**OMP1. Under what conditions will dynamic loop scheduling perform better than static scheduling?**

The scheduling where the computational workload is fixed and the execution of for loop is fixed then static is best whereas dynamic scheduling is best when the computational workload is varies and for each iteration is irregular Dynamic is best.

**OMP2. Give an example of a simple parallel forloop that would probably execute faster using schedule (static,1) than the default scheduling method. Your example should not have nested loops**

// OMP1-8\_program.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include <iostream>

#include <time.h>

#include <omp.h>

#define n 2000000

int i;

clock\_t t;

int x[n];

int y[n];

int main()

{

t = clock();

omp\_set\_num\_threads(4);

#pragma omp parallel for schedule(static,1)

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

x[i] = i;

}

printf("Clock : %d\n", clock() - t);

t = clock();

omp\_set\_num\_threads(4);

#pragma omp parallel for

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

y[i] = n - i;

}

printf("Clock : %d\n", clock() - t);

}

**OMP3. What performance advantages does guide scheduling offer over dynamic scheduling?**

Dynamic scheduling have fixed chuck size whereas in guided scheduling the size of the chuck depends upon the no of thread count and the unassign iteration.

**OMP4. The following program implements MatrixVector Multiplication ( a = B \* c ), where a and c are vectors, and B is a matrix.**

**OMP 4 (con’t) Parallelize the for loop in line 05 by providing the appropriate line in OpenMP. Which variables have to be private and which variables have to be shared?**

//matrix\_vector\_multiplication.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include <stdio.h>

#include <time.h>

#include <omp.h>

#include <math.h>

#define n 4

double A[n], B[n][n], C[n];

int i, j;

int threadCount = 4;

int main() {

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

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

B[i][j] = (i + 1.0) \* (j + 1.0);

}

C[i] = i + 2.0;

}

omp\_set\_num\_threads(threadCount);

#pragma omp parallel for private(j)

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

A[i] = 0.0;

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

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

}

}

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

printf("%f\n", A[i]);

}

}

**OMP 5. Which schedule would you propose for this parallelization? Explain your answer and briefly list the pros and contras for each of the three OpenMP work-sharing schedules (namely static, dynamic and guided) for this specific case.**

In OMP 4, I use default one which is a static one which divided chunk into approximately equal in size and it has low overhead and suitable for the given program.

But our program has a balanced workload so minimizing overhead static schedule is the best.

**OMP6. Make each of the following loops parallel, or explain why it is not suitable for parallel execution:**

omp\_set\_num\_threads(4);

#pragma omp parallel for private(x)

for (i=0; i < (int) sqrt(x); i++) {

a[i] = 2.3 \* i;

if (i < 10) b[i] = a[i];

}

===================================================================

flag = 0;

for (i=0; (i < n) && (!flag); i++) {

a[i] = 2.3 \* i;

if (a[i] < b[i]) flag = 1;

}

In this code flag is updated and read at same time when a[I]<b[I], so it has loop dependence to use parallel.

=======================================================================

omp\_set\_num\_threads(3);

#pragma omp parallel for

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

a[i] = foo(i);

**======================================================================**

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

a[i] = foo(i);

if (a[i] < b[i]) a[i] = b[i];

}

It has output loop dependence, so it can’t be used parallel

======================================================================

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

**a[i] = foo(i);**

**if (a[i] < b[i]) break;**

**}**

**It has conditional break statement, so can’t be used parallel.**

**======================================================================**

**dotp = 0;**

**omp\_set\_num\_threads(4);**

**#pragma omp parallel for reduction(+:dotp)**

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

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

**======================================================================**

**for (i=k; i < 2\*k; i++)**

**a[i] = a[i] + a[i-k];**

**It has loop dependency, so it can’t use parallel.**

**=====================================================================**

**for (i=k; i<n; i++)**

**a[i] = b \* a[i-k];**

**It has loop dependency, so it can’t use parallel.**

**OMP7. Run each of the following programs and explain the differences in the output.**

The first one is static schedule output with 1 chunk size and second one is default once. Both have thread size 3, so in static schedule index 0 was run by 1st thread, 1 was run by 2nd thread and 2 was run by 3rd thread and repeated. In this case the loop is divided according to the chunk size and run by thread assigned at runtime. For the default one all iterations are divided equally and assigned to each thread. And each thread runs the assigned iteration.

**OMP 8. For each of the following portions of code, is there any loop carried dependence? If yes, is it flow, anti, or output dependence?**

(a) for (i = 0; i < U1; i++)

for (j = 0; j < U2; j++)

a[i+4-j] = b[2\*i-j] + i\*j;

(b) for (j = 0; j < n; j++)

a[j] = a[j-1];

(c) for (j = 0; j < n; j++) {

c[j] = j;

c[j+1] = 5;}

(d) for (j = 0; j < n; j++)

b[j] = b[j+1];

ANS:

A: Loop-independent because both i and j loop are not dependent on other i and j loops values.

B: a[j] dependent on value of a[j-1] so it’s a loop-dependent of type flow.

C: this loop tries to write at c[j] and c[j+1] so for next iteration c[j] also try to write at current c[j+1] so it’s output dependency

D: a[j] dependent on value of a[j+1] so it’s a loop-dependent of type flow.