Name: Nitesh Sharma
Reg No: 23361
MTCS-103(P)
Parallel Processing Practicals
LabWork 13

==========

OMP_PROC_BIND bind:

The OMP_PROC_BIND environment variable is used to control the binding behavior of threads in an OpenMP program. When OMP_PROC_BIND is set to "true," it indicates that OpenMP threads should be bound to specific processor cores or hardware threads. Conversely, when set to "false," threads are not explicitly bound to specific resources. This variable can influence how threads are mapped to physical hardware resources, which can impact performance and resource utilization in parallel programs. By toggling the value of OMP_PROC_BIND, developers can experiment with different thread binding strategies to achieve better parallel performance or control the allocation of computational resources.

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main() {
  putenv("OMP_PROC_BIND=true");
  int num_threads = omp_get_max_threads();
  printf("Number of Threads: %d\n", num_threads);
  #pragma omp parallel
  {
     int thread_id = omp_get_thread_num();
     printf("Thread %d is executing\n", thread_id);
  }
  return 0;
}
```

Output:

```
et2023@dmacs13-OptiPlex-9020:~/OPENMP/ r( $ gcc -fopenmp omp_proc_bind.c
et2023@dmacs13-OptiPlex-9020:~/OPENMP/ r( $ time ./a.out
Number of Threads: 4
Thread 1 is executing
Thread 3 is executing
Thread 0 is executing
Thread 2 is executing
```

Explanation:
In the above code, the OMP_PROC_BIND environment variable is set to true using the putenv function, indicating that threads should be bound to specific processors. The omp_get_max_threads() function retrieves the maximum number of threads available for

parallel execution. The code then enters an OpenMP parallel region where each thread's execution is printed with its thread ID using omp_get_thread_num(). Since OMP_PROC_BIND is set to true, the OpenMP runtime system will attempt to bind threads to specific processors for improved performance

OMP_WAIT_POLICY policy:

The OMP_WAIT_POLICY environment variable is used to set the waiting behavior of threads in an OpenMP program. It influences the way threads wait during synchronization points. By setting this variable, you can control whether waiting threads actively consume processor cycles while waiting (ACTIVE policy) or if they wait passively without consuming CPU resources (PASSIVE policy). This choice can impact the overall resource utilization and system performance of the OpenMP program.

```c
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
int main() {
 setenv("OMP_THREAD_LIMIT", "2", 1);
 int num_processors = omp_get_num_procs();
 printf("Number of available processors: %d\n", num_processors);
 #pragma omp parallel num_threads(2)
 {
   int thread_num = omp_get_thread_num();
   printf("Thread ID: %d\n", thread_num);
   int num_active_threads = omp_get_num_threads();
   printf("Number of active threads: %d\n", num_active_threads);
 }
 return 0;
}
```

Output:

```
et2023@dmacs13-OptiPlex-9020:~/OPENMP/ r( $ gcc -fopenmp OMP_WAIT_POLICY.c
et2023@dmacs13-OptiPlex-9020:~/OPENMP/ r( $ time ./a.out
Number of available processors: 4
Thread ID: 0
Number of active threads: 2
Thread ID: 1
Number of active threads: 2
```

Explanation:

The code above sets the thread limit to 2 using the setenv() function. This tells the OpenMP runtime to only use 2 threads to execute the parallel region. The code then prints the number of available processors, which is 4. Next, the code starts a parallel region with 2 threads using the #pragma omp parallel num_threads(2) directive. Inside the parallel region, each thread prints its thread ID and the number of active threads. The thread ID is the number of the thread that is executing the code. The number of active threads is the number of threads that are currently running in the parallel region. In this case, the output of the program shows that there are 2 available processors, 2 threads in the parallel region, and

each thread has a thread ID of 0 or 1. This is because the OMP_THREAD_LIMIT environment variable is set to 2, so the OpenMP runtime only uses 2 threads to execute the parallel region. The output of the program also shows that the number of active threads is 2. This is because there are only 2 threads in the parallel region, so there are only 2 active threads.

OMP_MAX_ACTIVE_LEVELS:

The OMP_MAX_ACTIVE_LEVELS environment variable controls the maximum number of nested active parallel regions that OpenMP can support. The default value of this environment variable is 4. This means that OpenMP can support up to 4 levels of nested parallel regions.

```c
#include <stdio.h>
#include <omp.h>
#include<stdlib.h>
int main() {
 setenv("OMP_MAX_ACTIVE_LEVELS", "2", 1);
 int max_active_levels = omp_get_max_active_levels();
 printf("Max active levels: %d\n", max_active_levels);

 #pragma omp parallel num_threads(2)
 {
  int max_active_levels_in_child = omp_get_max_active_levels();
  printf("Max active levels in child: %d\n", max_active_levels_in_child);

  #pragma omp parallel num_threads(1)
  {
   int max_active_levels_in_grandchild = omp_get_max_active_levels();
   printf("Max active levels in grandchild: %d\n", max_active_levels_in_grandchild);
  }
 }

 return 0;
}
```

Output:

```
et2023@dmacs13-OptiPlex-9020:~/OPENMP/3rd$ export OMP_MAX_ACTIVE_LEVELS=2
et2023@dmacs13-OptiPlex-9020:~/OPENMP/3rd$ gcc -fopenmp max_active_levels.c
et2023@dmacs13-OptiPlex-9020:~/OPENMP/3rd$ ./a.out
Max active levels: 2
Max active levels in child: 2
Max active levels in grandchild: 2
Max active levels in child: 2
Max active levels in grandchild: 2
```

Explanation:
This code starts with a call to the setenv() function to set the OMP_MAX_ACTIVE_LEVELS environment variable to 2. This tells OpenMP that the maximum number of nested active parallel regions is 2. Next, the code prints the value of the OMP_MAX_ACTIVE_LEVELS environment variable. The output of this line is Max active levels: 2, which confirms that the environment variable has been set correctly.The code then starts a parallel region with 2 threads. Inside the parallel region, the code prints the value of the OMP_MAX_ACTIVE_LEVELS environment variable again. The output of this line is Max active levels in child: 2, which confirms that the maximum number of nested active parallel regions is still 2 inside the child parallel region.Finally, the code starts a nested parallel region with 1 thread. Inside the nested parallel region, the code prints the value of the OMP_MAX_ACTIVE_LEVELS environment variable again. The output of this line is also Max active levels in grandchild: 2, which confirms that the maximum number of nested active parallel regions is still 2 inside the grandchild parallel region.