BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)

CS ZG501 – Introduction to Parallel and Distributed Programming

Lab#4

Note: Please use programs under *Code* directory supplied with this sheet. Do not copy from this sheet.

The lab has the following objectives:
Giving practice programs for OpenMP.

Compiling and Running an OpenMP Program

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

int main(int argc, char* argv[]) {
    printf("Hello World\n");
    printf("No. of parallel process possible: %d\n", omp_get_num_procs());
#pragma omp parallel
    {
        // printf("I am a parallel region.\n");
        printf("Hi I'm parallel process no. : %d\n", omp_get_thread_num());
    }
    return 0;
}
```

Q?

1. Run the program with `gcc openmp_create.c -fopenmp`. How is the program deciding the number of parallel processes?
2. Find out if it's possible to change the number of available parallel processes for the program.

Monitoring an OpenMP Program

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

```

long fib(int n) {
    return (n < 2 ? 1 : fib(n - 1) + fib(n - 2));
}

int main(int argc, char* argv[]) {
    int n = 42;
#pragma omp parallel
    {
        int t = omp_get_thread_num();
        printf("%d: %ld\n", t, fib(n + t));
    }
    return 0;
}
```

Q?

1. Run the program with `env OMP_NUM_THREADS=8 time ./a.out`.
2. Real time means the actual or "wall-clock" time taken. The user time is the time taken together for all logical cores to run their respective program. Compare the two numbers. Why is one bigger than the other? Is it expected to be this way?

Scope of Variables and reduction clause

Refer to the program given in omp_trap1.c. This program computes integral of a function given its intervals. Trap function is parallelized using the following directive.

```
#   pragma omp parallel num_threads(thread_count)
    Trap(a, b, n, &global_result);
```

Q?

1. Check out whether this will give correct results. Each thread is adding to global_result variable. Modify the code to make sure that global_result variable is protected.
   You can use:
   ```
   #   pragma omp critical
       *global_result_p += my_result;
   ```
2. If we change the parallel directive in the following manner, will it improve

performance? Explain. You can check the time usage using $time ./a.out

```
global_result =0.0;
#  pragma omp parallel num_threads(thread_count)
   {
#      pragma omp critical
      global_result += Local_trap(a,b,n);
   }
```

3. If we change the parallel directive in the following manner, will it improve performance? Explain. You can check the time usage using $time ./a.out

```
global_result = 0.0;
#  pragma omp parallel num_threads(thread_count)
   {
      double my_result = 0.0;
      my_result += Local_trap(a, b, n);
#      pragma omp critical
      global_result += my_result;
   }
```

4. OpenMP provides a cleaner alternative in the form of reduction clause. If we change the parallel directive in the following manner, will it improve performance? Explain. You can check the time usage using $time ./a.out

```
#  pragma omp parallel num_threads(thread_count) \
      reduction(+: global_result)
   global_result += Local_trap(a, b, n);
```

5. If we use parallel for to parallelize loops instead of function, will it improve performance? Can you tell work done in iterations is uniform or not? Which is the best scheduling class and chunk partitioning in this case?

```
double Trap(double a, double b, int n, int thread_count) {
   double  h, approx;
   int  i;
   h = (b-a)/n;
   approx = (f(a) + f(b))/2.0;
#  pragma omp parallel for num_threads(thread_count) \
      reduction(+: approx)
   for (i = 1; i <= n-1; i++)
     approx += f(a + i*h);
   approx = h*approx;
```

```
    return approx;
}  /* Trap */
```

Loop-carried Dependencies

Consider the given code in file omp_pi.c. This program estimates the following series.

$$\pi = 4\left[1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right] = 4\sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}.$$

```
for (i = 0; i < n; i++) {
        sum += factor/(2*i+1);
        factor=-factor;
}
```

Q?

1. Does this loop carry a loop-carried dependence? How can we make it free from loop carried dependency? [Hint: factor = -factor can be eliminated]
2. If we compile and run the code given in omp_pi.c, it will not give correct results. What could be wrong? [Hint: Check the scope of factor variable.]

Odd-even Transposition Sort

Consider the following code for odd-even transposition sort.

```
for (phase = 0; phase < n; phase++) {
        if (phase % 2 == 0)
            for (i = 1; i < n; i += 2) {
                if (a[i-1] > a[i]) {
                    tmp = a[i-1];
                    a[i-1] = a[i];
                    a[i] = tmp;
                }
            }
        else
            for (i = 1; i < n-1; i += 2) {
                if (a[i] > a[i+1]) {
                    tmp = a[i+1];
                    a[i+1] = a[i];
                    a[i] = tmp;
                }
            }
```

Q?

1. Do these two loops carry a loop-carried dependence? Which one can be parallelized? What is the problem in parallelizing outer loop?
2. Write *omp parallel for* statements in omp_odd_even.c file. What should be the scope of i? What should be the scope of tmp? Run the program for 1..6 threads and note the time taken.
3. If we write omp parallel once outside outer for loop and only write “#pragma omp for” for inner loops, will it reduce time. Justify.

Scheduling Loops

OpenMP provides several scheduling classes: static, dynamic, guided, runtime, auto.
Consider Sum() and f() in the file omp_sin_sum.c:

Q?

1. Run the program with static schedule with both block and cyclic partitioning and dynamic schedule and guided schedule. Record the time taken. Explain the difference.

Run the following program

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

int main(int argc, char* argv[]) {
    long int max, sum = 0;
    sscanf(argv[1], "%ld", &max);
#pragma omp parallel for reduction (+:sum) schedule(runtime)
    for (int i = 1; i <= max; i++) {
        printf("%2d @ %d\n", i, omp_get_thread_num());
        sleep(i < 4 ? i + 1 : 1);
        sum += i;
    }
    printf("%ld\n", sum);
    return 0;
16. }
```

Q?

1. Run the above program `env OMP_SCHEDULE=static ./a.out 10`.
2. Change OMP_SCHEDULE to dynamic and see the difference in output.
3. Specify the chunk size for schedule in Line 8.

Combining the Results of Parallel Iterations

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

int main(int argc, char* argv[]) {
    int max, sum = 0;
    sscanf(argv[1], "%d", &max);
#pragma omp parallel for
    for (int i = 1; i <= max; i++)
#pragma omp atomic
        sum += i;
    printf("%d\n", sum);
    return 0;
}
```

Q?

1. Run the above program `./a.out 1000000`.
2. Observe how faster this is compared to critical sections (below).

OpenMP: critical sections

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

int main(int argc, char* argv[]) {
    int max, sum = 0;
    sscanf(argv[1], "%d", &max);
#pragma omp parallel for
    for (int i = 1; i <= max; i++)
#pragma omp critical
        sum += i;
    printf("%d\n", sum);
    return 0;
}
```

Q?

1. Run the above program `./a.out 1000000`.
2. Try removing the *#pragma omp critical*. Do you observe any unexpected output? Why is this?

Lock Free Stack Implementation

Lock free stack is implemented using CAS instructions. Code is given in lf_stack.c and lf_stack_main.c files. Two threads push and pop from a stack a large number of times.

Q?

1. Compile lf_stack_main.c file and run. Observe the output produced. Is it always popping the last pushed value? How can you explain this inconsistent operations although using atomic CAS instruction?[Hint: ABA problem]
2. Can you rectify this problem by delaying the memory freeing? [Hint: Line 47 free() can be commented]

End of lab1