NAME	JACOB JOHN
REGISTER NO.	16BCE2205
E-MAIL	jacob.john2016@vitstudent.ac.in

LAB ASSESSMENT #5

SCENARIO – 1

Write an OpenMP program to specify that the schedule (dynamic, chunk-size) clause of the loop construct specifies that the for loop has the dynamic scheduling type.

Brief about your approach:

Static schedule means that iterations blocks are mapped statically to the execution threads in a round-robin fashion. The nice thing with static scheduling is that OpenMP run-time guarantees that if you have two separate loops with the same number of iterations and execute them with the same number of threads using static scheduling, then each thread will receive exactly the same iteration range(s) in both parallel regions. This is quite important on NUMA systems: if you touch some memory in the first loop, it will reside on the NUMA node where the executing thread was. Then in the second loop the same thread could access the same memory location faster since it will reside on the same NUMA node.

Imagine there are two NUMA nodes: node 0 and node 1, e.g. a two-socket Intel Nehalem board with 4-core CPUs in both sockets. Then threads 0, 1, 2, and 3 will reside on node 0 and threads 4, 5, 6, and 7 will reside on node 1.

```
|             | core 0 | thread 0 |
| socket 0    | core 1 | thread 1 |
| NUMA node 0 | core 2 | thread 2 |
|             | core 3 | thread 3 |

|             | core 4 | thread 4 |
| socket 1    | core 5 | thread 5 |
| NUMA node 1 | core 6 | thread 6 |
|             | core 7 | thread 7 |
```

Each core can access memory from each NUMA node, but remote access is slower (1.5x - 1.9x slower on Intel) than local node access. You run something like this:

```
char *a = (char *)malloc(8*4096);

#pragma omp parallel for schedule(static,1) num_threads(8)
for (int i = 0; i < 8; i++)
   memset(&a[i*4096], 0, 4096);
```

Code:

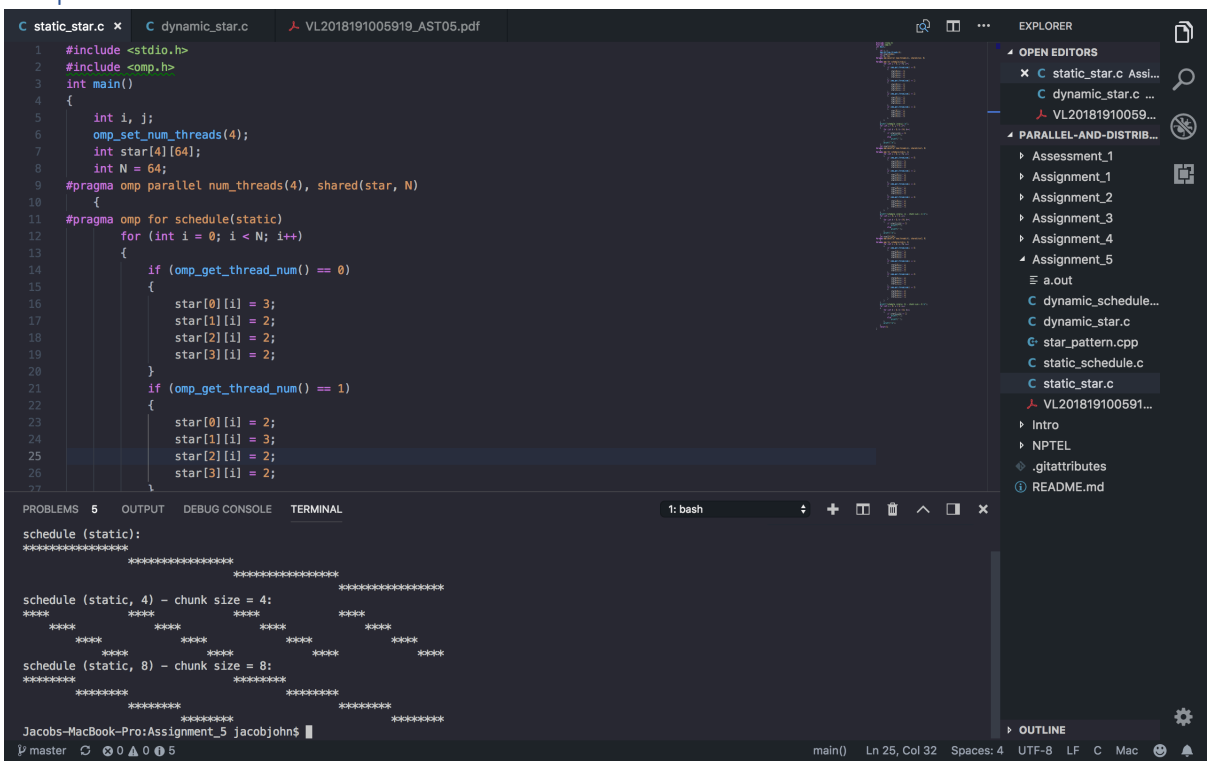
```
#include <stdio.h>
#include <omp.h>
int main()
{
    int i, j;
    omp_set_num_threads(4);
    int star[4][64];
    int N = 64;
#pragma omp parallel num_threads(4), shared(star, N)
    {
#pragma omp for schedule(static)
        for (int i = 0; i < N; i++)
        {
            if (omp_get_thread_num() == 0)
            {
                star[0][i] = 3;
                star[1][i] = 2;
                star[2][i] = 2;
                star[3][i] = 2;
            }
            if (omp_get_thread_num() == 1)
            {
                star[0][i] = 2;
                star[1][i] = 3;
                star[2][i] = 2;
                star[3][i] = 2;
            }
            if (omp_get_thread_num() == 2)
            {
                star[0][i] = 2;
                star[1][i] = 2;
                star[2][i] = 3;
                star[3][i] = 2;
            }
            if (omp_get_thread_num() == 3)
            {
                star[0][i] = 2;
                star[1][i] = 2;
                star[2][i] = 2;
                star[3][i] = 3;
            }
        }
    }
    printf("schedule (static):\n");
    for (int j = 0; j < 4; j++)
    {
        for (int k = 0; k < 64; k++)
        {
            if (star[j][k] == 3)
                printf("*");
```

```
            else
                printf(" ");
        }
        printf("\n");
    }
    int star1[4][64];
#pragma omp parallel num_threads(4), shared(star1, N)
    {
#pragma omp for schedule(static, 4)
        for (int i = 0; i < N; i++)
        {
            if (omp_get_thread_num() == 0)
            {
                star1[0][i] = 3;
                star1[1][i] = 2;
                star1[2][i] = 2;
                star1[3][i] = 2;
            }
            if (omp_get_thread_num() == 1)
            {
                star1[0][i] = 2;
                star1[1][i] = 3;
                star1[2][i] = 2;
                star1[3][i] = 2;
            }
            if (omp_get_thread_num() == 2)
            {
                star1[0][i] = 2;
                star1[1][i] = 2;
                star1[2][i] = 3;
                star1[3][i] = 2;
            }
            if (omp_get_thread_num() == 3)
            {
                star1[0][i] = 2;
                star1[1][i] = 2;
                star1[2][i] = 2;
                star1[3][i] = 3;
            }
        }
    }
    printf("schedule (static, 4) - chunk size = 4:\n");
    for (int j = 0; j < 4; j++)
    {
        for (int k = 0; k < 64; k++)
        {
            if (star1[j][k] == 3)
                printf("*");
            else
                printf(" ");
        }
```

```
        printf("\n");
    }
    int star2[4][64];
#pragma omp parallel num_threads(4), shared(star2, N)
    {
#pragma omp for schedule(static, 8)
        for (int i = 0; i < N; i++)
        {
            if (omp_get_thread_num() == 0)
            {
                star2[0][i] = 3;
                star2[1][i] = 2;
                star2[2][i] = 2;
                star2[3][i] = 2;
            }
            if (omp_get_thread_num() == 1)
            {
                star2[0][i] = 2;
                star2[1][i] = 3;
                star2[2][i] = 2;
                star2[3][i] = 2;
            }
            if (omp_get_thread_num() == 2)
            {
                star2[0][i] = 2;
                star2[1][i] = 2;
                star2[2][i] = 3;
                star2[3][i] = 2;
            }
            if (omp_get_thread_num() == 3)
            {
                star2[0][i] = 2;
                star2[1][i] = 2;
                star2[2][i] = 2;
                star2[3][i] = 3;
            }
        }
    }
    printf("schedule (static, 8) - chunk size = 8:\n");
    for (int j = 0; j < 4; j++)
    {
        for (int k = 0; k < 64; k++)
        {
            if (star2[j][k] == 3)
                printf("*");
            else
                printf(" ");
        }
        printf("\n");
    }
    return 0;
```

```
}
```

Output:



Execution

```
Jacobs-MacBook-Pro:Assignment_5 jacobjohn$ gcc-8 static_star.c -fopenmp
Jacobs-MacBook-Pro:Assignment_5 jacobjohn$ ./a.out
schedule (static):
****************
        ****************
                ****************
                        ****************
schedule (static, 4) - chunk size = 4:
****        ****        ****        ****
  ****        ****        ****        ****
    ****        ****        ****        ****
      ****        ****        ****        ****
schedule (static, 8) - chunk size = 8:
********            ********
    ********            ********
        ********            ********
```

SCENARIO – 2

Write an OpenMP program to specify that the schedule (dynamic, chunk-size) clause of the loop construct specifies that the for loop has the dynamic scheduling type.

Brief about your approach:

Dynamic scheduling works on a "first come, first served" basis. Two runs with the same number of threads might (and most likely would) produce completely different "iteration space" -> "threads" mappings as one can easily verify:

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

int main (void)
{
  int i;

  #pragma omp parallel num_threads(8)
  {
    #pragma omp for schedule(dynamic,1)
    for (i = 0; i < 8; i++)
      printf("[1] iter %0d, tid %0d\n", i, omp_get_thread_num());

    #pragma omp for schedule(dynamic,1)
    for (i = 0; i < 8; i++)
      printf("[2] iter %0d, tid %0d\n", i, omp_get_thread_num());
  }

  return 0;
}

$ icc -openmp -o dyn.x dyn.c

$ OMP_NUM_THREADS=8 ./dyn.x | sort
[1] iter 0, tid 2
[1] iter 1, tid 0
[1] iter 2, tid 7
[1] iter 3, tid 3
[1] iter 4, tid 4
[1] iter 5, tid 1
[1] iter 6, tid 6
[1] iter 7, tid 5
```

(same behavior is observed when gcc is used instead)

If the sample code from the static section was run with dynamic scheduling instead there will be only 1/70 (1.4%) chance that the original locality would be preserved and 69/70 (98.6%) chance that remote access would occur. This fact is often overlooked, and hence suboptimal performance is achieved.

Code:

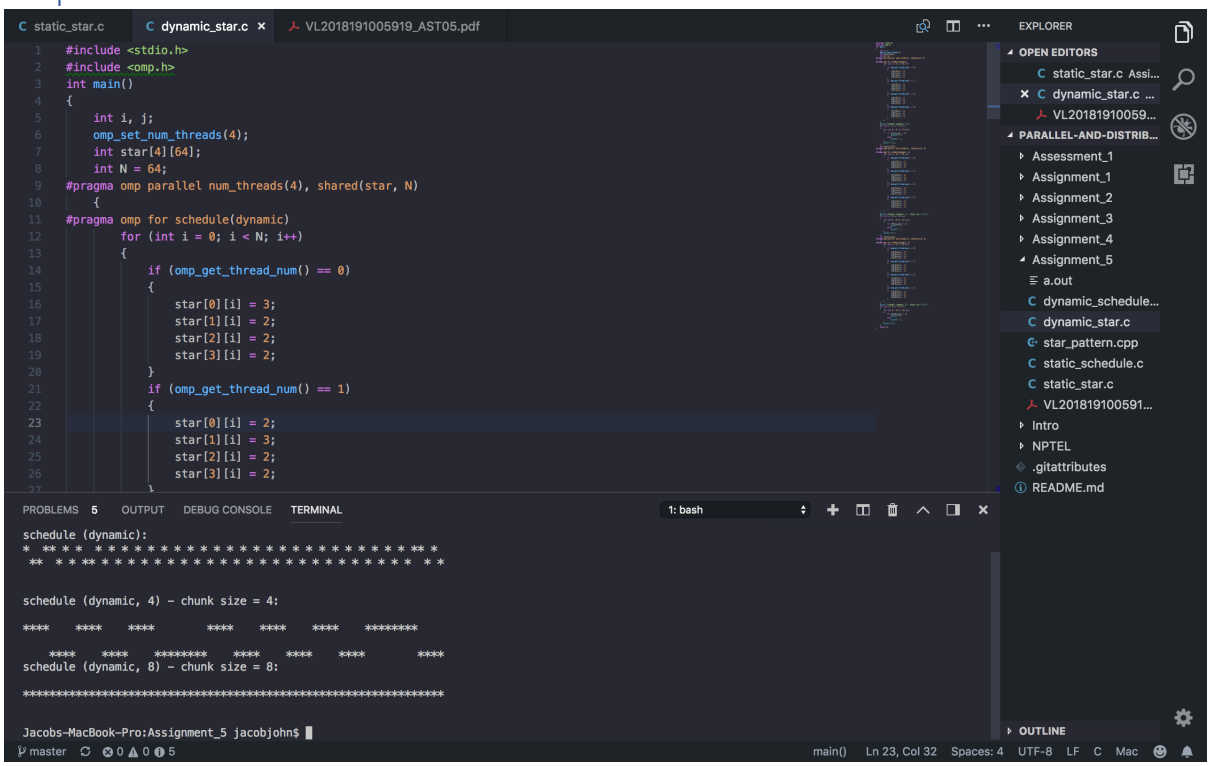
```
#include <stdio.h>
#include <omp.h>
int main()
{
    int i, j;
    omp_set_num_threads(4);
    int star[4][64];
    int N = 64;
#pragma omp parallel num_threads(4), shared(star, N)
```

```
    {
#pragma omp for schedule(dynamic)
        for (int i = 0; i < N; i++)
        {
            if (omp_get_thread_num() == 0)
            {
                star[0][i] = 3;
                star[1][i] = 2;
                star[2][i] = 2;
                star[3][i] = 2;
            }
            if (omp_get_thread_num() == 1)
            {
                star[0][i] = 2;
                star[1][i] = 3;
                star[2][i] = 2;
                star[3][i] = 2;
            }
            if (omp_get_thread_num() == 2)
            {
                star[0][i] = 2;
                star[1][i] = 2;
                star[2][i] = 3;
                star[3][i] = 2;
            }
            if (omp_get_thread_num() == 3)
            {
                star[0][i] = 2;
                star[1][i] = 2;
                star[2][i] = 2;
                star[3][i] = 3;
            }
        }
    }
    printf("schedule (dynamic):\n");
    for (int j = 0; j < 4; j++)
    {
        for (int k = 0; k < 64; k++)
        {
            if (star[j][k] == 3)
                printf("*");
            else
                printf(" ");
        }
        printf("\n");
    }
    int star1[4][64];
#pragma omp parallel num_threads(4), shared(star1, N)
    {
#pragma omp for schedule(dynamic, 4)
        for (int i = 0; i < N; i++)
```

```
        {
            if (omp_get_thread_num() == 0)
            {
                star1[0][i] = 3;
                star1[1][i] = 2;
                star1[2][i] = 2;
                star1[3][i] = 2;
            }
            if (omp_get_thread_num() == 1)
            {
                star1[0][i] = 2;
                star1[1][i] = 3;
                star1[2][i] = 2;
                star1[3][i] = 2;
            }
            if (omp_get_thread_num() == 2)
            {
                star1[0][i] = 2;
                star1[1][i] = 2;
                star1[2][i] = 3;
                star1[3][i] = 2;
            }
            if (omp_get_thread_num() == 3)
            {
                star1[0][i] = 2;
                star1[1][i] = 2;
                star1[2][i] = 2;
                star1[3][i] = 3;
            }
        }
    }
    printf("schedule (dynamic, 4) - chunk size = 4:\n");
    for (int j = 0; j < 4; j++)
    {
        for (int k = 0; k < 64; k++)
        {
            if (star1[j][k] == 3)
                printf("*");
            else
                printf(" ");
        }
        printf("\n");
    }
    int star2[4][64];
#pragma omp parallel num_threads(4), shared(star2, N)
    {
#pragma omp for schedule(dynamic, 8)
        for (int i = 0; i < N; i++)
        {
            if (omp_get_thread_num() == 0)
            {
```

```
                star2[0][i] = 3;
                star2[1][i] = 2;
                star2[2][i] = 2;
                star2[3][i] = 2;
            }
            if (omp_get_thread_num() == 1)
            {
                star2[0][i] = 2;
                star2[1][i] = 3;
                star2[2][i] = 2;
                star2[3][i] = 2;
            }
            if (omp_get_thread_num() == 2)
            {
                star2[0][i] = 2;
                star2[1][i] = 2;
                star2[2][i] = 3;
                star2[3][i] = 2;
            }
            if (omp_get_thread_num() == 3)
            {
                star2[0][i] = 2;
                star2[1][i] = 2;
                star2[2][i] = 2;
                star2[3][i] = 3;
            }
        }
    }
    printf("schedule (dynamic, 8) - chunk size = 8:\n");
    for (int j = 0; j < 4; j++)
    {
        for (int k = 0; k < 64; k++)
        {
            if (star2[j][k] == 3)
                printf("*");
            else
                printf(" ");
        }
        printf("\n");
    }
    return 0;
}
```

Output:



Execution

schedule (dynamic):

```
* ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 ** * * * * * * * * * * * * * * * * * * * * * * * * * * * ** *
```

schedule (dynamic, 4) - chunk size = 4:

```
****   ****   ****       ****   ****   ****   ********

    ****   ****   ********   ****   ****   ****       ****
```

schedule (dynamic, 8) - chunk size = 8:

```
************************************************************
```