**25th March,21**

**Private and shared variables**

**//compute the sum of two arrays in parallel**

#include < stdio.h >

#include < omp.h >

#define N 1000000

int main(void) {

float a[N], b[N], c[N];

int i;

/\* Initialize arrays a and b \*/

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

a[i] = i \* 2.0;

b[i] = i \* 3.0;

}

/\* Compute values of array c = a+b in parallel. \*/

#pragma omp parallel shared (a, b, c) private (i)

{

#pragma omp for

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

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

printf ("%f\n", c[10]);

}

}

}

**Scheduling (static)**

**//example4.c: add all elements in an array in parallel**

#include < stdio.h >

int main() {

const int N=100;

int a[N];

//initialize

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

a[i] = i;

//compute sum

int local\_sum, sum;

#pragma omp parallel private(local\_sum) shared(sum)

{

local\_sum =0;

//the array is distributde statically between threads

#pragma omp for schedule(static,1)

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

local\_sum += a[i];

}

//each thread calculated its local\_sum. ALl threads have to add to

//the global sum. It is critical that this operation is atomic.

#pragma omp critical

sum += local\_sum;

}

printf("sum=%d should be %d\n", sum, N\*(N-1)/2);

}

**Nowait clause**

#pragma omp parallel

{

#pragma omp for nowait

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

b[i] = (a[i] + a[i-1]) / 2.0;

#pragma omp for nowait

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

y[i] = sqrt(z[i]);

}

**Reduction clause**

sum **=** 0;

**#pragma omp parallel for shared(sum, a) reduction(+: sum)**

**for** (**auto** i **=** 0; i **<** 9; i**++**)

{

sum **+=** a[i]

}

**26th March,21**

**Vector Addition using dynamic schedule**

The following program adds two vectors together using a work-sharing approach to assign work to threads.

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define CHUNKSIZE 10

#define N 100

int main (int argc, char \*argv[]) { int nthreads, tid, i, chunk; float a[N], b[N], c[N];

for (i=0; i < N; i++)a[i] = b[i] = i \* 1.0; // initialize arrays chunk = CHUNKSIZE;

#pragma omp parallel shared(a,b,c,nthreads,chunk) private(i,tid)

{

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

if (tid == 0){

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

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

}

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

#pragma omp for schedule(dynamic,chunk)

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

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

printf("Thread %d: c[%d]= %f\n",tid,i,c[i]);

}

} /\* end of parallel section \*/

}

**Sum of elements in an array using critical**

#include<stdio.h>

#include<omp.h>

/\* Main Program \*/

main()

{

float \*Array, \*Check, serial\_sum, sum, partialsum;

int array\_size, i;

printf("Enter the size of the array\n");

scanf("%d", &array\_size);

if (array\_size <= 0) {

printf("Array Size Should Be Of Positive Value ");

exit(1);

}

/\* Dynamic Memory Allocation \*/

Array = (float \*) malloc(sizeof(float) \* array\_size);

Check = (float \*) malloc(sizeof(float) \* array\_size);

/\* Array Elements Initialization \*/

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

Array[i] = i \* 5;

Check[i] = Array[i];

}

printf("The Array Elements Are \n");

for (i = 0; i < array\_size; i++)

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

sum = 0.0;

partialsum = 0.0;

/\* OpenMP Parallel For Directive And Critical Section \*/

#pragma omp parallel for shared(sum)

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

#pragma omp critical

sum = sum + Array[i];

}

serial\_sum = 0.0;

/\* Serail Calculation \*/

for (i = 0; i < array\_size; i++)

serial\_sum = serial\_sum + Check[i];

if (serial\_sum == sum)

printf("\nThe Serial And Parallel Sums Are Equal\n");

else {

printf("\nThe Serial And Parallel Sums Are UnEqual\n");

exit(1);

}

/\* Freeing Memory \*/

free(Check);

free(Array);

printf("\nThe SumOfElements Of The Array Using OpenMP Directives Is %f\n", sum);

printf("\nThe SumOfElements Of The Array By Serial Calculation Is %f\n", serial\_sum);

}

**31st March,21**

**Barrier Clause**

#include <omp.h>

void work1(int k) {

// large amount of work

}

void work2(int k) {

// large amount of work that must all happen after work1 is finished

}

int main() {

int n=1000000;

#pragma omp parallel private(i) shared(n) {

#pragma omp for

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

work1(i);

#pragma omp barrier

#pragma omp for

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

work2(i);

}

return 0;

}

**Vector Addition and Multiplication using sections**

The program below adds elements of two vectors to form a third and also multiplies the elements of the arrays to produce a fourth vector.

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

#define N 50

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

int i, nthreads, tid;

float a[N], b[N], c[N], d[N];

for (i=0; i<N; i++) { // Some initializations, arbitrary values a[i] = i \* 1.5;

b[i] = i + 22.35;

c[i] = d[i] = 0.0;

}

#pragma omp parallel shared(a,b,c,d,nthreads) private(i,tid)

{

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

if (tid == 0) {

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

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

}

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

#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("Thread %d: c[%d]= %f\n",tid,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: d[%d]= %f\n",tid,i,d[i]);

}

}

} // end of sections printf("Thread %d done.\n",tid);

} // end of parallel section

}