
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int array[ARRAY_SIZE];
result tr[NUM_THREADS];

int main() {
     int i;
     clock_t tic = clock();

     printf("Start");

     build(array, ARRAY_SIZE);
//   for (i = 0; i < ARRAY_SIZE; i++) {
```

```
//        printf("\nArray %d, %d", i, array[i]);
//   }
     result r = stats(array, ARRAY_SIZE);

     printf("\nAvg = %d\n", r.avg);
     printf("\nMax = %d\n", r.max);
     printf("\nMin = %d\n", r.min);

     clock_t toc = clock();

     printf("Static: Elapsed: %f seconds\n", (double)(toc - tic) / CLOCKS_PER_SEC);

     return 0;
}

result stats(int *array, int size) {
     result r = {0, 0, 100};
     int i;
     for (i = 0; i < size; i++) {
          if (r.max < array[i]) {
               r.max = array[i];
          }
          if (r.min > array[i]) {
               r.min = array[i];
          }
          r.avg += array[i];
     }

     r.avg /= size;

     return r;
}

void build(int *a, int size) {
     int i;

     for (i = 0; i < size; i++) {
          a[i] = rand()%100;
     }
}
```

Listing 2: Corresponding header file

```
/*
 * statistics-threads.h
 *
 *  Created on: Apr 29, 2013
 *      Author: alvaroperedasancho
 */

#ifndef STATISTICS_THREADS_H_
#define STATISTICS_THREADS_H_
#define ARRAY_SIZE       500
#define NUM_THREADS     5
```

```
typedef struct {
     int avg;
     int max;
     int min;
} result;

result stats(int *array, int size);
void build(int *array, int size);
#endif /* STATISTICS_THREADS_H_ */
```

Part 2:

Next, discuss in no less than 500 words how each and every part of your program could be executed in parallel.

Part 3.

Using pthreads, implement your analysis from part 2. to build a parallel solution. Demonstrate the results of this in your answer PDF to show the maximum, minimum and mean are correct. To do this, it is best to run the serial code followed by the parallel code. In this section you will be marked on correctness of the solution. You must make sure that workload is evenly distributed across threads and the threads must all execute separate and independent computations.

The implementation discussed in can be read in Listing 3
The result is:
Start
static build: Elapsed: 0.000049 seconds
Avg = 508.8200
Max = 997
Min = 0
Static stats: Elapsed: 0.000009 seconds
Avg = 508.8200
Max = 997
Min = 0
dynamic stats: Elapsed: 0.000385 seconds
Selection Sort
2 6 5 1 4 3
1 6 5 2 4 3
1 2 5 6 4 3
1 2 3 6 4 5
1 2 3 4 6 5
1 2 3 4 5 6
Algorithm Sort
2 6 5 1 4 3
2 6 5 1 4 3
2 5 6 1 4 3
1 2 5 6 4 3
1 2 4 5 6 3
1 2 3 4 5 6

Listing 3: Parallel implementation in C.

```c
/*
 * statistics-threads.c
 *
 *  Created on: Apr 29, 2013
 *      Author: alvaroperedasancho
 */
#include "statistics-threads.h"
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <pthread.h>

int array[ARRAY_SIZE];
result tr[NUM_THREADS];
float avgs[NUM_THREADS];
int maxs[NUM_THREADS];
int mins[NUM_THREADS];

int main() {
     clock_t tic = clock();

     printf("Start\n");
```

```
25      build(array, ARRAY_SIZE);
//    for (int i = 0; i < ARRAY_SIZE; i++) {
//         printf("\nArray %d, %d", i, array[i]);
//    }
      clock_t toc = clock();
30      printf("static build: Elapsed: %f seconds\n",
                (double) (toc - tic) / CLOCKS_PER_SEC);
      tic = clock();

      result r = stats(array, ARRAY_SIZE);
35
      printf("Avg = %.4f\n", r.avg);
      printf("Max = %d\n", r.max);
      printf("Min = %d\n", r.min);

40      toc = clock();

      printf("Static stats: Elapsed: %f seconds\n",
                (double) (toc - tic) / CLOCKS_PER_SEC);

45      /*tic = clock();
      thread_starter(pt_build);
      toc = clock();
      printf("dynamic build: Elapsed: %f seconds\n",
                (double) (toc - tic) / CLOCKS_PER_SEC);
50  */
      tic = clock();
      thread_starter(pt_stats);
      r = pt_summarize();
      printf("Avg = %.4f\n", r.avg);
55      printf("Max = %d\n", r.max);
      printf("Min = %d\n", r.min);

      toc = clock();
      printf("dynamic stats: Elapsed: %f seconds\n",
60                (double) (toc - tic) / CLOCKS_PER_SEC);

      /* Last thing that main() should do */
      pthread_exit(NULL );

65      return 0;
  }

  result stats(int *array, int size) {
      result r = { 0.0, 0, 1000 };
70      for (int i = 0; i < size; i++) {
          if (r.max < array[i]) {
              r.max = array[i];
          }
          if (r.min > array[i]) {
75              r.min = array[i];
          }
```

```
            r.avg += array[i];
        }

        r.avg /= size;

        return r;
}

void build(int *a, int size) {
        for (int i = 0; i < size; i++) {
            a[i] = rand() % 1000;
        }
}

int thread_starter(void *exe) {
        pthread_t threads[NUM_THREADS];
        void * status;

        for (int t = 0; t < NUM_THREADS; t++) {
            int rc = pthread_create(&threads[t], NULL, exe, (void *) t);
        }

        for (int j = 0; j < NUM_THREADS; j++) {
            pthread_join(threads[j], &status);
        }

//      for (int i = 0; i < ARRAY_SIZE; i++) {
//          printf("\n Array built %d", array[i]);
//      }

        return 0;
}

void *pt_build(void *threadid) {
        int iter = ARRAY_SIZE / NUM_THREADS;
        int id = (int) threadid;
        for (int i = 0; i < iter; i++) {
            array[id * iter + i] = rand() % 1000;
        }
        pthread_exit(threadid);
        return threadid;
}

void *pt_stats(void *threadid) {
        int iter = ARRAY_SIZE / NUM_THREADS;
        int id = (int) threadid;
        result r = stats(&array[id * iter], iter);
        avgs[id] = r.avg;
        maxs[id] = r.max;
        mins[id] = r.min;

        pthread_exit(threadid);
        return threadid;
```

Q. 1 [Part 3.] continued on next page...

```
}

result pt_summarize() {
     result r = { 0, 0, 100 };
     for (int i = 0; i < NUM_THREADS; i++) {
          if (r.max < maxs[i]) {
               r.max = maxs[i];
          }
          if (r.min > mins[i]) {
               r.min = mins[i];
          }
          r.avg += avgs[i];
     }

     r.avg /= NUM_THREADS;

     return r;
}
```

Listing 4: Corresponding header file

```
/*
 * statistics-threads.h
 *
 *  Created on: Apr 29, 2013
 *      Author: alvaroperedasancho
 */

#ifndef STATISTICS_THREADS_H_
#define STATISTICS_THREADS_H_
#define ARRAY_SIZE       500
#define NUM_THREADS     5

typedef struct {
     float avg;
     int max;
     int min;
} result;

result stats(int *array, int size);
void build(int *array, int size);
int thread_starter (void *exe);
void *pt_build(void *threadid);
void *pt_stats(void *threadid);
result pt_summarize();

#endif /* STATISTICS_THREADS_H_ */
```