Section 1 Comparative Sequential Performance (2 marks)

Data Set	Go runtimes(s)	C runtimes (s)	Erlang Runtimes(s)
DS1	16.51070541	19.509976	28.184469
DS2	70.57904187	84.544474	120.200621

Table 1: Comparing C, Erlang, and Go Sequential Runtimes

Section 2 Two Worker Totient Range Erlang Program: totientrange2Workers (6 marks)

I think the interesting feature of this program is that I should create three processes in total. One for server and two for worker to calculate the sum of totient.

The source code is in Appendix C.

The output on range (1,4000) and (1,15000) screenshot sample in my own laptop is as below:

```
chongbin@CharlesRen-mac:~/Desktop/project/erlang_project/con/con2$ erl
Erlang/OTP 21 [erts-10.1.1] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [hipe] [dtrace]

Eshell V10.1.1  (abort with ^G)
1> c(con2).
{ok,con2}
2> con2:start_server().
true
3> server!{range,1,4000}.
Server : Compute range 1 4000
{range,1,4000}
Worker 1 : compute range 1 2000
Worker 2 : compute range 2001 4000
Server :Received Sum 1216588
Worker : Finished!
Server :Received Sum 3647014
Worker : Finished!
Server 0:Sum of totients: 4863602
Time taken in 1.033  seconds
4> con2:start_server().
true
5> server!{range,1,15000}.
Server : Compute range 1 15000
{range,1,15000}
Worker 1 : compute range 1 7500
Worker 2 : compute range 7501 15000
Server :Received Sum 17099412
Worker : Finished!
Server :Received Sum 51294904
Worker : Finished!
Server 0:Sum of totients: 68394316
Time taken in 15.664  seconds
6>
```

The output from execution on range(1,4000) is :

```
Eshell V10.1.1  (abort with ^G)
1> c(con2).
{ok,con2}
2> con2:start_server().
true
3> server!{range,1,4000}.
```

```
Server : Compute range 1 4000
{range,1,4000}
Worker 1 : compute range 1 2000
Worker 2 : compute range 2001 4000
Server :Received Sum 1216588
Worker : Finished!
Server :Received Sum 3647014
Worker : Finished!
Server 0:Sum of totients: 4863602
Time taken in 1.033  seconds
```

The output from execution on range(1,15000) is

```
4> con2:start_server().
true
5> server!{range,1,15000}.
Server : Compute range 1 15000
{range,1,15000}
Worker 1 : compute range 1 7500
Worker 2 : compute range 7501 15000
Server :Received Sum 17099412
Worker : Finished!
Server :Received Sum 51294904
Worker : Finished!
Server 0:Sum of totients: 68394316
Time taken in 15.664  seconds
```

Section 3 Multi Worker Erlang Totient Range Program: totientrangeNWorkers (10 marks)

Interesting features: I found there is no for or while loop in Erlang program because the variable cannot be changed once it was assigned. So in Erlang I have to use recursion instead of for loop to create N worker processes.

The source code is in Appendix D.

The output from execution on ranges (1,4000) with 4 workers:

```
1> c(conN).
conN.erl:69: Warning: variable 'LOW' is unused
conN.erl:69: Warning: variable 'NUM' is unused
conN.erl:69: Warning: variable 'UPPER' is unused
{ok,conN}
2> conN:start_server(4).
true
3> server!{range,1,4000}.
Server :received range 1 4000
calculate
{range,1,4000}
Worker : started
Worker : started
```

```
Worker : started
Worker : started
Worker : compute range 3001 4000
Worker : compute range 2001 3000
Worker : compute range 1001 2000
Worker : compute range 1 1000
Worker : finished
Server :received Sum 304192
Worker : finished
Server :received Sum 912396
Worker : finished
Server :received Sum 1519600
Worker : finished
Server :received Sum 2127414
Server : Sum of totients: 4863602
Time taken in 0.793  seconds
```

The output from execution on ranges (1,15000) with 6 workers:

```
{ok,conN}
2> conN:start_server(6).
true
3> server!{range,1,15000}.
Server :received range 1 15000
calculate
{range,1,15000}
Worker : started
Worker : started
Worker : started
Worker : started
Worker : started
Worker : started
Worker : compute range 12501 15000
Worker : compute range 10001 12500
Worker : compute range 7501 10000
Worker : compute range 5001 7500
Worker : compute range 2501 5000
Worker : compute range 1 2500
Worker : finished
Server :received Sum 1899878
Worker : finished
Server :received Sum 5700580
Worker : finished
Server :received Sum 9498954
Worker : finished
Server :received Sum 13298074
Worker : finished
Server :received Sum 17100872
Worker : finished
Server :received Sum 20895958
Server : Sum of totients: 68394316
Time taken in 9.184  seconds
```

Section 4 Reliable Multi Worker Erlang Totient Range Program: totientrangeNWorkersReliable (14 marks)

Interesting features: I should link every worker process with a watcher process and when the process killed I found the link process can catch it and do something. It is very powerful and reliable.

The source code is in Appendix E.

the output from execution with testRobust(4,3):

```
chongbin@CharlesRen-mac:~/Desktop/project/erlang_project/con/reliable$ erl
Erlang/OTP 21 [erts-10.1.1] [source] [64-bit] [smp:4:4] [ds:4:4:10] [async-threads:1] [hipe] [dtrace]
Eshell V10.1.1  (abort with ^G)
1> c(reliable).
{ok,reliable}
2> reliable:testRobust(4,3).
Server :received range 1 15000
calculate
worker4 : started
worker3 : started
worker2 : started
worker1 : started
Watcher : Watching  worker4
Watcher : Watching  worker3
Watcher : Watching  worker2
Watcher : Watching  worker1
worker4 : compute range 11251 15000
worker3 : compute range 7501 11250
worker2 : compute range 3751 7500
worker1 : compute range 1 3750
workerChaos killing worker2
Watcher: restart : worker2
worker2 : started
Watcher : Watching  worker2
worker2 : compute range 3751 7500
workerChaos killing worker3
Watcher: restart : worker3
worker3 : started
Watcher : Watching  worker3
worker3 : compute range 7501 11250
workerChaos killing worker3
Watcher: restart : worker3
worker3 : started
Watcher : Watching  worker3
[true,true,true]
worker3 : compute range 7501 11250
worker1 : finished
Server :received Sum 4275174
```

```
watcher: nomal : worker1
worker2 : finished
Server :received Sum 12824238
watcher: nomal : worker2
worker3 : finished
watcher: nomal : worker3
Server :received Sum 21371600
worker4 : finished
Server :received Sum 29923304
watcher: nomal : worker4
Server : Sum of totients: 68394316
Time taken in 11.439  seconds
```

the output from execution with testRobust(8,6)

```
3> c(reliable).
{ok,reliable}
4> reliable:testRobust(8,6).
Server :received range 1 15000
calculate
worker8 : started
worker7 : started
worker6 : started
worker5 : started
worker4 : started
worker3 : started
worker2 : started
worker1 : started
Watcher : Watching  worker8
Watcher : Watching  worker7
Watcher : Watching  worker6
Watcher : Watching  worker5
Watcher : Watching  worker4
Watcher : Watching  worker3
Watcher : Watching  worker2
Watcher : Watching  worker1
worker8 : compute range 13126 15000
worker7 : compute range 11251 13125
worker6 : compute range 9376 11250
worker5 : compute range 7501 9375
worker4 : compute range 5626 7500
worker3 : compute range 3751 5625
worker2 : compute range 1876 3750
worker1 : compute range 1 1875
workerChaos killing worker7
Watcher: restart : worker7
worker7 : started
Watcher : Watching  worker7
worker7 : compute range 11251 13125
worker1 : finished
Server :received Sum 1069230
watcher: nomal : worker1
workerChaos killing worker5
Watcher: restart : worker5
worker5 : started
```

```
Watcher : Watching  worker5
worker5 : compute range 7501 9375
workerChaos killing worker6
Watcher: restart : worker6
worker6 : started
Watcher : Watching  worker6
worker6 : compute range 9376 11250
workerChaos killing worker5
Watcher: restart : worker5
worker5 : started
Watcher : Watching  worker5
worker5 : compute range 7501 9375
worker2 : finished
Server :received Sum 3205944
watcher: nomal : worker2
workerChaos killing worker5
Watcher: restart : worker5
worker5 : started
Watcher : Watching  worker5
worker5 : compute range 7501 9375
workerChaos killing worker2
workerChaos already dead: worker2
[true,true,true,true,true,ok]
worker4 : finished
Server :received Sum 7480526
watcher: nomal : worker4
worker3 : finished
Server :received Sum 5343712
watcher: nomal : worker3
worker5 : finished
Server :received Sum 9617600
watcher: nomal : worker5
worker6 : finished
Server :received Sum 11754000
watcher: nomal : worker6
worker7 : finished
Server :received Sum 13894072
watcher: nomal : worker7
worker8 : finished
Server :received Sum 16029232
watcher: nomal : worker8
Server : Sum of totients: 68394316
Time taken in 10.218  seconds
```

the output from execution with testRobust(12,10)

```
5> c(reliable).
{ok,reliable}
6> reliable:testRobust(12,10).
Server :received range 1 15000
calculate
worker12 : started
worker11 : started
worker10 : started
worker9 : started
worker8 : started
worker7 : started
worker6 : started
```

worker5 : started
worker4 : started
worker3 : started
worker2 : started
worker1 : started
Watcher : Watching worker12
Watcher : Watching worker11
Watcher : Watching worker10
Watcher : Watching worker8
Watcher : Watching worker7
Watcher : Watching worker6
Watcher : Watching worker9
Watcher : Watching worker5
Watcher : Watching worker4
Watcher : Watching worker3
Watcher : Watching worker2
Watcher : Watching worker1
worker12 : compute range 13751 15000
worker11 : compute range 12501 13750
worker10 : compute range 11251 12500
worker9 : compute range 10001 11250
worker8 : compute range 8751 10000
worker7 : compute range 7501 8750
worker6 : compute range 6251 7500
worker5 : compute range 5001 6250
worker4 : compute range 3751 5000
worker3 : compute range 2501 3750
worker1 : compute range 1 1250
worker2 : compute range 1251 2500
workerChaos killing worker6
Watcher: restart : worker6
worker6 : started
Watcher : Watching worker6
worker6 : compute range 6251 7500
worker1 : finished
watcher: nomal : worker1
Server :received Sum 475306
workerChaos killing worker4
Watcher: restart : worker4
worker4 : started
Watcher : Watching worker4
worker4 : compute range 3751 5000
workerChaos killing worker7
Watcher: restart : worker7
worker7 : started
Watcher : Watching worker7
worker7 : compute range 7501 8750
worker2 : finished
Server :received Sum 1424572
watcher: nomal : worker2
workerChaos killing worker8
Watcher: restart : worker8
worker8 : started
Watcher : Watching worker8
worker8 : compute range 8751 10000
workerChaos killing worker7
Watcher: restart : worker7

worker7 : started
Watcher : Watching worker7
worker7 : compute range 7501 8750
workerChaos killing worker6
Watcher: restart : worker6
worker6 : started
Watcher : Watching worker6
worker6 : compute range 6251 7500
worker3 : finished
Server :received Sum 2375296
watcher: nomal : worker3
workerChaos killing worker7
Watcher: restart : worker7
worker7 : started
Watcher : Watching worker7
worker7 : compute range 7501 8750
workerChaos killing worker9
Watcher: restart : worker9
worker9 : started
Watcher : Watching worker9
worker9 : compute range 10001 11250
worker5 : finished
Server :received Sum 4274384
watcher: nomal : worker5
workerChaos killing worker10
Watcher: restart : worker10
worker10 : started
Watcher : Watching worker10
worker10 : compute range 11251 12500
workerChaos killing worker5
workerChaos already dead: worker5
[true,true,true,true,true,true,true,true,true,ok]
worker4 : finished
watcher: nomal : worker4
Server :received Sum 3325284
worker6 : finished
Server :received Sum 5224570
watcher: nomal : worker6
worker8 : finished
Server :received Sum 7122972
watcher: nomal : worker8
worker7 : finished
Server :received Sum 6175102
watcher: nomal : worker7
worker11 : finished
watcher: nomal : worker11
Server :received Sum 9970596
worker12 : finished
Server :received Sum 10925362
watcher: nomal : worker12
worker9 : finished
Server :received Sum 8073526
watcher: nomal : worker9
worker10 : finished
Server :received Sum 9027346
watcher: nomal : worker10
Server : Sum of totients: 68394316

Time taken in 9.983 seconds

Section 5 Comparative Parallel Performance Measurements (12 marks)

Section 5.1 Runtimes.

In my own program I changed the time precision display in console.

DS1: execution times for the sequential C, Erlang, and Go programs, and the parallel C+OpenMP, Erlang and Go programs on 1, 2, 4, 8, 12, 16, 24, 32, 48, 64 threads/goroutines/totientWorker processes on a GPG Cluster node.

Threads /goroutines	go runtime(s)	C runtime(s)	Erlang runtime(s)
sequential	16.51070541	19.509976	28.184469
1	16.56481488	19.723	28.196
2	12.80199701	15.062	21.943
4	7.784804143	9.038	13.273
8	4.193619101	5.013	7.338
12	2.901805619	3.461	5.212
16	2.28248696	2.644	4.021
24	1.546794145	1.819	2.952
32	1.298523879	1.407	2.328
48	1.037974181	1.049	2.009
64	0.926042525	1.108	1.766

DS2: execution times for the sequential C, Erlang, and Go programs, and the parallel C+OpenMP, Erlang and Go programs on 1, 2, 4, 8, 12, 16, 24, 32, 48, 64 threads/goroutines/totientWorker processes on a GPG Cluster node.

Threads /goroutines	go runtime(s)	C runtime(s)	Erlang runtime(s)
sequential	70.57904187	84.544556	120.200621
1	70.5348553	84.285	119.315
2	54.01022328	63.725	92.772
4	32.24430842	38.512	56.142
8	17.88690512	21.33	31.115
12	12.36136477	14.762	21.771
16	9.522891973	11.203	16.761
24	6.633120542	7.901	12.662
32	5.341379413	6.015	9.997
48	4.02412511	4.509	8.183
64	3.820356588	4.245	7.604

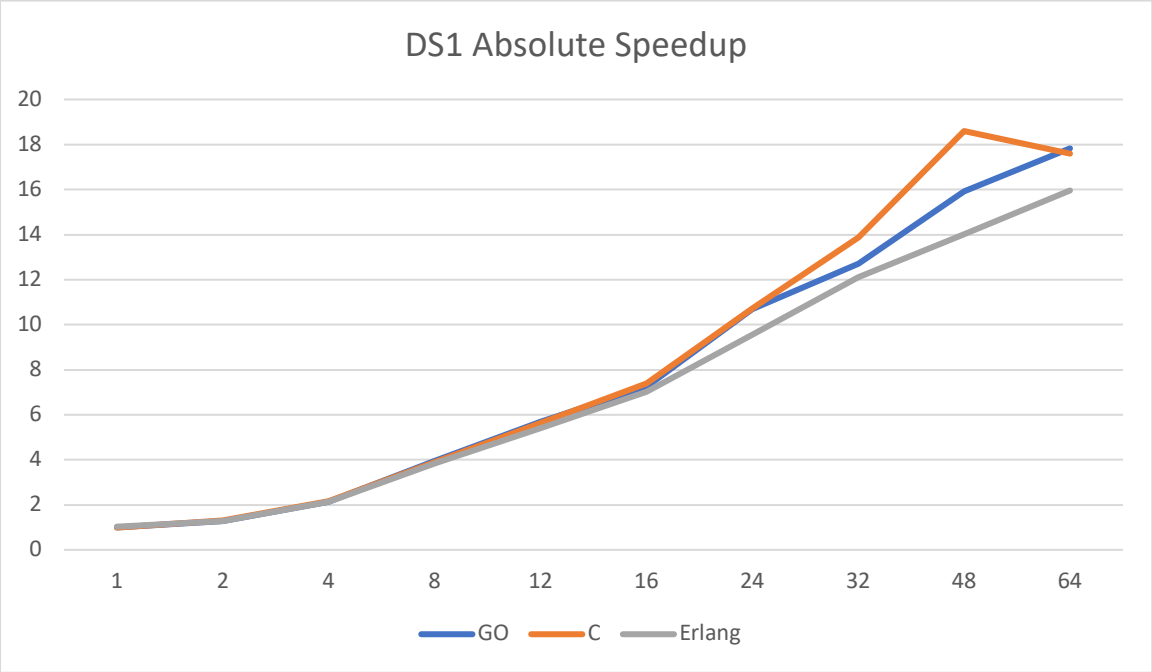
DS3: execution times for the parallel C+OpenMP, Erlang and Go programs on 8, 16, 32, 64, threads/ goroutines on a GPG Cluster node.

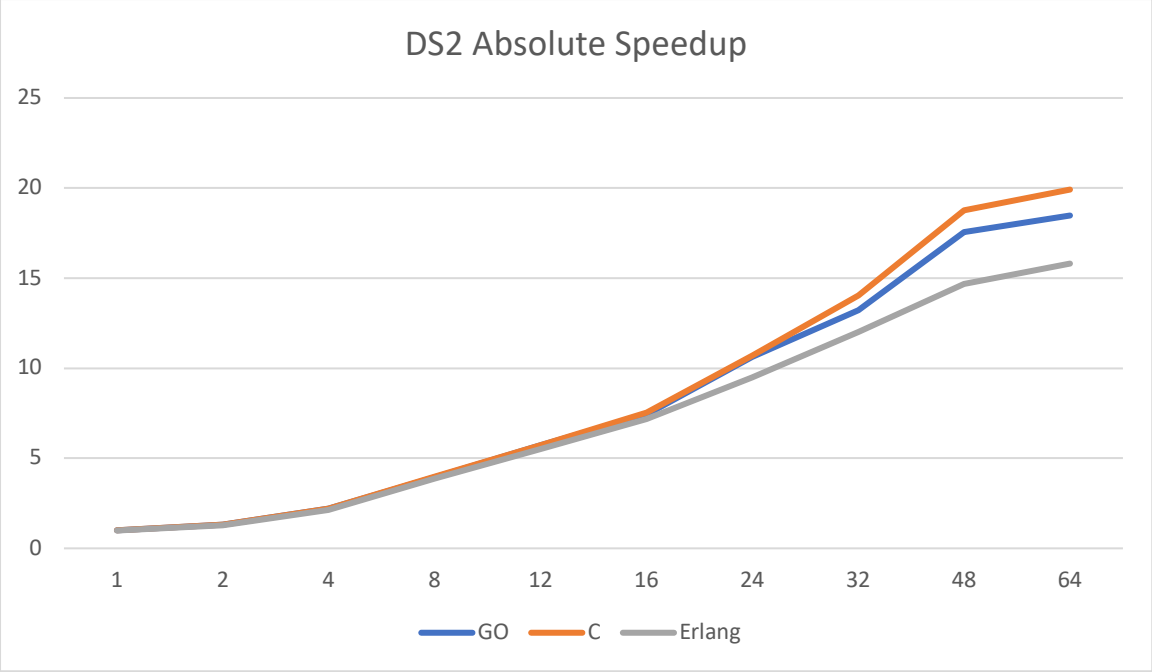
Threads /goroutines	go runtime(s)	C runtime(s)	Erlang runtime(s)

sequential	4m59.924110813	358.49216	508.647701
8	1m17.073047662	90.206	131.687
16	43.11157582	47.782	72.253
32	22.97898839	26.073	42.295
64	16.69879699	17.615	33.084

Section 5.2 Speedups.

X axis is threads/goroutines number, Y axis is absolute speedup




DS2 Absolute Speedup
25
20
15
10
5
0
1
2
4
8
12
16
24
32
48
64
GO
C
Erlang


DS3 Absolute Speedup
25
20
15
10
5
0
8
16
32
64
GO
C
Erlang

Section 5.3

Complete the table below summarising the sequential performance and the best parallel runtimes of your C+OpenMP and Go programs and Erlang program.

DS1

Language	Sequential runtimes(s)	Best parallel runtimes(s)	Best Speedup	No.cores
go	16.510	0.926	17.829	64
C+OpenMP	19.509	1.049	18.597	48
Erlang	28.184	1.766	15.959	64

DS2

Language	Sequential runtimes(s)	Best parallel runtimes(s)	Best Speedup	No.cores
go	70.579	3.82	18.476	64
C+OpenMP	84.544	4.245	19.916	64
Erlang	120.201	7.604	15.807	64

DS3

Language	Sequential runtimes(s)	Best parallel runtimes(s)	Best Speedup	No.cores
go	299.924	16.698	17.960	64
C+OpenMP	358.492	17.615	20.351	64
Erlang	508.647	33.084	15.374	64

Section 5.4

A discussion of the comparative performance of the C+OpenMP, Erlang, and Go programs.

From the runtimes data we can see that the C+OpenMP and go have similar runtimes and performance. However, the Erlang program's runtime is much slower in calculating the sum. In calculating the sum, go program performs the best in these three programing language. Additionally, C+OpenMP performs better in speedup.

Section 6 Programming Model Comparison (6 marks).

When I programming the parallel program, I should assign specific tasks for Go program. For example, I should assign 1-1000, 1001-2001….to goroutines. But for C+OpenMP, it will assign the tasks automatically. But Go perform better in computing sum of totient compared C+OpenMP in the same settings. In comparing DS3, go performs much better than C+OpenMP in 8 and 16 cores. However, when the number of cores increase to 48 and 64, there is even no difference in go and openMP.

Section 7 Reflection on Programming Models MSc Students Only (5 marks)

Go performs better both in sequential program and parallel program than C+OpenMP. Maybe the advantage of go is that it performs well and you can manipulate threads easier. For openMP I do think you could do less things about threads. The disadvantage of go is that it is

more difficult to learn but for c programmer you can master openMP in a very short time. OpenMP is more widely used in the world nowadays, but I think go language will be more and more popular in the future. I prefer to use Go to do parallel program if I have future parallel tasks. The go performance is better even though it is a bit harder to code for me.

Appendix A C+OpenMP TotientRange Program (5 marks)

I have some notation in my own language and that is easier for me to understand.

```
#include <stdio.h>
#include <sys/time.h>
#include <omp.h>  /* Here's the header file for OpenMP. */

// hcf x 0 = x
// hcf x y = hcf y (rem x y)

long hcf(long x, long y)
{
    long t;

    while (y != 0) {
        t = x % y;
        x = y;
        y = t;
    }
    return x;
}


// relprime x y = hcf x y == 1

int relprime(long x, long y)
{
    return hcf(x, y) == 1;
}


// euler n = length (filter (relprime n) [1 .. n-1])

long euler(long n)
{
```

```
    long length, i;

    length = 0;
    for (i = 1; i < n; i++)
        if (relprime(n, i))
            length++;
    return length;
}


// sumTotient lower upper = sum (map euler [lower, lower+1 .. upper])

long sumTotient(long lower, long upper)
{
    long sum, i;

    sum = 0;
    for (i = lower; i <= upper; i++)
        sum = sum + euler(i);
    return sum;
}


int main()
{
    unsigned long msec;
    double msperprime;
    struct timeval start, stop;

    long lower = 1;
    long upper = 30000;
    int number_t = 12;


    long sum = 0;
    long i = 0;

    gettimeofday(&start, NULL);       /* note start time */
//excute parallel
#pragma omp parallel for reduction(+:sum) num_threads(number_t)
    for (i = lower; i <= upper; i++)
        sum = sum + euler(i);

    gettimeofday(&stop, NULL);
    if (stop.tv_usec < start.tv_usec) {
        stop.tv_usec += 1000000;
        stop.tv_sec--;
    }
    msec = 1000 * (stop.tv_sec - start.tv_sec) +
           (stop.tv_usec - start.tv_usec) / 1000;

    printf("current number of threads is %d \n",number_t);
    printf("Sum of totient is  ----- %ld  \n",sum);
    printf("running time is ----- %lu  ms\n",msec);
    return 0
}
```

Appendix B Go TotientRange Program (5 marks)

```
package main

import (
   "fmt"
   "time"
)

// Compute the Highest Common Factor, hcf of two numbers x and y
//
// hcf x 0 = x
// hcf x y = hcf y (rem x y)

func hcf2(x, y int64) int64 {
   var t int64
   for (y != 0) {
      t = x % y
      x = y
      y = t
   }
   return x
}

// relprime determines whether two numbers x and y are relatively prime
//
// relprime x y = hcf x y == 1

func relprime2(x, y int64) bool {
   return hcf2(x, y) == 1;
}

// euler(n) computes the Euler totient function, i.e. counts the number of
// positive integers up to n that are relatively prime to n
//
// euler n = length (filter (relprime n) [1 .. n-1])

func euler2(n int64) int64 {
   var length, i int64

   length = 0
   for i = 1; i < n; i++ {
      if relprime2(n, i) {
         length++
      }
   }
   return length
}

// sumTotient lower upper sums the Euler totient values for all numbers
// between "lower" and "upper".
//
// sumTotient lower upper = sum (map euler [lower, lower+1 .. upper])

func sumTotient2(lower, upper int64, c chan int64) {
   var sum, i int64

   sum = 0
   for i = lower; i <= upper; i++ {
      sum = sum + euler2(i)

   }
   c <- sum
```

```
   fmt.Println("sumof", lower, "and", upper)

}

func main() {
   var lower, upper, totalNum,subTotal int64
   var dones int64 = 0
   c := make(chan int64)


   var numberOfThread int64 = 4
   upper = 15000
   lower = 1

   intervalNum := upper / numberOfThread
   // Record start time
   start := time.Now()
   //
   var i int64
   for i = 0; i < numberOfThread; i++ {
      base := int64(i) * intervalNum + 1
      if (i == numberOfThread - 1) {
         go sumTotient2(base, upper, c)
         break
      }
      go sumTotient2(base, base + intervalNum-1, c)

   }
   //取数
   for dones < numberOfThread {

      subTotal  = <- c

      fmt.Println(subTotal) //当打印出一个数的时候 dones 不变，然后再循环，必须等取到 10 个 0
时候结束，这个时候也就是表示 10 个线程执行完毕
      dones++
      totalNum = subTotal + totalNum
   }
   fmt.Println("Sum of Totients between", lower, "and", upper, "is",totalNum )
   // Record the elapsed time
   t := time.Now()
   elapsed := t.Sub(start)
   fmt.Println("Number of go routines is ", numberOfThread,"----Elapsed time is ",
elapsed)
}
```

Appendix C: Erlang totientrange2Workers Program (5 marks)

```
-module(con2).
% runing guide:  c(con2). -- con2:start_server(). -- server ! {range,
1, 4000}.
-export([start_server/0, worker_1/0, worker_2/0, server/2,
     hcf/2,
     relprime/2,
     euler/1,
     sumTotient/2]). % all method related to process has to export

%% TotientRange.erl - Sequential Euler Totient Function (Erlang
Version)

%% hcf x 0 = x
%% hcf x y = hcf y (rem x y) 两个数最大公约数

hcf(X,0) -> X;
hcf(X,Y) -> hcf(Y,X rem Y).

%% relprime x y = hcf x y == 1

relprime(X,Y) ->
  V = hcf(X,Y),
  if
    V == 1
      -> true;
    true
      -> false
  end.

%%euler n = length (filter (relprime n) (mkList n))

euler(N) ->
  RelprimeN = fun(Y) -> relprime(N,Y) end,
  length (lists:filter(RelprimeN,(lists:seq(1,N)))).



%%sumTotient lower upper = sum (map euler [lower, lower+1 .. upper])

sumTotient(Lower,Upper) ->
```

```
  Res = lists:sum(lists:map(fun euler/1,lists:seq(Lower, Upper))),%把lower 到upper 的数一个一个放到欧拉函数里面，欧拉函数只接收一个变量，然后返回结果列表，再计算结果的和
  io:format("Sum of totients: ~p~n", [Res]). %输出第二个参数需要是一个列表。

%% caculate time
printTime() ->
    {_, Time2} = statistics(wall_clock),
    U2 = Time2 * 1000,   % actual runtime
    io:format("Time taken in ~p  seconds~n",[U2/1000000]).


server(0,TOTAL) -> % finish sum and print time
    io:format("Server 0: Sum of totients: ~p ~n", [TOTAL]),
    printTime();

server(N,TOTAL) ->

     receive   % waiting for recieve
        {finished,SUM} ->
            io:format("Server :Received Sum ~p ~n", [SUM]),
            io:format("Worker : Finished! ~n", []),
            server(N-1,TOTAL+SUM); % keep receiving when the N is 0

        {range, LOW,UPPER} ->  % the order for recieving
            statistics(wall_clock), % start setting clock

            io:format("Server : Compute range ~p ~p ~n", [LOW,UPPER]),
            worker_1 ! {sum,round(LOW),round(UPPER/2)},  % send argument to worker and start it
            worker_2 ! {sum,round(UPPER/2+1), round(UPPER)},
            server(N-1 ,0)  % recall this method waiting for recieving
     end.

worker_1() ->
    receive
        {sum,L,U} ->
            io:format("Worker 1 : compute range ~p ~p ~n", [L,U]),
            SUM = lists:sum(lists:map(fun euler/1,lists:seq(L, U))),% worker calculate the sum
            server ! {finished,SUM}  % when finishing sum send it back to server
    end.
```

```
worker_2() ->
    receive
        {sum,L,U} ->
            io:format("Worker 2 : compute range ~p ~p ~n", [L,U]),
            SUM = lists:sum(lists:map(fun euler/1,lists:seq(L, U))), %
caculate erla
            server ! {finished,SUM}  % send sum to server
    end.

start_server() ->
    register(server, spawn(con2, server, [3,0])),   % create a server
processe and register name ,the last parameter is argument
    register(worker_1, spawn(con2, worker_1, [])),  % create two
workers and register name
    register(worker_2, spawn(con2, worker_2, [])).
```

Appendix D: Erlang totientrangeNWorkers Program (5 marks)

```
-module(conN).
% running guide :  c(conN). -- conN:start_server(16). -- server !
{range, 1, 15000}.
-export([start_server/1,
worker/0,server/2,workerName/1,printTime/0,setWorkerCaculate/4,
     hcf/2,
   relprime/2,
   euler/1,
   sumTotient/2]).

%% TotientRange.erl - Sequential Euler Totient Function (Erlang
Version)
%% hcf x 0 = x
%% hcf x y = hcf y (rem x y)

hcf(X,0) -> X;
hcf(X,Y) -> hcf(Y,X rem Y).

%% relprime x y = hcf x y == 1

relprime(X,Y) ->
  V = hcf(X,Y),
  if
    V == 1
      -> true;
    true
      -> false
  end.

%%euler n = length (filter (relprime n) (mkList n))

euler(N) ->
  RelprimeN = fun(Y) -> relprime(N,Y) end,
  length (lists:filter(RelprimeN,(lists:seq(1,N)))).

%%sumTotient lower upper = sum (map euler [lower, lower+1 .. upper])

sumTotient(Lower,Upper) ->
  Res = lists:sum(lists:map(fun euler/1,lists:seq(Lower, Upper))),
  io:format("Sum of totients: ~p~n", [Res]).
```

```
%% caculate time
printTime() ->
    {_, Time2} = statistics(wall_clock),
    U2 = Time2 * 1000,   % actual runtime
    io:format("Time taken in ~p  seconds~n",[U2/1000000]).

%% produce worker name
workerName(Num) ->
list_to_atom( "worker" ++ integer_to_list( Num )).

server(0,TOTAL) ->
    io:format("Server : Sum of totients: ~p ~n", [TOTAL]),
    printTime();
server(N,TOTAL) ->
     receive
     {finished,SUM} ->
            io:format("Server :received Sum ~p ~n", [SUM]),
            server(N-1,TOTAL+SUM); % waiting for N worker's result and
loop
      {range, LOW,UPPER} ->  % order to recieve value
            statistics(wall_clock), % start time clock
            io:format("Server :received range ~p ~p ~n", [LOW,UPPER]),
            setWorkerCaculate(LOW,UPPER,N,N), %
            server(N,TOTAL)
     end.

%% this method is used for loop to produce  N worker process
setWorkerCaculate(_,_,0,_) -> % _ is occupation , it means we do not
care this argument
  io:format("calculate  ~n", []);  % finish start N workers
setWorkerCaculate(LOW,UPPER,N,NUM) ->
            register(workerName(N), spawn(conN, worker, [])), %
register one worker
            workerName(N) ! {caculate_sum,round((UPPER/NUM)*(N-
1)+1),round((UPPER/NUM)*N)}, % give the worker range and start worker
to work
            setWorkerCaculate(LOW,UPPER,N-1,NUM).  % loop to produce N
workers
worker() ->
    receive
        {caculate_sum,L,U} ->
            io:format("Worker : started ~n", []),
            io:format("Worker : compute range ~p ~p ~n", [L,U]),
```

```
            SUM = lists:sum(lists:map(fun euler/1,lists:seq(L, U))),%
calculate sum and pass it to server
            io:format("Worker : finished ~n", []),
            server ! {finished,SUM}
    end.

%% start server process and pass the number of processes as N
start_server(N) ->
    register(server, spawn(conN, server, [N,0])).
```

Appendix E: Erlang totientrangeNWorkersReliable Program (5 marks)

```
-module(reliable).
%% running guide :  c(reliable). -- reliable:testRobust(4,3).
-export([start_server/1,
    testRobust/2,
    workerChaos/2,
    workerWatcher/3,
    worker/1,
    setWorkerCaculate/4,
    server/2,
    workerName/1,
    printTime/0,

   hcf/2,
   relprime/2,
   euler/1,
   sumTotient/2]).

%% TotientRange.erl - Sequential Euler Totient Function (Erlang
Version)
%% hcf x 0 = x
%% hcf x y = hcf y (rem x y)

hcf(X,0) -> X;
hcf(X,Y) -> hcf(Y,X rem Y).

%% relprime x y = hcf x y == 1

relprime(X,Y) ->
  V = hcf(X,Y),
  if
    V == 1
      -> true;
    true
      -> false
  end.

%%euler n = length (filter (relprime n) (mkList n))

euler(N) ->
  RelprimeN = fun(Y) -> relprime(N,Y) end,
  length (lists:filter(RelprimeN,(lists:seq(1,N)))).
```

```
%%sumTotient lower upper = sum (map euler [lower, lower+1 .. upper])

sumTotient(Lower,Upper) ->
  Res = lists:sum(lists:map(fun euler/1,lists:seq(Lower, Upper))),
  io:format("Sum of totients: ~p~n", [Res]).




%% caculate time
printTime() ->
    {_, Time2} = statistics(wall_clock),
    U2 = Time2 * 1000,   % actual runtime
    io:format("Time taken in ~p  seconds~n",[U2/1000000]).

%% produce worker name
workerName(Num) ->
list_to_atom( "worker" ++ integer_to_list( Num )).

server(0,TOTAL) ->
    io:format("Server : Sum of totients: ~p ~n", [TOTAL]),
    printTime();
server(N,TOTAL) ->
     receive
     {finished,SUM} ->
            io:format("Server :received Sum ~p ~n", [SUM]),
            server(N-1,TOTAL+SUM); % waiting for N worker's result and
loop
      {range, LOW,UPPER} ->  % order to recieve value
            statistics(wall_clock), % start time clock
            io:format("Server :received range ~p ~p ~n", [LOW,UPPER]),
            setWorkerCaculate(LOW,UPPER,N,N), %
            server(N,TOTAL)
     end.

%% this method is used for loop to produce  N worker process
setWorkerCaculate(_,_,0,_) ->
  io:format("calculate  ~n", []);  % finish start N workers
setWorkerCaculate(LOW,UPPER,N,NUM) ->
            register(workerName(N), spawn(reliable, worker, [N])), %
register one worker
            workerName(N) ! {caculate_sum,round((UPPER/NUM)*(N-
1)+1),round((UPPER/NUM)*N)}, % give the worker range and start worker
to work
```

```
            setWorkerCaculate(LOW,UPPER,N-1,NUM).  % loop to produce N
workers
worker(WorkerNum) ->
    receive
        {caculate_sum,L,U} ->
            spawn_link(reliable,workerWatcher,[WorkerNum,L,U]), % set a
watcher link
            io:format("~p : started ~n", [workerName(WorkerNum)]),
            io:format("~p : compute range ~p ~p ~n",
[workerName(WorkerNum),L,U]),
            SUM = lists:sum(lists:map(fun euler/1,lists:seq(L, U))),%
calculate sum and pass it to server
            io:format("~p : finished ~n", [workerName(WorkerNum)]),
            server ! {finished,SUM}
    end.
%% set workerWatcher
workerWatcher(WorkerNum,L,U) ->
    io:format("Watcher : Watching  ~p~n", [workerName(WorkerNum)]),
    process_flag(trap_exit, true),   % get Exit infromation
    receive
        {'EXIT',_,chaos} ->  % if kill restart the worker
        io:format("Watcher: restart : ~p~n", [workerName(WorkerNum)]),
        register(workerName(WorkerNum), spawn(reliable, worker,
[WorkerNum])),
        workerName(WorkerNum) ! {caculate_sum,L,U};

        {'EXIT',_,normal} ->
          io:format("watcher: nomal : ~p~n", [workerName(WorkerNum)])
    end.
%% chaos
workerChaos(NVictims,NWorkers) ->
  lists:map(fun( _ ) ->timer:sleep(500), %% Sleep for .5s%% Choose a
random victim
        WorkerNum = rand:uniform(NWorkers),
        io:format("workerChaos killing ~p~n",
        [workerName(WorkerNum)]),
        WorkerPid = whereis(workerName(WorkerNum)),
        if   %% Check if victim is alive
          WorkerPid == undefined ->
            io:format("workerChaos already dead:
~p~n",[workerName(WorkerNum)]);
          true ->   %% Kill Kill Kill
            exit(whereis(workerName(WorkerNum)),chaos)
            end
```

```
          end,
  lists:seq( 1, NVictims ) ).


%% start server process and pass the number of processes as N
start_server(N) ->
    register(server, spawn(reliable, server, [N,0])).

  %% test method
testRobust(NWorkers,NVictims) ->
      ServerPid = whereis(server),
      if ServerPid == undefined ->
          start_server(NWorkers);
        true ->
          ok
        end,
      server ! {range, 1, 15000},
      workerChaos(NVictims,NWorkers).
```