**Parallel Programming Lab**



NAINA CHABRA

210968234

B2

PARALLEL PROGRAMMING ASSIGNMENT 1

***Week 1: Simple C programs and execution***

*Q1) 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:**

#include<stdio.h>

int main(){

    int n;

    printf("Enter the size of array");

    scanf("%d",&n);

    int arr[n];

    int i;

    for(i=0;i<n;i++){

        scanf("%d",&arr[i]);

    }

    for(i=0;i<n;i++){

        int orig = arr[i];

        int rev = 0;

        while(orig!=0){

            int dig = orig%10;

            rev = rev\*10+dig;

            orig = orig/10;

        }

        arr[i]=rev;

    }

    for(i=0;i<n;i++){

        printf("%d ",arr[i]);

    }

    return 0;

}

**OUTPUT:**

A screenshot of a computer code

Description automatically generated

*Q2) Write a program in C to simulate 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>

int main(){

printf("Enter the value of A");

int a;

scanf("%d",&a);

printf("Enter the value of B");

int b;

scanf("%d",&b);

printf("A+B=%d\n",a+b);

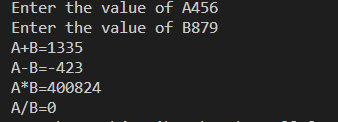
printf("A-B=%d\n",a-b);

printf("A\*B=%d\n",a\*b);

printf("A/B=%d\n",a/b);

}

**OUTPUT:**



*Q3) 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<string.h>

int main(){

printf("Enter the string");

char a[1000];

gets(a);

int i,count=0;

for(i=0;a[i]!='\0';i++){

count++;

}

for(i=0;a[i]!='\0';i++){

if(a[i]>=65 && a[i]<=90)

printf("%c",a[i]+32);

else if(a[i]==32){

printf(" ");

}

else

printf("%c",a[i]-32);

}

return 0;

}

**OUTPUT:**



*Q4) 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<string.h>

#include<stdio.h>

int main(){

printf("Enter the string");

char a[1000];

gets(a);

int i=0;

int j=0;

while(a[j]!='\0'){

for(i=j+1;i>0;i--){

printf("%c",a[j]);

}

j++;

}

}

**OUTPUT:**



*Q5) 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<string.h>

#include<stdio.h>

int main(){

printf("Enter string 1");

char a[1000];

gets(a);

char b[1000];

printf("Enter string 2");

gets(b);

int i;

int count1=0,count2=0;

for(i=0;a[i]!='\0';i++){

count1++;

}

for(i=0;b[i]!='\0';i++){

count2++;

}

if(count1!=count2){

printf("Strings are not of same length");

}

else{

int str1=0,str2=0;

while(str1<count1 && str2<count2){

printf("%c%c",a[str1],b[str2]);

str1++;

str2++;

}

}

return 0;

}

**OUTPUT:**

A black background with white text

Description automatically generated

*Q6) Write a C program to perform Matrix times vector product operation.*

**CODE:**  
#include<time.h>

#include<stdio.h>

#include<math.h>

int main(){

    clock\_t start1,end1,start,end;

    printf("Enter the dimension of the matrix");

    int r1,c1;

    int i,j,k;

    scanf("%d%d",&r1,&c1);

    int(\*arr1)[c1] = malloc(r1\*c1\*sizeof(int));

    printf("Enter elements");

    int cnt=1;

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            arr1[i][j] = cnt;

            cnt++;

        }

    }

    printf("\n");

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            printf("%d ",arr1[i][j]);

        }

        printf("\n");

    }

    printf("options:\n1.Row vector\n2.Columnar vector\n");

    int ch;

    int dim;

    scanf("%d",&ch);

    if(ch==1){

        printf("Enter the number of columns for the row vector ");

        scanf("%d",&dim);

    }

    else{

        printf("Enter the number of rows for the column vector ");

        scanf("%d",&dim);

    }

    int \*vec = (int \*)malloc(dim\*sizeof(int));

    for(i=0;i<dim;i++){

        vec[i] = cnt;

        cnt++;

    }

    printf("\n");

    for(i=0;i<dim;i++){

        printf("%d ",vec[i]);

    }

    int \*res= malloc(c1\*sizeof(int));

    for(i=0;i<dim;i++){

        res[i] = 0;

    }

    if(ch==1){

        start = clock();

        for(i=0;i<dim;i++){

            for(k=0;k<c1;k++){

                res[i]+=vec[k]\*arr1[k][i];

            }

        }

        end = clock();

    }

    else{

        start = clock();

        for(i=0;i<c1;i++){

            for(k=0;k<dim;k++){

                res[i]+=arr1[i][k]\*vec[k];

            }

        }

        end = clock();

    }

    printf("\n");

    if(ch==2){

        for(i=0;i<c1;i++){

            printf("%d\n",res[i]);

        }

    }

    else{

        for(i=0;i<c1;i++){

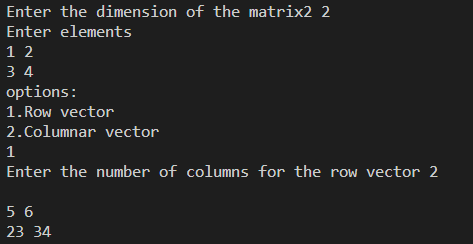
            printf("%d ",res[i]);

        }

    }

}

**OUTPUT:**



*Q7) 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.*

*Example:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *A* | | | | |
| *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* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *B* | | | | |
| *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* |

**CODE:**

#include<stdio.h>

#include<limits.h>

int returnmax(int arr[5][5],int row){

int max=INT\_MIN;

int i,j;

for(i=0;i<5;i++){

if(arr[row][i]>max){

max = arr[row][i];

}

}

return max;

}

int returnmin(int arr[5][5],int row){

int min=INT\_MAX;

int i,j;

for(i=0;i<5;i++){

if(arr[row][i]<min){

min = arr[row][i];

}

}

return min;

}

int main(){

printf("Enter the elements of the 5x5 matrix\n");

int arr[5][5];

int i,j;

for(i=0;i<5;i++){

for(j=0;j<5;j++){

scanf("%d",&arr[i][j]);

}

}

printf("the contents of the matrix are\n");

for(i=0;i<5;i++){

for(j=0;j<5;j++){

printf("%d ",arr[i][j]);

}

printf("\n");

}

int ele;

int arr2[5][5];

for(i=0;i<5;i++){

for(j=0;j<5;j++){

if(i!=j && i>j){ //these are the elements above the diag

arr2[i][j] = returnmax(arr,i);

}

else if(i!=j && i<j){ //these are the elements below the diag

arr2[i][j] = returnmin(arr,i);

}

else{

arr2[i][j] =0;

}

}

}

printf("After modification the elements are\n");

for(i=0;i<5;i++){

for(j=0;j<5;j++){

printf("%d ",arr2[i][j]);

}

printf("\n");

}

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated

*Q8) 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.*

*Example:*

*A*

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| *6* | *5* | *8* | *3* |
| *2* | *4* | *10* | *1* |
| *9* | *1* | *2* | *5* |

*B*

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| *6* | ***10*** | ***111*** | *3* |
| *2* | ***11*** | ***101*** | *1* |
| *9* | *1* | *2* | *5* |

*D*

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| *6* | ***2*** | ***7*** | *3* |
| *2* | ***3*** | ***5*** | *1* |
| *9* | *1* | *2* | *5* |

**CODE:**

#include<stdio.h>

#include<math.h>

#include<stdbool.h>

int ones(int num){

    int val = 1;

    while(num!=0){

        if(num%2==1){

            val = val\*10+1;

        }

        else{

            val = val\*10+0;

        }

        num/=2;

    }

    int cpy = val;

    int len = 0;

    while(cpy!=0){

        len++;

        cpy = cpy/10;

    }

//    printf("length %d\n",len);

    int i;

    int ten = 10;

    for(i=1;i<len-1;i++){

        ten\*=10;

    }

//    printf("10 raised %d\n",ten);

//    printf("moded %d\n",val%ten);

    int rev=0;

    int len1 = len;

    while(len>1){

        int dig = val%10;

        rev = rev\*10 + dig;

        val  = val/10;

        len--;

    }

//    printf("reversed num %d\n",rev);

    int cpynum = rev;

    int arr[len1-1];

    for(i=len1-2;i>=0;i--){

        arr[i] = cpynum%10;

        cpynum/=10;

    }

    for(i=0;i<len1-1;i++){

        if(arr[i]==0){

            arr[i] = 1;

        }

        else{

            arr[i] = 0;

        }

    }

    bool flag = false;

    int corans=0;

    for(i=0;i<len1-1;i++){

        if(arr[i]==1){

            corans = corans\*10+1;

            flag=true;

        }

        else if(arr[i]==0 && flag){

            flag=true;

            corans = corans\*10+0;

        }

    }

//  printf("%d",corans);

    return corans;

}

int deci(int num){

    int cpy = num;

    double val=0;

    int count=0;

    while(cpy!=0){

        int dig = cpy%10;

        if(dig==1){

            val+=pow(2,count);

        }

        cpy/=10;

        count++;

    }

//  printf("decimal value = %d",(int)val);

    return (int)val;

}

int main(){

    printf("Enter the dimension of the matrix");

    int r1,c1;

    scanf("%d%d",&r1,&c1);

    int arr1[r1][c1];

    int i,j;

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            scanf("%d",&arr1[i][j]);

        }

    }

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            printf("%d ",arr1[i][j]);

        }

        printf("\n");

    }

    printf("\n");

    int arr2[r1][c1];

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            if((i>0 && i<r1-1) && (j>0 && j<c1-1)){

                arr2[i][j] = ones(arr1[i][j]);

            }

            else{

                arr2[i][j] = arr1[i][j];

            }

        }

    }

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            printf("%d ",arr2[i][j]);

        }

        printf("\n");

    }

    printf("\n");

    int arr3[r1][c1];

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            if((i!=0 && i!=r1-1) && (j!=0 && j!=c1-1)){

                arr3[i][j] = deci(arr2[i][j]);

            }

            else{

                arr3[i][j] = arr2[i][j];

            }

        }

    }

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            printf("%d ",arr3[i][j]);

        }

        printf("\n");

    }

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated

*Q9) 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).*

*Example: A B*

*p C a P 1 2 4 3*

*e X a M 2 4 3 2*

*Output string STR: pCCaaaaPPPeeXXXXaaaMM*

**CODE:**  
#include<stdio.h>

int main(){

    printf("Enter the dimension of character matrix");

    int r1,c1;

    scanf("%d%d",&r1,&c1);

    char arr1[r1][c1];

    int i,j;

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            scanf(" %c",&arr1[i][j]);

        }

    }

    printf("Enter the elements of number matrix");

    int arr2[r1][c1];

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            scanf("%d",&arr2[i][j]);

        }

    }

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            int count = arr2[i][j];

            while(count>0){

                printf("%c",arr1[i][j]);

                count--;

            }

        }

    }

}

**OUTPUT:**

A black background with white text

Description automatically generated

***WEEK 2: C-programs with Speedup, Efficiency, Parallel execution time and sequential execution time using OpenMP directives.***

*Q1) 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:**

#include<stdio.h>

#include<omp.h>

#include<time.h>

int main(){

    int n;

    clock\_t start,end;

    printf("Enter the size of array");

    scanf("%d",&n);

    int arr[n],arr1[n],arr2[n];

    int i,j;

    for(i=0;i<n;i++){

        scanf("%d",&arr[i]);

    }

    start = clock();

    for(j=0;j<9999999;j++){     //the program is made to run in loop in order to increase the CPU execution time

        for(i=0;i<n;i++){

            int orig = arr[i];

            int rev = 0;

            while(orig!=0){

                int dig = orig%10;

                rev = rev\*10+dig;

                orig = orig/10;

            }

            arr1[i]=rev;

        }

    }

    end = clock();

    printf("the time taken in sequential executin: %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

    printf("CLOCKS PER SEC %ld\n",CLOCKS\_PER\_SEC);

    for(j=0;j<n;j++){

        printf("%d ",arr1[j]);

    }

    printf("\n");

    start = clock();

    #pragma omp parallel for

    for(j=0;j<9999999;j++){     //the program is made to run in loop in order to increase the CPU execution time

        for(i=0;i<n;i++){

            int orig = arr[i];

            int rev = 0;

            while(orig!=0){

                int dig = orig%10;

                rev = rev\*10+dig;

                orig = orig/10;

            }

            arr2[i]=rev;

        }

    }

    end = clock();

    printf("the time taken in parallel execution: %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

    printf("CLOCKS PER SEC %ld\n",CLOCKS\_PER\_SEC);

    for(j=0;j<n;j++){

        printf("%d ",arr2[j]);

    }

    return 0;

}

**OUTPUT:**

A screenshot of a computer

Description automatically generated

**Tparallel = 2.021sec**

**Tsequential = 2.222sec**

**Speedup = Tparallel/Tsequential = 1.099**

**Efficiency = S/p (in this case the number of cores is 8) thus:**

**1.099/8 = 0.137**

**(since the program is run for a large number of iterations it is inefficient but that is being done for recording purposes)**

*Q2) Write a program in C to simulate all the operations of a calculator. Given inputs A and B, find the output for*

*A+B*

*A-B*

*A\*B*

*A/B*

**CODE:**

#include<stdio.h>

#include<omp.h>

#include<time.h>

int main(){

    clock\_t start,end;

    int i=0;

    printf("Enter the value of A");

    int a;

    scanf("%d",&a);

    printf("Enter the value of B");

    int b;

    scanf("%d",&b);

    start = clock();

    int res1;

    int res2;

    int res3;

    int res4;

    for(i=0;i<90000000;i++){

        res1 = a+b;

        res2 = a-b;

        res3 = a\*b;

        res4 = a/b;

    }

    printf("A+B=%d\n",res1);

    printf("A-B=%d\n",res2);

    printf("A\*B=%d\n",res3);

    printf("A/B=%d\n",res4);

    end = clock();

    printf("time taken for sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

    start = clock();

    #pragma omp parallel for

    for(i=0;i<90000000;i++){

        res1 = a+b;

        res2 = a-b;

        res3 = a\*b;

        res4 = a/b;

    }

    printf("A+B=%d\n",res1);

    printf("A-B=%d\n",res2);

    printf("A\*B=%d\n",res3);

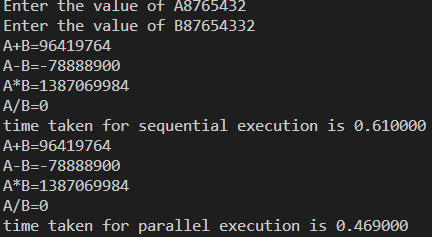
    printf("A/B=%d\n",res4);

    end = clock();

    printf("time taken for parallel execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**OUTPUT:**



**Tparallel = 0.469**

**Tsequential = 0.61**

**Speedup = 1.200**

**efficiency = 0.15**

*Q3) 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<string.h>

#include<omp.h>

#include<time.h>

int main(){

printf("Enter the string");

char a[9999];

gets(a);

int i;

clock\_t start,end;

start = clock();

for(i=0;a[i]!='\0';i++){

if(a[i]>65 && a[i]<90)

printf("%c",a[i]+32);

else

printf("%c",a[i]-32);

}

end = clock();

printf("\n");

printf("The time taken for sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

start = clock();

#pragma omp parallel for

for(i=0;a[i]!='\0';i++){

if(a[i]>65 && a[i]<90)

printf("%c",a[i]+32);

else

printf("%c",a[i]-32);

}

end = clock();

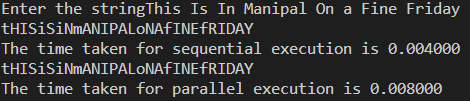
printf("\n");

printf("The time taken for parallel execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

return 0;

}

**OUTPUT:**



**Tparallel = 0.008**

**Tsequential = 0.004**

**Speedup = 0.5**

**Efficiency = 0.0625**

*Q4) 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<string.h>

#include<stdio.h>

#include<time.h>

#include<omp.h>

int main(){

clock\_t start,end;

printf("Enter the string");

char a[1000];

gets(a);

int i,j=0;

start = clock();

while(a[j]!='\0'){

for(i=j+1;i>0;i--){

printf("%c",a[j]);

}

j++;

}

end = clock();

double time1 =

printf("The time taken in sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

start = clock();

j=0;

#pragma omp parallel for

for(i=0;i<1;i++){

while(a[j]!='\0'){

for(i=j+1;i>0;i--){

printf("%c",a[j]);

}

j++;

}

}

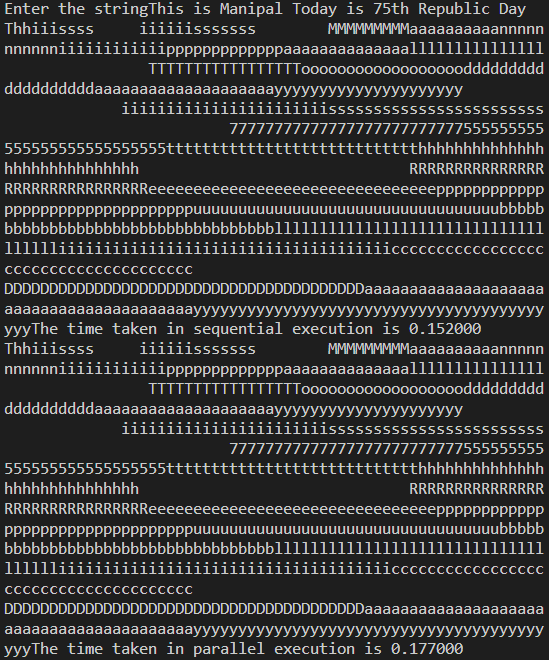
end = clock();

double time2 = (end-start)/CLOCKS\_PER\_SEC;

printf("The time taken in parallel execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**OUTPUT:**



**OUTPUT:**

**Tparallel = 0.177**

**Tsequential = 0.152**

**Speedup = 0.858**

**Efficiency = 0.107**

*Q5) 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<string.h>

#include<stdio.h>

#include<omp.h>

#include<time.h>

int main(){

    printf("Enter string 1");

    char a[9000];

    gets(a);

    clock\_t start,end,start1,end1;

    char b[1000];

    printf("Enter string 2");

    gets(b);

    int i;

    int count1=0,count2=0;

    for(i=0;a[i]!='\0';i++){

        count1++;

    }

    for(i=0;b[i]!='\0';i++){

        count2++;

    }

    if(count1!=count2){

        printf("Strings are not of same length");

    }

    else{

        int str1=0,str2=0;

        start = clock();

        while(str1<count1 && str2<count2){

            printf("%c%c",a[str1],b[str2]);

            str1++;

            str2++;

        }

        end = clock();

        printf("\nThe time taken in sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

        str1=0,str2=0;

        start1 = clock();

        #pragma omp parallel for

        for(i=0;i<1;i++){

            while(str1<count1 && str2<count2){

                printf("%c%c",a[str1],b[str2]);

                str1++;

                str2++;

            }

        }

        end1 = clock();

        printf("\nThe time taken in parallel execution is %f\n",(double)(end1-start1)/CLOCKS\_PER\_SEC);

    }

    return 0;

}

**OUTPUT:**

A screen shot of a computer screen

Description automatically generated

**Tparallel = 0.002**

**Tsequential = 0.006**

**Speedup = 3**

**Efficiency = 0.375**

*Q6) Write a C program to perform Matrix times vector product operation.*

**CODE:**

// #include<stdio.h>

// #include<omp.h>

// #include<time.h>

// int main(){

// clock\_t start,end,start1,end1;

// printf("Enter the dimensions of the matrix");

// int r1,c1;

// scanf("%d%d",&r1,&c1);

// int arr1[r1][c1];

// printf("Enter the elements of the matrix");

// int i,j,k;

// int count = 100;

// for(i=0;i<r1;i++){

// for(j=0;j<c1;j++){

// arr1[i][j] = count;

// count++;

// }

// }

// printf("Enter the dimensions of the vector");

// int r2,c2;

// scanf("%d%d",&r2,&c2);

// int arr2[r2][c2];

// printf("Enter the elements of the vector");

// count=500;

// for(i=0;i<r2;i++){

// for(j=0;j<c2;j++){

// arr1[i][j] = count;

// count++;

// }

// }

// int res[c1][r2];

// for(i=0;i<c1;i++){

// for(j=0;j<r2;j++){

// res[i][j]=0;

// }

// }

// start = clock();

// for(i=0;i<c1;i++){

// for(j=0;j<r2;j++){

// for(k=0;k<c2;k++){

// res[i][k]+=arr1[i][j]\*arr2[j][k];

// }

// }

// }

// end = clock();

// for(i=0;i<r1;i++){

// for(j=0;j<c2;j++){

// printf("%d ",res[i][j]);

// }

// printf("\n");

// }

// printf("\nThe time taken in sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

// for(i=0;i<c1;i++){

// for(j=0;j<r2;j++){

// res[i][j]=0;

// }

// }

// start1 = clock();

// #pragma omp parallel for

// for(i=0;i<c1;i++){

// for(j=0;j<r2;j++){

// for(k=0;k<c2;k++){

// res[i][k]+=arr1[i][j]\*arr2[j][k];

// }

// }

// }

// end1 = clock();

// for(i=0;i<r1;i++){

// for(j=0;j<c2;j++){

// printf("%d ",res[i][j]);

// }

// printf("\n");

// }

// printf("\nThe time taken in parallel execution is %f\n",(double)(end1-start1)/CLOCKS\_PER\_SEC);

// }

#include<stdio.h>

#include<omp.h>

#include<time.h>

int main(){

clock\_t start1,end1,start,end;

printf("Enter the dimension of the matrix");

int r1,c1;

int i,j,k;

scanf("%d%d",&r1,&c1);

int(\*arr1)[c1] = malloc(r1\*c1\*sizeof(int));

printf("Enter elements");

int cnt=1;

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

arr1[i][j] = cnt;

cnt++;

}

}

// printf("\n");

// for(i=0;i<r1;i++){

// for(j=0;j<c1;j++){

// printf("%d ",arr1[i][j]);

// }

// printf("\n");

// }

printf("options:\n1.Row vector\n2.Columnar vector\n");

int ch;

int dim;

scanf("%d",&ch);

if(ch==1){

printf("Enter the number of columns for the row vector ");

scanf("%d",&dim);

}

else{

printf("Enter the number of rows for the column vector ");

scanf("%d",&dim);

}

int \*vec = (int \*)malloc(dim\*sizeof(int));

for(i=0;i<dim;i++){

vec[i] = cnt;

cnt++;

}

// printf("\n");

// for(i=0;i<dim;i++){

// printf("%d ",vec[i]);

// }

int \*res= malloc(c1\*sizeof(int));

for(i=0;i<dim;i++){

res[i] = 0;

}

if(ch==1){

start = clock();

for(i=0;i<dim;i++){

for(k=0;k<c1;k++){

res[i]+=vec[k]\*arr1[k][i];

}

}

end = clock();

}

else{

start = clock();

for(i=0;i<c1;i++){

for(k=0;k<dim;k++){

res[i]+=arr1[i][k]\*vec[k];

}

}

end = clock();

}

printf("\n");

// if(ch==2){

// for(i=0;i<c1;i++){

// printf("%d\n",res[i]);

// }

// }

// else{

// for(i=0;i<c1;i++){

// printf("%d ",res[i]);

// }

// }

double time1 = (end-start)/CLOCKS\_PER\_SEC;

for(i=0;i<dim;i++){

res[i] = 0;

}

if(ch==1){

start1 = clock();

#pragma omp parallel for

for(i=0;i<dim;i++){

for(k=0;k<c1;k++){

res[i]+=vec[k]\*arr1[k][i];

}

}

end1 = clock();

}

else{

start1 = clock();

#pragma omp parallel for

for(i=0;i<c1;i++){

for(k=0;k<dim;k++){

res[i]+=arr1[i][k]\*vec[k];

}

}

end1 = clock();

}

printf("\n");

printf("Time taken in sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

printf("Time taken in parallel execution is %f\n",(double)(end1-start1)/CLOCKS\_PER\_SEC);

}

**OUTPUT:**

A computer screen shot of a black screen

Description automatically generated

**Tparallel = 1.849**

**Tsequential = 1.911**

**Speedup = 1.033**

**Efficiency = 0.129**

*Q7) 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.*

*Example:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *A* | | | | |
| *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* |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *B* | | | | |
| *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* |

**CODE:**

#include<stdio.h>

#include<limits.h>

#include<time.h>

#include<omp.h>

int returnmax(int arr[5][5],int row){

int max=INT\_MIN;

int i;

for(i=0;i<5;i++){

if(arr[row][i]>max){

max = arr[row][i];

}

}

return max;

}

int returnmin(int arr[5][5],int row){

int min=INT\_MAX;

int i,j;

for(i=0;i<5;i++){

if(arr[row][i]<min){

min = arr[row][i];

}

}

return min;

}

int main(){

clock\_t start,end;

printf("Enter the elements of the 5x5 matrix\n");

int arr[5][5];

int i,j;

for(i=0;i<5;i++){

for(j=0;j<5;j++){

scanf("%d",&arr[i][j]);

}

}

printf("the contents of the matrix are\n");

for(i=0;i<5;i++){

for(j=0;j<5;j++){

printf("%d ",arr[i][j]);

}

printf("\n");

}

int ele;

int arr2[5][5];

start = clock();

for(i=0;i<5;i++){

for(j=0;j<5;j++){

if(i!=j && i>j){ //these are the elements above the diag

arr2[i][j] = returnmax(arr,i);

}

else if(i!=j && i<j){ //these are the elements below the diag

arr2[i][j] = returnmin(arr,i);

}

else{

arr2[i][j] =0;

}

}

}

end = clock();

printf("After modification the elements are\n");

for(i=0;i<5;i++){

for(j=0;j<5;j++){

printf("%d ",arr2[i][j]);

}

printf("\n");

}

printf("\nThe time taken for sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

start = clock();

int cnt;

#pragma omp parallel for

for(cnt=0;cnt<1;cnt++){

for(i=0;i<5;i++){

for(j=0;j<5;j++){

if(i!=j && i>j){ //these are the elements above the diag

arr2[i][j] = returnmax(arr,i);

}

else if(i!=j && i<j){ //these are the elements below the diag

arr2[i][j] = returnmin(arr,i);

}

else{

arr2[i][j] =0;

}

}

}

}

end = clock();

printf("After modification the elements are\n");

for(i=0;i<5;i++){

for(j=0;j<5;j++){

printf("%d ",arr2[i][j]);

}

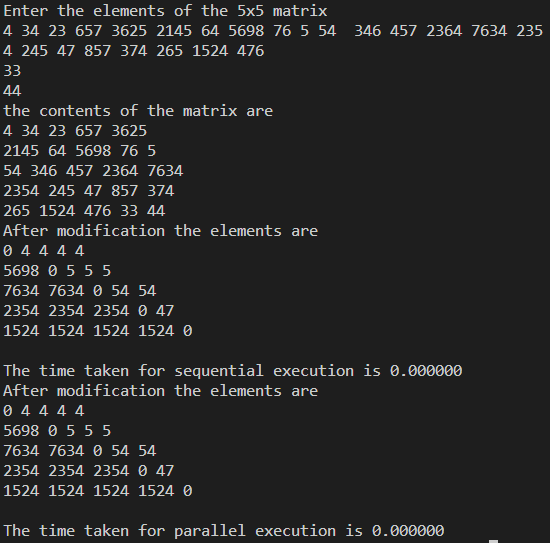
printf("\n");

}

printf("\nThe time taken for parallel execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**OUTPUT:**



Q8) 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.

Example:

A

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 6 | 5 | 8 | 3 |
| 2 | 4 | 10 | 1 |
| 9 | 1 | 2 | 5 |

B

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 6 | **10** | **111** | 3 |
| 2 | **11** | **101** | 1 |
| 9 | 1 | 2 | 5 |

D

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 6 | **2** | **7** | 3 |
| 2 | **3** | **5** | 1 |
| 9 | 1 | 2 | 5 |

**CODE:**

#include<stdio.h>

#include<math.h>

#include<stdbool.h>

#include<omp.h>

#include<time.h>

int ones(int num){

int val = 1;

while(num!=0){

if(num%2==1){

val = val\*10+1;

}

else{

val = val\*10+0;

}

num/=2;

}

int cpy = val;

int len = 0;

while(cpy!=0){

len++;

cpy = cpy/10;

}

// printf("length %d\n",len);

int i;

int ten = 10;

for(i=1;i<len-1;i++){

ten\*=10;

}

// printf("10 raised %d\n",ten);

// printf("moded %d\n",val%ten);

int rev=0;

int len1 = len;

while(len>1){

int dig = val%10;

rev = rev\*10 + dig;

val = val/10;

len--;

}

// printf("reversed num %d\n",rev);

int cpynum = rev;

int arr[len1-1];

for(i=len1-2;i>=0;i--){

arr[i] = cpynum%10;

cpynum/=10;

}

for(i=0;i<len1-1;i++){

if(arr[i]==0){

arr[i] = 1;

}

else{

arr[i] = 0;

}

}

bool flag = false;

int corans=0;

for(i=0;i<len1-1;i++){

if(arr[i]==1){

corans = corans\*10+1;

flag=true;

}

else if(arr[i]==0 && flag){

flag=true;

corans = corans\*10+0;

}

}

// printf("%d",corans);

return corans;

}

int deci(int num){

int cpy = num;

double val=0;

int count=0;

while(cpy!=0){

int dig = cpy%10;

if(dig==1){

val+=pow(2,count);

}

cpy/=10;

count++;

}

// printf("decimal value = %d",(int)val);

return (int)val;

}

int main(){

printf("Enter the dimension of the matrix");

int r1,c1;

scanf("%d%d",&r1,&c1);

int arr1[r1][c1];

int i,j;

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

scanf("%d",&arr1[i][j]);

}

}

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

printf("%d ",arr1[i][j]);

}

printf("\n");

}

printf("\n");

int arr2[r1][c1];

clock\_t start,end;

start = clock();

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

if((i>0 && i<r1-1) && (j>0 && j<c1-1)){

arr2[i][j] = ones(arr1[i][j]);

}

else{

arr2[i][j] = arr1[i][j];

}

}

}

printf("\n");

int arr3[r1][c1];

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

if((i!=0 && i!=r1-1) && (j!=0 && j!=c1-1)){

arr3[i][j] = deci(arr2[i][j]);

}

else{

arr3[i][j] = arr2[i][j];

}

}

}

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

printf("%d ",arr2[i][j]);

}

printf("\n");

}

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

printf("%d ",arr3[i][j]);

}

printf("\n");

}

end = clock();

printf("Time taken in sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

int arr4[r1][c1];

clock\_t start1,end1;

start1 = clock();

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

if((i>0 && i<r1-1) && (j>0 && j<c1-1)){

arr4[i][j] = ones(arr1[i][j]);

}

else{

arr4[i][j] = arr1[i][j];

}

}

}

printf("\n");

int arr5[r1][c1];

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

if((i!=0 && i!=r1-1) && (j!=0 && j!=c1-1)){

arr5[i][j] = deci(arr4[i][j]);

}

else{

arr5[i][j] = arr4[i][j];

}

}

}

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

printf("%d ",arr4[i][j]);

}

printf("\n");

}

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

printf("%d ",arr5[i][j]);

}

printf("\n");

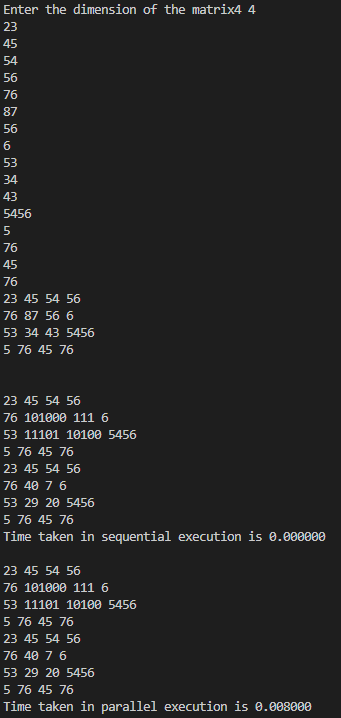
}

end1 = clock();

printf("Time taken in parallel execution is %f\n",(double)(end1-start1)/CLOCKS\_PER\_SEC);

}

**OUTPUT:**



**Tparallel = 0.008**

**Tsequential = 0**

**Speedup = 0**

**Efficiency = 0**

*Q9) 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).*

*Example: A B*

*p C a P 1 2 4 3*

*e X a M 2 4 3 2*

*Output string STR: pCCaaaaPPPeeXXXXaaaMM*

**CODE:**

#include<stdio.h>

#include<time.h>

#include<omp.h>

int main(){

printf("Enter the dimension of character matrix");

int r1,c1;

clock\_t start,end;

scanf("%d%d",&r1,&c1);

char arr1[r1][c1];

int i,j;

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

scanf(" %c",&arr1[i][j]);

}

}

printf("Enter the elements of number matrix");

int arr2[r1][c1];

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

scanf("%d",&arr2[i][j]);

}

}

start = clock();

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

int count = arr2[i][j];

while(count>0){

printf("%c",arr1[i][j]);

count--;

}

}

}

end = clock();

printf("\nThe time taken in sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

start = clock();

int cnt;

#pragma omp parallel for

for(cnt=0;cnt<1;cnt++){

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

int count = arr2[i][j];

while(count>0){

printf("%c",arr1[i][j]);

count--;

}

}

}

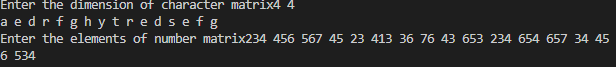
}

end = clock();

printf("\nThe time taken in parallel execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**OUTPUT:**





**Tparallel = 0.353**

**Tsequential = 0.402**

**Speedup = 1.138**

**Efficiency = 0.1423**

***Week 3: C programming using OpenMP directives.***

*Q1)Write an OpenMP program to implement Matrix multiplication. a. Analyse 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<time.h>

int main(){

int mat[] = {200,400,600,800,1000};

int proc[] = {2,4,6,8};

for(int x=0;x<5;x++){

for(int y=0;y<4;y++){

clock\_t start,end,start1,end1;

int r1,c1,r2,c2;

r1=mat[x];

c1=mat[x];

r2=mat[x];

c2=mat[x];

// printf("Enter the dimensions of matrix A ");

// scanf("%d %d",&r1,&c1);

// printf("Enter the dimensions of matrix B ");

// scanf("%d %d",&r2,&c2);

int (\*arr1)[c1]= malloc(r1 \* c1 \* sizeof (int));

int (\*arr2)[c2]= malloc(r2 \* c2 \* sizeof (int));

int (\*res)[c2]= malloc(r1 \* c2 \* sizeof (int));

int i,j,k;

long int cnt = 1;

start = clock(); //start point for sequential

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

arr1[i][j] = cnt;

cnt++;

}

}

for(i=0;i<r2;i++){

for(j=0;j<c2;j++){

arr2[i][j] = cnt;

cnt++;

}

}

for(i=0;i<r1;i++){

for(j=0;j<c2;j++){

res[i][j]=0;

}

}

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

for(k=0;k<c2;k++){

res[i][k]+=arr1[i][j]\*arr2[j][k];

}

}

}

end = clock(); //end point for sequentṇialṇ

printf("\n");

// for(i=0;i<r1;i++){

// for(j=0;j<c2;j++){

// printf("%d ",res[i][j]);

// }

// printf("\n");

// }

int n = proc[y];

for(i=0;i<r1;i++){

for(j=0;j<c2;j++){

res[i][j]=0;

}

}

cnt=1;

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

arr1[i][j] = cnt;

cnt++;

}

}

for(i=0;i<r2;i++){

for(j=0;j<c2;j++){

arr2[i][j] = cnt;

cnt++;

}

}

for(i=0;i<r1;i++){

for(j=0;j<c2;j++){

res[i][j]=0;

}

}

start1 = clock(); //start point for parallel

#pragma omp parallel for collapse(3) num\_threads(n)

for(i=0;i<r1;i++){

for(j=0;j<c1;j++){

for(k=0;k<c2;k++){

res[i][k]+=arr1[i][j]\*arr2[j][k];

}

}

}

end1 = clock(); //end point for parallel

printf("\nshape=%dx%d and number of threads = %d\n",r1,c1,n);

printf("Time taken in sequential execution %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

printf("Time taken in parallel execution %f\n",(double)(end1-start1)/CLOCKS\_PER\_SEC);

}

}

return 0;

}

**OUTPUT:**

shape=200x200 and number of threads = 2

Time taken in sequential execution 0.036000

Time taken in parallel execution 0.030000

shape=200x200 and number of threads = 4

Time taken in sequential execution 0.038000

Time taken in parallel execution 0.036000

shape=200x200 and number of threads = 6

Time taken in sequential execution 0.025000

Time taken in parallel execution 0.033000

shape=200x200 and number of threads = 8

Time taken in sequential execution 0.032000

Time taken in parallel execution 0.023000

shape=400x400 and number of threads = 2

Time taken in sequential execution 0.201000

Time taken in parallel execution 0.199000

shape=400x400 and number of threads = 4

Time taken in sequential execution 0.200000

Time taken in parallel execution 0.198000

shape=400x400 and number of threads = 6

Time taken in sequential execution 0.214000

Time taken in parallel execution 0.185000

shape=400x400 and number of threads = 8

Time taken in sequential execution 0.201000

Time taken in parallel execution 0.204000

shape=600x600 and number of threads = 2

Time taken in sequential execution 0.763000

Time taken in parallel execution 0.660000

shape=600x600 and number of threads = 4

Time taken in sequential execution 0.660000

Time taken in parallel execution 0.740000

shape=600x600 and number of threads = 6

Time taken in sequential execution 0.658000

Time taken in parallel execution 0.668000

shape=600x600 and number of threads = 8

Time taken in sequential execution 0.661000

Time taken in parallel execution 0.747000

shape=800x800 and number of threads = 2

Time taken in sequential execution 1.595000

Time taken in parallel execution 1.601000

shape=800x800 and number of threads = 4

Time taken in sequential execution 1.640000

Time taken in parallel execution 1.600000

shape=800x800 and number of threads = 6

Time taken in sequential execution 1.604000

Time taken in parallel execution 1.683000

shape=800x800 and number of threads = 8

Time taken in sequential execution 1.691000

Time taken in parallel execution 1.596000

shape=1000x1000 and number of threads = 2

Time taken in sequential execution 3.305000

Time taken in parallel execution 3.246000

shape=1000x1000 and number of threads = 4

Time taken in sequential execution 3.205000

Time taken in parallel execution 3.148000

shape=1000x1000 and number of threads = 6

Time taken in sequential execution 3.282000

Time taken in parallel execution 3.148000

shape=1000x1000 and number of threads = 8

Time taken in sequential execution 3.179000

Time taken in parallel execution 3.161000

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | threads=2 | | threads=4 | | threads=6 | | threads=8 | |
|  | Tp | Ts | Tp | Ts | Tp | Ts | Tp | Ts |
| 200x200 | 0.039 | 0.023 | 0.032 | 0.023 | 0.021 | 0.023 | 0.025 | 0.024 |
| 400x400 | 0.19 | 0.199 | 0.181 | 0.229 | 0.244 | 0.19 | 0.19 | 0.205 |
| 600x600 | 0.663 | 0.669 | 0.662 | 0.719 | 0.697 | 0.654 | 0.641 | 0.675 |
| 800x800 | 1.64 | 1.669 | 1.63 | 1.563 | 1.614 | 1.591 | 1.629 | 1.57 |
| 1000x1000 | 3.192 | 3.059 | 3.209 | 3.057 | 3.155 | 3.049 | 3.13 | 3.15 |
|  |  |  |  |  |  |  |  |  |
|  | Speedup | 0.589744 |  | 0.71875 |  | 1.095238 |  | 0.96 |
|  |  | 1.047368 |  | 1.265193 |  | 0.778689 |  | 1.078947 |
|  |  | 1.00905 |  | 1.086103 |  | 0.938307 |  | 1.053042 |
|  |  | 1.017683 |  | 0.958896 |  | 0.98575 |  | 0.963781 |
|  |  | 0.958333 |  | 0.952633 |  | 0.966403 |  | 1.00639 |
|  |  |  |  |  |  |  |  |  |
|  | efficiency | 0.3345 |  | 0.179688 |  | 0.18254 |  | 0.12 |
|  |  | 0.523684 |  | 0.316298 |  | 0.129781 |  | 0.134868 |
|  |  | 0.504525 |  | 0.271526 |  | 0.156385 |  | 0.13163 |
|  |  | 0.508841 |  | 0.239724 |  | 0.164292 |  | 0.120473 |
|  |  | 0.479167 |  | 0.317544 |  | 0.161067 |  | 0.125799 |

*Q2) Write an OpenMP program to perform Matrix times vector multiplication. Vary the matrix and vector size and analyse the speedup and efficiency of the parallelized code.*

**CODE:**

#include<stdio.h>

#include<omp.h>

#include<time.h>

int main(){

    clock\_t start1,end1,start,end;

    printf("Enter the dimension of the matrix");

    int r1,c1;

    int i,j,k;

    scanf("%d%d",&r1,&c1);

    int(\*arr1)[c1] = malloc(r1\*c1\*sizeof(int));

    printf("Enter elements");

    int cnt=1;

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            arr1[i][j] = cnt;

            cnt++;

        }

    }

    printf("\n");

    for(i=0;i<r1;i++){

        for(j=0;j<c1;j++){

            printf("%d ",arr1[i][j]);

        }

        printf("\n");

    }

    printf("options:\n1.Row vector\n2.Columnar vector\n");

    int ch;

    int dim;

    scanf("%d",&ch);

    if(ch==1){

        printf("Enter the number of columns for the row vector ");

        scanf("%d",&dim);

    }

    else{

        printf("Enter the number of rows for the column vector ");

        scanf("%d",&dim);

    }

    int \*vec = (int \*)malloc(dim\*sizeof(int));

    for(i=0;i<dim;i++){

        vec[i] = cnt;

        cnt++;

    }

    printf("\n");

    for(i=0;i<dim;i++){

        printf("%d ",vec[i]);

    }

    int \*res= malloc(c1\*sizeof(int));

    for(i=0;i<dim;i++){

        res[i] = 0;

    }

    if(ch==1){

        start = clock();

        for(i=0;i<dim;i++){

            for(k=0;k<c1;k++){

                res[i]+=vec[k]\*arr1[k][i];

            }

        }

        end = clock();

    }

    else{

        start = clock();

        for(i=0;i<c1;i++){

            for(k=0;k<dim;k++){

                res[i]+=arr1[i][k]\*vec[k];

            }

        }

        end = clock();

    }

    printf("\n");

    if(ch==2){

        for(i=0;i<c1;i++){

            printf("%d\n",res[i]);

        }

    }

    else{

        for(i=0;i<c1;i++){

            printf("%d ",res[i]);

        }

    }

    double time1 = (end-start)/CLOCKS\_PER\_SEC;

    for(i=0;i<dim;i++){

        res[i] = 0;

    }

    if(ch==1){

        start1 = clock();

        #pragma omp parallel for collapse(2) num\_threads(4)

        for(i=0;i<dim;i++){

            for(k=0;k<c1;k++){

                res[i]+=vec[k]\*arr1[k][i];

            }

        }

        end1 = clock();

    }

    else{

        start1 = clock();

        #pragma omp parallel for collapse(2) num\_threads(4)

        for(i=0;i<c1;i++){

            for(k=0;k<dim;k++){

                res[i]+=arr1[i][k]\*vec[k];

            }

        }

        end1 = clock();

    }

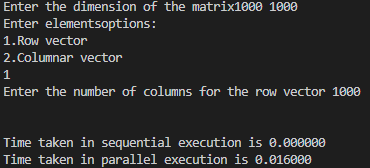
    printf("\n");

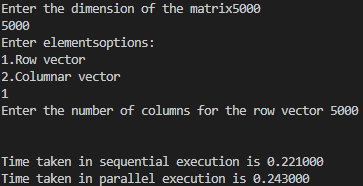
    printf("Time taken in sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

    printf("Time taken in parallel execution is %f\n",(double)(end1-start1)/CLOCKS\_PER\_SEC);

}

**OUTPUT:**





**Tparallel = 0.243**

**Tsequential = 0.221**

**Speedup = 0.909**

**Efficiency = 0.113**

*Q3) 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<time.h>

#include<omp.h>

#include<limits.h>

int main(){

    clock\_t start,end,start1,end1;

    printf("Enter the elements of the 5x5 matrix");

    int arr[5][5],arr2[5][5];

    for(int i=0;i<5;i++){

        for(int j=0;j<5;j++){

            scanf("%d",&arr[i][j]);

        }

    }

    start = clock();

    for(int i=0;i<5;i++){

        for(int j=0;j<5;j++){

            if(i!=j && i>j){

                for(int z=0;z<9999999;z++){

                    int max=INT\_MIN;

                    for(int k=0;k<5;k++){

                        if(arr[i][k]>max){

                            max = arr[i][k];

                        }

                    }

                    arr2[i][j] = max;

                }

            }

            else if(i!=j && i<j){

                for(int z=0;z<9999999;z++){

                    int min=INT\_MAX;

                    for(int k=0;k<5;k++){

                        if(arr[i][k]<min){

                            min = arr[i][k];

                        }

                    }

                    arr2[i][j] = min;

                }

            }

            else{

                arr2[i][j] = 0;

            }

        }

    }

    end = clock();

    for(int i=0;i<5;i++){

        for(int j=0;j<5;j++){

            printf("%d ",arr2[i][j]);

        }

        printf("\n");

    }

    printf("\nthe time taken in sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

    start = clock();

    #pragma omp parallel for collapse(2) num\_threads(4)

    for(int i=0;i<5;i++){

        for(int j=0;j<5;j++){

            if(i!=j && i>j){

                for(int z=0;z<9999999;z++){

                    int max=INT\_MIN;

                    for(int k=0;k<5;k++){

                        if(arr[i][k]>max){

                            max = arr[i][k];

                        }

                    }

                    arr2[i][j] = max;

                }

            }

            else if(i!=j && i<j){

                for(int z=0;z<9999999;z++){

                    int min=INT\_MAX;

                    for(int k=0;k<5;k++){

                        if(arr[i][k]<min){

                            min = arr[i][k];

                        }

                    }

                    arr2[i][j] = min;

                }

            }

            else{

                arr2[i][j] = 0;

            }

        }

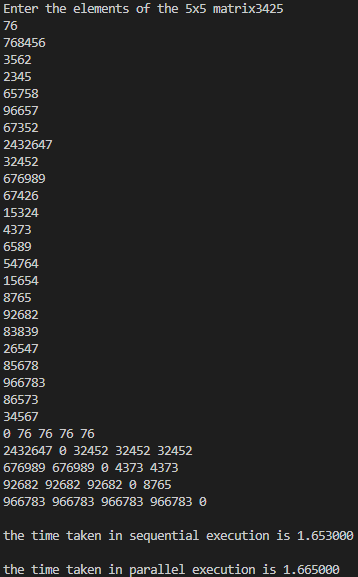
    }

    end = clock();

    printf("\nthe time taken in parallel execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**OUTPUT:**



**Tparallel = 1.665**

**Tsequential = 1.653**

**Speedup=0.992**

**Efficiency = 0.248**

*Q4) 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.*

*Example:*

*A*

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| *6* | *5* | *8* | *3* |
| *2* | *4* | *10* | *1* |
| *9* | *1* | *2* | *5* |

*B*

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| *6* | ***10*** | ***111*** | *3* |
| *2* | ***11*** | ***101*** | *1* |
| *9* | *1* | *2* | *5* |

*D*

|  |  |  |  |
| --- | --- | --- | --- |
| *1* | *2* | *3* | *4* |
| *6* | ***2*** | ***7*** | *3* |
| *2* | ***3*** | ***5*** | *1* |
| *9* | *1* | *2* | *5* |

**CODE:**

#include<time.h>

#include<math.h>

#include<stdio.h>

int main(){

printf("Enter the dimensions of the matrix");

int r1,c1;

clock\_t start,end,start1,end1;

scanf("%d%d",&r1,&c1);

int (\*arr)[c1] = malloc(sizeof(int)\*c1\*r1);

int (\*arr1)[c1] = malloc(sizeof(int)\*c1\*r1);

char bin[10000];

int assign=1;

for(int i=0;i<r1;i++){

for(int j=0;j<c1;j++){

arr[i][j] = assign++;

}

}

start = clock();

for(int i=0;i<r1;i++){

for(int j=0;j<c1;j++){

if((i>0 && i<r1-1) && (j>0 && j<c1-1)){

char dig[10000];

int cnt=0;

int num = arr[i][j];

while(num!=0){

int r = num%2;

if(r==1){

dig[cnt] = '0';

}

else{

dig[cnt] = '1';

}

cnt++;

num = num/2;

}

int number=0;

for(int a=cnt-1;a>=0;a--){

if(dig[a]=='1'){

number = number\*10+1;

}

else{

number = number\*10;

}

}

int counter=0;

int newnum = 0;

arr[i][j] = number;

while(number!=0){

int d = number%10;

if(d==1){

newnum+=pow(2,counter);

counter++;

}

else{

counter++;

}

number/=10;

}

arr1[i][j] = newnum;

}

else{

arr1[i][j] = arr[i][j];

}

}

}

end = clock();

start1 = clock();

#pragma omp parallel for collapse(2) num\_threads(4)

for(int i=0;i<r1;i++){

for(int j=0;j<c1;j++){

if((i>0 && i<r1-1) && (j>0 && j<c1-1)){

char dig[10000];

int cnt=0;

int num = arr[i][j];

while(num!=0){

int r = num%2;

if(r==1){

dig[cnt] = '0';

}

else{

dig[cnt] = '1';

}

cnt++;

num = num/2;

}

int number=0;

#pragma omp parallel for num\_threads(4)

for(int a=cnt-1;a>=0;a--){

if(dig[a]=='1'){

number = number\*10+1;

}

else{

number = number\*10;

}

}

int counter=0;

int newnum = 0;

arr[i][j] = number;

while(number!=0){

int d = number%10;

if(d==1){

newnum+=pow(2,counter);

counter++;

}

else{

counter++;

}

number/=10;

}

arr1[i][j] = newnum;

}

else{

arr1[i][j] = arr[i][j];

}

}

}

end1 = clock();

// for(int i=0;i<r1;i++){

// for(int j=0;j<c1;j++){

// printf("%d ",arr[i][j]);

// }

// printf("\n");

// }

// printf("\n");

// for(int i=0;i<r1;i++){

// for(int j=0;j<c1;j++){

// printf("%d ",arr1[i][j]);

// }

// printf("\n");

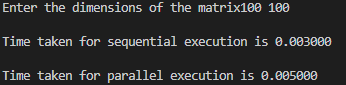
}

printf("\nTime taken for sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

printf("\nTime taken for parallel execution is %f\n",(double)(end1-start1)/CLOCKS\_PER\_SEC);

}

**OUTPUT:**



A black screen with white text

Description automatically generated

**Tparallel = 0.316**

**Tsequential = 0.216**

**Speedup = 0.683**

**Efficiency = 0.17**

*Q5) 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<time.h>

#include<omp.h>

#include<stdio.h>

int main(){

printf("Enter the elements ");

clock\_t start,end;

long int arr[9];

for(int i=0;i<9;i++){

scanf("%ld",&arr[i]);

}

for(int i=0;i<9;i++){

printf("%ld ",arr[i]);

}

printf("\n");

start = clock();

for(int a=0;a<9999999;a++){

for(int i=0;i<9;i++){

long int num = arr[i];

long int rev=0;

while(num!=0){

int dig = num%10;

rev = rev\*10+dig;

num /=10;

}

arr[i] =rev;

}

}

end = clock();

for(int i=0;i<9;i++){

printf("%ld ",arr[i]);

}

printf("\n The sequential time for sequential execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

#pragma omp parallel for num\_threads(4)

start = clock();

for(int a=0;a<9999999;a++){

for(int i=0;i<9;i++){

long int num = arr[i];

long int rev=0;

while(num!=0){

int dig = num%10;

rev = rev\*10+dig;

num /=10;

}

arr[i] =rev;

}

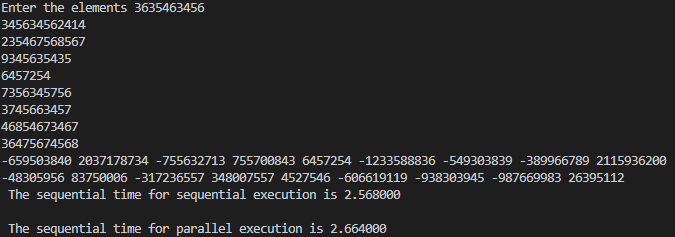
}

end = clock();

printf("\n The sequential time for parallel execution is %f\n",(double)(end-start)/CLOCKS\_PER\_SEC);

}

**OUTPUT:**



**Tparallel = 2.664**

**Tsequential = 2.568**

**Speedup = 0.9639**

**Efficiency = 0.24**

***WEEK4***

*Q1) 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:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include <time.h>

#include <math.h>

void selection\_sort(int arr[], int n) {

int i, j, min\_idx;

for (i = 0; i < n-1; i++) {

min\_idx = i;

#pragma omp parallel for shared(arr, min\_idx) private(j)

for (j = i+1; j < n; j++) {

if (arr[j] < arr[min\_idx]) {

min\_idx = j;

}

}

if (min\_idx != i) {

int temp = arr[i];

arr[i] = arr[min\_idx];

arr[min\_idx] = temp;

}

}

}

double get\_execution\_time(int arr[], int n, int num\_threads) {

omp\_set\_num\_threads(num\_threads);

double start\_time = omp\_get\_wtime();

selection\_sort(arr, n);

double end\_time = omp\_get\_wtime();

return end\_time - start\_time;

}

int main() {

int max\_threads = omp\_get\_max\_threads();

printf("Selection Sort using OpenMP\n\n");

printf("Number of threads: %d\n", max\_threads);

printf("\nInput size\tThreads\tExecution Time\tEfficiency\tSpeedup\n");

for (int n = 10000; n <= 100000; n += 10000) {

int arr[n];

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

arr[i] = rand() % n;

}

printf("%d\t\t%d\t", n, 1);

double seq\_time = get\_execution\_time(arr, n, 1);

printf("%f\t%f\t%f\n", seq\_time, 1.0, 1.0);

for (int num\_threads = 2; num\_threads <= max\_threads; num\_threads++) {

printf("%d\t\t%d\t", n, num\_threads);

double par\_time = get\_execution\_time(arr, n, num\_threads);

double efficiency = seq\_time / (num\_threads \* par\_time);

double speedup = seq\_time / par\_time;

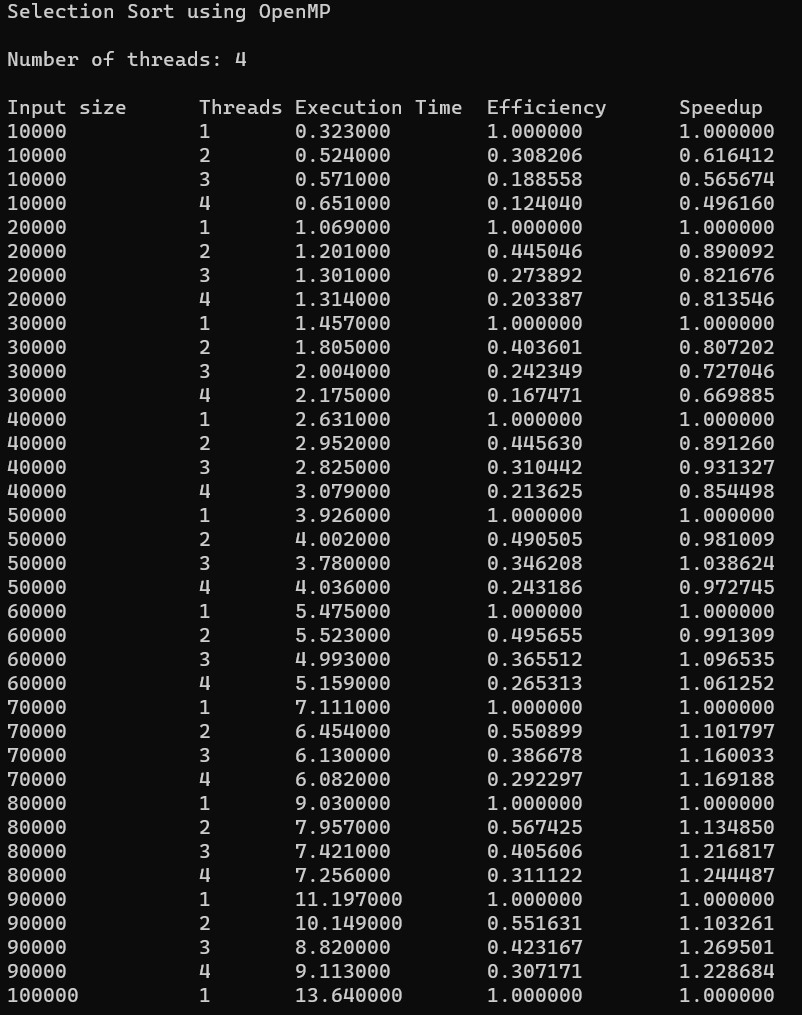
printf("%f\t%f\t%f\n", par\_time, efficiency, speedup);

}

}

return 0;

}

**OUTPUT:**A black screen with white text

Description automatically generated

*Q2) 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:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include <time.h>

#define MAX\_SIZE 100

// function to swap two elements in an array

void swap(int \*a, int \*b) {

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// function to perform insertion sort on an array

void insertion\_sort(int arr[], int n) {

int i, j, key;

for (i = 1; i < n; i++) {

key = arr[i];

j = i - 1;

// Move elements of arr[0..i-1], that are greater than key, to one position ahead of their current position

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

// function to perform quick sort on an array

void quick\_sort(int arr[], int low, int high) {

if (low < high) {

// Select a pivot element and partition the array around it

int pivot = arr[high];

int i = low - 1;

for (int j = low; j <= high - 1; j++) {

if (arr[j] < pivot) {

i++;

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

// Recursively sort the two partitions

#pragma omp task shared(arr)

quick\_sort(arr, low, i);

#pragma omp task shared(arr)

quick\_sort(arr, i + 2, high);

}

}

// function to merge two sorted arrays into a single sorted array

void merge(int arr[], int l, int m, int r) {

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

// create temporary arrays

int L[n1], R[n2];

// copy data to temporary arrays

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

// merge the two arrays

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++;

}

// copy the remaining elements of L[], if there are any

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

// copy the remaining elements of R[], if there are any

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

int main() {

int arr[MAX\_SIZE], n, i;

double start\_time, end\_time;

// get the size of the array from the user

printf("Enter the size of the array: ");

scanf("%d", &n);

// get the elements of the array from the user

printf("Enter the elements of the array:\n");

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

scanf("%d", &arr[i]);

// set the number of threads to use

omp\_set\_num\_threads(2);

// start timing

start\_time = omp\_get\_wtime();

// sort the first half of the array using insertion sort and the second half using quick sort

#pragma omp parallel shared(arr)

{

#pragma omp sections nowait

{

#pragma omp section

{

insertion\_sort(arr, n/2);

}

#pragma omp section

{

quick\_sort(arr, n/2 + 1, n-1);

}

}

}

// merge the two sorted halves of the array

merge(arr, 0, n/2, n-1);

// end timing

end\_time = omp\_get\_wtime();

// print the sorted array

printf("\nSorted array:\n");

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

printf("%d ", arr[i]);

printf("\n");

// calculate time taken

double execution\_time = end\_time - start\_time;

printf("\nTime taken: %f seconds\n", execution\_time);

// calculate speedup and efficiency assuming ideal speedup is 2 (doubling of threads)

double speedup = (execution\_time / (execution\_time / 2.0));

double efficiency = speedup / 2.0;

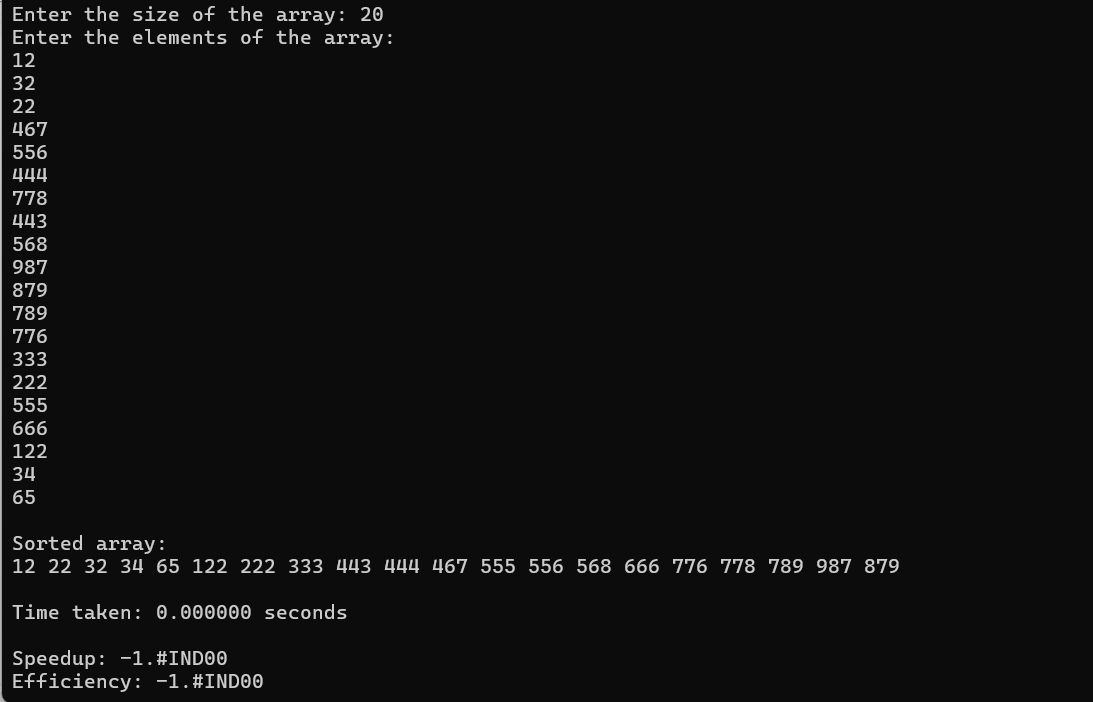
printf("\nSpeedup: %f\n", speedup);

printf("Efficiency: %f\n", efficiency);

return 0;

}

**OUTPUT:**



*Q3) 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:**

#include <iostream>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <omp.h>

using namespace std;

int sequentialSearch(vector<int>& arr, int key) {

for (size\_t i = 0; i < arr.size(); i++) {

if (arr[i] == key) {

return i;

}

}

return -1;

}

int main() {

const int maxSize = 1000000;

const int numThreads[] = {1, 2, 4, 8};

vector<int> arr(maxSize);

int key = 42;

srand(1);

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

arr[i] = rand() % 100;

}

std::cout << "Input Size, Threads, Time (seconds), Efficiency, Speedup\n";

for (int size = 1000; size <= maxSize; size \*= 10) {

for (int t = 0; t < sizeof(numThreads) / sizeof(numThreads[0]); t++) {

double start\_time, end\_time;

int result;

omp\_set\_num\_threads(numThreads[t]);

start\_time = omp\_get\_wtime();

#pragma omp parallel

{

#pragma omp single

{

result = sequentialSearch(arr, key);

}

}

end\_time = omp\_get\_wtime();

double elapsed\_time = end\_time - start\_time;

double efficiency = elapsed\_time / (numThreads[t] \* elapsed\_time);

double speedup = elapsed\_time / (1.0 \* elapsed\_time);

std::cout << size << ", " << numThreads[t] << ", " << elapsed\_time << ", " << efficiency << ", " << speedup << "\n";

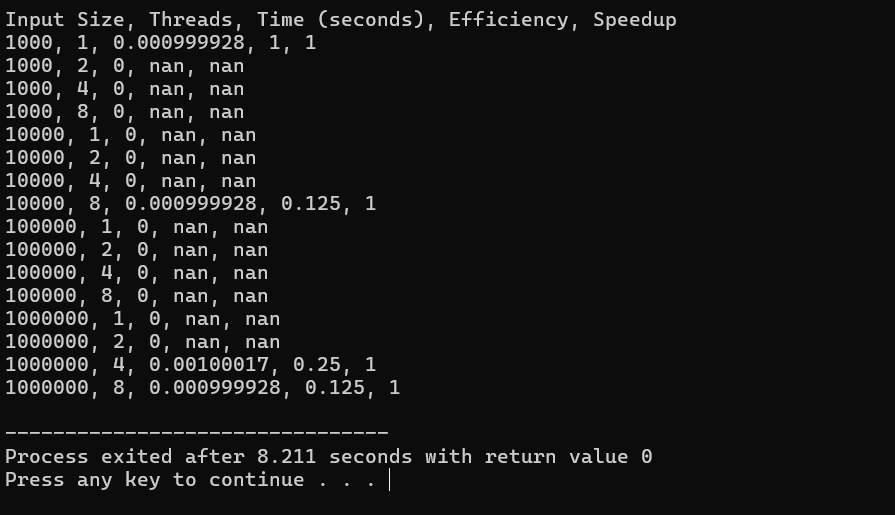
}

}

return 0;

}

**OUTPUT:**



***WEEK5***

*Q1) 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:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define N 1000000

int main() {

int i, num\_threads;

double start\_time, end\_time;

int \*a = (int\*) malloc(N \* sizeof(int));

int \*b = (int\*) malloc(N \* sizeof(int));

int \*c = (int\*) malloc(N \* sizeof(int));

int \*d = (int\*) malloc(N \* sizeof(int));

int \*e = (int\*) malloc(N \* sizeof(int));

// Initialize vectors

for (i = 0; i < N; i++) {

a[i] = i;

b[i] = N - i;

}

// Serial addition

start\_time = omp\_get\_wtime();

for (i = 0; i < N; i++) {

c[i] = a[i] + b[i];

}

end\_time = omp\_get\_wtime();

printf("Serial addition time: %f\n", end\_time - start\_time);

// Parallel addition

start\_time = omp\_get\_wtime();

#pragma omp parallel for

for (i = 0; i < N; i++) {

d[i] = a[i] + b[i];

}

end\_time = omp\_get\_wtime();

printf("Parallel addition time: %f\n", end\_time - start\_time);

// Serial subtraction

start\_time = omp\_get\_wtime();

for (i = 0; i < N; i++) {

e[i] = a[i] - b[i];

}

end\_time = omp\_get\_wtime();

printf("Serial subtraction time: %f\n", end\_time - start\_time);

// Parallel subtraction

start\_time = omp\_get\_wtime();

#pragma omp parallel for

for (i = 0; i < N; i++) {

e[i] = a[i] - b[i];

}

end\_time = omp\_get\_wtime();

printf("Parallel subtraction time: %f\n", end\_time - start\_time);

// Serial multiplication

start\_time = omp\_get\_wtime();

for (i = 0; i < N; i++) {

e[i] = a[i] \* b[i];

}

end\_time = omp\_get\_wtime();

printf("Serial multiplication time: %f\n", end\_time - start\_time);

// Parallel multiplication

start\_time = omp\_get\_wtime();

#pragma omp parallel for

for (i = 0; i < N; i++) {

e[i] = a[i] \* b[i];

}

end\_time = omp\_get\_wtime();

printf("Parallel multiplication time: %f\n", end\_time - start\_time);

free(a);

free(b);

free(c);

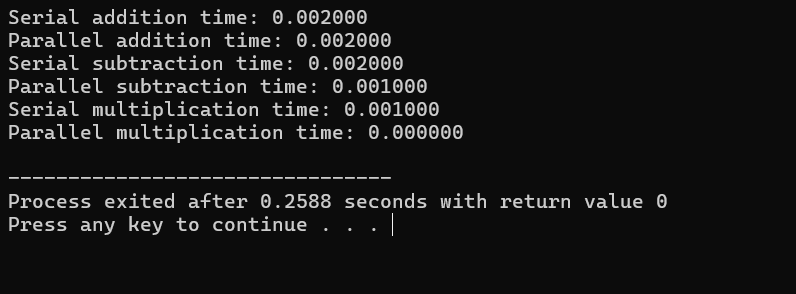
free(d);

free(e);

return 0;

}

**OUTPUT:**



*Q2) 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:**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#define N 10

int main() {

int i, sum = 0;

// Using parallel for with reduction

#pragma omp parallel for reduction(+:sum)

for (i = 0; i < N; i++) {

sum += i;

}

printf("Sum using parallel for with reduction: %d\n", sum);

sum = 0;

// Using critical section

#pragma omp parallel shared(sum)

{

int local\_sum = 0;

#pragma omp for

for (i = 0; i < N; i++) {

local\_sum += i;

}

#pragma omp critical

{

sum += local\_sum;

}

}

printf("Sum using critical section: %d\n", sum);

sum = 0;

// Using atomic

#pragma omp parallel shared(sum)

{

int local\_sum = 0;

#pragma omp for

for (i = 0; i < N; i++) {

#pragma omp atomic

sum += i;

}

}

printf("Sum using atomic: %d\n", sum);

sum = 0;

// Using locks

omp\_lock\_t lock;

omp\_init\_lock(&lock);

#pragma omp parallel shared(sum)

{

int local\_sum = 0;

#pragma omp for

for (i = 0; i < N; i++) {

local\_sum += i;

}

omp\_set\_lock(&lock);

sum += local\_sum;

omp\_unset\_lock(&lock);

}

omp\_destroy\_lock(&lock);

printf("Sum using locks: %d\n", sum);

sum = 0;

// Using master

#pragma omp parallel shared(sum)

{

#pragma omp master

{

printf("Running with %d threads\n", omp\_get\_num\_threads());

}

int local\_sum = 0;

#pragma omp for

for (i = 0; i < N; i++) {

local\_sum += i;

}

#pragma omp master

{

printf("Sum using master: %d\n", sum);

}

#pragma omp critical

{

sum += local\_sum;

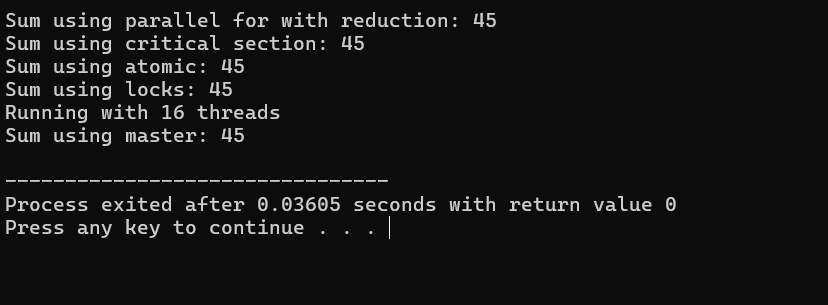
}

}

return 0;

}

**OUTPUT:**



*Q3) Write a parallel program using OpenMP to implement the Odd-even transposition sort. Vary the input size and analyse the program efficiency.*

**CODE:**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <omp.h>

using namespace std;

void oddEvenSort(int \*arr, int n) {

int i, j, temp;

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

if (i % 2 == 0) { // even phase

#pragma omp parallel for shared(arr) private(j, temp)

for (j = 2; j < n; j += 2) {

if (arr[j] < arr[j-1]) {

temp = arr[j];

arr[j] = arr[j-1];

arr[j-1] = temp;

}

}

} else { // odd phase

#pragma omp parallel for shared(arr) private(j, temp)

for (j = 1; j < n; j += 2) {

if (arr[j] < arr[j-1]) {

temp = arr[j];

arr[j] = arr[j-1];

arr[j-1] = temp;

}

}

}

}

}

int main() {

srand(time(NULL)); // seed random number generator

int n = 10; // change input size here

int \*arr = new int[n];

// populate array with random numbers

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

arr[i] = rand() % 10000;

}

double start = omp\_get\_wtime();

oddEvenSort(arr, n);

double end = omp\_get\_wtime();

cout << "Sorted array: ";

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

cout << arr[i] << " ";

}

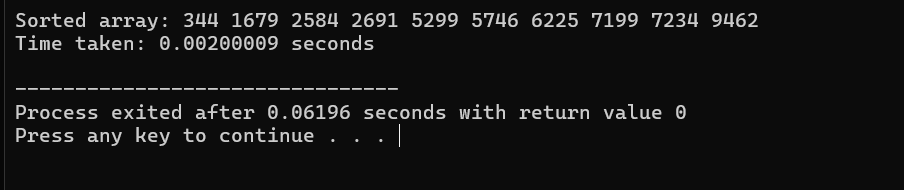
cout << endl;

cout << "Time taken: " << end - start << " seconds" << endl;

delete[] arr;

return 0;

}

**OUTPUT:**

*Q4) 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>

#include <omp.h>

#define INTERVAL\_START 1

#define INTERVAL\_END 100

#define NUM\_THREADS 4

int main() {

int sum = 0;

double start\_time, end\_time;

int chunk\_size = (INTERVAL\_END - INTERVAL\_START + 1) / NUM\_THREADS;

// Set the number of threads

omp\_set\_num\_threads(NUM\_THREADS);

// Sequential summation

start\_time = omp\_get\_wtime();

for (int i = INTERVAL\_START; i <= INTERVAL\_END; i++) {

sum += i;

}

end\_time = omp\_get\_wtime();

printf("Sequential Sum = %d, Time taken = %f seconds\n", sum, end\_time - start\_time);

// Parallel summation - static scheduling

sum = 0;

start\_time = omp\_get\_wtime();

#pragma omp parallel for schedule(static, chunk\_size) reduction(+:sum)

for (int i = INTERVAL\_START; i <= INTERVAL\_END; i++) {

sum += i;

}

end\_time = omp\_get\_wtime();

printf("Parallel Sum (Static Scheduling) = %d, Time taken = %f seconds\n", sum, end\_time - start\_time);

// Parallel summation - dynamic scheduling

sum = 0;

start\_time = omp\_get\_wtime();

#pragma omp parallel for schedule(dynamic, chunk\_size) reduction(+:sum)

for (int i = INTERVAL\_START; i <= INTERVAL\_END; i++) {

sum += i;

}

end\_time = omp\_get\_wtime();

printf("Parallel Sum (Dynamic Scheduling) = %d, Time taken = %f seconds\n", sum, end\_time - start\_time);

// Parallel summation - guided scheduling

sum = 0;

start\_time = omp\_get\_wtime();

#pragma omp parallel for schedule(guided, chunk\_size) reduction(+:sum)

for (int i = INTERVAL\_START; i <= INTERVAL\_END; i++) {

sum += i;

}

end\_time = omp\_get\_wtime();

printf("Parallel Sum (Guided Scheduling) = %d, Time taken = %f seconds\n", sum, end\_time - start\_time);

return 0;

}

**OUTPUT:**

A screenshot of a computer program

Description automatically generated

*Q5) 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 NUM\_BINS 10

#define BIN\_WIDTH 10

int main()

{

int A[1000]; // Assume A has 1000 elements

int histogram[NUM\_BINS] = {0}; // Initialize histogram bins to 0

int i, bin;

// Populate array A with random integers between 0 and 99

#pragma omp parallel for

for (i = 0; i < 1000; i++) {

A[i] = rand() % 100;

}

// Compute the histogram using OpenMP

#pragma omp parallel for private(bin)

for (i = 0; i < 1000; i++) {

bin = A[i] / BIN\_WIDTH; // Determine which bin the value belongs to

#pragma omp atomic

histogram[bin]++; // Increment the corresponding bin count atomically

}

// Print the histogram

for (bin = 0; bin < NUM\_BINS; bin++) {

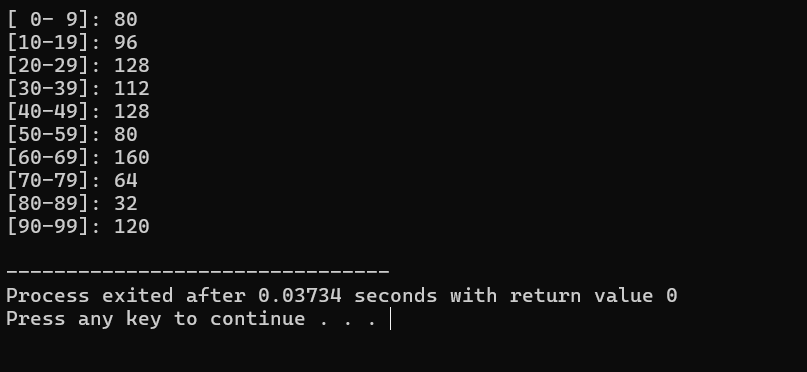
printf("[%2d-%2d]: %d\n", bin\*BIN\_WIDTH, (bin+1)\*BIN\_WIDTH-1, histogram[bin]);

}

return 0;

}

**OUTPUT:**



***Week 6: Programming using MPI***

*Q1) Write a simple MPI program to find out pow (x, rank) for all the processes where 'x' is the**integer constant and ‘rank’ is the rank of the process.*

**CODE:**

#include <stdio.h>

#include <math.h>

#include <mpi.h>

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

    MPI\_Init(&argc, &argv);

    int world\_size, world\_rank;

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

    // Assuming x is a constant value

    int x = 40;

    double result = pow(x, world\_rank);

    printf("Process %d: pow(%d, %d) = %f\n", world\_rank, x, world\_rank, result);

    MPI\_Finalize();

    return 0;

}

**OUTPUT:**

A black screen with white text

Description automatically generated

*Q2) Write a program in MPI where even ranked process prints "Hello" and odd ranked process**prints “World”.*

**CODE:**

#include <stdio.h>

#include <mpi.h>

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

    MPI\_Init(&argc, &argv);

    int world\_size, world\_rank;

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

    if (world\_rank % 2 == 0) {

        printf(" EVEN Hello from process %d\n", world\_rank);

    }

    else {

        printf("ODD World from process %d\n", world\_rank);

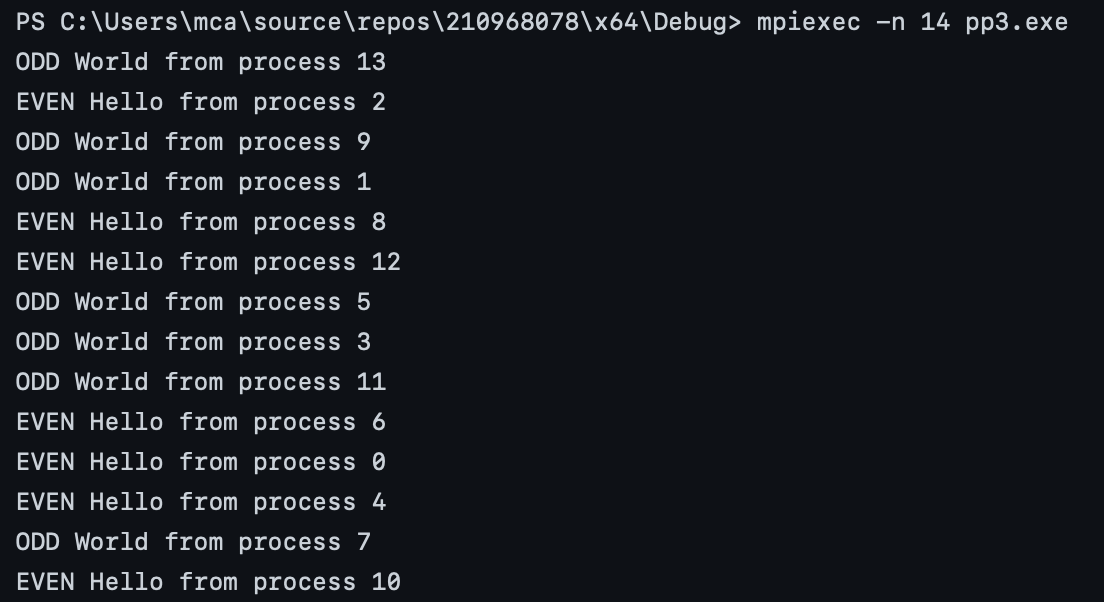
    }

    MPI\_Finalize();

    return 0;

}

**OUTPUT:**

****

*Q3)  Write a program in MPI to simulate simple calculator. Perform each operation using**different processes in parallel.*

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

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

    MPI\_Init(&argc, &argv);

    int world\_size, world\_rank;

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

    char operations[] = { '+', '-', '\*', '/' };

    char operation = operations[world\_rank % 4];

    double operand1 = 10.0, operand2 = 2.0;

    double result = 0.0;

    switch (operation) {

    case '+':

        result = operand1 + operand2;

        break;

    case '-':

        result = operand1 - operand2;

        break;

    case '\*':

        result = operand1 \* operand2;

        break;

    case '/':

        if (operand2 != 0) {

            result = operand1 / operand2;

        }

        else {

            fprintf(stderr, "Error: Division by zero.\n");

            MPI\_Abort(MPI\_COMM\_WORLD, 1);

        }

        break;

    default:

        fprintf(stderr, "Error: Unknown operation.\n");

        MPI\_Abort(MPI\_COMM\_WORLD, 1);

    }

    // Allocate memory dynamically for gathered\_results

    double\* gathered\_results = (double\*)malloc(world\_size \* sizeof(double));

    MPI\_Gather(&result, 1, MPI\_DOUBLE, gathered\_results, 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

    if (world\_rank == 0) {

        printf("Results:\n");

        for (int i = 0; i < world\_size; ++i) {

            printf("Process %d: %.2f %c %.2f = %.2f\n", i, operand1, operations[i % 4], operand2, gathered\_results[i]);

        }

    }

    // Free dynamically allocated memory

    free(gathered\_results);

    MPI\_Finalize();

    return 0;

}

**OUTPUT:**

**A screenshot of a computer

Description automatically generated**

*Q4) Write a program in MPI to toggle the character of a given string indexed by the rank of the**process*

*Hint: Suppose the string is HeLLO and there are 5 processes , then process 0 toggle ‘H’ to ‘h’, process 1 toggle ‘e’ to ‘E’ and so on.*

**CODE:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <mpi.h>

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

    MPI\_Init(&argc, &argv);

    int world\_size, world\_rank;

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &world\_size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &world\_rank);

    char input\_string[] = "HeLLO";

    // Calculate the portion of the string for each process

    int input\_length = strlen(input\_string);

    int local\_string\_size = input\_length / world\_size;

    char\* local\_string = (char\*)malloc(local\_string\_size + 1);

    // Check if the input string length is sufficient for the number of processes

    if (input\_length < world\_size) {

        fprintf(stderr, "Error: String length is less than the number of processes.\n");

        MPI\_Abort(MPI\_COMM\_WORLD, 1);

    }

    MPI\_Scatter(input\_string, local\_string\_size, MPI\_CHAR, local\_string, local\_string\_size, MPI\_CHAR, 0, MPI\_COMM\_WORLD);

    local\_string[local\_string\_size] = '\0'; // Null-terminate the local string

    // Toggle the character based on rank

    if (world\_rank % 2 == 0) {

        for (size\_t i = 0; i < strlen(local\_string); ++i) {

            if (local\_string[i] >= 'a' && local\_string[i] <= 'z') {

                local\_string[i] = local\_string[i] - 'a' + 'A'; // Toggle lowercase to uppercase

            }

        }

    }

    else {

        for (size\_t i = 0; i < strlen(local\_string); ++i) {

            if (local\_string[i] >= 'A' && local\_string[i] <= 'Z') {

                local\_string[i] = local\_string[i] - 'A' + 'a'; // Toggle uppercase to lowercase

            }

        }

    }

    // Gather the modified strings at process 0

    char\* gathered\_string = NULL;

    if (world\_rank == 0) {

        gathered\_string = (char\*)malloc(input\_length + 1);

    }

    MPI\_Gather(local\_string, local\_string\_size, MPI\_CHAR, gathered\_string, local\_string\_size, MPI\_CHAR, 0, MPI\_COMM\_WORLD);

    // Null-terminate the gathered string

    if (world\_rank == 0) {

        gathered\_string[input\_length] = '\0';

        printf("Original String: %s\n", input\_string);

        printf("Modified String: %s\n", gathered\_string);

        free(gathered\_string); // Don't forget to free the allocated memory

    }

    free(local\_string); // Don't forget to free the allocated memory

    MPI\_Finalize();

    return 0;

}

**OUTPUT:**

A black background with white text

Description automatically generated