PP Lab Record

Sagar Kumar

210968002

Batch B3

Week 1

1. Write a program in C to reverse the digits of the following integer array of size 9. Initialize the input array to the following values. Input array: 18, 523, 301, 1234, 2, 14, 108, 150, 1928 Output array: 81, 325, 103, 4321, 2, 41, 801, 51, 8291

   Code

   ```c
   #include <stdio.h>
   #include <math.h>

   int rev(int num)
   {
       int buf = 0;
       while (num > 0)
       {
           buf = (10 * buf) + num % 10;
           num = num / 10;
       }

       return buf;
       printf("%d\n", buf);
   }

   int main()
   {
       int arr[] = {18, 523, 301, 1234, 2, 14, 108, 150, 1928}, n = 9;

       for (int i = 0; i < n; i++)
       {
           arr[i] = rev(arr[i]);
           printf("%d ", arr[i]);
       }
       printf("\n");

       return 0;
   }
   ```

   Output

   ```
   81 325 103 4321 2 41 801 51 8291
   ```

2. Write a program in C to simulate the all the operations of a calculator. Given inputs A and B, find the output for A+B, A-B, A*B and A/B.

   Code

```c
#include <stdio.h>

int main()
{
    int a, b;

    printf("Enter num1, num2\n");
    scanf("%d%d", &a, &b);

    printf("Sum %d\n", a + b);
    printf("Difference %d\n", a - b);
    printf("Product %d\n", a * b);
    printf("Quotient %f\n", (double)a / b);

    return 0;
}
```

Output

```
Enter num1, num2
2 3
Sum 5
Difference -1
Product 6
Quotient 0.666667
```

3. Write a program in C to toggle the character of a given string. Example: suppose the string is “HeLLo”, then the output should be “hEllO”.

Code

```c
#include <stdio.h>

int main()
{
    char str[100];

    printf("Enter string\n");
    gets(str);

    char i = 0;
    while (str[i] != '\0')
    {
        str[i] = (int)str[i] < 91 ? (char)((int)str[i] + 32) : (char)((int)str[i] -
         32);
        i++;
    }
    printf("%s\n", str);

    return 0;
}
```

Output

```
Enter string
hElLo
HeLlO
```

4. Write a C program to read a word of length N and produce the pattern as shown

in the example. Example: Input: PCBD Output: PCCBBBDDDD

Code

```
#include <stdio.h>

int main()
{
    int n;
    printf("Enter num\n");
    scanf("%d", &n);

    int i = 0;
    char str[] = "PCBD";
    for (; i < n; i++)
    {
        int j = 0;
        for (; j <= i; j++)
        {
            printf("%c", str[i % 4]);
        }
    }
}
```

Output

```
Enter num
5
PCCBBBDDDDPPPPP
```

5. Write a C program to read two strings S1 and S2 of same length and produce the resultant string as shown below. S1: string S2: length Resultant String: slternigntgh

Code

```
#include <stdio.h>

int main()
{
    char str1[100], str2[100];

    printf("Enter 2 strings\n");

    gets(str1);
    gets(str2);

    char buf[100];

    int i = 0;
    int ctr = 0;
    while (str1[i] != '\0')
    {
        buf[ctr] = str1[i];
        ctr++;
        buf[ctr] = str2[i];
        i++;
        ctr++;
    }
    i = 0;
```

```c
    printf("%s\n", buf);

    return 0;
}
```

Output

```
Enter 2 strings
hello
world
hweolrllod
```

6. Write a C program to perform Matrix times vector product operation.

Code

```c
#include <stdio.h>

int main()
{
    int m[100][100], v[100][100], res[100][100];
    int m1, m2, v1, v2;

    printf("Enter mat dims\n");
    scanf("%d%d", &m1, &m2);
    printf("Enter vec dims\n");
    scanf("%d%d", &v1, &v2);

    if (m2 != v1)
    {
        printf("Can't multiply\n");
        return 0;
    }

    printf("Enter matrix\n");
    int i, j, k;
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            scanf("%d", &m[i][j]);
        }
    }

    printf("Enter vec\n");
    for (i = 0; i < v1; i++)
    {
        for (j = 0; j < v2; j++)
        {
            scanf("%d", &v[i][j]);
        }
    }

    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            res[i][j] = 0;
        }
    }
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < v2; j++)
        {
            for (k = 0; k < m2; k++)
            {
                res[i][j] += m[i][k] * v[k][j];
            }
        }
    }

    printf("Result\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < v2; j++)
        {
            printf("%d ", res[i][j]);
        }
        printf("\n");
    }

    printf("\n");

    return 0;
}
```

Output

```
Enter mat dims
2 2
Enter vec dims
2 1
Enter matrix
1 0 0 1
Enter vec
1 0
Result
1
0
```

7. Write a C program to read a matrix A of size 5x5. It produces a resultant matrix B of size 5x5. It sets all the principal diagonal elements of B matrix with 0. It replaces each row elements in the B matrix in the following manner. If the element is below the principal diagonal it replaces it with the maximum value of the row in the A matrix having the same row number of B. If the element is above the principal diagonal it replaces it with the minimum value of the row in the A matrix having the same row number of B.

Code

```
#include <stdio.h>

int main()
{
    int m[100][100], res[100][100];
    int max[100], min[100];
    int m1 = 5, m2 = 5;

    printf("Enter matrix\n");
    int i, j;
    for (i = 0; i < m1; i++)
    {
        int maxx = 0;
        int minn = 9999999;
        for (j = 0; j < m2; j++)
        {
            scanf("%d", &m[i][j]);
            if (m[i][j] > maxx)
            {
                maxx = m[i][j];
            }
            if (m[i][j] < minn)
            {
                minn = m[i][j];
            }
        }
        max[i] = maxx;
        min[i] = minn;
    }

    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == j)
            {
                res[i][j] = 0;
            }
            else if (i < j)
            {
                res[i][j] = min[i];
            }
            else
            {
                res[i][j] = max[i];
            }
        }
    }

    printf("Result\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            printf("%d ", res[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

Output

```
Enter matrix
1 2 3 4 5
5 4 3 2 4
10 3 13 14 15
11 2 11 33 44
1 12 5 4 6
Result
0 1 1 1 1
5 0 2 2 2
15 15 0 3 3
```

```
44 44 44 0 2
12 12 12 12 0
```

8. Write a C program that reads a matrix of size MxN and produce an output matrix B of same size such that it replaces all the non-border elements of A with its equivalent 1's complement and remaining elements same as matrix A. Also produce a matrix D as shown below.

   Code

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

int binary_to_int(char *str)
{
    int val = 0;
    int power = 1;
    int len = strlen(str);

    for (int i = 4; i >= 0; i--)
    {
        val += (str[i] - '0') * power;
        power *= 2;
    }

    return val;
}

int main()
{
    int m[100][100], res[100][100], ress[100][100];
    int m1, m2;

    printf("Enter mat dims\n");
    scanf("%d%d", &m1, &m2);

    printf("Enter matrix\n");
    int i, j, k;
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            scanf("%d", &m[i][j]);
        }
    }

    printf("B\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == 0 || i == m1 - 1 || j == 0 || j == m2 - 1)
            {
                printf("%d ", m[i][j]);
            }
            else
            {
                char *buf = calloc(sizeof(int) * 8 + 1, sizeof(int));
                itoa(m[i][j], buf, 2);

                for (int z = 0; z < sizeof(int) * 8; z++)
                {
                    buf[z] = (buf[z] == '0') ? '1' : '0';
                };

                printf("%d ", binary_to_int(buf));
            }
        }
        printf("\n");
    }

    printf("C\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == 0 || i == m1 - 1 || j == 0 || j == m2 - 1)
            {
                printf("%d ", m[i][j]);
            }
            else
            {
                char *buf = calloc(sizeof(int) * 2 + 1, sizeof(int));
                itoa(m[i][j], buf, 2);

                for (int z = 0; z < sizeof(int) * 8; z++)
                {
                    buf[z] = (buf[z] == '0') ? '1' : '0';
                };

                printf("%s ", buf);
            }
        }
        printf("\n");
    }

    return 0;
}
```

Output

```
Enter mat dims
4 4
Enter matrix
1 2 3 4
6 5 8 3
2 4 10 1
9 1 2 5
B
1 2 3 4
6 8 14 3
2 12 10 1
9 1 2 5
C
1 2 3 4
6 01000000000000000000000000000000 01110000000000000000000000000000 3
2 01100000000000000000000000000000 01010000000000000000000000000000 1
9 1 2 5
```

9. Write a C program that reads a character type matrix and integer type matrix B of size MxN. It produces and output string STR such that, every character of A is repeated r times (where r is the integer value in matrix B which is having the

same index as that of the character taken in A).

Code

```c
#include <stdio.h>

int main()
{
    int m[100][100];
    char c[100][100];
    char s[1000];
    int m1, m2;

    printf("Enter mat dims\n");
    scanf("%d%d", &m1, &m2);

    printf("Enter matrix\n");
    int i, j, k;
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            getchar();
            scanf("%c ", &c[i][j]);
            scanf("%d", &m[i][j]);
        }
    }

    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            for (k = 0; k < m[i][j]; k++)
            {
                printf("%c", c[i][j]);
            }
        }
    }

    return 0;
}
```

Output

```
Enter mat dims
4 2
Enter matrix
p 1
C 2
a 4
P 3
e 2
X 4
a 3
M 2
pCCaaaaPPPeeXXXXaaaMM
```

Week 2

1. Write a program in C to reverse the digits of the following integer array of size 9. Initialize the input array to the following values. Input array: 18, 523, 301, 1234,

2, 14, 108, 150, 1928 Output array: 81, 325, 103, 4321, 2, 41, 801, 51, 8291

Code

```c
#include <stdio.h>
#include <math.h>
#include <omp.h>

int rev(int num)
{
    int buf = 0;
    while (num > 0)
    {
        buf = (10 * buf) + num % 10;
        num = num / 10;
    }

    return buf;
    printf("%d\n", buf);
}

int main()
{
    int arr1[] = {18, 523, 301, 1234, 2, 14, 108, 150, 1928};
    int arr2[] = {18, 523, 301, 1234, 2, 14, 108, 150, 1928};
    int n = 9;

    // Sequential
    double begin1 = omp_get_wtime();

    for (int i = 0; i < n; i++)
    {
        arr1[i] = rev(arr1[i]);
        printf("%d ", arr1[i]);
    }
    printf("\n");

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    double begin2 = omp_get_wtime();
    int num_threads;

#pragma omp parallel
    {
        num_threads = omp_get_num_threads();

#pragma omp for
        for (int i = 0; i < n; i++)
        {
            arr2[i] = rev(arr2[i]);
            printf("%d ", arr2[i]);
        }
    }
    printf("\n");

    double end2 = omp_get_wtime();
    double par = end2 - begin2;
    printf("Parallel time: %f\n", par / num_threads);

    double speedup = seq / par;
    printf("Speedup: %f\n", speedup);
    printf("Efficiency: %f\n", speedup / num_threads);

    return 0;
}
```

Output

```
81 325 103 4321 2 41 801 51 8291
Sequential time: 0.000000
103 801 325 81 41 4321 8291 51 2
Parallel time: 0.000125
Speedup: 0.000000
Efficiency: 0.000000
```

2. Write a program in C to simulate the all the operations of a calculator. Given inputs A and B, find the output for A+B, A-B, A*B and A/B.

Code

```
#include <stdio.h>
#include <math.h>
#include <omp.h>

int main()
{
    int a, b;

    printf("Enter num1, num2\n");
    scanf("%d%d", &a, &b);

    // Sequential
    double begin1 = omp_get_wtime();

    printf("Sum %d\n", a + b);
    printf("Difference %d\n", a - b);
    printf("Product %d\n", a * b);
    printf("Quotient %d\n", a / b);

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    double begin2 = omp_get_wtime();
    int num_threads;

#pragma omp parallel num_threads(4)
    {
        num_threads = omp_get_num_threads();

#pragma omp sections
        {

#pragma omp section
            {
                printf("Sum %d\n", a + b);
            }
#pragma omp section
            {
                printf("Difference %d\n", a - b);
            }
#pragma omp section
            {
                printf("Product %d\n", a * b);
            }
#pragma omp section
            {
                printf("Quotient %d\n", a / b);
            }
        }
    }
    double end2 = omp_get_wtime();
    double par = end2 - begin2;
    printf("Parallel time: %f\n", par / num_threads);

    double speedup = seq / par;
    printf("Speedup: %f\n", speedup);
    printf("Efficiency: %f\n", speedup / num_threads);

    return 0;
}
```

Output

```
Enter num1, num2
2 3
Sum 5
Difference -1
Product 6
Quotient 0
Sequential time: 0.000000
Sum 5
Difference -1
Product 6
Quotient 0
Parallel time: 0.000250
Speedup: 0.000000
Efficiency: 0.000000
```

3. Write a program in C to toggle the character of a given string. Example: suppose the string is “HeLLo”, then the output should be “hEllO”.

Code

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
#include <string.h>

int main()
{
    char str1[100], str2[100];

    printf("Enter string\n");
    gets(str1);
    int j = 0;
    for (j = 0; str1[j] != '\0'; j++)
    {
        str2[j] = str1[j];
    }
    // Sequential
    double begin1 = omp_get_wtime();

    char i = 0;
    while (str1[i] != '\0')
    {
        str1[i] = (int)str1[i] < 91 ? (char)((int)str1[i] + 32) : (char)((int)str1[i]
        - 32);
        i++;
    }
    printf("%s\n", str1);

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    int len = strlen(str2);

    double begin2 = omp_get_wtime();
    int num_threads;

#pragma omp parallel
    {
        num_threads = omp_get_num_threads();

#pragma omp for
        for (int j = 0; j < len; j++)
        {
            str2[j] = (int)str2[j] < 91 ? (char)((int)str2[j] + 32) : (char)
        ((int)str2[j] - 32);
        }
    }
    printf("%s\n", str2);

    double end2 = omp_get_wtime();
    double par = end2 - begin2;
    printf("Parallel time: %f\n", par / num_threads);

    double speedup = seq / par;
    printf("Speedup: %f\n", speedup);
    printf("Efficiency: %f\n", speedup / num_threads);

    return 0;
}
```

Output

```
Enter string
hEllOo
HeLLoO
Sequential time: 0.000000
HeLLoO
Parallel time: 0.000125
Speedup: 0.000000
Efficiency: 0.000000
```

4. Write a C program to read a word of length N and produce the pattern as shown in the example. Example: Input: PCBD Output: PCCBBBDDDD

Code

```c
#include <stdio.h>
#include <omp.h>

int main()
{
    int n;
    printf("Enter num\n");
    scanf("%d", &n);

    char str[] = "PCBD";

    // Sequential
    double begin1 = omp_get_wtime();

    for (int i = 0; i < n; i++)
    {

        for (int j = 0; j <= i; j++)
        {
            printf("%c", str[i % 4]);
        }
    }
    printf("\n");

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    double begin2 = omp_get_wtime();
    double num_threads = omp_get_num_threads();

#pragma omp parallel for
    for (int i = 0; i < n; i++)
    {

        for (int j = 0; j <= i; j++)
        {
            printf("%c", str[i % 4]);
        }
    }
    printf("\n");

    double end2 = omp_get_wtime();
    double par = end2 - begin2;
    printf("Parallel time: %f\n", par);

    double speedup = seq / par;
    printf("Speedup: %f\n", speedup);
    printf("Efficiency: %f\n", speedup / num_threads);
}
```

Output

```
Enter num
5
PCCBBBDDDDPPPPP
Sequential time: 0.002000
CCPBBBDDDDPPPPP
Parallel time: 0.002000
```

```
Speedup: 1.000119
Efficiency: 1.000119
```

5. Write a C program to read two strings S1 and S2 of same length and produce the resultant string as shown below. S1: string S2: length Resultant String: slternigntgh

Code

```c
#include <stdio.h>
#include <omp.h>
#include <string.h>

int main()
{
    char str1[100], str2[100];

    printf("Enter 2 strings\n");

    gets(str1);
    gets(str2);

    // Sequential
    char buf1[100];
    int i = 0;
    int ctr = 0;

    double begin1 = omp_get_wtime();
    while (str1[i] != '\0')
    {
        buf1[ctr] = str1[i];
        ctr++;
        buf1[ctr] = str2[i];
        i++;
        ctr++;
    }
    i = 0;
    printf("%s\n", buf1);

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    char buf2[100];
    int ctrr = 0;

    double begin2 = omp_get_wtime();
    double num_threads = omp_get_num_threads();
    int len = strlen(str1);

#pragma omp parallel for
    for (int i = 0; i < len; i++)
    {
        buf2[ctrr] = str1[i];
        ctrr++;
        buf2[ctrr] = str2[i];
        i++;
        ctrr++;
    }
    printf("%s\n", buf2);
    double end2 = omp_get_wtime();
    double par = end2 - begin2;
    printf("Parallel time: %f\n", par);

    double speedup = seq / par;
    printf("Speedup: %f\n", speedup);
    printf("Efficiency: %f\n", speedup / num_threads);

    return 0;
}
```

Output

```
Enter 2 strings
Hello
World
HWeolrllloda
Sequential time: 0.000000
eoodllHWlr☺
Parallel time: 0.002000
Speedup: 0.000000
Efficiency: 0.000000
```

6. Write a C program to perform Matrix times vector product operation.

Code

```
#include <stdio.h>
#include <omp.h>

int main()
{
    int m[100][100], v[100][100], res1[100][100], res2[100][100];
    int m1, m2, v1, v2;

    printf("Enter mat dims\n");
    scanf("%d%d", &m1, &m2);
    printf("Enter vec dims\n");
    scanf("%d%d", &v1, &v2);

    if (m2 != v1)
    {
        printf("Can't multiply\n");
        return 0;
    }

    printf("Enter matrix\n");
    int i, j, k;
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            scanf("%d", &m[i][j]);
        }
    }

    printf("Enter vec\n");
    for (i = 0; i < v1; i++)
    {
        for (j = 0; j < v2; j++)
```

```c
            {
                scanf("%d", &v[i][j]);
            }
        }

        for (i = 0; i < m1; i++)
        {
            for (j = 0; j < m2; j++)
            {
                res1[i][j] = 0;
                res2[i][j] = 0;
            }
        }

        // Sequential
        double begin1 = omp_get_wtime();
        for (i = 0; i < m1; i++)
        {
            for (j = 0; j < v2; j++)
            {
                for (k = 0; k < m2; k++)
                {
                    res1[i][j] += m[i][k] * v[k][j];
                }
            }
        }

        printf("Seq Result\n");
        for (i = 0; i < m1; i++)
        {
            for (j = 0; j < v2; j++)
            {
                printf("%d ", res1[i][j]);
            }
            printf("\n");
        }
        double end1 = omp_get_wtime();
        double seq = end1 - begin1;
        printf("Sequential time: %f\n", seq);

        // Parallel
        double begin2 = omp_get_wtime();
        double num_threads = omp_get_num_threads();

    #pragma omp parallel for collapse(3)
        for (i = 0; i < m1; i++)
        {
            for (j = 0; j < v2; j++)
            {
                for (k = 0; k < m2; k++)
                {
                    res2[i][j] += m[i][k] * v[k][j];
                }
            }
        }

        printf("Parallel Result\n");
        for (i = 0; i < m1; i++)
        {
            for (j = 0; j < v2; j++)
            {
                printf("%d ", res2[i][j]);
            }
            printf("\n");
        }

        printf("\n");
        double end2 = omp_get_wtime();
        double par = end2 - begin2;
        printf("Parallel time: %f\n", par);

        double speedup = seq / par;
        printf("Speedup: %f\n", speedup);
        printf("Efficiency: %f\n", speedup / num_threads);

        return 0;
    }
```

Output

```
Enter mat dims
2 2
Enter vec dims
2 1
Enter matrix
1 0 0 1
Enter vec
1 0
Seq Result
1
0
Sequential time: 0.002000
Parallel Result
1
0

Parallel time: 0.001000
Speedup: 2.000238
Efficiency: 2.000238
```

7. Write a C program to read a matrix A of size 5x5. It produces a resultant matrix B of size 5x5. It sets all the principal diagonal elements of B matrix with 0. It replaces each row elements in the B matrix in the following manner. If the element is below the principal diagonal it replaces it with the maximum value of the row in the A matrix having the same row number of B. If the element is above the principal diagonal it replaces it with the minimum value of the row in the A matrix having the same row number of B.

Code

```
#include <stdio.h>
#include <omp.h>

int main()
{
    int m[100][100], res1[100][100], res2[100][100];
    int max[100], min[100];
    int m1 = 5, m2 = 5;

    printf("Enter matrix\n");
    int i, j;
    for (i = 0; i < m1; i++)
    {
        int maxx = 0;
        int minn = 9999999;
```

```c
        for (j = 0; j < m2; j++)
        {
            scanf("%d", &m[i][j]);
            if (m[i][j] > maxx)
            {
                maxx = m[i][j];
            }
            if (m[i][j] < minn)
            {
                minn = m[i][j];
            }
        }
        max[i] = maxx;
        min[i] = minn;
    }

    // Sequential
    double begin1 = omp_get_wtime();
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == j)
            {
                res1[i][j] = 0;
            }
            else if (i < j)
            {
                res1[i][j] = min[i];
            }
            else
            {
                res1[i][j] = max[i];
            }
        }
    }

    printf("Seq Result\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            printf("%d ", res1[i][j]);
        }
        printf("\n");
    }

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    double begin2 = omp_get_wtime();
    double num_threads = omp_get_num_threads();

#pragma omp parallel for collapse(2)
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == j)
            {
                res2[i][j] = 0;
            }
            else if (i < j)
            {
                res2[i][j] = min[i];
            }
            else
            {
                res2[i][j] = max[i];
            }
        }
    }

    printf("Parallel Result\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            printf("%d ", res2[i][j]);
        }
        printf("\n");
    }

    printf("\n");
    double end2 = omp_get_wtime();
    double par = end2 - begin2;
    printf("Parallel time: %f\n", par);

    double speedup = seq / par;
    printf("Speedup: %f\n", speedup);
    printf("Efficiency: %f\n", speedup / num_threads);

    return 0;
}
```

Output

```
Enter matrix
1 2 3 4 5
5 4 3 2 4
10 3 13 14 15
11 2 11 33 44
1 12 5 4 6
Seq Result
0 1 1 1 1
5 0 2 2 2
15 15 0 3 3
44 44 44 0 2
12 12 12 12 0
Sequential time: 0.002000
Parallel Result
0 1 1 1 1
5 0 2 2 2
15 15 0 3 3
44 44 44 0 2
12 12 12 12 0

Parallel time: 0.004000
Speedup: 0.500000
Efficiency: 0.500000
```

8. Write a C program that reads a matrix of size MxN and produce an output matrix B of same size such that it replaces all the non-border elements of A with its

equivalent 1’s complement and remaining elements same as matrix A. Also produce a matrix D as shown below.

Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int binary_to_int(char *str)
{
    int val = 0;
    int power = 1;
    int len = strlen(str);

    for (int i = 4; i >= 0; i--)
    {
        val += (str[i] - '0') * power;
        power *= 2;
    }

    return val;
}

int main()
{
    int m[100][100], res[100][100], ress[100][100];
    int m1, m2;

    printf("Enter mat dims\n");
    scanf("%d%d", &m1, &m2);

    printf("Enter matrix\n");
    int i, j, k;
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            scanf("%d", &m[i][j]);
        }
    }

    // Sequential
    double begin1 = omp_get_wtime();

    printf("Seq B\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == 0 || i == m1 - 1 || j == 0 || j == m2 - 1)
            {
                printf("%d ", m[i][j]);
            }
            else
            {
                char *buf = calloc(sizeof(int) * 8 + 1, sizeof(int));
                itoa(m[i][j], buf, 2);

                for (int z = 0; z < sizeof(int) * 8; z++)
                {
                    buf[z] = (buf[z] == '0') ? '1' : '0';
                };
                printf("%d ", binary_to_int(buf));
            }
        }
        printf("\n");
    }

    printf("Seq C\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == 0 || i == m1 - 1 || j == 0 || j == m2 - 1)
            {
                printf("%d ", m[i][j]);
            }
            else
            {
                char *buf = calloc(sizeof(int) * 2 + 1, sizeof(int));
                itoa(m[i][j], buf, 2);

                for (int z = 0; z < sizeof(int) * 8; z++)
                {
                    buf[z] = (buf[z] == '0') ? '1' : '0';
                };

                printf("%s ", buf);
            }
        }
        printf("\n");
    }

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    double begin2 = omp_get_wtime();
    double num_threads = omp_get_num_threads();

    printf("Parallel B\n");
#pragma omp parallel for collapse(2)
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == 0 || i == m1 - 1 || j == 0 || j == m2 - 1)
            {
                printf("%d ", m[i][j]);
            }
            else
            {
                char *buf = calloc(sizeof(int) * 8 + 1, sizeof(int));
                itoa(m[i][j], buf, 2);

                for (int z = 0; z < sizeof(int) * 8; z++)
                {
                    buf[z] = (buf[z] == '0') ? '1' : '0';
                };

                printf("%d ", binary_to_int(buf));
            }
        }
    }
```

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

    printf("Parallel C\n");
#pragma omp parallel for collapse(2)
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == 0 || i == m1 - 1 || j == 0 || j == m2 - 1)
            {
                printf("%d ", m[i][j]);
            }
            else
            {
                char *buf = calloc(sizeof(int) * 2 + 1, sizeof(int));
                itoa(m[i][j], buf, 2);

                for (int z = 0; z < sizeof(int) * 8; z++)
                {
                    buf[z] = (buf[z] == '0') ? '1' : '0';
                };

                printf("%s ", buf);
            }
        }
    }
    printf("\n");

    double end2 = omp_get_wtime();
    double par = end2 - begin2;
    printf("Parallel time: %f\n", par);

    double speedup = seq / par;
    printf("Speedup: %f\n", speedup);
    printf("Efficiency: %f\n", speedup / num_threads);

    return 0;
}
```

Output

```
Enter mat dims
4 4
Enter matrix
1 2 3 4
6 5 8 3
2 4 10 1
9 1 2 5
Seq B
1 2 3 4
6 8 14 3
2 12 10 1
9 1 2 5
Seq C
1 2 3 4
6 01000000000000000000000000000000 01110000000000000000000000000000 3
2 01100000000000000000000000000000 01010000000000000000000000000000 1
9 1 2 5
Sequential time: 5.000000
Parallel B
2 6 4 3 14 3 1 2 9 1 2 1 12 8 10 5
Parallel C
2 3 6 4 1 01000000000000000000000000000000 01110000000000000000000000000000 1 2 3 2 5
```

```
9 01010000000000000000000000000000 01100000000000000000000000000000 1
Parallel time: 5.000000
Speedup: 1.000000
Efficiency: 1.000000
```

9. Write a C program that reads a character type matrix and integer type matrix B of size MxN. It produces and output string STR such that, every character of A is repeated r times (where r is the integer value in matrix B which is having the same index as that of the character taken in A).

   Code

```c
#include <stdio.h>

int main()
{
    int m[100][100];
    char c[100][100];
    char s[1000];
    int m1, m2;

    printf("Enter mat dims\n");
    scanf("%d%d", &m1, &m2);

    printf("Enter matrix\n");
    int i, j, k;
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            getchar();
            scanf("%c ", &c[i][j]);
            scanf("%d", &m[i][j]);
        }
    }

    // Sequential
    double begin1 = omp_get_wtime();

    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            for (k = 0; k < m[i][j]; k++)
            {
                printf("%c", c[i][j]);
            }
        }
    }
    printf("\n");

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    double begin2 = omp_get_wtime();
    double num_threads = omp_get_num_threads();

#pragma omp parallel for collapse(2)
    for (i = 0; i < m1; i++)
```

```
        {
            for (j = 0; j < m2; j++)
            {
                for (k = 0; k < m[i][j]; k++)
                {
                    printf("%c", c[i][j]);
                }
            }
        }
        printf("\n");

        double end2 = omp_get_wtime();
        double par = end2 - begin2;
        printf("Parallel time: %f\n", par);

        double speedup = seq / par;
        printf("Speedup: %f\n", speedup);
        printf("Efficiency: %f\n", speedup / num_threads);

        return 0;
    }
```

Output

```
Enter mat dims
2 4
Enter matrix
p 1
C 2
a 4
P 3
e 2
X 4
a 3
M 2
pCCaaaaPPPeeXXXXaaaMM
Sequential time: 2.000000
eeMapaPXC
Parallel time: 1.000000
Speedup: 2.000000
Efficiency: 2.000000
```

Week 3

1. Write an OpenMP program to implement Matrix multiplication.

a. Analyze the speedup and efficiency of the parallelized code.

b. Vary the size of your matrices from 200, 400, 600, 800 and 1000 and measure the runtime with one thread and four threads.

c. For each matrix size, change the number of threads from 2,4,6 and 8 and plot the speedup versus the number of threads. Compute the efficiency.

Code

```
#include <stdio.h>
#include <omp.h>
#include <malloc.h>
#include <stdlib.h>
```

```c
int main()
{
    int sizes[] = {200, 400, 600, 800, 1000};
    int threads[] = {2, 4, 6, 8};
    int times[5];

    // Sequential
    for (int n = 0; n < 5; n++)
    {
        int size = sizes[n];

        int *mat1 = (int *)calloc(size * size, sizeof(int));
        int *mat2 = (int *)calloc(size * size, sizeof(int));
        int *res = (int *)calloc(size * size, sizeof(int));

        for (int j = 0; j < size * size; j++)
        {
            int r = rand() % 10 + 1;
            mat1[j] = r;
            r = rand() % 10 + 1;
            mat2[j] = r;
            res[j] = 0;
        }

        double begin = omp_get_wtime();
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                for (int k = 0; k < n; k++)
                {
                    res[i * n + j] += mat1[i * n + k] * mat2[k * n + j];
                }
            }
        }
        double end = omp_get_wtime();
        times[n] = end - begin;
    }

    // Parallel
    for (int n = 0; n < 5; n++)
    {
        int size = sizes[n];

        int *mat1 = (int *)calloc(size * size, sizeof(int));
        int *mat2 = (int *)calloc(size * size, sizeof(int));
        int *res = (int *)calloc(size * size, sizeof(int));

        for (int j = 0; j < size * size; j++)
        {
            int r = rand() % 10 + 1;
            mat1[j] = r;
            r = rand() % 10 + 1;
            mat2[j] = r;
            res[j] = 0;
        }

        for (int t = 0; t < 4; t++)
        {
            double start = omp_get_wtime();
            omp_set_num_threads(threads[t]);

#pragma omp parallel for collapse(3)
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    for (int k = 0; k < n; k++)
                    {
                        res[i * n + j] += mat1[i * n + k] * mat2[k * n + j];
                    }
                }
            }

            double end = omp_get_wtime();
            double time = end - start;
            double speedup = times[n] / time;
            double eff = speedup / (int)omp_get_num_threads();

            printf("Size: %d | Thread: %d | Time: %f | Speedup: %f | Efficiency: 
        %f\n", size, threads[t], time, speedup, eff);
        }
    }

    return 0;
}
```

Output

```
Size: 200 | Thread: 2 | Time: 0.001000 | Speedup: 0.000000 | Efficiency: 0.000000
Size: 200 | Thread: 4 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 200 | Thread: 6 | Time: 0.001000 | Speedup: 0.000000 | Efficiency: 0.000000
Size: 200 | Thread: 8 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 400 | Thread: 2 | Time: 0.001000 | Speedup: 0.000000 | Efficiency: 0.000000
Size: 400 | Thread: 4 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 400 | Thread: 6 | Time: 0.001000 | Speedup: 0.000000 | Efficiency: 0.000000
Size: 400 | Thread: 8 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 600 | Thread: 2 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 600 | Thread: 4 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 600 | Thread: 6 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 600 | Thread: 8 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 800 | Thread: 2 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 800 | Thread: 4 | Time: 0.001000 | Speedup: 0.000000 | Efficiency: 0.000000
Size: 800 | Thread: 6 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 800 | Thread: 8 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 1000 | Thread: 2 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 1000 | Thread: 4 | Time: 0.001000 | Speedup: 0.000000 | Efficiency: 0.000000
Size: 1000 | Thread: 6 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
Size: 1000 | Thread: 8 | Time: 0.000000 | Speedup: -1.#IND00 | Efficiency: -1.#IND00
```

2. Write an OpenMP program to perform Matrix times vector multiplication. Vary the matrix and vector size and analyze the speedup and efficiency of the parallelized code.

Code

```
#include <stdio.h>
#include <omp.h>
#include <malloc.h>
#include <stdlib.h>

int main()
{
    int m[300][300], v[300][300], res1[300][300], res2[300][300];
    int sizes[3] = {100, 200, 300};
    for (int size = 0; size < 3; size++)
    {
        int m1 = sizes[size];
        int m2 = m1;
        int v2 = m1 / 10;

        printf("Size: %d\n", sizes[size]);
        for (int i = 0; i < m1; i++)
        {
            for (int j = 0; j < m2; j++)
            {

                res1[i][j] = 0;
                res2[i][j] = 0;
                m[i][j] = rand() % 10 + 1;
                v[i][j] = rand() % 10 + 1;
            }
        }

        // Sequential
        double begin1 = omp_get_wtime();
        for (int i = 0; i < m1; i++)
        {
            for (int j = 0; j < v2; j++)
            {
                for (int k = 0; k < m2; k++)
                {
                    res1[i][j] += m[i][k] * v[k][j];
                }
            }
        }

        double end1 = omp_get_wtime();
        double seq = end1 - begin1;
        printf("Sequential time: %f\n", seq);

        // Parallel
        double begin2 = omp_get_wtime();
        double num_threads = omp_get_num_threads();

#pragma omp parallel for collapse(3)
        for (int i = 0; i < m1; i++)
        {
            for (int j = 0; j < v2; j++)
            {
                for (int k = 0; k < m2; k++)
                {
                    res2[i][j] += m[i][k] * v[k][j];
                }
            }
        }

        double end2 = omp_get_wtime();
        double par = end2 - begin2;
        printf("Parallel time: %f\n", par);

        double speedup = seq / par;
        printf("Speedup: %f\n", speedup);
        printf("Efficiency: %f\n", speedup / num_threads);

        printf("\n\n");
    }
    return 0;
}
```

Output

```
Size: 100
Sequential time: 0.001000
Parallel time: 0.001000
Speedup: 0.999762
Efficiency: 0.999762


Size: 200
Sequential time: 0.002000
Parallel time: 0.001000
Speedup: 2.000238
Efficiency: 2.000238


Size: 300
Sequential time: 0.008000
Parallel time: 0.002000
Speedup: 4.000358
Efficiency: 4.000358
```

3. Write an OpenMp program to read a matrix A of size 5x5. It produces a resultant matrix B of size 5x5. It sets all the principal diagonal elements of B matrix with 0. It replaces each row elements in the B matrix in the following manner. If the element is below the principal diagonal it replaces it with the maximum value of the row in the A matrix having the same row number of B. If the element is above the principal diagonal it replaces it with the minimum value of the row in the A matrix having the same row number of B. Analyze the speedup and efficiency of the parallelized code.

Code

```
#include <stdio.h>
#include <omp.h>

int main()
{
    int m[100][100], res1[100][100], res2[100][100];
    int max[100], min[100];
    int m1 = 5, m2 = 5;

    printf("Enter matrix\n");
    int i, j;
    for (i = 0; i < m1; i++)
    {
        int maxx = 0;
        int minn = 9999999;
        for (j = 0; j < m2; j++)
        {
            scanf("%d", &m[i][j]);
            if (m[i][j] > maxx)
            {
                maxx = m[i][j];
            }
            if (m[i][j] < minn)
            {
```

```c
                minn = m[i][j];
            }
        }
        max[i] = maxx;
        min[i] = minn;
    }

    // Sequential
    double begin1 = omp_get_wtime();
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == j)
            {
                res1[i][j] = 0;
            }
            else if (i < j)
            {
                res1[i][j] = min[i];
            }
            else
            {
                res1[i][j] = max[i];
            }
        }
    }

    printf("Seq Result\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            printf("%d ", res1[i][j]);
        }
        printf("\n");
    }

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    double begin2 = omp_get_wtime();
    double num_threads = omp_get_num_threads();

#pragma omp parallel for collapse(2)
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == j)
            {
                res2[i][j] = 0;
            }
            else if (i < j)
            {
                res2[i][j] = min[i];
            }
            else
            {
                res2[i][j] = max[i];
            }
        }
    }

    printf("Parallel Result\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            printf("%d ", res2[i][j]);
        }
        printf("\n");
    }

    printf("\n");
    double end2 = omp_get_wtime();
    double par = end2 - begin2;
    printf("Parallel time: %f\n", par);

    double speedup = seq / par;
    printf("Speedup: %f\n", speedup);
    printf("Efficiency: %f\n", speedup / num_threads);

    return 0;
}
```

Output

```
Enter matrix
1 2 3 4 5
5 4 3 2 4
10 3 13 14 15
11 2 11 33 44
1 12 5 4 6
Seq Result
0 1 1 1 1
5 0 2 2 2
15 15 0 3 3
44 44 44 0 2
12 12 12 12 0
Sequential time: 0.002000
Parallel Result
0 1 1 1 1
5 0 2 2 2
15 15 0 3 3
44 44 44 0 2
12 12 12 12 0

Parallel time: 0.005000
Speedup: 0.400029
Efficiency: 0.400029
```

4. Write a parallel program using OpenMP that reads a matrix of size MxN and produce an output matrix B of same size such that it replaces all the non-border elements of A with its equivalent 1’s complement and remaining elements same as matrix A. Also produce a matrix D as shown below.

Code

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

```c
int binary_to_int(char *str)
{
    int val = 0;
    int power = 1;
    int len = strlen(str);

    for (int i = 4; i >= 0; i--)
    {
        val += (str[i] - '0') * power;
        power *= 2;
    }

    return val;
}

int main()
{
    int m[100][100], res[100][100], ress[100][100];
    int m1, m2;

    printf("Enter mat dims\n");
    scanf("%d%d", &m1, &m2);

    printf("Enter matrix\n");
    int i, j, k;
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            scanf("%d", &m[i][j]);
        }
    }

    // Sequential
    double begin1 = omp_get_wtime();

    printf("Seq B\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == 0 || i == m1 - 1 || j == 0 || j == m2 - 1)
            {
                printf("%d ", m[i][j]);
            }
            else
            {
                char *buf = calloc(sizeof(int) * 8 + 1, sizeof(int));
                itoa(m[i][j], buf, 2);

                for (int z = 0; z < sizeof(int) * 8; z++)
                {
                    buf[z] = (buf[z] == '0') ? '1' : '0';
                };

                printf("%d ", binary_to_int(buf));
            }
        }
        printf("\n");
    }

    printf("Seq C\n");
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == 0 || i == m1 - 1 || j == 0 || j == m2 - 1)
            {
                printf("%d ", m[i][j]);
            }
            else
            {
                char *buf = calloc(sizeof(int) * 2 + 1, sizeof(int));
                itoa(m[i][j], buf, 2);

                for (int z = 0; z < sizeof(int) * 8; z++)
                {
                    buf[z] = (buf[z] == '0') ? '1' : '0';
                };

                printf("%s ", buf);
            }
        }
        printf("\n");
    }

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    double begin2 = omp_get_wtime();
    double num_threads = omp_get_num_threads();

    printf("Parallel B\n");
#pragma omp parallel for collapse(2)
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == 0 || i == m1 - 1 || j == 0 || j == m2 - 1)
            {
                printf("%d ", m[i][j]);
            }
            else
            {
                char *buf = calloc(sizeof(int) * 8 + 1, sizeof(int));
                itoa(m[i][j], buf, 2);

                for (int z = 0; z < sizeof(int) * 8; z++)
                {
                    buf[z] = (buf[z] == '0') ? '1' : '0';
                };

                printf("%d ", binary_to_int(buf));
            }
        }
    }
    printf("\n");

    printf("Parallel C\n");
#pragma omp parallel for collapse(2)
    for (i = 0; i < m1; i++)
    {
        for (j = 0; j < m2; j++)
        {
            if (i == 0 || i == m1 - 1 || j == 0 || j == m2 - 1)
            {
                printf("%d ", m[i][j]);
            }
            else
            {
                char *buf = calloc(sizeof(int) * 2 + 1, sizeof(int));
                itoa(m[i][j], buf, 2);

                for (int z = 0; z < sizeof(int) * 8; z++)
                {
                    buf[z] = (buf[z] == '0') ? '1' : '0';
                };

                printf("%s ", buf);
            }
        }
    }
    printf("\n");

    double end2 = omp_get_wtime();
    double par = end2 - begin2;
    printf("Parallel time: %f\n", par);

    double speedup = seq / par;
    printf("Speedup: %f\n", speedup);
    printf("Efficiency: %f\n", speedup / num_threads);

    return 0;
}
```

Output

```
Enter mat dims
4 4
Enter matrix
1 2 3 4
6 5 8 3
2 4 10 1
9 1 2 5
Seq B
1 2 3 4
6 8 14 3
2 12 10 1
9 1 2 5
Seq C
1 2 3 4
6 01000000000000000000000000000000 01110000000000000000000000000000 3
2 01100000000000000000000000000000 01010000000000000000000000000000 1
9 1 2 5
Sequential time: 5.000000
Parallel B
6 2 4 12 8 1 14 2 3 5 9 1 2 1 10 3
Parallel C
6 01110000000000000000000000000000 2 01100000000000000000000000000000 9 2
01000000000000000000000000000000 1 2 4 5 01010000000000000000000000000000 1 3 3 1
Parallel time: 6.000000
Speedup: 0.833333
Efficiency: 0.833333
```

5. Write a parallel program in OpenMP to reverse the digits of the following integer array of size 9. Initialize the input array to the following values:

a. Input array: 18, 523, 301, 1234, 2, 14, 108, 150, 1928

b. Output array: 81, 325, 103, 4321, 2, 41, 801, 51, 8291

Code

```
#include <stdio.h>
#include <math.h>
#include <omp.h>

int rev(int num)
{
    int buf = 0;
    while (num > 0)
    {
        buf = (10 * buf) + num % 10;
        num = num / 10;
    }

    return buf;
    printf("%d\n", buf);
}

int main()
{
    int arr[9] = {18, 523, 301, 1234, 2, 14, 108, 150, 1928};

    // Sequential
    double begin1 = omp_get_wtime();

    for (int i = 0; i < 9; i++)
    {
        printf("%d ", rev(arr[i]));
    }
    printf("\n");

    double end1 = omp_get_wtime();
    double seq = end1 - begin1;
    printf("Sequential time: %f\n", seq);

    // Parallel
    double begin2 = omp_get_wtime();
    int num_threads;

#pragma omp parallel
    {
        num_threads = omp_get_num_threads();

#pragma omp for
        for (int i = 0; i < 9; i++)
        {
            printf("%d ", rev(arr[i]));
        }
    }
    printf("\n");

    double end2 = omp_get_wtime();
    double par = end2 - begin2;
    printf("Parallel time: %f\n", par / num_threads);

    double speedup = seq / par;
    printf("Speedup: %f\n", speedup);
    printf("Efficiency: %f\n", speedup / num_threads);
    return 0;
}
```

Output

```
81 325 103 4321 2 41 801 51 8291
Sequential time: 0.000000
325 103 41 4321 2 81 801 8291 51
Parallel time: 0.000125
Speedup: 0.000000
Efficiency: 0.000000
```

Week 4

1. Write a parallel program using OpenMP to implement the Selection sort algorithm. Compute the efficiency and plot the speed up for varying input size and thread number.

   Code

```c
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    for (i = 0; i < n - 1; i++)
    {
        min_idx = i;
        for (j = i + 1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;

        if (min_idx != i)
            swap(&arr[min_idx], &arr[i]);
    }
}

struct Compare
{
    int val;
    int index;
};
#pragma omp declare reduction(maximum : struct Compare : omp_out = omp_in.val >
        omp_out.val ? omp_in : omp_out)

void parallelSelectionSort(int arr[], int size)
{
    for (int i = size - 1; i > 0; --i)
    {
        struct Compare max;
        max.val = arr[i];
        max.index = i;
#pragma omp parallel for reduction(maximum : max)
        for (int j = i - 1; j >= 0; --j)
        {
            if (arr[j] > max.val)
            {
                max.val = arr[j];
                max.index = j;
            }
        }
        int tmp = arr[i];
        arr[i] = max.val;
        arr[max.index] = tmp;
    }
}

int main()
{
    int arr1[1000], arr2[1000], n;
    int sizes[] = {100, 200, 300, 500, 1000};
    int threads[] = {2, 3, 4, 5};

    for (int size = 0; size < 5; size++)
    {
        printf("\n\nInput size: %d\n", sizes[size]);
        n = sizes[size];

        for (int i = 0; i < n; i++)
        {
            arr1[i] = rand() % 10 + 1;
            arr2[i] = arr1[i];
        }

        // Sequential
        double start = omp_get_wtime();

        selectionSort(arr1, n);

        double end = omp_get_wtime();
        double seq = end - start;
        printf("Sequential time: %f\n", seq);

        // Parallel
        for (int t = 0; t < 4; t++)
        {
            int thread = threads[t];
            omp_set_num_threads(thread);
            printf("Threads: %d\n", thread);

            start = omp_get_wtime();

            parallelSelectionSort(arr2, n);

            end = omp_get_wtime();
            double par = end - start;
            printf("Parallel time: %f\n", par);

            double speedup = seq / par;
            double eff = speedup / omp_get_num_threads();

            printf("Speedup: %f, Efficiency: %f\n", speedup, eff);
        }
    }
    return 0;
}
```

Output

```
Input size: 100
Sequential time: 0.000000
Threads: 2
Parallel time: 0.005000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 3
Parallel time: 0.006000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 4
Parallel time: 0.007000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 5
Parallel time: 0.008000
Speedup: 0.000000, Efficiency: 0.000000


Input size: 200
Sequential time: 0.000000
Threads: 2
Parallel time: 0.012000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 3
Parallel time: 0.013000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 4
Parallel time: 0.017000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 5
Parallel time: 0.018000
Speedup: 0.000000, Efficiency: 0.000000


Input size: 300
Sequential time: 0.000000
Threads: 2
Parallel time: 0.016000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 3
Parallel time: 0.017000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 4
Parallel time: 0.021000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 5
Parallel time: 0.025000
Speedup: 0.000000, Efficiency: 0.000000


Input size: 500
Sequential time: 0.001000
Threads: 2
Parallel time: 0.029000
Speedup: 0.034480, Efficiency: 0.034480
Threads: 3
Parallel time: 0.036000
Speedup: 0.027776, Efficiency: 0.027776
Threads: 4
```

```
Parallel time: 0.042000
Speedup: 0.023808, Efficiency: 0.023808
Threads: 5
Parallel time: 0.040000
Speedup: 0.024998, Efficiency: 0.024998


Input size: 1000
Sequential time: 0.001000
Threads: 2
Parallel time: 0.046000
Speedup: 0.021743, Efficiency: 0.021743
Threads: 3
Parallel time: 0.060000
Speedup: 0.016669, Efficiency: 0.016669
Threads: 4
Parallel time: 0.066000
Speedup: 0.015154, Efficiency: 0.015154
Threads: 5
Parallel time: 0.082000
Speedup: 0.012197, Efficiency: 0.012197
```

2. Write a parallel program using openMP to implement the following: Take an array of input size m. Divide the array into two parts and sort the first half using insertion sort and second half using quick sort. Use two threads to perform these tasks. Use merge sort to combine the results of these two sorted arrays.

   Code

```c
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>

void insertionSort(int arr[], int n)
{
    int i, key, j;
    for (i = 1; i < n; i++)
    {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int cmp(const void *a, const void *b)
{
    return (*(int *)a - *(int *)b);
}

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;
    int L[100], R[100];

    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}

int main()
{
    int m;
    int arr1[100], arr2[100];

    printf("Enter the size of the array: \n");
    scanf("%d", &m);

    printf("Enter %d elements:\n", m);
    for (int i = 0; i < m; i++)
    {
        scanf("%d", &arr1[i]);
        arr2[i] = arr1[i];
    }

    // Sequential
    double begin = omp_get_wtime();

    insertionSort(arr1, m / 2);
    qsort(arr1 + m / 2, m - m / 2, sizeof(int), cmp);

    mergeSort(arr1, 0, m - 1);

    for (int i = 0; i < m; i++)
    {
        printf("%d ", arr1[i]);
    }
    printf("\n");

    double end = omp_get_wtime();
    double seq = end - begin;
    printf("Sequential time: %f\n", seq);

// Parallel
#pragma omp parallel num_threads(2)
    {
        begin = omp_get_wtime();

#pragma omp sections
        {
#pragma omp section
            {
                insertionSort(arr2, m / 2);
            }

#pragma omp section
            {
                qsort(arr2 + m / 2, m - m / 2, sizeof(int), cmp);
            }
        }

        mergeSort(arr2, 0, m - 1);

        for (int i = 0; i < m; i++)
        {
            printf("%d ", arr2[i]);
        }
        printf("\n");

        end = omp_get_wtime();
        double par = end - begin;
        printf("Parallel time: %f\n", par);

        double speedup = seq / par;
        double eff = speedup / omp_get_num_threads();

        printf("Speedup: %f\nEfficiency: %f\n", speedup, eff);
    }
    return 0;
}
```

Output

```
Enter the size of the array:
```

```
10
Enter 10 elements:
1 2 56 2 5 242 7 43 86 2
1 2 2 2 5 7 43 56 86 242
Sequential time: 0.001000
1 2 2 2 5 7 43 56 86 242
Parallel time: 0.000000
Speedup: 1.#INF00
Efficiency: 1.#INF00
1 2 2 2 5 7 43 56 86 242
Parallel time: 0.002000
Speedup: 0.500119
Efficiency: 0.250060
```

3. Write a parallel program using OpenMP to implement sequential search algorithm. Compute the efficiency and plot the speed up for varying input size and thread number.

   Code

```c
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>

int main()
{
    int arr[1000], n, target;
    int sizes[] = {100, 200, 300, 500, 1000};
    int threads[] = {2, 3, 4, 5};

    for (int size = 0; size < 5; size++)
    {
        printf("\n\nInput size: %d\n", sizes[size]);
        n = sizes[size];

        for (int i = 0; i < n; i++)
        {
            arr[i] = rand() % 10 + 1;
        }
        target = rand() % 10 + 1;

        // Sequential
        double start = omp_get_wtime();

        int res = -1;
        for (int i = 0; i < n; i++)
        {
            if (arr[i] == target)
            {
                res = i;
            }
        }

        if (res == -1)
        {
            printf("Not found\n");
        }
        else
        {
            printf("Found at %d index\n", res);
        }
        double end = omp_get_wtime();
        double seq = end - start;
        printf("Sequential time: %f\n", seq);

        // Parallel
        for (int t = 0; t < 4; t++)
        {
            int thread = threads[t];
            omp_set_num_threads(thread);
            printf("Threads: %d\n", thread);

            start = omp_get_wtime();
            res = -1;

#pragma omp parallel for
            for (int i = 0; i < n; i++)
            {
                if (arr[i] == target)
                {
                    res = i;
                }
            }

            if (res == -1)
            {
                printf("Not found\n");
            }
            else
            {
                printf("Found at %d index\n", res);
            }

            end = omp_get_wtime();
            double par = end - start;
            printf("Parallel time: %f\n", par);

            double speedup = seq / par;
            double eff = speedup / omp_get_num_threads();

            printf("Speedup: %f, Efficiency: %f\n", speedup, eff);
        }
    }

    return 0;
}
```

Output

```
Input size: 100
Found at 93 index
Sequential time: 0.000000
Threads: 2
Found at 93 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00
Threads: 3
Found at 93 index
Parallel time: 0.001000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 4
Found at 93 index
Parallel time: 0.001000
```

```
Speedup: 0.000000, Efficiency: 0.000000
Threads: 5
Found at 93 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00


Input size: 200
Found at 198 index
Sequential time: 0.000000
Threads: 2
Found at 99 index
Parallel time: 0.001000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 3
Found at 198 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00
Threads: 4
Found at 198 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00
Threads: 5
Found at 198 index
Parallel time: 0.001000
Speedup: 0.000000, Efficiency: 0.000000


Input size: 300
Found at 299 index
Sequential time: 0.000000
Threads: 2
Found at 138 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00
Threads: 3
Found at 299 index
Parallel time: 0.001000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 4
Found at 299 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00
Threads: 5
Found at 299 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00


Input size: 500
Found at 451 index
Sequential time: 0.000000
Threads: 2
Found at 244 index
Parallel time: 0.001000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 3
Found at 451 index
Parallel time: 0.001000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 4
Found at 451 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00
```

```
Threads: 5
Found at 451 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00


Input size: 1000
Found at 992 index
Sequential time: 0.000000
Threads: 2
Found at 488 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00
Threads: 3
Found at 992 index
Parallel time: 0.000000
Speedup: -1.#IND00, Efficiency: -1.#IND00
Threads: 4
Found at 992 index
Parallel time: 0.001000
Speedup: 0.000000, Efficiency: 0.000000
Threads: 5
Found at 992 index
Parallel time: 0.001000
Speedup: 0.000000, Efficiency: 0.000000
```

Week 5

1. Write a parallel program using OpenMP to perform vector addition, subtraction, multiplication. Demonstrate task level parallelism. Analyze the speedup and efficiency of the parallelized code.

   Code

```c
#include <stdio.h>
#include <omp.h>

int main()
{
int vec1[100];
int vec2[100];
int sum[100], diff[100], mul[100];
int n;

printf("Enter size of vec\n");
scanf("%d", &n);

printf("Enter corresponding elements\n");
for (int i = 0; i < n; i++)
{
    scanf("%d%d", &vec1[i], &vec2[i]);
}

// Sequential
double start = omp_get_wtime();

for (int i = 0; i < n; i++)
{
    sum[i] = vec1[i] + vec2[i];
    diff[i] = vec1[i] - vec2[i];
    mul[i] = vec1[i] * vec2[i];
}

printf("Sum, Diff, Mul\n");
for (int i = 0; i < n; i++)
{
    printf("%d %d %d", sum[i], diff[i], mul[i]);
    printf("\n");
}

double end = omp_get_wtime();
double seq = end - start;
printf("Seq time: %f\n", seq);

// Parallel
start = omp_get_wtime();

#pragma omp parallel num_threads(3)
#pragma omp sections
{

#pragma omp section
    for (int i = 0; i < n; i++)
    {
    sum[i] = vec1[i] + vec2[i];
    }

#pragma omp section
    for (int i = 0; i < n; i++)
    {
    diff[i] = vec1[i] - vec2[i];
    }

#pragma omp section
    for (int i = 0; i < n; i++)
    {
    mul[i] = vec1[i] * vec2[i];
    }
}

printf("Sum, Diff, Mul\n");
for (int i = 0; i < n; i++)
{
    printf("%d %d %d", sum[i], diff[i], mul[i]);
    printf("\n");
}

end = omp_get_wtime();
double par = end - start;
printf("Parallel time: %f\n", par);

double speedup = seq / par;
double eff = speedup / omp_get_num_threads();

printf("Speedup: %f, Efficiency: %f\n", speedup, eff);

return 0;
}
```

Output

```
Enter size of vec
5
Enter corresponding elements
```

```
1 2 3 4 5 6 7 8 9 10
Sum, Diff, Mul
3 -1 2
7 -1 12
11 -1 30
15 -1 56
19 -1 90
Seq time: 0.001000
Sum, Diff, Mul
3 -1 2
7 -1 12
11 -1 30
15 -1 56
19 -1 90
Parallel time: 0.002000
Speedup: 0.499940, Efficiency: 0.499940
```

2. Write a parallel program using OpenMP to find sum of N numbers using the following constructs/clauses.

a. Critical section

b. Atomic

c. Reduction

d. Master

e. Locks

Code

```c
#include <stdio.h>
#include <omp.h>

int main()
{
int n, vec[100];
int sum1, sum2, sum3, sum4, sum5;
omp_lock_t lock;

printf("Enter number of elements\n");
scanf("%d", &n);

printf("Enter elements\n");
for (int i = 0; i < n; i++)
{
    scanf("%d", &vec[i]);
}

#pragma omp parallel for
for (int i = 0; i < n; i++)
{
#pragma omp critical
    sum1 += vec[i];
}

#pragma omp parallel for
for (int i = 0; i < n; i++)
{
#pragma omp atomic
    sum2 += vec[i];
}

#pragma omp parallel for reduction(+ : sum3)
for (int i = 0; i < n; i++)
{
    sum3 += vec[i];
}

#pragma omp parallel
for (int i = 0; i < n; i++)
{
#pragma omp master
    sum4 += vec[i];
}

#pragma omp parallel
{
    omp_init_lock(&lock);

#pragma omp for
    for (int i = 0; i < n; i++)
    {
    omp_set_lock(&lock);
    sum5 += vec[i];
    omp_unset_lock(&lock);
    }
    omp_destroy_lock(&lock);
}

printf("Critical, Atomic, Reduction, Master, Locks\n");
printf("%d %d %d %d %d", sum1, sum2, sum3, sum4, sum5);
printf("\n");

return 0;
}
```

Output

```
Enter number of elements
5
Enter elements
1 2 3 4 5
Critical, Atomic, Reduction, Master, Locks
31 15 15 15804559 15
```

3. Write a parallel program using OpenMP to implement the Odd-even transposition sort. Vary the input size and analyse the program efficiency.

   Code

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

void transpositionSort(int *a, int n)
{
    int temp;

    for (int phase = 0; phase < n; ++phase)
    {
        if (phase % 2 == 0)
        {

            for (int i = 1; i < n - 1; i += 2)
            {
                if (a[i] > a[i + 1])
                {
                    temp = a[i];
                    a[i] = a[i + 1];
                    a[i + 1] = temp;
                }
            }
        }
        else
        {

            for (int i = 0; i < n - 1; i += 2)
            {
                if (a[i] > a[i + 1])
                {
                    temp = a[i];
                    a[i] = a[i + 1];
                    a[i + 1] = temp;
                }
            }
        }
    }
}

void parallelTranspositionSort(int *a, int n)
{
    int temp;

    for (int phase = 0; phase < n; ++phase)
    {
        if (phase % 2 == 0)
        {

#pragma omp parallel for private(temp)
            for (int i = 1; i < n - 1; i += 2)
            {
                if (a[i] > a[i + 1])
                {
                    temp = a[i];
                    a[i] = a[i + 1];
                    a[i + 1] = temp;
                }
            }
        }
        else
        {

#pragma omp parallel for private(temp)
            for (int i = 0; i < n - 1; i += 2)
            {
                if (a[i] > a[i + 1])
                {
                    temp = a[i];
                    a[i] = a[i + 1];
                    a[i + 1] = temp;
                }
            }
        }
    }
}
```

```c
int main()
{
    int arr1[1000], arr2[1000], n;
    int sizes[] = {100, 200, 300, 500, 1000};
    int threads[] = {2, 3, 4, 5};

    for (int size = 0; size < 5; size++)
    {
        printf("\n\nInput size: %d\n", sizes[size]);
        n = sizes[size];

        for (int i = 0; i < n; i++)
        {
            arr1[i] = rand() % 1000;
            arr2[i] = arr1[i];
        }

        // Sequential
        double start = omp_get_wtime();

        transpositionSort(arr1, n);

        printf("Sorted array\n");
        for (int i = 0; i < n; i++)
        {
            printf("%d ", arr1[i]);
        }
        printf("\n");

        double end = omp_get_wtime();
        double seq = end - start;
        printf("Sequential time: %f\n", seq);

        // Parallel
        for (int t = 0; t < 4; t++)
        {
            int thread = threads[t];
            omp_set_num_threads(thread);
            printf("Threads: %d\n", thread);

            start = omp_get_wtime();

            parallelTranspositionSort(arr2, n);

            end = omp_get_wtime();
            double par = end - start;
            printf("Parallel time: %f\n", par);

            double speedup = seq / par;
            double eff = speedup / omp_get_num_threads();

            printf("Speedup: %f, Efficiency: %f\n", speedup, eff);
        }
    }

    return 0;
}
```

Output

```
Input size: 100
Sorted array
```

```
6 35 35 37 40 41 82 84 101 106 118 141 145 153 169 190 253 264 281 288 292 299 308
316 322 323 333 334 350 358 370 376 382 391 393 421 436 439 446 447 464 467 478 491
500 529 537 538 538 541 547 548 604 623 626 629 644 648 662 664 667 673 703 705 711
716 718 723 724 726 729 741 756 757 771 778 805 811 827 827 840 842 859 868 869 890
894 895 902 912 929 931 942 942 944 954 961 962 966 995
Sequential time: 0.008000
Threads: 2
Parallel time: 0.004000
Speedup: 2.000060, Efficiency: 2.000060
Threads: 3
Parallel time: 0.006000
Speedup: 1.333347, Efficiency: 1.333347
Threads: 4
Parallel time: 0.008000
Speedup: 1.000000, Efficiency: 1.000000
Threads: 5
Parallel time: 0.007000
Speedup: 1.142881, Efficiency: 1.142881


Input size: 200
Sorted array
2 3 7 8 9 20 21 21 21 30 31 38 41 52 53 55 72 75 88 93 97 107 115 127 142 150 154 156
157 161 168 169 179 182 189 190 191 191 195 199 200 202 209 221 224 249 255 270 272
281 281 285 287 290 291 292 303 306 309 310 314 322 328 337 343 348 350 350 355 359
362 374 383 386 389 401 410 413 416 418 421 422 423 426 430 433 448 451 457 458 467
472 476 483 483 484 485 503 506 510 511 512 514 519 523 537 538 548 556 557 573 574
580 585 587 588 589 591 595 596 600 600 602 609 616 617 617 618 622 624 634 636 639
646 648 651 655 655 657 658 668 673 688 699 702 704 712 721 724 728 734 745 753 757
758 762 767 777 788 789 796 798 798 798 807 813 815 829 832 833 836 844 844 861 869
869 875 881 886 888 892 893 900 900 909 924 930 932 935 938 941 945 946 958 966 977
986 989 998 999
Sequential time: 0.018000
Threads: 2
Parallel time: 0.008000
Speedup: 2.249978, Efficiency: 2.249978
Threads: 3
Parallel time: 0.015000
Speedup: 1.200019, Efficiency: 1.200019
Threads: 4
Parallel time: 0.016000
Speedup: 1.125006, Efficiency: 1.125006
Threads: 5
Parallel time: 0.014000
Speedup: 1.285729, Efficiency: 1.285729


Input size: 300
Sorted array
3 7 10 11 18 18 19 22 22 23 24 30 31 41 43 44 53 53 58 60 60 61 64 67 70 72 86 87 99
102 105 108 109 111 112 113 114 116 119 123 125 129 129 139 140 142 142 144 145 152
153 154 159 161 164 168 170 170 173 177 181 185 186 187 188 192 193 195 195 196 200
202 202 213 216 222 222 223 227 229 235 253 260 261 263 264 270 270 271 279 282 285
286 286 292 296 297 297 302 303 313 313 313 314 315 316 321 324 326 329 334 337 350
355 356 357 357 357 360 361 365 368 369 371 372 384 391 405 411 413 414 416 423 423
423 425 428 432 434 437 439 439 441 450 455 457 460 464 466 466 474 474 477 477 477
480 481 483 484 485 487 487 488 492 505 510 512 518 520 520 525 526 527 529 532 535
540 543 547 549 549 550 555 556 558 559 565 565 576 576 577 578 584 585 593 593 596
607 617 624 625 625 625 626 627 627 629 635 646 649 654 658 659 662 667 668 671 675
678 687 692 694 694 695 696 701 704 711 713 717 718 725 734 734 737 745 753 756 757
758 760 760 763 763 771 773 786 787 790 800 801 802 808 823 824 829 832 833 833 835
848 850 851 851 855 866 869 874 882 887 896 896 900 900 902 905 911 912 924 924 926
928 932 938 940 945 949 958 958 962 962 963 972 972 974 976 982 985 993 996
Sequential time: 0.022000
```

```
Threads: 2
Parallel time: 0.011000
Speedup: 2.000022, Efficiency: 2.000022
Threads: 3
Parallel time: 0.016000
Speedup: 1.375002, Efficiency: 1.375002
Threads: 4
Parallel time: 0.018000
Speedup: 1.222234, Efficiency: 1.222234
Threads: 5
Parallel time: 0.016000
Speedup: 1.375002, Efficiency: 1.375002


Input size: 500
Sorted array
1 3 8 8 8 10 11 12 15 15 17 21 22 25 27 28 28 28 35 36 37 37 38 39 40 40 44 49 50 52
54 58 60 64 67 71 71 72 72 73 75 75 75 77 80 80 82 84 84 85 87 87 88 90 93 98 98 103
109 110 112 113 114 114 115 116 117 124 124 129 131 131 132 132 136 141 142 142 145
146 146 148 152 152 153 155 158 164 168 169 171 173 174 175 176 181 184 185 186 192
193 196 196 200 200 205 205 212 213 213 215 215 220 221 221 221 224 226 230 232 233
234 238 240 240 245 247 248 249 256 256 257 258 259 262 262 264 264 264 268 269 269
278 279 282 287 288 289 290 292 292 293 295 296 300 303 303 303 307 309 309 313 313
314 315 317 318 318 318 318 323 330 333 335 336 340 342 342 345 347 352 353 353 355
355 359 361 363 363 369 375 380 392 392 392 397 398 402 404 405 409 410 411 413 415
416 421 422 423 423 423 424 426 428 429 430 439 443 448 452 454 458 459 461 462 462
463 467 472 479 481 482 484 486 488 489 489 491 493 496 497 498 498 503 503 504 508
508 511 515 515 516 519 527 528 531 534 536 539 540 540 541 541 542 543 545 546 548
549 549 555 556 556 556 561 561 565 565 567 570 575 584 584 588 589 589 591 598 600
600 601 601 601 602 602 604 605 606 608 608 610 611 619 619 620 625 626 626 627 629
629 630 633 634 634 637 637 637 641 643 647 648 650 650 651 652 653 659 662 662 663
666 667 670 673 674 676 676 678 678 679 679 681 681 683 685 685 686 686 689 690 690
693 695 698 700 701 704 704 705 705 706 708 710 712 712 721 722 722 723 726 726 734
736 740 741 745 748 748 750 752 754 758 759 759 762 762 763 766 769 774 775 778 781
783 783 786 788 790 799 806 807 809 812 813 814 815 815 816 818 818 824 825 825 826
829 830 831 831 838 841 843 847 850 853 855 855 858 864 864 865 866 867 869 870 870
873 875 877 878 881 882 885 888 893 898 901 902 902 903 907 909 913 913 913 918 923
923 925 932 934 936 937 938 941 941 943 944 944 945 948 948 949 951 954 954 955 956
958 959 961 962 964 964 969 970 971 971 971 974 975 977 985 990 992 993 994 995 996
997
Sequential time: 0.039000
Threads: 2
Parallel time: 0.025000
Speedup: 1.560001, Efficiency: 1.560001
Threads: 3
Parallel time: 0.026000
Speedup: 1.499991, Efficiency: 1.499991
Threads: 4
Parallel time: 0.033000
Speedup: 1.181812, Efficiency: 1.181812
Threads: 5
Parallel time: 0.039000
Speedup: 1.000000, Efficiency: 1.000000


Input size: 1000
Sorted array
0 1 4 5 5 6 6 7 7 8 9 11 12 12 15 15 15 18 18 19 20 21 21 21 22 22 23 27 28 28 28 29
31 32 32 33 33 38 40 41 42 42 43 47 48 52 53 55 55 56 56 58 58 58 60 63 63 63 64 65
65 68 71 72 72 72 73 74 74 74 74 74 75 75 75 76 78 78 79 80 80 80 80 81 82 83 84 85
85 88 91 91 93 93 95 95 95 97 97 97 100 104 105 105 105 106 106 109 110 111 111 112
118 118 119 119 122 124 125 128 128 130 134 135 135 135 135 136 136 136 140 141 144
144 146 146 147 149 149 151 152 152 153 154 156 159 159 161 161 162 162 163 164 165
166 166 166 168 168 168 169 169 170 171 172 172 172 174 176 177 178 179 180 181 181
```

```
182 182 182 183 183 184 184 185 186 187 188 189 189 189 191 192 193 193 194 195 196
196 196 197 197 198 199 199 199 199 200 202 204 206 206 208 208 208 209 210 213 214
216 216 217 218 220 221 221 222 223 223 224 225 225 226 227 229 229 230 231 232 234
236 237 237 238 239 240 241 243 243 244 244 245 245 247 248 249 249 251 252 252 253
253 253 255 257 258 259 261 262 264 264 267 267 268 268 271 271 271 274 276 276 276
277 278 278 279 279 281 281 282 283 283 286 287 289 289 290 290 290 293 293 296 296
300 301 303 310 311 313 315 318 318 319 320 321 323 324 326 327 328 329 330 330 330
331 331 333 333 335 335 336 337 337 337 337 338 338 339 340 340 343 343 348 350 351
351 352 353 355 358 360 360 361 363 363 363 364 364 365 365 367 367 368 369 370 372
372 372 373 375 376 376 380 381 381 382 386 387 387 388 390 391 392 393 393 394 396
396 396 396 398 400 406 408 408 411 413 415 415 418 418 419 420 420 421 422 422 426
426 428 431 431 431 432 432 432 432 433 434 434 435 439 440 442 443 444 445 445 445
446 447 447 447 449 449 450 450 452 454 455 455 455 455 455 457 458 460 460 464 465
465 469 469 470 470 472 472 473 474 474 474 474 475 476 476 477 479 479 481 481 482
483 486 486 486 486 487 487 487 487 487 489 489 489 489 490 490 490 492 492 493 493
495 495 497 497 497 501 501 503 503 503 505 507 508 508 509 510 512 513 514 515 517
517 520 521 522 523 523 524 525 525 528 528 529 529 529 531 532 534 535 537 538 538
540 540 542 544 545 545 546 547 549 549 550 551 552 556 556 557 559 560 560 561 561
562 563 565 565 565 567 568 569 570 572 573 573 577 577 577 577 578 578 578 578 578
579 580 580 582 583 584 584 585 586 588 588 589 589 590 591 591 593 594 594 594 595
597 598 599 599 601 602 604 604 604 605 606 607 608 608 608 608 609 610 610 611 611
611 612 614 614 615 615 615 616 617 620 621 623 623 626 629 630 633 634 635 636 638
640 640 640 641 643 643 643 646 647 647 648 649 650 650 651 651 652 652 652 653 654
655 656 658 658 661 661 663 663 664 665 666 666 667 668 669 670 670 671 671 672 675
676 677 677 677 678 678 678 679 680 680 680 681 681 681 683 684 684 692 692 693 693
694 695 697 697 697 697 699 700 702 703 705 707 707 710 710 713 713 714 714 714 716
716 717 718 718 719 720 721 722 723 723 724 725 727 729 733 734 735 737 737 737 737
738 740 741 741 741 742 742 747 748 750 750 750 750 752 753 753 753 754 754 756 758
759 759 759 761 761 762 762 764 766 768 768 769 770 771 773 774 774 776 777 779 779
779 779 779 780 781 783 783 784 785 785 787 787 788 789 790 790 790 791 791 791 792
793 793 794 795 797 801 801 805 805 805 805 806 807 807 808 810 810 812 812 813 813
814 815 816 819 821 821 822 823 826 826 827 828 828 830 832 833 834 835 836 836 837
838 840 840 841 841 844 846 850 851 851 854 855 855 856 857 859 860 860 861 862 864
865 865 867 868 869 871 872 874 874 874 875 876 877 878 880 881 882 882 882 885 885
888 890 890 891 892 893 893 895 896 897 899 899 901 901 902 903 904 905 905 906 907
908 908 910 910 912 912 916 916 919 920 922 922 922 923 924 925 925 925 926 926 928
928 929 931 932 933 935 936 938 940 941 943 945 945 946 947 947 948 948 949 949 950
954 955 955 955 958 961 962 962 962 963 964 965 967 967 968 968 969 969 970 971 972
972 973 974 974 975 976 978 978 979 979 981 981 982 982 984 986 987 987 987 989 990
990 991 993 995 998 998
Sequential time: 0.083000
Threads: 2
Parallel time: 0.050000
Speedup: 1.660000, Efficiency: 1.660000
Threads: 3
Parallel time: 0.066000
Speedup: 1.257575, Efficiency: 1.257575
Threads: 4
Parallel time: 0.074000
Speedup: 1.121623, Efficiency: 1.121623
Threads: 5
Parallel time: 0.081000
Speedup: 1.024690, Efficiency: 1.024690
```

4. Write an OpenMP program to find the Summation of integers from a given interval. Analyze the performance of various iteration scheduling strategies.

   Code

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

```c
#include <omp.h>

#define ll long

int main()
{
    ll a, b, sum = 0;

    printf("Enter range\n");
    scanf("%ld%ld", &a, &b);

    // Static
    double start = omp_get_wtime();

#pragma omp parallel for schedule(static) reduction(+ : sum)
    for (int i = 0; i < b - a; i++)
    {
        sum += i + b;
    }

    double end = omp_get_wtime();
    double stat = end - start;
    printf("Time taken for static scheduling: %f\n", stat);

    // Dynamic
    start = omp_get_wtime();

#pragma omp parallel for schedule(dynamic) reduction(+ : sum)
    for (int i = 0; i < b - a; i++)
    {
        sum += i + b;
    }

    end = omp_get_wtime();
    double dyn = end - start;
    printf("Time taken for dynamic scheduling: %f\n", dyn);

    // Guided
    start = omp_get_wtime();

#pragma omp parallel for schedule(guided) reduction(+ : sum)
    for (int i = 0; i < b - a; i++)
    {
        sum += i + b;
    }

    end = omp_get_wtime();
    double guided = end - start;
    printf("Time taken for guided scheduling: %f\n", guided);

    // Runtime
    start = omp_get_wtime();

#pragma omp parallel for schedule(runtime) reduction(+ : sum)
    for (int i = 0; i < b - a; i++)
    {
        sum += i + b;
    }

    end = omp_get_wtime();
    double runtime = end - start;
    printf("Time taken for runtime scheduling: %f\n", runtime);

    return 0;
}
```

Output

```
Enter range
0 99999999
Time taken for static scheduling: 0.032000
Time taken for dynamic scheduling: 1.817000
Time taken for guided scheduling: 0.029000
Time taken for runtime scheduling: 1.883000
```

5. Write a parallel program using OpenMP to generate the histogram of the given array A.

Code

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

#define ll long

int main()
{
    double arr[] = {1.3, 2.9, 0.4, 0.3, 1.3, 4.4, 1.7, 0.4, 3.2, 0.3, 4.9, 2.4, 3.1,
        4.4, 3.9, 0.4, 4.2, 4.5, 4.9, 0.9};
    int histogram[100] = {0};

    double max = arr[0];
    double min = arr[0];

    for (int i = 1; i < 20; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
        }
        if (arr[i] < min)
        {
            min = arr[i];
        }
    }

    double binSize = (max - min) / 10;

#pragma omp parallel for
    for (int i = 0; i < 20; i++)
    {
        int bin = (int)((arr[i] - min) / binSize);

#pragma omp critical
        {
            histogram[bin]++;
        }
    }

    printf("[Start, End) - Frequency\n");
    for (int i = 0; i < 10; i++)
    {
        double start = min + i * binSize;
        double end = start + binSize;
        printf("[%0.1f, %0.1f) - %d\n", start, end, histogram[i]);
    }
}
```

Output

```
[Start, End) - Frequency
[0.3, 0.8) - 5
[0.8, 1.2) - 1
[1.2, 1.7) - 2
[1.7, 2.1) - 1
[2.1, 2.6) - 1
[2.6, 3.1) - 1
[3.1, 3.5) - 2
[3.5, 4.0) - 1
[4.0, 4.4) - 3
[4.4, 4.9) - 1
```