Abstract

Making observations on each code and extracting the running time.

1 Introduction

Implementing all the codes and observing the results after each run to check the performance.

2 Implementation

1- Reproducing the serial code:

Listing 1: C Example

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

int main()
{
        int             array_size;
        int             counter;
        int *           rand_arr;
        double          duration;
        clock_t         start;
        clock_t         end;

        array_size      = 10000000;
        counter         = 0;
        rand_arr        = calloc(array_size, sizeof(int));
        srand(time(NULL));
        start           = clock();

        for (int i = 0; i < array_size; i++)
    {
        rand_arr[i] = rand() % 10;
        if (rand_arr[i] == 3)
        {
            counter++;
        }
    }

        end             = clock();
        duration= ((double)(end - start) / CLOCKS_PER_SEC) * 1000;
        printf("There are %d 3s and it takes %fms", counter, duration);
        return 0;
}
```

2- Implementing data race:

Listing 2: C Example

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

#define MaxThreads 1000
```

```c
void* count3s_thread(void* id);
pthread_t tid[MaxThreads];

int t;          /* number of threads */
int * array;
int length;
int count;

void count3s()
{
   int i;
   count = 0;
   /* Create t threads */
   for(i = 0; i < t; i++)
   {
      pthread_create(&tid[i], NULL, count3s_thread, (void*)i);
   }

   /*** wait for the threads to finish ***/
   for(i = 0; i < t; i++)
   {
      pthread_join(tid[i], NULL);
   }
}

void* count3s_thread(void* id)
{
   int i;
   /* Compute portion of the array that this thread should work on */
   int length_per_thread = length / t;
   int start = (intptr_t)id * length_per_thread;

   for(i = start; i < start+length_per_thread; i++)
   {
      if(array[i] == 3)
      {
         count++;
      }
   }
   return 0;
}


int main(int argc, char *argv[])
{
   int i;
   length = 1048576;  /*  2^20  */
   t = 40;  /*** be sure that t divides length!! ***/

   array = calloc(length, sizeof(int));

   /* initialize the array with random integers between 0 and 9 */
   srand(time(NULL));  /* seed the random number generator with current time */
   for(i = 0; i < length; i++)
   {
      array[i] = rand()%10;
```

```
    }

    clock_t start = clock();
    count3s();
    clock_t end = clock();
    double time_spent = ((double)(end - start) / CLOCKS_PER_SEC) * 1000.0;
    printf("It takes %fms\n", time_spent);

    printf("Parallel: The number of 3's is %d\n", count);

    count = 0;
    for (i = 0; i < length; i++)
        if (array[i] == 3)
            count++;
    printf("Serial: The number of 3's is %d\n", count);

    return 0;
}
```

3- Implementing data race with locks only:

Listing 3: C Example

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

#define MaxThreads 1000
void* count3s_thread(void* id);
pthread_t tid[MaxThreads];

int t;          /* number of threads */
int * array;
int length;
int count;

pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

void count3s()
{
    int i;
    count = 0;
    /* Create t threads */
    for(i = 0; i < t; i++)
    {
        pthread_create(&tid[i], NULL, count3s_thread, (void*)i);
    }

    /*** wait for the threads to finish ***/
    for(i = 0; i < t; i++)
    {
        pthread_join(tid[i], NULL);
    }
}

void* count3s_thread(void* id)
{
    int i;
    /* Compute portion of the array that this thread should work on */
    int length_per_thread = length / t;
    int start = (intptr_t)id * length_per_thread;

    for(i = start; i < start+length_per_thread; i++)
    {
        if(array[i] == 3)
        {
            pthread_mutex_lock(&m);
            count++;
            pthread_mutex_unlock(&m);
        }
    }
    return 0;
}


int main(int argc, char *argv[])
{
    int i;
    length = 1048576;  /*  2^20  */
    t = 40;  /*** be sure that t divides length!! ***/

    array = calloc(length, sizeof(int));

    /* initialize the array with random integers between 0 and 9 */
    srand(time(NULL));  /* seed the random number generator with current time */
    for(i = 0; i < length; i++)
    {
        array[i] = rand()%10;
    }

    clock_t start = clock();
    count3s();
    clock_t end = clock();
    double time_spent = ((double)(end - start) / CLOCKS_PER_SEC) * 1000.0;
    printf("It takes %fms\n", time_spent);

    printf("Parallel: The number of 3's is %d\n", count);

    count = 0;
    for (i = 0; i < length; i++)
        if (array[i] == 3)
            count++;
    printf("Serial: The number of 3's is %d\n", count);

    return 0;
}
```

4- data race with locks and padding:

Listing 4: C Example

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

```c
#define MaxThreads 1000
void* count3s_thread(void* id);
pthread_t tid[MaxThreads];

int t;          /* number of threads */
int * array;
int length;
int count;

struct padded_int
{
     int value;
     char padding[60];
} private_count[MaxThreads];
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

void count3s()
{
   int i;
   count = 0;
   /* Create t threads */
   for(i = 0; i < t; i++)
   {
      pthread_create(&tid[i], NULL, count3s_thread, (void*)i);
   }

   /*** wait for the threads to finish ***/
   for(i = 0; i < t; i++)
   {
      pthread_join(tid[i], NULL);
   }
}

void* count3s_thread(void* id)
{
   int i;
   /* Compute portion of the array that this thread should work on */
   int length_per_thread = length / t;
   int start = (int)id * length_per_thread;

   for(i = start; i < start+length_per_thread; i++)
   {
      if(array[i] == 3)
      {
         private_count[(int)id].value++;
      }
   }
   pthread_mutex_lock(&m);
   count += private_count[(int)id].value;
   pthread_mutex_unlock(&m);

   return 0;
}
```

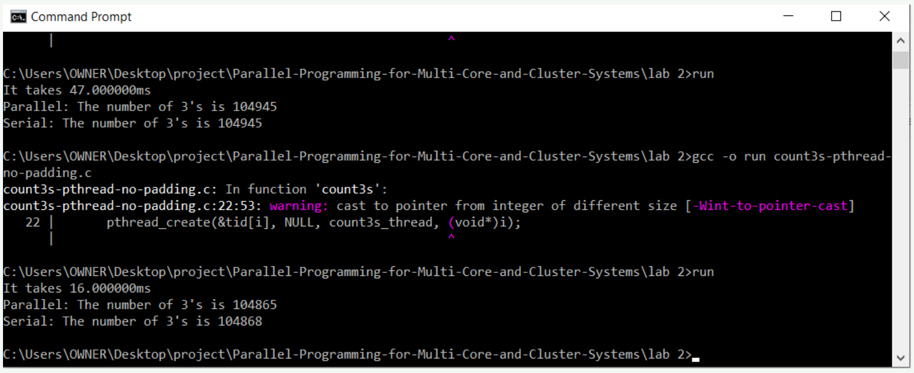
```
int main(int argc, char *argv[])
{
    int i;
    length = 1048576;  /*  2^20  */
    t = 40;  /*** be sure that t divides length!! ***/

    array = calloc(length, sizeof(int));

    /* initialize the array with random integers between 0 and 9 */
    srand(time(NULL));  /* seed the random number generator with current time */
    for(i = 0; i < length; i++)
    {
        array[i] = rand()%10;
    }

    clock_t start = clock();
    count3s();
    clock_t end = clock();
    double time_spent = ((double)(end - start) / CLOCKS_PER_SEC) * 1000.0;
    printf("It takes %fms\n", time_spent);

    printf("Parallel: The number of 3's is %d\n", count);

    count = 0;
    for (i = 0; i < length; i++)
        if (array[i] == 3)
            count++;
    printf("Serial: The number of 3's is %d\n", count);

    return 0;
}
```

3 Experimental Platform

Windows 10, Sublime text editor and a GCC compiler

4 Results

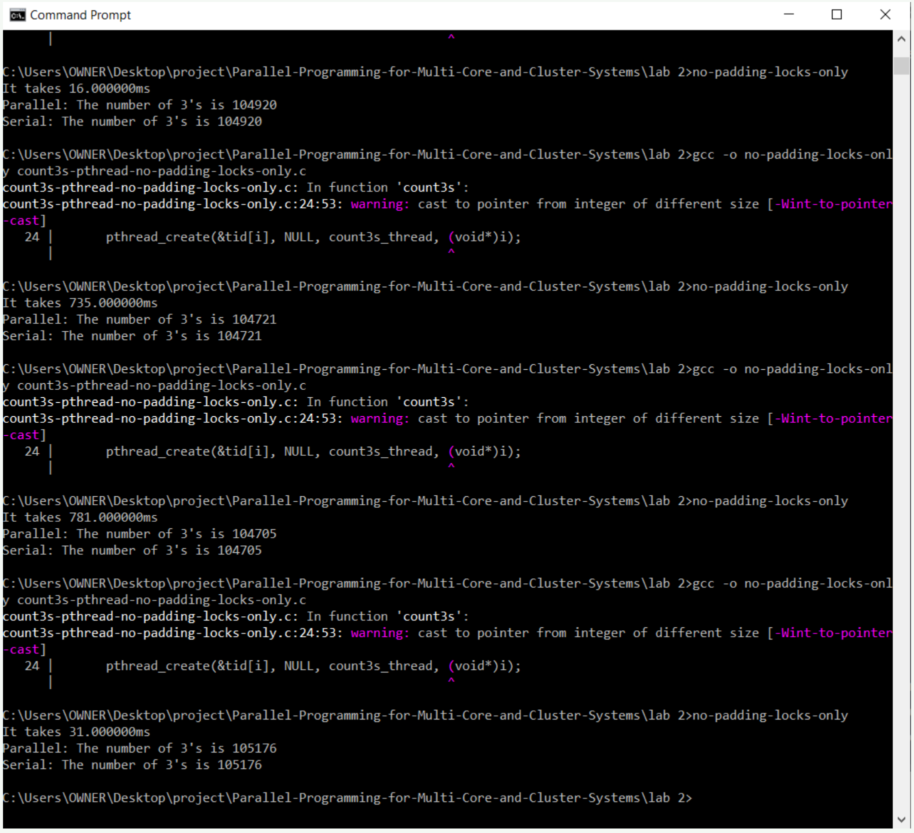
no padding:



```
      |                                                     ^

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>run
It takes 47.000000ms
Parallel: The number of 3's is 104945
Serial: The number of 3's is 104945

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>gcc -o run count3s-pthread-
no-padding.c
count3s-pthread-no-padding.c: In function 'count3s':
count3s-pthread-no-padding.c:22:53: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
   22 |       pthread_create(&tid[i], NULL, count3s_thread, (void*)i);
      |                                                     ^

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>run
It takes 16.000000ms
Parallel: The number of 3's is 104865
Serial: The number of 3's is 104868

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>
```

Figure 1: A screenshot of the terminal for no padding

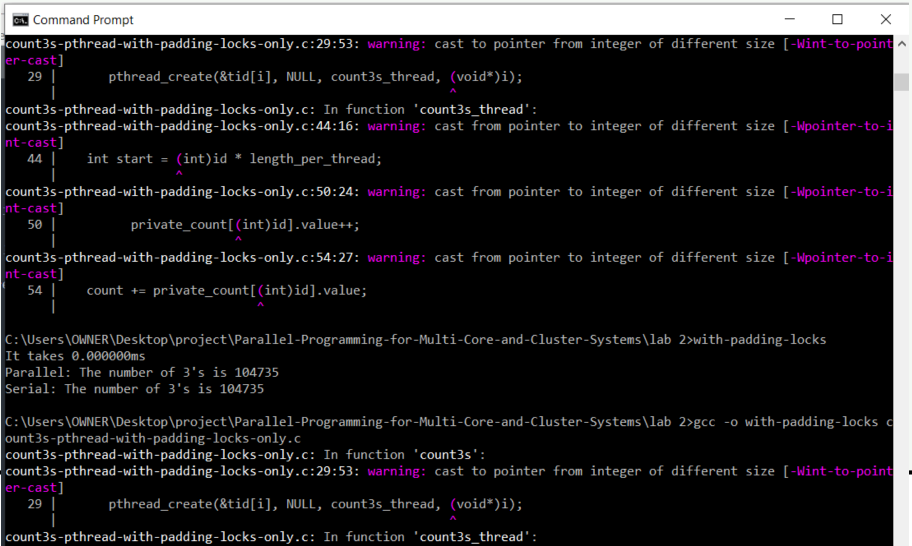
no padding but locks only:



```
      |                                                     ^

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>no-padding-locks-only
It takes 16.000000ms
Parallel: The number of 3's is 104920
Serial: The number of 3's is 104920

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>gcc -o no-padding-locks-onl
y count3s-pthread-no-padding-locks-only.c
count3s-pthread-no-padding-locks-only.c: In function 'count3s':
count3s-pthread-no-padding-locks-only.c:24:53: warning: cast to pointer from integer of different size [-Wint-to-pointer
-cast]
   24 |       pthread_create(&tid[i], NULL, count3s_thread, (void*)i);
      |                                                     ^

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>no-padding-locks-only
It takes 735.000000ms
Parallel: The number of 3's is 104721
Serial: The number of 3's is 104721

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>gcc -o no-padding-locks-onl
y count3s-pthread-no-padding-locks-only.c
count3s-pthread-no-padding-locks-only.c: In function 'count3s':
count3s-pthread-no-padding-locks-only.c:24:53: warning: cast to pointer from integer of different size [-Wint-to-pointer
-cast]
   24 |       pthread_create(&tid[i], NULL, count3s_thread, (void*)i);
      |                                                     ^

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>no-padding-locks-only
It takes 781.000000ms
Parallel: The number of 3's is 104705
Serial: The number of 3's is 104705

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>gcc -o no-padding-locks-onl
y count3s-pthread-no-padding-locks-only.c
count3s-pthread-no-padding-locks-only.c: In function 'count3s':
count3s-pthread-no-padding-locks-only.c:24:53: warning: cast to pointer from integer of different size [-Wint-to-pointer
-cast]
   24 |       pthread_create(&tid[i], NULL, count3s_thread, (void*)i);
      |                                                     ^

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>no-padding-locks-only
It takes 31.000000ms
Parallel: The number of 3's is 105176
Serial: The number of 3's is 105176

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>
```

Figure 2: A screenshot of the terminal for no padding but locks only

With padding and locks:



```
count3s-pthread-with-padding-locks-only.c:29:53: warning: cast to pointer from integer of different size [-Wint-to-point
er-cast]
   29 |       pthread_create(&tid[i], NULL, count3s_thread, (void*)i);
      |                                                     ^
count3s-pthread-with-padding-locks-only.c: In function 'count3s_thread':
count3s-pthread-with-padding-locks-only.c:44:16: warning: cast from pointer to integer of different size [-Wpointer-to-i
nt-cast]
   44 |   int start = (int)id * length_per_thread;
      |             ^
count3s-pthread-with-padding-locks-only.c:50:24: warning: cast from pointer to integer of different size [-Wpointer-to-i
nt-cast]
   50 |         private_count[(int)id].value++;
      |                       ^
count3s-pthread-with-padding-locks-only.c:54:27: warning: cast from pointer to integer of different size [-Wpointer-to-i
nt-cast]
   54 |   count += private_count[(int)id].value;
      |                          ^

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>with-padding-locks
It takes 0.000000ms
Parallel: The number of 3's is 104735
Serial: The number of 3's is 104735

C:\Users\OWNER\Desktop\project\Parallel-Programming-for-Multi-Core-and-Cluster-Systems\lab 2>gcc -o with-padding-locks c
ount3s-pthread-with-padding-locks-only.c
count3s-pthread-with-padding-locks-only.c: In function 'count3s':
count3s-pthread-with-padding-locks-only.c:29:53: warning: cast to pointer from integer of different size [-Wint-to-point
er-cast]
   29 |       pthread_create(&tid[i], NULL, count3s_thread, (void*)i);
      |                                                     ^
count3s-pthread-with-padding-locks-only.c: In function 'count3s_thread':
```

5 Discussion

We notice after running the codes with different array sizes, that the bigger the array the lower the performance. Also, locks and padding make the results more correct. And, when the number of threads is increases the code runs faster.

6 Conclusion

Race conditions are hard to deal with in parallel programming