

Lab-Report for Lab-3 (submitted by Parth Parashar)

This is a great lab for learning the workings of OPENMP.

I have never worked with OPENMP and this is a great learning curve for me. The first few exercises tested my basic knowledge of OPENMP.

I did not have any particular trouble with Hello-World (hello-omp.c). It was a great starting point to start working with OpenMP.

The loop-nest did give me some troubles because of the way the threads were working. It became very vague at first and I had to go step by step to understand what was going on. But once I understood the flow of it, It became a little easier. It helped me understand the critical directive.

This helped me in the next questions as well. The recursive routines did give me some troubles as recursion in itself is a tricky concept and when paired with parallelism, it became even trickier. But with the help of sections, it became a little easier to implement.

The lock-ownership program helped me immensely to understand the concept of lock in a practical parallel environment.

The next set of questions were a little tricky. It helped me gain invaluable experience of working on the sections directive, task directive, for directive and how these concepts function to perform task and nested parallelism.

I did get stuck on the sections-parallelism because the sections have to be defined within the scope of the “sections” directive which I was not doing. After spending quite a bit of time on it, I was able to determine the importance of scope in sections directive.

The bank-omp.c and prime-omp.cpp programs were terrific in stimulating our brain to apply the OPENMP directives in some real-world scenario problems.

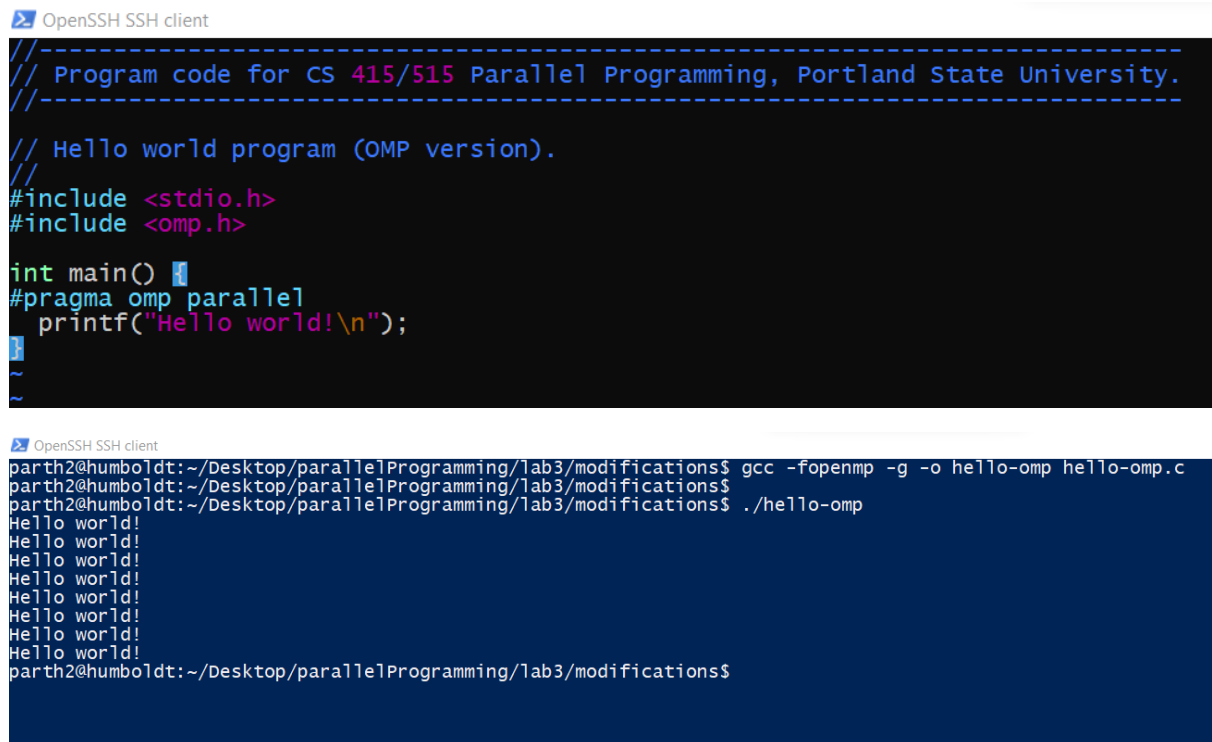
I had to spend a lot of time on it to understand how to approach such problems and which directives can be used where to achieve the desired results.

Overall, I had fun learning about sections, for directives, tasks and how OPENMP is useful in implementing parallel programming problems.

Hello World (hello-omp.c)

- 1) Upon analysing the code, we can safely say that 8 copies of Hello World!! will be printed. This is because it is running on a server with 8 cores and since we have not provided any argument for the number of threads in the directive, it will be equal to the number of cores.

The code along with the output is given below: -



```
OpenSSH SSH client
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----
// Hello world program (OMP version).
//
#include <stdio.h>
#include <omp.h>

int main() {
#pragma omp parallel
    printf("Hello world!\n");
}

~
~

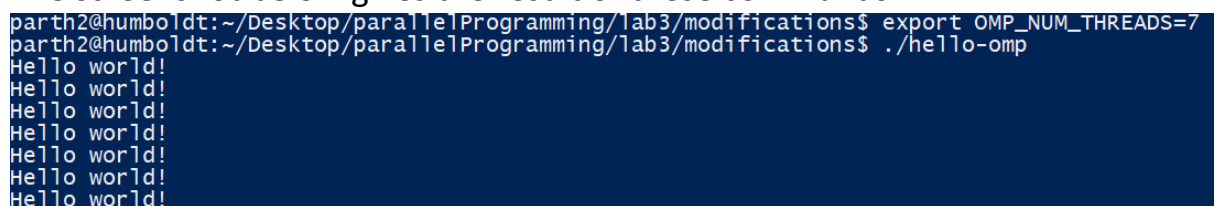
OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ gcc -fopenmp -g -o hello-omp hello-omp.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./hello-omp
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

- 2) A) Environment Variable

This can be done by using the export statement and then running the program as it is.

```
export OMP_NUM_THREADS=7
./hello-omp
```

The screenshot below gives the result of these commands



```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ export OMP_NUM_THREADS=7
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./hello-omp
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
```

B) In-Program Directive

This can be done by adding the in-program directive: -

```
#pragma omp parallel num_threads(4)
```

The modified code is given below: -

```
OpenSSH SSH client
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----
// Hello world program (OMP version).
//
#include <stdio.h>
#include <omp.h>

int main() {
#pragma omp parallel num_threads(4)
    printf("Hello world!\n");
}
~
~
```

Upon compiling and running the program, we get the following output: -

```
OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi hello-omp.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ gcc -fopenmp -g -o hello-omp hello-omp.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./hello-omp
Hello world!
Hello world!
Hello world!
Hello world!
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

Scenario: - When both the environment variables and the in-program directives are changed and run, then, the preference is given to the in-program directive.

3) The code for getting the required output is given below: -

```
OpenSSH SSH client
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----
// Hello world program (OMP version).
//
#include <stdio.h>
#include <omp.h>
#include <sched.h>

int main() {
#pragma omp parallel num_threads(4)
{
    int tid = omp_get_thread_num();
    int core = sched_getcpu();
    printf("Hello world! --thread %d on core %d \n",tid,core);
}
}
```

When this is compiled and run, we get the following output: -

```
OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi hello-omp.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./hello-omp
Hello world! --thread 0 on core 7
Hello world! --thread 3 on core 6
Hello world! --thread 1 on core 4
Hello world! --thread 2 on core 1
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

LOOP NEST (loop-omp*.c)

The code for loop.c is given below: -

```
OpenSSH SSH client
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----
// A simple double loop.
//
#include <stdio.h>
#define N 4
int main() {
    int total=0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            total += i + j;
            printf("[%d,%d]\n", i, j);
        }
    }
    printf("Total = %d (should be 48)\n", total);
}
```

The output of this program after compiling and running it is given below: -

```
OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./loop
[0,0]
[0,1]
[0,2]
[0,3]
[1,0]
[1,1]
[1,2]
[1,3]
[2,0]
[2,1]
[2,2]
[2,3]
[3,0]
[3,1]
[3,2]
[3,3]
Total = 48 (should be 48)
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

1) The code for loop-omp1.c is given below: -

OpenSSH SSH client

```
//-----  
// Program code for CS 415/515 Parallel Programming, Portland State University.  
//-----  
  
// A simple double loop (OMP version 1).  
//  
#include <stdio.h>  
#include <omp.h>  
  
#define N 4  
  
int main() {  
    int total=0;  
  
    #pragma omp parallel for  
    for (int i = 0; i < N; i++) {  
        for (int j = 0; j < N; j++) {  
            total += i + j;  
            printf("[%d,%d]\n", i, j);  
        }  
    }  
    printf("Total = %d (should be 48)\n", total);  
}
```

The output of this program after compiling and running is given below: -

Run-1

OpenSSH SSH client

```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./loop-omp1  
[1,0]  
[1,1]  
[1,2]  
[1,3]  
[3,0]  
[3,1]  
[3,2]  
[3,3]  
[0,0]  
[0,1]  
[0,2]  
[0,3]  
[2,0]  
[2,1]  
[2,2]  
[2,3]  
Total = 46 (should be 48)  
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

Run-2

```

OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./loop-omp1
[0,0]
[0,1]
[0,2]
[0,3]
[1,0]
[1,1]
[1,2]
[1,3]
[2,0]
[2,1]
[2,2]
[2,3]
[3,0]
[3,1]
[3,2]
[3,3]
Total = 42 (should be 48)
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$

```

As we can see that with every run the result is different. This is because of the variable total which is being simultaneously changed by the threads and hence the wrong output.

The code for loop-omp2.c is given below: -

```

OpenSSH SSH client
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----
// A simple double loop (OMP version 2).
//
#include <stdio.h>
#include <omp.h>

#define N 4

int main() {
    int total=0;

    for (int i = 0; i < N; i++) {
#pragma omp parallel for
        for (int j = 0; j < N; j++) {
            total += i + j;
            printf("[%d,%d]\n", i, j);
        }
    }
    printf("Total = %d (should be 48)\n", total);
}

```

The output for this code after compilation is given below: -

Run-1: -

```

parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ gcc -fopenmp -g -o loop-omp2 loop-omp2.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./loop-omp2
[0,0]
[0,2]
[0,3]
[0,1]
[1,0]
[1,2]
[1,3]
[1,1]
[2,3]
[2,1]
[2,2]
[2,0]
[3,1]
[3,3]
[3,2]
[3,0]
Total = 38 (should be 48)

```

Run-2: -

```

parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./loop-omp2
[0,3]
[0,0]
[0,2]
[0,1]
[1,2]
[1,3]
[1,1]
[1,0]
[2,0]
[2,2]
[2,3]
[2,1]
[3,2]
[3,3]
[3,1]
[3,0]
Total = 28 (should be 48)
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$

```

We can see that the outputs vary with each run

2) The code for loop-omp3.c is: -

A) After modifying the code for loop-omp1.c by adding the critical directive to the variable total, we get,

OpenSSH SSH client

```
//-----  
// Program code for CS 415/515 Parallel Programming, Portland State University.  
//-----  
  
// A simple double loop (OMP version 1).  
//  
#include <stdio.h>  
#include <omp.h>  
  
#define N 4  
  
int main() {  
    int total=0;  
  
    #pragma omp parallel for  
    for (int i = 0; i < N; i++) {  
        for (int j = 0; j < N; j++) {  
            #pragma omp critical  
            total += i + j;  
            printf("[%d,%d]\n", i, j);  
        }  
    }  
    printf("Total = %d (should be 48)\n", total);  
}
```

The output of this code is: -

OpenSSH SSH client

```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./loop-omp3  
[0,0]  
[0,1]  
[0,2]  
[0,3]  
[1,0]  
[1,1]  
[1,2]  
[1,3]  
[2,0]  
[2,1]  
[2,2]  
[2,3]  
[3,0]  
[3,1]  
[3,2]  
[3,3]  
Total = 48 (should be 48)  
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./loop-omp3  
[3,0]  
[3,1]  
[3,2]  
[3,3]  
[1,0]  
[1,1]  
[1,2]  
[1,3]  
[2,0]  
[2,1]  
[2,2]  
[2,3]  
[0,0]  
[0,1]  
[0,2]  
[0,3]  
Total = 48 (should be 48)  
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

B) After adding the critical directive to the total variable for loop-omp2, we get,

OpenSSH SSH client

```
//-----  
// Program code for CS 415/515 Parallel Programming, Portland State University.  
//-----  
  
// A simple double loop (OMP version 2).  
//  
#include <stdio.h>  
#include <omp.h>  
  
#define N 4  
  
int main() {  
    int total=0;  
  
    for (int i = 0; i < N; i++) {  
#pragma omp parallel for  
        for (int j = 0; j < N; j++) {  
#pragma omp critical  
            total += i + j;  
            printf("[%d,%d]\n", i, j);  
        }  
    }  
    printf("Total = %d (should be 48)\n", total);  
}
```

The output of this is given below: -

OpenSSH SSH client

```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi loop-omp4.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./loop-omp4
[0,3]
[0,0]
[0,1]
[0,2]
[1,0]
[1,3]
[1,2]
[1,1]
[2,2]
[2,1]
[2,3]
[2,0]
[3,2]
[3,1]
[3,3]
[3,0]
Total = 48 (should be 48)
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./loop-omp4
[0,1]
[0,0]
[0,3]
[0,2]
[1,1]
[1,3]
[1,2]
[1,0]
[2,0]
[2,3]
[2,1]
[2,2]
[3,2]
[3,3]
[3,1]
[3,0]
Total = 48 (should be 48)
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

RECURSIVE ROUTINES(rec-omp*.c)

Code and output for rec.c is given below: -

```
OpenSSH SSH client
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----
// A simple recursive routine.
//
#include <stdio.h>

int array[8] = {1,2,3,4,5,6,7,8};

// array a[]'s size n is assumed to be a power of 2
//
void rec(int a[], int n) {
    if (n == 1) {
        printf("%d\n", a[0]);
    } else {
        rec(a, n/2);
        rec(a+n/2, n/2);
    }
}

int main() {
    rec(array, 8);
}
~
~

OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ gcc -fopenmp -g -o rec rec.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec
1
2
3
4
5
6
7
8
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

1) rec-omp1.c

The code for rec-omp1.c is given below: -

OpenSSH SSH client

```
//-----  
// Program code for CS 415/515 Parallel Programming, Portland State University.  
//-----  
  
// A simple recursive routine (OMP version 1).  
//  
#include <stdio.h>  
#include <omp.h>  
  
int array[8] = {1,2,3,4,5,6,7,8};  
  
// array a[]'s size n is assumed to be a power of 2  
//  
void rec(int a[], int n) {  
    if (n == 1) {  
        printf("%d by thread %d\n", a[0], omp_get_thread_num());  
    } else {  
        #pragma omp parallel sections  
        {  
            #pragma omp section  
                rec(a,n/2);  
            #pragma omp section  
                rec(a+n/2, n/2);  
        }  
    }  
  
int main() {  
    rec(array, 8);  
}
```

OpenSSH SSH client

```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi rec-omp1.c  
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ gcc -fopenmp -g -o rec-omp1 rec-omp1.c  
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp1  
1 by thread 0  
2 by thread 0  
3 by thread 0  
4 by thread 0  
5 by thread 0  
6 by thread 0  
7 by thread 0  
8 by thread 0  
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp1  
1 by thread 0  
2 by thread 0  
5 by thread 0  
6 by thread 0  
7 by thread 0  
8 by thread 0  
3 by thread 0  
4 by thread 0  
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp1  
5 by thread 0  
6 by thread 0  
7 by thread 0  
8 by thread 0  
1 by thread 0  
2 by thread 0  
3 by thread 0  
4 by thread 0  
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

As we can see from the output that output is in different order each time the program is run.

Rec-omp2.c

The code and output after running the code is given below: -

```

OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi rec-omp2.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ gcc -fopenmp -g -o rec-omp2 rec-omp2.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp2
5 by thread 0
6 by thread 0
7 by thread 0
8 by thread 0
1 by thread 0
2 by thread 0
3 by thread 0
4 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp2
5 by thread 0
6 by thread 0
7 by thread 0
8 by thread 0
1 by thread 0
2 by thread 0
3 by thread 0
4 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp2
1 by thread 0
2 by thread 0
3 by thread 0
4 by thread 0
5 by thread 0
6 by thread 0
7 by thread 0
8 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp2
5 by thread 0
6 by thread 0
1 by thread 0
2 by thread 0
7 by thread 0
3 by thread 0
4 by thread 0
8 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$

```

No, I do not see the expected parallelism in both codes as only thread executes the task and then prints it.

2) A) Modifications to the rec-omp1.c to support parallel programming

```

OpenSSH SSH client
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----
// A simple recursive routine (OMP version 1).
//
#include <stdio.h>
#include <omp.h>

int array[8] = {1,2,3,4,5,6,7,8};
// array a[]'s size n is assumed to be a power of 2
//
void rec(int a[], int n) {
    if (n == 1) {
#pragma omp single
        printf("%d by thread %d\n", a[0], omp_get_thread_num());
    } else {
#pragma omp parallel sections
    {
#pragma omp section
    {
        rec(a,n/2);
    }
#pragma omp section
    {
        rec(a+n/2, n/2);
    }
    }
}

int main() {
    rec(array, 8);
}

```

The output when run multiple times is given below: -

OpenSSH SSH client

```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi rec-omp3.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp3
1 by thread 0
2 by thread 0
3 by thread 0
4 by thread 0
5 by thread 0
6 by thread 0
7 by thread 0
8 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp3
1 by thread 0
2 by thread 0
3 by thread 0
4 by thread 0
5 by thread 0
6 by thread 0
7 by thread 0
8 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp3
5 by thread 0
6 by thread 0
7 by thread 0
8 by thread 0
1 by thread 0
2 by thread 0
3 by thread 0
4 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

Modifications to the rec-omp2.c to achieve parallelism is given below: -

OpenSSH SSH client

```
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----

// A simple recursive routine (OMP version 2).
//
#include <stdio.h>
#include <omp.h>

int array[8] = {1,2,3,4,5,6,7,8};

// array a[]'s size n is assumed to be a power of 2
void rec(int a[], int n) {
    if (n == 1) {
        printf("%d by thread %d\n", a[0], omp_get_thread_num());
    } else {
        #pragma omp parallel
        #pragma omp single
        {
            #pragma omp task
            rec(a,n/2);
            #pragma omp taskwait
            rec(a+n/2, n/2);
        }
    }
}

int main() {
    rec(array, 8);
}
```

The output is given below: -

OpenSSH SSH client

```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi rec-omp4.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ gcc -fopenmp -g -o rec-omp4 rec-omp4.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp4
1 by thread 0
2 by thread 0
3 by thread 0
4 by thread 0
5 by thread 0
6 by thread 0
7 by thread 0
8 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp4
1 by thread 0
2 by thread 0
3 by thread 0
4 by thread 0
5 by thread 0
6 by thread 0
7 by thread 0
8 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp4
1 by thread 0
2 by thread 0
3 by thread 0
4 by thread 0
5 by thread 0
6 by thread 0
7 by thread 0
8 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./rec-omp4
1 by thread 0
2 by thread 0
3 by thread 0
4 by thread 0
5 by thread 0
6 by thread 0
7 by thread 0
8 by thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```


LOCK OWNERSHIP(lock-omp.c)

The code given in the lock-omp.c file is given below: -

```
OpenSSH SSH client
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----
// An OMP lock ownership demo.
//
#include <stdio.h>
#include <omp.h>

int main() {
    omp_lock_t lck;
    omp_init_lock(&lck);
    #pragma omp parallel num_threads(2)
    {
        if ((omp_get_thread_num()) == 0) {
            omp_set_lock(&lck);
            printf("Lock set by thread 0\n");
        } else {
            omp_unset_lock(&lck); // !!!
            omp_set_lock(&lck);
            printf("Lock re-set by thread 1\n");
        }
    }
}
```

To check whether the lock locked by thread A is unlocked by thread B, we can compile and run the program and if it goes into an infinite loop, then the condition is validated.

Upon compiling and running the program, we have the following output: -

```
OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi lock-omp.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ gcc -fopenmp -g -o lock-omp lock-omp.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./lock-omp
Lock set by thread 0
Lock re-set by thread 1
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

Here, we can see that the lock is now reset by the other thread.

This can also be verified by checking the thread number which invokes the lock.

The code for the same is given below: -

```
//-----  
// Program code for CS 415/515 Parallel Programming, Portland State University.  
//-----  
  
// An OMP lock ownership demo.  
//  
#include <stdio.h>  
#include <omp.h>  
  
int main() {  
    omp_lock_t lck;  
    omp_init_lock(&lck);  
    #pragma omp parallel num_threads(2)  
    {  
        if ((omp_get_thread_num()) == 0) {  
            omp_set_lock(&lck);  
            printf("Lock set by thread 0 with id: - %d\n", omp_get_thread_num());  
        } else {  
            omp_unset_lock(&lck); // !!!  
            omp_set_lock(&lck);  
            printf("Lock re-set by thread 1 with id: - %d\n", omp_get_thread_num());  
        }  
    }  
}
```

The output when this code is run is: -

```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./lock-omp  
Lock set by thread 0 with id: - 0  
Lock re-set by thread 1 with id: - 1  
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

NESTED PARALLELISM (nested-omp.c)

1)

The code is given as: -

```
Select OpenSSH SSH client
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----
// OMP nested parallel region demo.
//
#include <stdio.h>
#include <omp.h>

void f() {
#pragma omp parallel num_threads(2)
    printf("Hi from thread %d\n", omp_get_thread_num());
}

int main() {
#pragma omp parallel num_threads(2)
    {
        printf("Hello from thread %d\n", omp_get_thread_num());
        f();
    }
}
```

This code produces 4 lines of output as given below: -

```
OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./nested-omp
Hello from thread 0
Hi from thread 0
Hello from thread 1
Hi from thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./nested-omp
Hello from thread 0
Hello from thread 1
Hi from thread 0
Hi from thread 0
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

The output has randomly selected threads running the function. This is because when threads are spawned, they all have the access to the same critical section and same memory area. This means each one of them will execute the resource as and when they get it.

The two parallel directives were run and they were allocated the critical section which leads to a race between the threads and whichever is invoked at the end gets the resource and runs it.

2) When the variable OMP_NESTED is turned to true, then the following output is produced.

```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ export OMP_NESTED=true
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./nested-omp
Hello from thread 0
Hello from thread 1
Hi from thread 0
Hi from thread 1
Hi from thread 0
Hi from thread 1
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

This is because when the OMP_NESTED is turned to true, the threads are run in parallel and each has its own mechanism to prevent the overriding of obtaining the critical section.

TASK PARALLELISM (bank-omp.c)

1) A) sections:-

The code for the same along with the output is given below in the screenshots

OpenSSH SSH client

```
//-----
// Program code for CS 415/515 Parallel Programming, Portland State University.
//-----

// A deposit-withdraw program.
// Usage: ./bank
//

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>

#define INIT_BALANCE 1000
#define NUM_DEPOSITS 10
#define NUM_WITHDRAWS 10
#define DEPOSIT_AMT 100
#define WITHDRAW_AMT 200

int total; // account balance

void deposit(int i) {
    int amount = rand() % DEPOSIT_AMT;
    int oldtotal = total;
    total += amount;
    printf("Deposit-%d %5d (%3d -> %3d)\n", i, amount, oldtotal, total);
}

void withdraw(int i) {
    int amount = rand() % WITHDRAW_AMT;
    int oldtotal = total;
    if (amount < total) {
        total -= amount;
        printf("Withdraw-%d %4d (%3d -> %3d)\n", i, amount, oldtotal, total);
    } else {
        printf("Withdraw-%d %4d (%3d) **aborted**\n", i, amount, oldtotal);
    }
}

int main() {
    total = INIT_BALANCE;
    srand(time(NULL));
#pragma omp parallel sections shared(total)
    {
#pragma omp section
        for (int i = 1; i <= NUM_DEPOSITS; i++)
        {
```

```

int main() {
    total = INIT_BALANCE;
    srand(time(NULL));
#pragma omp parallel sections shared(total)
    {
#pragma omp section
        for (int i = 1; i <= NUM_DEPOSITS; i++)
        {
#pragma omp critical
            {
//                printf("deposit run\n");
                deposit(i);
            }
        }

#pragma omp section
        for (int i = 1; i <= NUM_WITHDRAWS; i++)
        {
#pragma omp critical
            {
//                printf("withdraw run\n");
                withdraw(i);
            }
        }
    }
}

```

The output is as expected and is given below: -

```

OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi bank.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ gcc -fopenmp -g -o bank bank.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./bank
Deposit-1    25 (1000 -> 1025)
Deposit-2    69 (1025 -> 1094)
Deposit-3    74 (1094 -> 1168)
Deposit-4    67 (1168 -> 1235)
Deposit-5    77 (1235 -> 1312)
Deposit-6    76 (1312 -> 1388)
Deposit-7    66 (1388 -> 1454)
Deposit-8    74 (1454 -> 1528)
Deposit-9    86 (1528 -> 1614)
Deposit-10   52 (1614 -> 1666)
Withdraw-1   101 (1666 -> 1565)
Withdraw-2   168 (1565 -> 1397)
Withdraw-3    46 (1397 -> 1351)
Withdraw-4    33 (1351 -> 1318)
Withdraw-5    19 (1318 -> 1299)
Withdraw-6     2 (1299 -> 1297)
Withdraw-7    59 (1297 -> 1238)
Withdraw-8    14 (1238 -> 1224)
Withdraw-9     9 (1224 -> 1215)
Withdraw-10   91 (1215 -> 1124)
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$

```

B) Tasks

The code and the output is given below: -

```
//-----  
// Program code for CS 415/515 Parallel Programming, Portland State University.  
//-----  
  
// A deposit-withdraw program.  
// Usage: ./bank  
//  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <omp.h>  
  
#define INIT_BALANCE 1000  
#define NUM_DEPOSITS 10  
#define NUM_WITHDRAWS 10  
#define DEPOSIT_AMT 100  
#define WITHDRAW_AMT 200  
  
int total; // account balance  
  
void deposit(int i) {  
    int amount = rand() % DEPOSIT_AMT;  
    int oldtotal = total;  
    total += amount;  
#pragma omp task  
    printf("Deposit-%d %5d (%3d -> %3d)\n", i, amount, oldtotal, total);  
}  
  
void withdraw(int i) {  
    int amount = rand() % WITHDRAW_AMT;  
    int oldtotal = total;  
    if (amount < total) {  
        total -= amount;  
#pragma omp task  
        printf("Withdraw-%d %4d (%3d -> %3d)\n", i, amount, oldtotal, total);  
    } else {  
        printf("Withdraw-%d %4d (%3d) **aborted**\n", i, amount, oldtotal);  
    }  
}  
  
int main() {  
    total = INIT_BALANCE;  
    srand(time(NULL));  
#pragma omp parallel shared(total)  
#pragma omp single  
    {
```

```

int main() {
    total = INIT_BALANCE;
    srand(time(NULL));
#pragma omp parallel shared(total)
#pragma omp single
    {
        for (int i = 1; i <= NUM_DEPOSITS; i++)
        {
#pragma omp taskwait
            {
                deposit(i);
            }
        }

        for (int i = 1; i <= NUM_WITHDRAWS; i++)
        {
#pragma omp taskwait
            {
                withdraw(i);
            }
        }
    }
}

```

The output is given below: -

```

OpenSSH SSH client
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi bank.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./bank
Deposit-1      49 (1000 -> 1049)
Deposit-2      92 (1049 -> 1141)
Deposit-3       3 (1141 -> 1144)
Deposit-4      19 (1144 -> 1163)
Deposit-5      47 (1163 -> 1210)
Deposit-6       1 (1210 -> 1211)
Deposit-7      80 (1211 -> 1291)
Deposit-8      39 (1291 -> 1330)
Deposit-9      98 (1330 -> 1428)
Deposit-10     26 (1428 -> 1454)
Withdraw-1      28 (1454 -> 1426)
Withdraw-2     197 (1426 -> 1229)
Withdraw-3     190 (1229 -> 1039)
Withdraw-4     181 (1039 -> 858)
Withdraw-5     154 (858 -> 704)
Withdraw-6       8 (704 -> 696)
Withdraw-7     142 (696 -> 554)
Withdraw-8      20 (554 -> 534)
Withdraw-9     105 (534 -> 429)
Withdraw-10    161 (429 -> 268)
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$

```

C) For loop

The code and the output is given below: -


```
#define INIT_BALANCE 1000
#define NUM_DEPOSITS 10
#define NUM_WITHDRAWS 10
#define DEPOSIT_AMT 100
#define WITHDRAW_AMT 200

int total; // account balance

void deposit(int i) {
    int amount = rand() % DEPOSIT_AMT;
    int oldtotal = total;
    total += amount;
    printf("Deposit-%d %5d (%3d -> %3d)\n", i, amount, oldtotal, total);
}

void withdraw(int i) {
    int amount = rand() % WITHDRAW_AMT;
    int oldtotal = total;
    if (amount < total) {
        total -= amount;
        printf("Withdraw-%d %4d (%3d -> %3d)\n", i, amount, oldtotal, total);
    } else {
        printf("Withdraw-%d %4d (%3d) **aborted**\n", i, amount, oldtotal);
    }
}

int main() {
    total = INIT_BALANCE;
    srand(time(NULL));
#pragma omp parallel for shared(total)
    {
        for (int i = 1; i <= NUM_DEPOSITS; i++)
        {
            {
                deposit(i);
            }
        }

        for (int i = 1; i <= NUM_WITHDRAWS; i++)
        {
            {
                withdraw(i);
            }
        }
    }
}
```

OpenSSH SSH client

```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi bank.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./bank
Deposit-1    49 (1000 -> 1049)
Deposit-2    92 (1049 -> 1141)
Deposit-3     3 (1141 -> 1144)
Deposit-4    19 (1144 -> 1163)
Deposit-5    47 (1163 -> 1210)
Deposit-6     1 (1210 -> 1211)
Deposit-7    80 (1211 -> 1291)
Deposit-8    39 (1291 -> 1330)
Deposit-9    98 (1330 -> 1428)
Deposit-10   26 (1428 -> 1454)
Withdraw-1   28 (1454 -> 1426)
Withdraw-2  197 (1426 -> 1229)
Withdraw-3  190 (1229 -> 1039)
Withdraw-4  181 (1039 -> 858)
Withdraw-5  154 (858 -> 704)
Withdraw-6    8 (704 -> 696)
Withdraw-7  142 (696 -> 554)
Withdraw-8   20 (554 -> 534)
Withdraw-9  105 (534 -> 429)
Withdraw-10 161 (429 -> 268)
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$
```

2) There is a need for synchronization because the threads use a lot of directives and that makes the task of debugging difficult. Also, the process slows down as well.

To implement synchronization, I would use task and taskwithdraw.

The code and the output is given below: -

```
//-----  
// Program code for CS 415/515 Parallel Programming, Portland State University.  
//-----  
  
// A deposit-withdraw program.  
// Usage: ./bank  
//  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <omp.h>  
  
#define INIT_BALANCE 1000  
#define NUM_DEPOSITS 10  
#define NUM_WITHDRAWS 10  
#define DEPOSIT_AMT 100  
#define WITHDRAW_AMT 200  
  
int total; // account balance  
  
void deposit(int i) {  
    int amount = rand() % DEPOSIT_AMT;  
    int oldtotal = total;  
    total += amount;  
#pragma omp task  
    printf("Deposit-%d %5d (%3d -> %3d)\n", i, amount, oldtotal, total);  
}  
  
void withdraw(int i) {  
    int amount = rand() % WITHDRAW_AMT;  
    int oldtotal = total;  
    if (amount < total) {  
        total -= amount;  
#pragma omp task  
        printf("Withdraw-%d %4d (%3d -> %3d)\n", i, amount, oldtotal, total);  
    } else {  
        printf("Withdraw-%d %4d (%3d) **aborted**\n", i, amount, oldtotal);  
    }  
}  
  
int main() {  
    total = INIT_BALANCE;  
    srand(time(NULL));  
#pragma omp parallel shared(total)  
#pragma omp single  
    {
```

```

int main() {
    total = INIT_BALANCE;
    srand(time(NULL));
#pragma omp parallel shared(total)
#pragma omp single
    {
        for (int i = 1; i <= NUM_DEPOSITS; i++)
        {
#pragma omp taskwait
            {
                deposit(i);
            }
        }

        for (int i = 1; i <= NUM_WITHDRAWS; i++)
        {
#pragma omp taskwait
            {
                withdraw(i);
            }
        }
    }
}

```

OpenSSH SSH client

```

parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ vi bank.c
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./bank
Deposit-1      49 (1000 -> 1049)
Deposit-2      92 (1049 -> 1141)
Deposit-3       3 (1141 -> 1144)
Deposit-4      19 (1144 -> 1163)
Deposit-5      47 (1163 -> 1210)
Deposit-6       1 (1210 -> 1211)
Deposit-7      80 (1211 -> 1291)
Deposit-8      39 (1291 -> 1330)
Deposit-9      98 (1330 -> 1428)
Deposit-10     26 (1428 -> 1454)
Withdraw-1     28 (1454 -> 1426)
Withdraw-2    197 (1426 -> 1229)
Withdraw-3    190 (1229 -> 1039)
Withdraw-4    181 (1039 -> 858)
Withdraw-5    154 (858 -> 704)
Withdraw-6     8 (704 -> 696)
Withdraw-7    142 (696 -> 554)
Withdraw-8     20 (554 -> 534)
Withdraw-9    105 (534 -> 429)
Withdraw-10   161 (429 -> 268)
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$

```

PRIME FINDING (prime.cpp)

The code given is: -

```
OpenSSH SSH client
-----
Program code for CS 415P/S15 Parallel Programming, Portland State University
-----
// A prime-finding program (Sequential version).
// Usage:
// linux> ./prime N
#include <iostream>
#include <cmath>
#include <omp.h>
using namespace std;

int main(int argc, char **argv) {
    int N = 2;
    if (argc < 2) {
        cout << "Usage: ./prime N\n";
        exit(0);
    }
    if ((N=atoi(argv[1])) < 2) {
        cout << "N must be greater than 1\n";
        exit(0);
    }
    bool *compo = new bool[N+1]{};

    for (int i = 2; i <= sqrt(N); i++)
        if (!compo[i])
            #pragma omp parallel
            for (int j = i+i; j <= N; j += i)
            {
                int threadnum = omp_get_thread_num();
                #pragma omp critical
                cout << "Thread["<<threadnum<<"]<<" working on prime number "<<i<<" (first composite number:- "<<j<<)"<<endl;
                compo[j] = true;
            }

    int cnt = 0;
    for (int i = 2; i <= N; i++)
        if (!compo[i]) cnt++;
    cout << "Found " << cnt << " primes in 2.." << N << "\n";
    for (int i = 2; i <= N; i++)
        if (!compo[i]) cout << to_string(i) + ",";
    cout << "\n";
}
```

The output for the same is given below: -

OpenSSH SSH client

```
parth2@humboldt:~/Desktop/parallelProgramming/lab3/modifications$ ./prime-omp 10
Thread[0] working on prime number 2
Thread[3] working on prime number 2
Thread[3] working on prime number 2
Thread[3] working on prime number 2
Thread[3] working on prime number 2
Thread[0] working on prime number 2
Thread[2] working on prime number 2
Thread[2] working on prime number 2
Thread[2] working on prime number 2
Thread[2] working on prime number 2
Thread[1] working on prime number 2
Thread[1] working on prime number 2
Thread[1] working on prime number 2
Thread[1] working on prime number 2
Thread[0] working on prime number 2
Thread[0] working on prime number 2
Thread[3] working on prime number 3
Thread[2] working on prime number 3
Thread[1] working on prime number 3
Thread[2] working on prime number 3
Thread[1] working on prime number 3
Thread[3] working on prime number 3
Thread[0] working on prime number 3
Thread[0] working on prime number 3
Found 4 primes in 2..10
2,3,5,7,
```