Umeå University
Department of Computing Science

Parallel Programming 7.5 p
5DV152

Exercises, Chapter/Topic 1

Submitted	2017-01-26
Author:	Lorenz Gerber (dv15lgr@cs.umu.se lozger03@student.umu.se)
Instructor:	Lars Karlsson / Mikael Ränner

Contents

1 Introduction

This report is part of the mandatory coursework. It describes the solutions for several chosen exercises from the course book [**?**].

2 4.1 - Generalization of matrix-vector multiplication

If we keep the same scheme of parallelization as mentioned in the book (outer for...loop), generalzation can be implemented rather easy, bascially in the same way as already shown in exercise 1.1:

```
my_first_i = k * m / p + ( k < m mod p ? k : m mod p )
my_last_i = (k + 1) * m / p + (k + 1 < m mod p ? k + 1 : m mod p)
```

It is not useful to parallelize into n as one thread needs to process as this would create a mutex for acces to shared variables.

3 4.2 - Physical data distribution

The source code for this exercise can be found in appendix **??**.

4 4.8 - Deadlock

5 4.11 - Linked list troubles

6 4.12 - Linked list insert and delete with read-write lock

7 4.17 - False sharing

8 A4.1 - Histogram

9 A4.3 - Trapezoidal rule

10 A4.4 - Fork/join overhead

11 A4.5 - Task queue

A C Source Code for Exercise 4.2

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include "timer.h"

/* Global variables */
int     thread_count;
int     m, n;
int     input_counter = 0;
int     output_counter = 0;
pthread_mutex_t mutex_input;
pthread_mutex_t mutex_output;
```

2(5)

```
pthread_cond_t cond_var_input;
pthread_cond_t cond_var_output;

/* Serial functions */
void Usage(char* prog_name);
void Gen_matrix(double A[], int m, int n);
void Read_matrix(char* prompt, double A[], int m, int n);
void Gen_vector(double x[], int n);
void Read_vector(char* prompt, double x[], int n);
void Print_matrix(char* title, double A[], int m, int n);
void Print_vector(char* title, double y[], double m);

/* Parallel function */
void *Pth_mat_vect(void* rank);


/*------------------------------------------------------------------*/
int main(int argc, char* argv[]) {

  long       thread;
  pthread_t* thread_handles;

  if (argc != 4) Usage(argv[0]);
  thread_count = strtol(argv[1], NULL, 10);
  m = strtol(argv[2], NULL, 10);
  n = strtol(argv[3], NULL, 10);

# ifdef DEBUG
  printf("thread_count =  %d, m = %d, n = %d\n", thread_count, m, n);
# endif

  thread_handles = malloc(thread_count*sizeof(pthread_t));

  for (thread = 0; thread < thread_count; thread++)
     pthread_create(&thread_handles[thread], NULL,
        Pth_mat_vect, (void*) thread);

  for (thread = 0; thread < thread_count; thread++)
     pthread_join(thread_handles[thread], NULL);

   return 0;
}  /* main */


void Usage (char* prog_name) {
   fprintf(stderr, "usage: %s <thread_count> <m> <n>\n", prog_name);
   exit(0);
}  /* Usage */
```

```
void Read_matrix(char* prompt, double A[], int m, int n) {
   int             i, j;

   printf("%s\n", prompt);
   for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
         scanf("%lf", &A[i*n+j]);
}  /* Read_matrix */


void Gen_matrix(double A[], int m, int n) {
   int i, j;
   for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
         A[i*n+j] = random()/((double) RAND_MAX);
}  /* Gen_matrix */


void Gen_vector(double x[], int n) {
   int i;
   for (i = 0; i < n; i++)
      x[i] = random()/((double) RAND_MAX);
}

void Read_vector(char* prompt, double x[], int n) {
   int   i;

   printf("%s\n", prompt);
   for (i = 0; i < n; i++)
      scanf("%lf", &x[i]);
}


void *Pth_mat_vect(void* rank) {
  double* A;
  double* x;
  double* y;
  long my_rank = (long) rank;
  int i;
  int j;
  int local_m = m/thread_count;
  int my_first_row = 0; //my_rank*local_m;
  int my_last_row = m/thread_count; //my_first_row + local_m;
  register int sub = my_first_row*n;
  double start, finish;
  double temp;
```

4(5)

```
# ifdef DEBUG
  printf("Thread %ld > local_m = %d, sub = %d\n",
        my_rank, local_m, sub);
# endif
  // scheduling data input
  pthread_mutex_lock(&mutex_input);

  while(input_counter != my_rank){
    pthread_cond_wait(&cond_var_input, &mutex_input);
  }

  A = malloc(m/thread_count*n*sizeof(double));
  x = malloc(n*sizeof(double));
  y = malloc(m/thread_count*sizeof(double));

  Gen_matrix(A, local_m, n);
# ifdef DEBUG
  Print_matrix("We generated", A, local_m, n);
# endif

  Gen_vector(x, n);
# ifdef DEBUG
  Print_vector("We generated", x, n);
# endif


  input_counter++;
  pthread_cond_broadcast(&cond_var_input);
  pthread_mutex_unlock(&mutex_input);

  GET_TIME(start);
  for (i = my_first_row; i < my_last_row; i++) {
     y[i] = 0.0;
     for (j = 0; j < n; j++) {
         temp = A[sub++];
         temp *= x[j];
         y[i] += temp;
     }
  }
  GET_TIME(finish);

  pthread_mutex_lock(&mutex_output);

  while(output_counter != my_rank){
    pthread_cond_wait(&cond_var_output, &mutex_output);
  }
  printf("Thread %ld > Elapsed time = %e seconds\n",
     my_rank, finish - start);
```

```
# ifdef DEBUG
  Print_vector("The Product is", y, m/thread_count);
# endif

  output_counter++;
  pthread_cond_broadcast(&cond_var_output);
  pthread_mutex_unlock(&mutex_input);


  free(A);
  free(x);
  free(y);

  return NULL;
}


void Print_matrix( char* title, double A[], int m, int n) {
   int   i, j;

   printf("%s\n", title);
   for (i = 0; i < m; i++) {
      for (j = 0; j < n; j++)
         printf("%6.3f ", A[i*n + j]);
      printf("\n");
   }
}


void Print_vector(char* title, double y[], double m) {
   int   i;

   printf("%s\n", title);
   for (i = 0; i < m; i++)
      printf("%6.3f ", y[i]);
   printf("\n");
}
```