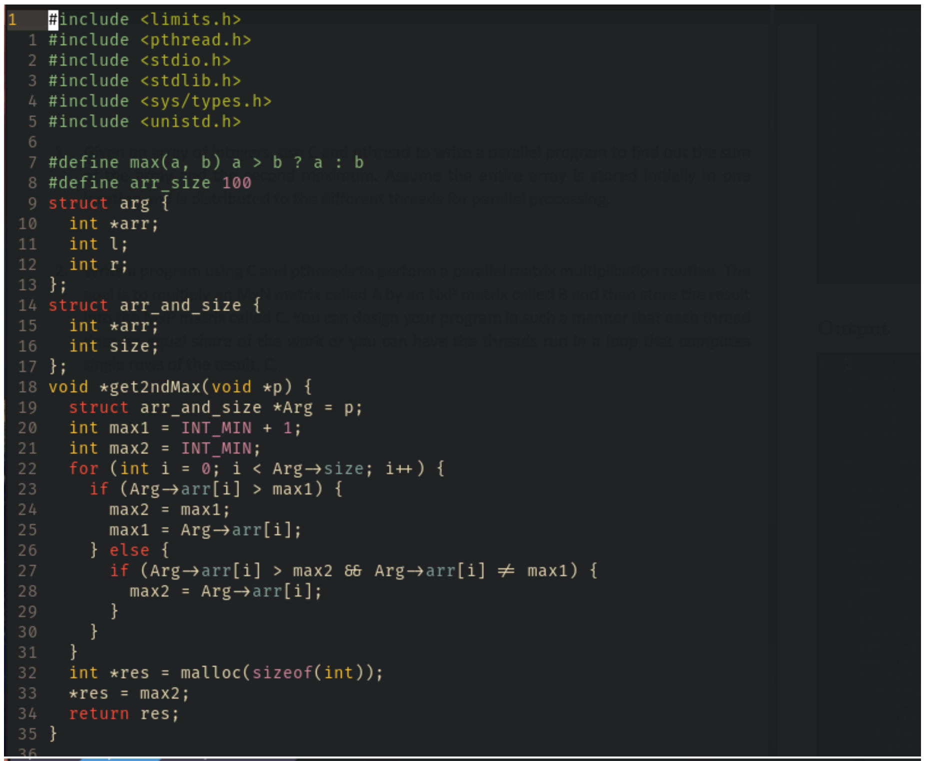
106119029 , OS Lab 7

Dipesh Kafle

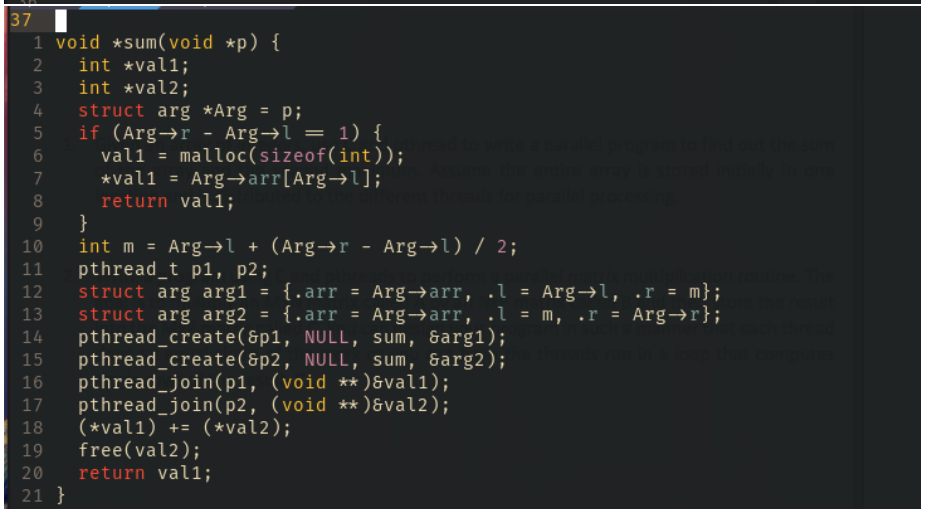
Question 1

- Given an array of integers, use C and pthread to write a parallel program to find out the sum of the array and the second maximum. Assume the entire array is stored initially in one location and is distributed to the different threads for parallel processing.

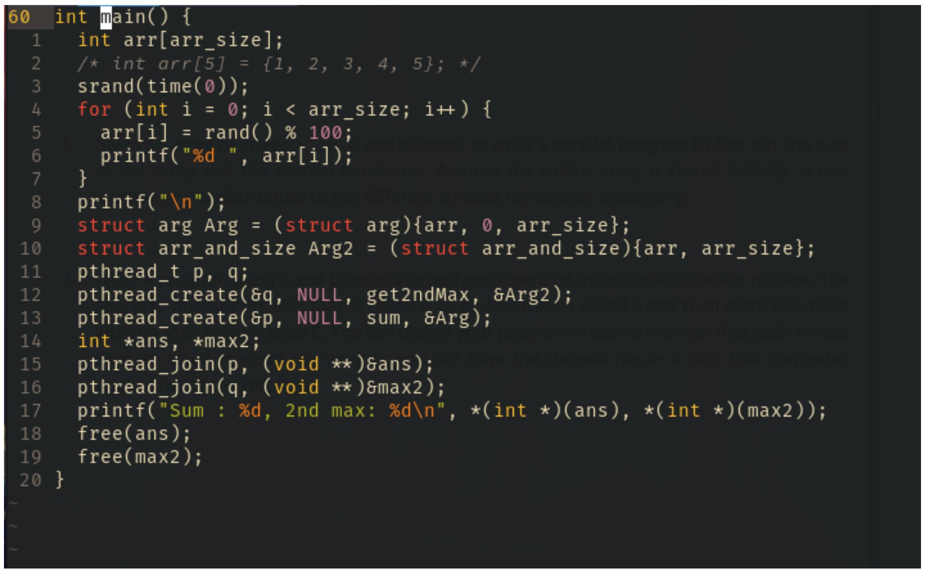
Code



```c
#include <limits.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

#define max(a, b) a > b ? a : b
#define arr_size 100
struct arg {
  int *arr;
  int l;
  int r;
};
struct arr_and_size {
  int *arr;
  int size;
};
void *get2ndMax(void *p) {
  struct arr_and_size *Arg = p;
  int max1 = INT_MIN + 1;
  int max2 = INT_MIN;
  for (int i = 0; i < Arg->size; i++) {
    if (Arg->arr[i] > max1) {
      max2 = max1;
      max1 = Arg->arr[i];
    } else {
      if (Arg->arr[i] > max2 && Arg->arr[i] != max1) {
        max2 = Arg->arr[i];
      }
    }
  }
  int *res = malloc(sizeof(int));
  *res = max2;
  return res;
}
```

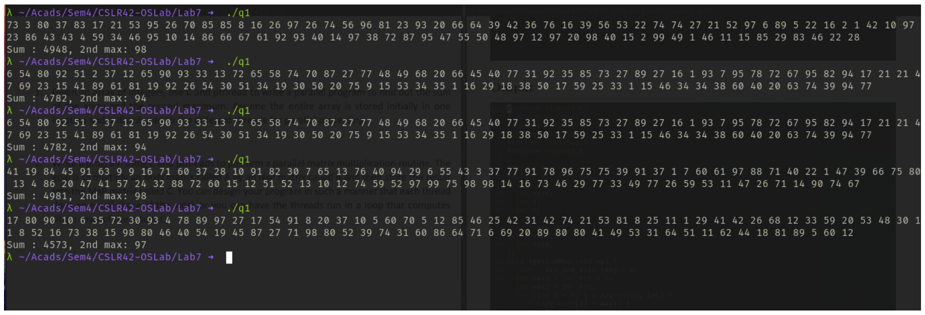


```c
void *sum(void *p) {
  int *val1;
  int *val2;
  struct arg *Arg = p;
  if (Arg->r - Arg->l == 1) {
    val1 = malloc(sizeof(int));
    *val1 = Arg->arr[Arg->l];
    return val1;
  }
  int m = Arg->l + (Arg->r - Arg->l) / 2;
  pthread_t p1, p2;
  struct arg arg1 = {.arr = Arg->arr, .l = Arg->l, .r = m};
  struct arg arg2 = {.arr = Arg->arr, .l = m, .r = Arg->r};
  pthread_create(&p1, NULL, sum, &arg1);
  pthread_create(&p2, NULL, sum, &arg2);
  pthread_join(p1, (void **)&val1);
  pthread_join(p2, (void **)&val2);
  (*val1) += (*val2);
  free(val2);
  return val1;
}
```



```c
int main() {
  int arr[arr_size];
  /* int arr[5] = {1, 2, 3, 4, 5}; */
  srand(time(0));
  for (int i = 0; i < arr_size; i++) {
    arr[i] = rand() % 100;
    printf("%d ", arr[i]);
  }
  printf("\n");
  struct arg Arg = (struct arg){arr, 0, arr_size};
  struct arr_and_size Arg2 = (struct arr_and_size){arr, arr_size};
  pthread_t p, q;
  pthread_create(&q, NULL, get2ndMax, &Arg2);
  pthread_create(&p, NULL, sum, &Arg);
  int *ans, *max2;
  pthread_join(p, (void **)&ans);
  pthread_join(q, (void **)&max2);
  printf("Sum : %d, 2nd max: %d\n", *(int *)(ans), *(int *)(max2));
  free(ans);
  free(max2);
}
```

Output



```
λ ~/Acads/Sem4/CSLR42-OSLab/Lab7 →  ./q1
73 3 80 37 83 17 21 53 95 26 70 85 85 8 16 26 97 26 74 56 96 81 23 93 20 66 64 39 42 36 76 16 39 56 53 22 74 74 27 21 52 97 6 89 5 22 16 2 1 42 10 97
23 86 43 43 4 59 34 46 95 10 14 86 66 67 61 92 93 40 14 97 38 72 87 95 47 55 50 48 97 12 97 20 98 40 15 2 99 49 1 46 11 15 85 29 83 46 22 28
Sum : 4948, 2nd max: 98
λ ~/Acads/Sem4/CSLR42-OSLab/Lab7 →  ./q1
6 54 80 92 51 2 37 12 65 90 93 33 13 72 65 58 74 70 87 27 77 48 49 68 20 66 45 40 77 31 92 35 85 73 27 89 27 16 1 93 7 95 78 72 67 95 82 94 17 21 21 4
7 69 23 15 41 89 61 81 19 92 26 54 30 51 34 19 30 50 20 75 9 15 53 34 35 1 16 29 18 38 50 17 59 25 33 1 15 46 34 34 38 60 40 20 63 74 39 94 77
Sum : 4782, 2nd max: 94
λ ~/Acads/Sem4/CSLR42-OSLab/Lab7 →  ./q1
6 54 80 92 51 2 37 12 65 90 93 33 13 72 65 58 74 70 87 27 77 48 49 68 20 66 45 40 77 31 92 35 85 73 27 89 27 16 1 93 7 95 78 72 67 95 82 94 17 21 21 4
7 69 23 15 41 89 61 81 19 92 26 54 30 51 34 19 30 50 20 75 9 15 53 34 35 1 16 29 18 38 50 17 59 25 33 1 15 46 34 34 38 60 40 20 63 74 39 94 77
Sum : 4782, 2nd max: 94
λ ~/Acads/Sem4/CSLR42-OSLab/Lab7 →  ./q1
41 19 84 45 91 63 9 9 16 71 60 37 28 10 91 82 30 7 65 13 76 40 94 29 6 55 43 3 37 77 91 78 96 75 75 39 91 37 1 7 60 61 97 88 71 40 22 1 47 39 66 75 80
 13 4 86 20 47 41 57 24 32 88 72 60 15 12 51 52 13 10 12 74 59 52 97 99 75 98 98 14 16 73 46 29 77 33 49 77 26 59 53 11 47 26 71 14 90 74 67
Sum : 4981, 2nd max: 98
λ ~/Acads/Sem4/CSLR42-OSLab/Lab7 →  ./q1
17 80 90 10 6 35 72 30 93 4 78 89 97 27 17 54 91 8 20 37 10 5 60 70 5 12 85 46 25 42 31 42 74 21 53 81 8 25 11 1 29 41 42 26 68 12 33 59 20 53 48 30 1
1 8 52 16 73 38 15 98 80 46 40 54 19 45 87 27 71 98 80 52 39 74 31 60 86 64 71 6 69 20 89 80 80 41 49 53 31 64 51 11 62 44 18 81 89 5 60 12
Sum : 4573, 2nd max: 97
λ ~/Acads/Sem4/CSLR42-OSLab/Lab7 →
```

Remark

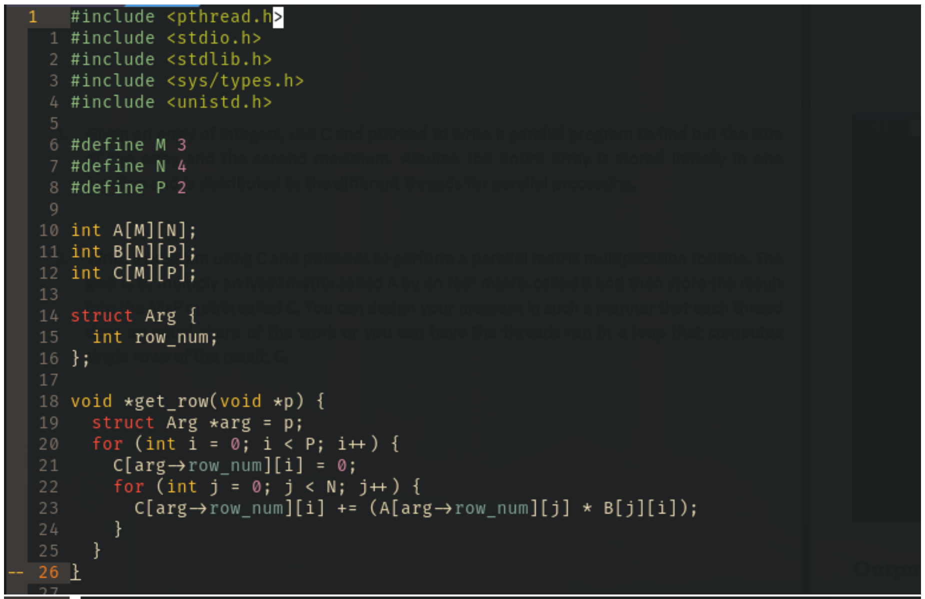
- I have first put 100 elements in an array (all random numbers from 0 to 99).
- I have spawned a thread which will calculate the 2nd max of the array and return it.
- I have spawned another thread which calculates the sum of the array elements using divide and conquer strategy. Every time we calculate the sum of the half of the array and sum it and return each time.

Question 2

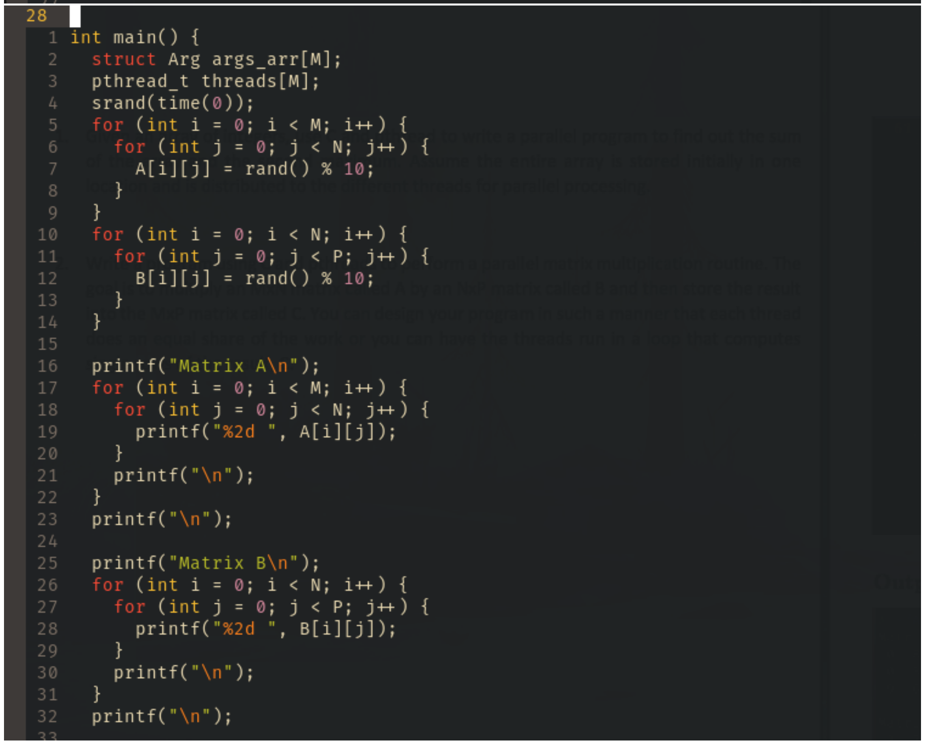
- Write a program using C and pthreads to perform a parallel matrix multiplication routine. The goal is to multiply an MxN matrix called A by an NxP matrix called B and then store the result into the MxP matrix called

C. You can design your program in such a manner that each thread does an equal share of the work or you can have the threads run in a loop that computes

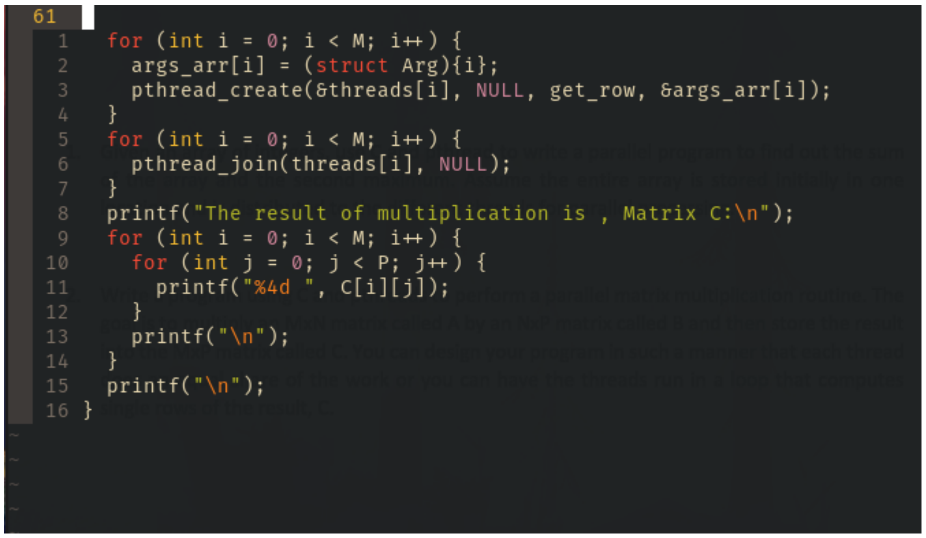
Code



```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

#define M 3
#define N 4
#define P 2

int A[M][N];
int B[N][P];
int C[M][P];

struct Arg {
  int row_num;
};

void *get_row(void *p) {
  struct Arg *arg = p;
  for (int i = 0; i < P; i++) {
    C[arg->row_num][i] = 0;
    for (int j = 0; j < N; j++) {
      C[arg->row_num][i] += (A[arg->row_num][j] * B[j][i]);
    }
  }
}
```

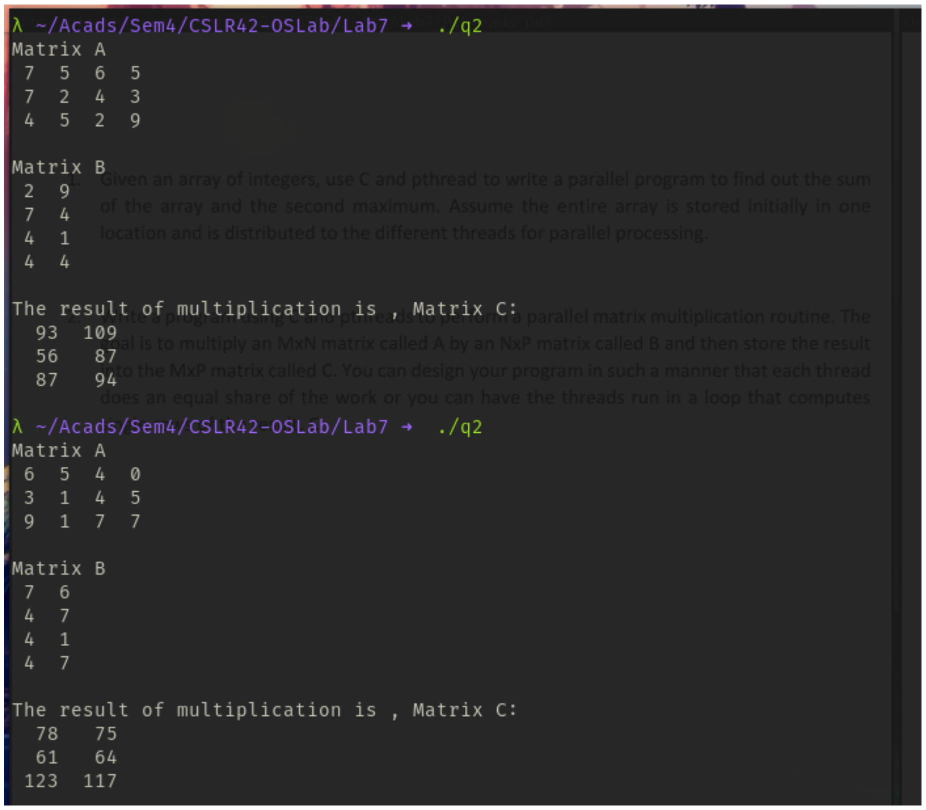


```c
int main() {
  struct Arg args_arr[M];
  pthread_t threads[M];
  srand(time(0));
  for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
      A[i][j] = rand() % 10;
    }
  }
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < P; j++) {
      B[i][j] = rand() % 10;
    }
  }

  printf("Matrix A\n");
  for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
      printf("%2d ", A[i][j]);
    }
    printf("\n");
  }
  printf("\n");

  printf("Matrix B\n");
  for (int i = 0; i < N; i++) {
    for (int j = 0; j < P; j++) {
      printf("%2d ", B[i][j]);
    }
    printf("\n");
  }
  printf("\n");
```



```c

  for (int i = 0; i < M; i++) {
    args_arr[i] = (struct Arg){i};
    pthread_create(&threads[i], NULL, get_row, &args_arr[i]);
  }
  for (int i = 0; i < M; i++) {
    pthread_join(threads[i], NULL);
  }
  printf("The result of multiplication is , Matrix C:\n");
  for (int i = 0; i < M; i++) {
    for (int j = 0; j < P; j++) {
      printf("%4d ", C[i][j]);
    }
    printf("\n");
  }
  printf("\n");
}
```

Output



```
λ ~/Acads/Sem4/CSLR42-OSLab/Lab7 →  ./q2
Matrix A
 7  5  6  5
 7  2  4  3
 4  5  2  9

Matrix B
 2  9
 7  4
 4  1
 4  4

The result of multiplication is , Matrix C:
  93  109
  56   87
  87   94
λ ~/Acads/Sem4/CSLR42-OSLab/Lab7 →  ./q2
Matrix A
 6  5  4  0
 3  1  4  5
 9  1  7  7

Matrix B
 7  6
 4  7
 4  1
 4  7

The result of multiplication is , Matrix C:
  78   75
  61   64
 123  117
```

Remark

- I have filled two matrices with random elements,both are actually defined in global scope along with the resultant matrix(which is also in global scope).
- I then assigned each thread the duty to calculate the row of the resultant matrix by multiplying the needed row and columns of input matrices ,then joined all the threads at the end and printed the resultant matrix.