

Task-1 [10 Marks]

(a) Consider the following C codes and perform dependence analysis (*Dependence type, Distance vectors, Direction Vectors*) for all the dependencies (both loop-carried and loop-independent). [4 Marks]

(b) Change the code (if required, using valid transformations only) and parallelize it using OpenMP. [4 Marks]

(c) Execute using 2, 4, and 8 processes. Compare the performance of the parallelized version with the original code version. Draw the speedup graph and explain the obtained results [2 Marks].

```
#define N 1024

int main()
{
    int i, k=10;
    int a[10]={0,1,2,3,4,5,6,7,8,9};
    int c[1000];
    int b[N][N];
    int loc=-1;
    int tmp=-1;

    for(i=0; i<k; i++)
        b[i][k]=b[a[i]][k];

    printf("%d %d", a[0], b[0][0]);

    for(i=0; i<1000; i++)
    {
        tmp = tmp+1;
        c[i]=tmp;
    }

    for(i=0; i<1000; i++)
        if (c[i]%4==0)
            loc = i;

    return 0;
}
```

Task-2 [10 Marks]

(a) Consider the following C codes and perform dependence analysis (*Dependence type, Distance vectors, Direction Vectors*) for all the dependencies (both loop-carried and loop-independent). [4 Marks]

(b) Change the code (if required, using valid transformations only) and parallelize it using OpenMP. [4 Marks]

(c) Execute using 2, 4, and 8 processes. Compare the performance of the parallelized version with the original code version. Draw the speedup graph and explain the obtained results [2 Marks].

```
#include <stdio.h>
#define N 800

int function_call(int j) {
    int a;
    a=2*2+j;
    return a;
}

int main()
{
    int i,j;
    int a[N][N];
    int b[N][N];
    int c[N][N];

    for(i=1;i<=N;i++)
        for(j=0;j<N;j++)
            b[i-1][j]=function_call(j);

    for(j=0;j<N-10;j++)
        for(i=0;i<N-10;i++)
        {
            a[i][j+2] = b[i+2][j];
            c[i+1][j] = b[i][j+3];
        }

    return 0;
}
```

Task-3 [10 Marks]

(a) Consider the following C codes and perform dependence analysis (*Dependence type, Distance vectors, Direction Vectors*) for all the dependencies (both loop-carried and loop-independent). [4 Marks]

(b) Change the code (if required, using valid transformations only) and parallelize it using OpenMP. [4 Marks]

(c) Execute using 2, 4, and 8 processes. Compare the performance of the parallelized version with the original code version. Draw the speedup graph and explain the obtained results [2 Marks].

```
#define N 1024

int main()
{
    int i,j; int
    X[N][N];
    int Y[N];
    int Z[N];

    int k=1;
    for(i=0;i<N;i++)
    {
        Y[i] = k;
        k = k*2;
        Z[i]=-1;

        for(j=0;j<N;j++)
            X[i][j]=2;
    }

    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
            Z[i] += Y[j] + X[i][j];

    return 0;
}
```

Task-4 [10 Marks]

(a) Improve the following code (using valid transformations and OpenMP constructs). [6 Marks]

(b) Compare the results in terms of **Speedup** of the improved and original code using 1, 2, 4, 8 processes. You have to submit both the original code, improved code, and the speedup graph showing the performance difference (if any).

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

int main()
{
    int m=4,n=16384;
    int i,j;
    double a[m][n];
    double s[m];

    #pragma omp parallel for private(i,j), shared(s,a)
    for (i=0;i<m;i++)
    {
        s[i]=1.0;
        for(j=0;j<n;j++)
            s[i]=s[i]+a[i][j];
    }
    printf("%f",s[1]);
}
```