**HPC EXTERNAL LAB ANSWERS**

**Ques 1-a Write an OpenMP program to perform addition of two arrays A & B store the result in the array C (Using Scheduling concept)**

#include<stdio.h>

#include<omp.h>

#define N 10

#define CHUNK 1 int main()

{

int A[N],B[N],C[N]; int i,tid; omp\_set\_num\_threads(10);

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

A[i] = B[i] = 1; C[i] = 0;

}

#pragma omp parallel for schedule(static,CHUNK)

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

{

C[i] = A[i] + B[i];

printf("Thread %d did C[%d] = %d\n",omp\_get\_thread\_num(),i,C[i]);

}

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

printf("C[%d] = %d\n",i,C[i]);

}

}

**Ques1-b Write a CUDA program to print the message “Hello World” and demonstrate**

**blocks by varying NUM\_BLOCKS to different sizes.**

#include<stdio.h>

#define BLOCKS 10

#define THREADS 2

global\_\_ void hello()

{

printf("This is thread %d of block %d with a block dim of

%d\n",threadIdx.x,blockIdx.x,blockDim.x);

}

int main()

{

hello<<<BLOCKS,THREADS>>>(); cudaDeviceSynchronize(); printf("Done\n");

return 0;

}

**Ques2-a Write an OpenMP program which performs C=A+B & D=A-B in separate blocks/sections where A,B,C & D are arrays.**

#include<stdio.h>

#include<omp.h>

#define N 10 int main()

{

int A[N],B[N],C[N],D[N];

int tid,n\_threads,i,j;

//init

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

{

A[i] = B[i] = i; C[i] = D[i] = 0;

}

#pragma omp parallel shared(A,B,C,D,n\_threads) private(i,j,tid)

{

tid = omp\_get\_thread\_num(); n\_threads = omp\_get\_num\_threads(); if(tid == 0)

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

#pragma omp sections nowait

{

#pragma omp section

{

printf("Thread %d doing section 1\n",tid);

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

C[i] = A[i] + B[i];

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

}

#pragma omp section

{

printf("Thread %d doing section 2\n",tid);

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

D[i] = A[i] - B[i];

}

}

printf("Thread %d done\n",tid);

}

printf("Finally:-\n");

printf("C is \n");

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

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

printf("\nD is \n");

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

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

}

**Ques2-b Write a MPI program to send the message from a process whose rank=3 to all other remaining processes.**

#include<stdio.h>

#include<mpi.h>

#include<stdlib.h>

#include<string.h>

#define BUFF\_SIZE 100

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

{

int rank,size,i,master=3; char msg[BUFF\_SIZE]; MPI\_Status status; MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank); MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

if(rank == master){

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

}

}

else

{

strcpy(msg,"Hello msrit");

if(i==master)

continue; MPI\_Send(&msg,BUFF\_SIZE,MPI\_CHAR,i,i,MPI\_COMM\_WORLD);

MPI\_Recv(&msg,BUFF\_SIZE,MPI\_CHAR,master,rank,MPI\_COMM\_WORLD,&status);

printf("%d recieved %s\n",rank,msg);

} MPI\_Finalize();

}

**Ques3-a Write an OpenMP program which demonstrates the usage of Critical Directive.**

#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

#define MAXIMUM 65536

int main()

{

int \*array, i, n\_elements, cur\_max, current\_value;

printf("Enter the number of elements\n");

scanf("%d", &n\_elements);

if (n\_elements <= 0) {

printf("The array elements cannot be stored\n");

exit(1);

}

array = (int\*)malloc(sizeof(int)\*n\_elements);

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

array[i] = i;

cur\_max = 0;

#pragma omp parallel for for(i=0;i<n\_elements;i++)

{

#pragma omp critical if(array[i] > cur\_max)

cur\_max = array[i];

}

current\_value = 0;

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

{

if(array[i] > current\_value)

current\_value = array[i];

}

printf("%d %d\n",cur\_max,current\_value);

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

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

printf("\n");

}

**Ques3-b Write a CUDA program to demonstrate different types of memory**

#include<stdio.h>

#define THREADS 10

#define BLOCKS 10

global\_\_ void local\_mem(float a)

{

float b;

b = a;

}

global\_\_ void global\_mem(int \*arr)

{

arr[blockIdx.x\*blockDim.x + threadIdx.x] = 2\*threadIdx.x;

}

global\_\_ void shared\_mem(int \*arr)

{

shared int shared\_arr[THREADS]; shared\_arr[threadIdx.x] = arr[threadIdx.x]; int i;

syncthreads();

int sum = 0; float average; for(i=0;i<threadIdx.x;i++)

{

sum+=shared\_arr[i];

}

average = (float)sum/(float)threadIdx.x;

if(arr[threadIdx.x] > average) arr[threadIdx.x] = average;

printf("thread id %d sum is %d average is %0.2f\n",threadIdx.x,sum,average);

}

int main()

{

float a = 0.2;

int \*arr,size,i;

size\_t n = 20;

size = sizeof(int)\*n;

arr = (int \*)malloc(size); int \*d\_arr; cudaMalloc(&d\_arr,size);

local\_mem<<<BLOCKS,THREADS>>>(a);

global\_mem<<<BLOCKS,THREADS>>>(d\_arr);

cudaMemcpy(arr,d\_arr,size,cudaMemcpyDeviceToHost);

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

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

printf("\n");

shared\_mem<<<BLOCKS,THREADS>>>(d\_arr);

cudaMemcpy(arr,d\_arr,n,cudaMemcpyDeviceToHost);

cudaDeviceSynchronize();

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

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

return 0;

}

**Ques4-a Write an OpenMP program to add all the elements of two arrays A & B each of size 1000 and store their sum in a variable using reduction clause.**

#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

#define N 1000 int main()

{

int a[N],b[N],i,result;

#pragma omp for for(i=0;i<N;i++){

a[i] = b[i] = i;

}

#pragma omp parallel for reduction(+:result)

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

result += a[i] \* b[i];

}

printf("%d\n",result);

}

**Ques4-b Write a MPI program to calculate and print the value of PI.**

#include<stdio.h>

#include<mpi.h>

double func(double x)

{

return 4/(1+x\*x);

}

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

{

int rank,size,master = 0,i,n\_intervals; double h,pi,mypi,x,sum; MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank); MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

if(rank == master) n\_intervals = 200000; MPI\_Bcast(&n\_intervals,1,MPI\_INT,master,MPI\_COMM\_WORLD);

h = 1.0/n\_intervals; sum = 0; for(i=rank+1;i<=n\_intervals;i+=size)

{

x = h \* ((double)i-0.5);

sum += func(x);

}

mypi = h \* sum; MPI\_Reduce(&mypi,&pi,1,MPI\_DOUBLE,MPI\_SUM,master,MPI\_COMM\_WORLD);

if(rank == master)

printf("Pie is approximately %f\n",pi); MPI\_Finalize();

}

**Ques5-a Write an OpenMP program to multiply two matrices A & B and find the resultant matrix C**

#include<stdio.h>

#include<omp.h>

#define A\_rows 10

#define A\_columns 8

#define B\_rows A\_columns

#define B\_columns 20

int main()

{

int i,j,k,chunk=3,tid,n\_threads;

int A[A\_rows][A\_columns], B[B\_rows][B\_columns], C[A\_rows][B\_columns];

#pragma omp parallel shared(A,B,chunk,n\_threads) private(tid,i,j,k)

{

tid = omp\_get\_thread\_num();

if(tid == 0)

{

n\_threads = omp\_get\_num\_threads();

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

}

#pragma omp for schedule(static,chunk)

for(i=0;i<A\_rows;i++)

for(j=0;j<A\_columns;j++)

A[i][j] = 1;

#pragma omp for schedule(static,chunk)

for(i=0;i<B\_rows;i++)

for(j=0;j<B\_columns;j++) B[i][j] = 1;

#pragma omp for schedule(static,chunk)

for(i=0;i<A\_rows;i++)

for(j=0;j<B\_columns;j++) C[i][j] = 0;

#pragma omp for schedule(static, chunk)

for(i=0;i<A\_rows;i++)

{

printf("Thread %d did row %d\n",tid,i);

for(j=0;j<B\_columns;j++)

for(k=0;k<A\_columns;k++)

C[i][j] += A[i][k] \* B[k][j];

}

}

for(i=0;i<A\_rows;i++)

{

for(j=0;j<B\_columns;j++)

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

printf("\n");

}

}

**Ques5-b Write a MPI program to send the message from a process whose rank=3 to all other remaining processes**

#include<stdio.h>

#include<mpi.h>

#include<stdlib.h>

#include<string.h>

#define BUFF\_SIZE 100

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

{

int rank,size,i,master=3; char msg[BUFF\_SIZE]; MPI\_Status status; MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank); MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

if(rank == master){

for(i=0;i<size;i++){ strcpy(msg,"Hello msrit"); if(i==master)

continue;

MPI\_Send(&msg,BUFF\_SIZE,MPI\_CHAR,i,i,MPI\_COMM\_WORLD);

}

}

else

{

MPI\_Recv(&msg,BUFF\_SIZE,MPI\_CHAR,master,rank,MPI\_COMM\_WORLD,&status);

printf("%d recieved %s\n",rank,msg);

} MPI\_Finalize();

}

**Ques6-a Write an OpenMP program to find the number of processes, number of threads, etc (the environment information).**

#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

int main()

{

int tid,nthreads,procs,maxt,inpar,dynamic,nested;

#pragma omp parallel private(nthreads,tid)

{

tid = omp\_get\_thread\_num(); procs = omp\_get\_num\_procs(); nthreads = omp\_get\_num\_threads(); maxt = omp\_get\_max\_threads(); inpar = omp\_in\_parallel();

dynamic = omp\_get\_dynamic(); nested = omp\_get\_nested(); if(tid==0)

{

printf("Thread %d getting env info\n",tid);

printf("Env info:-\n");

printf("Number of processes = %d\n",procs); printf("Number of threads = %d\n",nthreads); printf("Max threads = %d\n", maxt);

printf("In parallel? = %d\n", inpar);

printf("Dynamic threads enabled? = %d\n", dynamic);

printf("Nested parallelism supported? = %d\n", nested);

}

}

}

**Ques6-b Write a MPI program scatter the information to different processes (Consider at least Six Processes)**

#include<stdio.h>

#include<mpi.h>

#include<stdlib.h>

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

int rank,size,master = 0;

int data[]={1,2,3,4,5,6,7},rec;

MPI\_Init(&argc,&argv); MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);

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

MPI\_Scatter(&data,1,MPI\_INT,&rec,1,MPI\_INT,master,MPI\_COMM\_WORLD);

printf ("[%d] Received data = %d\n", rank, rec); MPI\_Finalize();

}

**Ques7-a Write an OpenMP program to find the largest element in an array using critical section**

#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

#define MAXIMUM 65536

int main()

{

int \*array, i, n\_elements, cur\_max, current\_value;

printf("Enter the number of elements\n");

scanf("%d", &n\_elements);

if (n\_elements <= 0) {

printf("The array elements cannot be stored\n");

exit(1);

}

array = (int\*)malloc(sizeof(int)\*n\_elements);

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

array[i] = i;

cur\_max = 0;

#pragma omp parallel for for(i=0;i<n\_elements;i++)

{

#pragma omp critical if(array[i] > cur\_max)

cur\_max = array[i];

}

current\_value = 0;

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

{

if(array[i] > current\_value)

current\_value = array[i];

}

printf("%d %d\n",cur\_max,current\_value);

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

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

printf("\n");

}

**Ques7-b Write a MPI program where each processor sends an integer number and its rank to the master processor, where the master gathers all the information and prints the data accordingly**

#include<stdio.h>

#include<mpi.h>

#include<string.h>

#define MAX\_SIZE 100

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

{

int rank,size,i,master=0;

int msg,recv[MAX\_SIZE];

MPI\_Init(&argc,&argv); MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank); MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

msg = 23+rank; MPI\_Gather(&msg,1,MPI\_INT,&recv,1,MPI\_INT,master,MPI\_COMM\_WORLD); if(rank==master)

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

printf(" Integer is %d and rank is %d\n",recv[i],i); MPI\_Finalize();

}

**Ques8-a Write an OpenMP program to find the largest element in an array using locks**

#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

#define N 10000

int main()

{

int \*a,i,cur\_max=0;

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

omp\_lock\_t lock;

#pragma omp for for(i = 0;i<N;i++)

a[i] = i;

omp\_init\_lock(&lock);

#pragma omp parallel shared(a) private(i)

{

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

{

omp\_set\_lock(&lock);

if(a[i] > cur\_max)

cur\_max = a[i];

omp\_unset\_lock(&lock);

}

}

printf("Max is %d\n",cur\_max);

}

**Ques8-b Write a CUDA program for adding two vectors**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define BLOCK\_SIZE 20

global\_\_ void vecAdd(double \*a,double \*b,double \*c,int n)

{

int id = blockIdx.x\*blockDim.x + threadIdx.x;

if(id<n)

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

}

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

{

int n = 100,i;

double \*h\_a,\*h\_b,\*h\_c;

double \*d\_a,\*d\_b,\*d\_c;

size\_t bytes = sizeof(double)\*n; h\_a = (double \*)malloc(bytes); h\_b = (double \*)malloc(bytes); h\_c = (double \*)malloc(bytes);

cudaMalloc(&d\_a,bytes); cudaMalloc(&d\_b,bytes); cudaMalloc(&d\_c,bytes);

for(i=0;i<n;i++) h\_a[i] = h\_b[i] = i;

cudaMemcpy(d\_a,h\_a,bytes,cudaMemcpyHostToDevice);

cudaMemcpy(d\_b,h\_b,bytes,cudaMemcpyHostToDevice);

int gridSize = n/BLOCK\_SIZE;

vecAdd<<<gridSize,BLOCK\_SIZE>>>(d\_a,d\_b,d\_c,n);

cudaMemcpy(h\_c,d\_c,bytes,cudaMemcpyDeviceToHost);

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

printf("%f + %f = %f\n",h\_a[i],h\_b[i],h\_c[i]);

cudaDeviceSynchronize();

}

**Ques9-a Write an OpenMP program to find the sum of an array A and store the result in a variable. (Using Reduction clause)**

#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

#define N 1000 int main()

{

int a[N],i,result=0;

#pragma omp for

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

a[i]=i;

}

#pragma omp parallel for reduction(+:result)

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

result+=a[i];

}

printf("%d\n",result);

}

**Ques9-b Write a CUDA program to multiply two matrices**

#include<stdio.h>

#define WA 3

#define HA 3

#define WB 3

#define HB WA

#define WC WB

#define HC HA

#define BLOCK HC

#define THREADS WC

void init(float \*arr,int size)

{

int i;

for(i = 0; i<size;i++) arr[i] = i;

}

global\_\_ void mm(float \*A,float \*B,float \*C)

{

int i,j,sum = 0,id,row\_start,column\_start,row\_end;

id = threadIdx.x + blockIdx.x\*blockDim.x; row\_start = blockIdx.x\*blockDim.x; row\_end = row\_start+WA;

column\_start = threadIdx.x;

for(i=row\_start,j = column\_start;i<row\_end;i++,j+=blockDim.x)

{

sum += A[i]\*B[j];

}

C[id] = sum;

}

int main()

{

float \*h\_a,\*h\_b,\*h\_c,\*d\_a,\*d\_b,\*d\_c;

int i,size\_A,size\_B,size\_C;

size\_t mem\_A,mem\_B,mem\_C;

size\_A = WA\*HA; size\_B = WB\*HB; size\_C = WC\*HC;

mem\_A = sizeof(float) \* size\_A; mem\_B = sizeof(float) \* size\_B; mem\_C = sizeof(float) \* size\_C;

h\_a = (float \*)malloc(mem\_A);

h\_b = (float \*)malloc(mem\_B);

h\_c = (float \*)malloc(mem\_C);

cudaMalloc(&d\_a,mem\_A); cudaMalloc(&d\_b,mem\_B); cudaMalloc(&d\_c,mem\_C);

init(h\_a,size\_A); init(h\_b,size\_B); cudaMemcpy(d\_a,h\_a,mem\_A,cudaMemcpyHostToDevice); cudaMemcpy(d\_b,h\_b,mem\_B,cudaMemcpyHostToDevice);

mm<<<BLOCK,THREADS>>>(d\_a,d\_b,d\_c);

cudaMemcpy(h\_c,d\_c,mem\_C,cudaMemcpyDeviceToHost);

cudaDeviceSynchronize();

for(i=0;i<size\_A;i++)

{

printf("%0.2f ",h\_a[i]);

if((i+1)%WA == 0) printf("\n");

}

printf("\n");

for(i=0;i<size\_B;i++)

{

printf("%0.2f ",h\_b[i]);

if((i+1)%WB == 0) printf("\n");

}

printf("\n");

for(i=0;i<size\_C;i++)

{

printf("%0.2f ",h\_c[i]);

if((i+1)%WC == 0) printf("\n");

}

printf("\n");

}

**Ques10-a Write an OpenMP program to print all the letters of the alphabet A- Z using threads**

#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

#define CHUNK 2 int main(){

char c;

omp\_set\_num\_threads(5);

#pragma omp parallel for schedule(static,CHUNK)

for(c='A';c<='Z';c++){

printf("Thread %d print %c\n",omp\_get\_thread\_num(),c);

}

}

**Ques10-b Write a CUDA program to print the message “Hello World” and demonstrate**

**blocks by varying NUM\_BLOCKS to different sizes**

#include<stdio.h>

#define BLOCKS 10

#define THREADS 2

global\_\_ void hello()

{

printf("This is thread %d of block %d with a block dim of

%d\n",threadIdx.x,blockIdx.x,blockDim.x);

}

int main()

{

hello<<<BLOCKS,THREADS>>>(); cudaDeviceSynchronize(); printf("Done\n");

return 0;

}

**Ques11-a Write an OpenMP program to show how thread private clause works**

#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

int a,b,tid;

float x;

#pragma omp threadprivate(a,x)

int main()

{

omp\_set\_dynamic(0);

printf("1st parallel region\n");

#pragma omp parallel private(tid,b)

{

tid = omp\_get\_thread\_num();

a = rand();

b = rand();

x = 1.1\*tid + 1;

printf("Thread %d ,a,b,x = %d,%d,%f\n",tid,a,b,x);

}

printf("\nMaster thread doing serial work\n\n");

#pragma omp parallel private(tid,b)

{

tid = omp\_get\_thread\_num();

printf("Thread %d ,a,b,x = %d,%d,%f\n",tid,a,b,x);

}

}

**Ques11-b Write a MPI program to find sum of 'n' integers on 'p' processors using point- to-point communication libraries call**

#include<stdio.h>

#include<mpi.h>

#include<stdlib.h>

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

{

int rank,size,master = 0,i; int val,sum= 0; MPI\_Status status; MPI\_Init(&argc,&argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank); MPI\_Comm\_size(MPI\_COMM\_WORLD,&size); if(rank == master)

{

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

{

if(i==master)

continue; MPI\_Recv(&val,1,MPI\_INT,i,0,MPI\_COMM\_WORLD,&status); sum += val;

}

}

else

{

}

printf("Sum is %d\n",sum);

val = rank; MPI\_Send(&val,1,MPI\_INT,master,0,MPI\_COMM\_WORLD);

MPI\_Finalize();

}

**Ques12-a Write an OpenMP program to show how first private clause works.(Factorial program)**

#include<stdio.h>

#include<omp.h>

long long factorial(long long n)

{

int i;

long result = 1;

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

result \*= i;

return result;

}

int main()

{

int i,j,n\_threads;

long long n=12; long long x[n]; omp\_set\_num\_threads(5);

printf("Max threads %d\n",omp\_get\_max\_threads());

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

{

x[i] = factorial(i);

}

j = 0;

#pragma omp parallel for firstprivate(x,j)

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

{

j+=i;

x[i] = j\*x[i-1];

}

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

printf("factorial %d is %lld %lld\n",i,factorial(i),x[i]);

return 0;

}

**Ques12-b Write an MPI program where the master processor broadcasts a message**

**“HELLO MSRIT” to the remaining processors using broadcast system call**

#include<stdio.h>

#include<mpi.h>

#include<string.h>

#define BUFF\_SIZE 20

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

{

int rank,size,master = 0;

char msg[BUFF\_SIZE]; MPI\_Init(&argc,&argv); MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank); MPI\_Comm\_size(MPI\_COMM\_WORLD,&size);

//sprintf(msg,"%s","Hello msrit");

strcpy(msg,"Hello msrit");

MPI\_Bcast(msg,BUFF\_SIZE,MPI\_CHAR,master,MPI\_COMM\_WORLD); MPI\_Barrier(MPI\_COMM\_WORLD);

if(rank!=master)

printf("Process %d has %s\n",rank,msg); MPI\_Finalize();

}

**Ques13-a Write an OpenMP program to find prime numbers (split)**

#include<stdio.h>

#include<omp.h>

#include<stdlib.h>

#define N 10000

int main() {

int i,k,\*a,found,count = 0;

double t1,t2;

a = (int \*)malloc(sizeof(int)\*(N+1));

#pragma omp parallel for for(i=0;i<N;i++)

a[i] = 1;

k = 2;

t1 = omp\_get\_wtime();

#pragma omp parallel firstprivate(k) private(i,found)

while(k\*k <= N)

{

#pragma omp for for(i=k\*k;i<=N;i+=k){

a[i] = 0;

}

found = 0;

for(i=k+1;!found;i++){

if(a[i]){

found = 1;

k=i;

}

}

}

t2 = omp\_get\_wtime();

#pragma omp for for(i=2;i<=N;i++){

if(a[i]){

count++;

}

}

printf("Time %f\n",t2-t1);

printf("Total prime numbers are %d\n",count);

}

**Ques13-b Write a CUDA program for adding two vectors**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

#define BLOCK\_SIZE 20

global\_\_ void vecAdd(double \*a,double \*b,double \*c,int n)

{

int id = blockIdx.x\*blockDim.x + threadIdx.x;

if(id<n)

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

}

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

{

int n = 100,i;

double \*h\_a,\*h\_b,\*h\_c;

double \*d\_a,\*d\_b,\*d\_c;

size\_t bytes = sizeof(double)\*n; h\_a = (double \*)malloc(bytes); h\_b = (double \*)malloc(bytes); h\_c = (double \*)malloc(bytes);

cudaMalloc(&d\_a,bytes); cudaMalloc(&d\_b,bytes); cudaMalloc(&d\_c,bytes);

for(i=0;i<n;i++) h\_a[i] = h\_b[i] = i;

cudaMemcpy(d\_a,h\_a,bytes,cudaMemcpyHostToDevice);

cudaMemcpy(d\_b,h\_b,bytes,cudaMemcpyHostToDevice);

int gridSize = n/BLOCK\_SIZE;

vecAdd<<<gridSize,BLOCK\_SIZE>>>(d\_a,d\_b,d\_c,n);

cudaMemcpy(h\_c,d\_c,bytes,cudaMemcpyDeviceToHost);

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

printf("%f + %f = %f\n",h\_a[i],h\_b[i],h\_c[i]);

cudaDeviceSynchronize();

}

**Ques14-a Write an OpenMP program to multiply two matrices A & B and find the resultant matrix C**

#include<stdio.h>

#include<omp.h>

#define A\_rows 10

#define A\_columns 8

#define B\_rows A\_columns

#define B\_columns 20

int main()

{

int i,j,k,chunk=3,tid,n\_threads;

int A[A\_rows][A\_columns], B[B\_rows][B\_columns], C[A\_rows][B\_columns];

#pragma omp parallel shared(A,B,chunk,n\_threads) private(tid,i,j,k)

{

tid = omp\_get\_thread\_num();

if(tid == 0)

{

n\_threads = omp\_get\_num\_threads();

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

}

#pragma omp for schedule(static,chunk)

for(i=0;i<A\_rows;i++)

for(j=0;j<A\_columns;j++) A[i][j] = 1;

#pragma omp for schedule(static,chunk)

for(i=0;i<B\_rows;i++)

for(j=0;j<B\_columns;j++) B[i][j] = 1;

#pragma omp for schedule(static,chunk)

for(i=0;i<A\_rows;i++)

for(j=0;j<B\_columns;j++) C[i][j] = 0;

#pragma omp for schedule(static, chunk)

for(i=0;i<A\_rows;i++)

{

printf("Thread %d did row %d\n",tid,i);

for(j=0;j<B\_columns;j++)

for(k=0;k<A\_columns;k++)

C[i][j] += A[i][k] \* B[k][j];

}

}

for(i=0;i<A\_rows;i++)

{

for(j=0;j<B\_columns;j++)

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

printf("\n");

}

}

**Ques14-b Write a CUDA program to print the message “Hello World” and demonstrate**

**threads by varying BLOCK\_WIDTH to different sizes**

#include<stdio.h>

#define BLOCKS 2

#define THREADS 10

global\_\_ void hello()

{

printf("This is thread %d of block %d with a block dim of

%d\n",threadIdx.x,blockIdx.x,blockDim.x);

}

int main()

{

hello<<<BLOCKS,THREADS>>>(); cudaDeviceSynchronize(); printf("Done\n");

return 0;

}