Unit - 3

① If it is def, the OPENMP macro is a decimal Int. what is the significan of the value?

Sol^n:- The val of OPENMP is a date having the form YYYYmm where YYYY is a 4 digit year and mm is a 2-digit month. Ex:- 200505. The OPENMP std states that when the macro is def, it will be year and month of the version of OPENMP. std that has been implemented

-x-

② Recall that OPENMP creates private var for red^n var and these private var are initialized to the identity value. for the red^n operator Ex:- If the operator is add^n the private var are initialized to 0, while if the operator is multiplication, the private var are initialized to 1. what are the identity val for the operators &&, ||, &, |, ^. ?

Sol^n:-

Operator	Identity value
&& (and)	1
|| (OR)	0
& bin AND	11 : .. 11 2
| bin OR	0
^ XOR	0

-x-

③ In our 1st attempt to ||lize the prog for estimating π, our prog was incorrect. In fact we used the result of the prog when it was run with one thread as evidence that the prog run with two threads was incorrect. Explain why we could trust when the result of the prog when it was run with one thread?

Sol^n:- when the prog is run with one thread, the parallel for directive has no effect, and the prog is effectively same as the preceding serial prog. In particular there is no loop_carried dependence, since there is only one thread.

-x-

④ Consider the loop

```
a[0] = 0;
for (i = 1; i < n; i++)
    a[i] = a[i-1] + i;
```

There is clearly loop carried dependency, as the val of a[i] can't be computed without the val of a[i-1]. Can you see any way to eliminate this dependence and parallize the loop.?

Sol^n:- observe that

$a[0] = 0$

$a[1] = a[0] + 1 = 0 + 1$

$a[2] = a[1] + 2 = 0 + 1 + 2$

$a[3] = a[2] + 3 = 0 + 1 + 2 + 3$

$a[4] = a[3] + 4 = 0 + 1 + 2 + 3 + 4$

$\therefore \quad a[i] = \sum_{j=0}^{i} j \quad$ But $\sum_{j=0}^{i} j = \frac{i(i+1)}{2}$.

So we can rewrite the code as:

```
for (i=0; i<n; i++)
    a[i] = i*(i+1)/2
```

In this loop the result of any iteration is not used again. So the code can be parallelizable

(8)
```
# pragma OMP parallel for num_thread (thread_count) \
    default (none) private (i) shared (a, n)
    for (i=0; i<n; i++)
        a[i] = i*(i+1)/2;
```

— x —

(5) Recall that in C, the funcⁿ that takes 2D array arg must specify the no of col in the argument list. So it is quite common for C progrs to use 1-D arrays and to write explicit code for converting pairs of subscripts in to a single dimension. Modify the OpenMP matrix_vector multipⁿ so that it uses the 1-D array for the matrix

Sol:- The following code will do the job

```
# pragma OMP parallel for num_thread (thread_count) \
    default (none) private (i, j), shared (A, x, y, m, n).
    for (i=0; i<m; i++)
    {   y[i] = 0.0;
        for (j=0; j<n; j++)
            // y[i] += A[i][j] * x[j];
            y[i] += A[i*n+j] * x[j];
    }
```

— x —

(6) Recall the matrix_vector multⁿ ex with 8000×8000 i/p. Suppose that the thread 0 and thread 2 are assigned to diff processors. If the cache line contains the 64 bytes of 8 doubles, is it possible for false sharing b/w threads 0 and 2 to occur for any part of the vector y? why? what about if thread 0 and 3 are assigned to diff processors; is it possible for false sharing to occur b/w them for any part of y?

Solⁿ:- with 8000 elements y will be partitioned as

Th 0 :	y[0], y[1] ...	y[1999]
Th 1 :	y[2000], y[2001] ...	y[3999]
Th 2 :	y[4000], y[4001] ...	y[5999]
Th 3 :	y[6000], y[6001] ...	y[7999].

In order for the false sharing to occur b/w Th 0 & 2 there must be elements of y that belong to the same cache line. but are assigned to diff threads.

On Thread 0 the cache line that closest to the elements assigned to Th 2 is the line Containing Y[1999]. But even if this is the first element of the Cache line, the highest possible index for an element of Y that belong to this line is 2006.

Y[1999]	Y[2000]	Y[2001]	Y[2002]	Y[2003]	Y[2004]	Y[2005]	Y[2006]

Since the least index of an element of Y assigned to Th 2 is 4000. There can't be a cache line that has elements belonging to Th0 and Th2.
Similar reasoning applies to Th0 and Th3.

—x—

7 (a) what is the min no of Cache lines that are needed to store the vector Y

(b) what is ~~max~~ -"- to store vecto Y

(c) If the boundaries of the Cache line always coincide with the boundaries of 8-byte doubles, in how many diff ways can the components of Y be assigned to Cache lines.

Soln:- If we look at the locn Y[0] in the 1st Cache line Containing all or part of Y, we see that Y can be distributed across the cache lines in 8 diff ways. If Y[0] is the 1st element of Cache line, then we'll have the following assignment.

1st line	Y[0]	Y[1]	Y[2]	Y[3]	Y[4]	Y[5]	Y[6]	Y[7]

If Y[0] is the 2nd element of the Cache line then we have

1st line	–	Y[0]	Y[1]	Y[2]	Y[3]	Y[4]	Y[5]	Y[6]
2nd line	Y[7]							

As a final ex, If Y[0] is the last element of the 1st line then

–	–	–	–	–	–	–	Y[0]
Y[1]	Y[2]	Y[3]	Y[4]	Y[5]	Y[6]	Y[7]	–

(a) It is possible for Y to fit into a single Cache line

(b) In most cases, Y, will be split into two Cache lines

(c) There are 8 ways the doubles can be assigned. 8 ways correspond to the 8 diff possible locn for Y[0] in the first line.