We consider a Pthreads application computing the dot product of two real *m*-length vectors *x* and *y* to illustrate parallel programming an *n*-processor SMP computer with thread libraries. This MT application divides vectors *x* and *y* into *n* sub-vectors. Meanwhile the first *n*-1 sub-vectors are of the same length $\frac{m}{n}$, the last *n*-th subvector may be shorter if *m* is not a multiply of *n*. This application uses *n* parallel threads with *i*-th thread computing its fraction of the total dot product by multiplying sub-vectors x_i and y_i. The *n* parallel threads share a data object accumulating the dot product, and synchronize their access to the data object with a mutex. The main thread creates the *n* threads, waits for them to complete their computations via joining with each of the threads, and then outputs the result.

The source code of the application is as follows:

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

#define MAXTHRDS 124

typedef struct {
             double *my_x;
             double *my_y;
             double my_dot_prod;
             double *global_dot_prod;
             pthread_mutex_t *mutex;
             int my_vec_len;
} dot_product_t;

void *serial_dot_product(void *arg) {
   dot_product_t *dot_data;
   int i;

   dot_data = arg;
   for(i=0; i<dot_data->my_vec_len; i++)
     dot_data->my_dot_prod += dot_data->my_x[i]*dot_data->my_y[i];

   pthread_mutex_lock(dot_data->mutex);
   *(dot_data->global_dot_prod) += dot_data->my_dot_prod;
   pthread_mutex_unlock(dot_data->mutex);

   pthread_exit(NULL);
}

int main()
{
   double *x, *y, dot_prod;
   pthread_t *working_thread;
   dot_product_t *thrd_dot_prod_data;
   void *status;
   pthread_mutex_t *mutex_dot_prod;

   int num_of_thrds;
   int vec_len;
```

```
    int subvec_len;
    int i;

    printf("Number of processors = ");
    if(scanf("%d", &num_of_thrds) < 1 || num_of_thrds > MAXTHRDS) {
      printf("Check input for number of processors. Bye.\n");
      return -1;
    }
    printf("Vector length = ");
    if(scanf("%d", &vec_len)<1) {
      printf("Check input for vector length. Bye.\n");
      return -1;
    }
    subvec_len = vec_len/num_of_thrds;

    x = malloc(vec_len*sizeof(double));
    y = malloc(vec_len*sizeof(double));
    for(i=0; i<vec_len; i++) {
      x[i] = 1.;
      y[i] = 1.;
    }

    working_thread = malloc(num_of_thrds*sizeof(pthread_t));
    thrd_dot_prod_data = malloc(num_of_thrds*sizeof(dot_product_t));

    mutex_dot_prod = malloc(sizeof(pthread_mutex_t));
    pthread_mutex_init(mutex_dot_prod, NULL);

    for(i=0; i<num_of_thrds; i++) {
      thrd_dot_prod_data[i].my_x = x + i*subvec_len;
      thrd_dot_prod_data[i].my_y = y + i*subvec_len;
      thrd_dot_prod_data[i].my_dot_prod = 0.;
      thrd_dot_prod_data[i].global_dot_prod = &dot_prod;
      thrd_dot_prod_data[i].mutex = mutex_dot_prod;
      thrd_dot_prod_data[i].my_vec_len =
          (i==num_of_thrds-1)? vec_len-(num_of_thrds-1)*subvec_len
                             : subvec_len;
      pthread_create(&working_thread[i], NULL, serial_dot_product,
                     (void*)&thrd_dot_prod_data[i]);
    }
    for(i=0; i<num_of_threds; i++)
      pthread_join(working_thread[i], &status);

    printf("Dot product = %f\n", dot_prod);

    free(x);
    free(y);
    free(working_thread);
    free(thrd_dot_prod_data);
    pthread_mutex_destroy(mutex_dot_prod);
    free(mutex_dot_prod);
}
```