Mod4 Analysis Report

Woohyuk Yang

yangwo@oregonstate.edu

Analysis and discussion on the results

1. Recursive vs. Iterative implementation of Fibonacci Number Algorithm

To compare execution time for each style, I referenced the code from

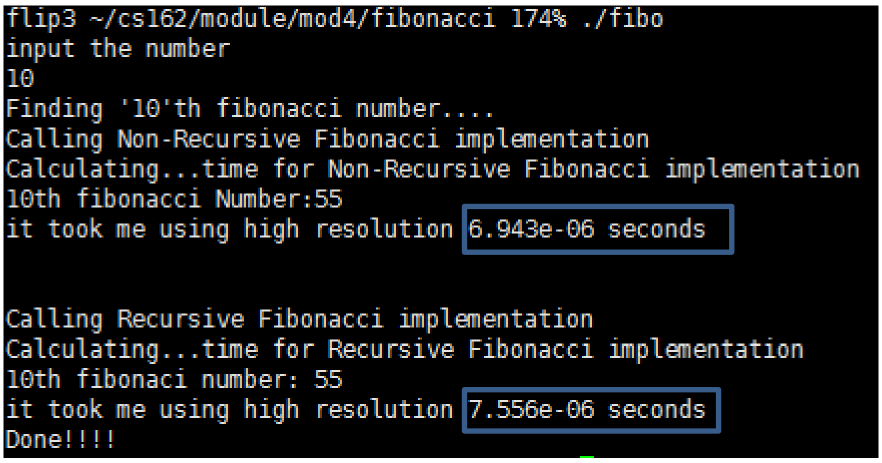
http://www.codeproject.com/Tips/109443/Fibonacci-Recursive-and-Non-Recursive-C

and modified them to measure the time using std::chrono::high_resolution_clock::now() function.

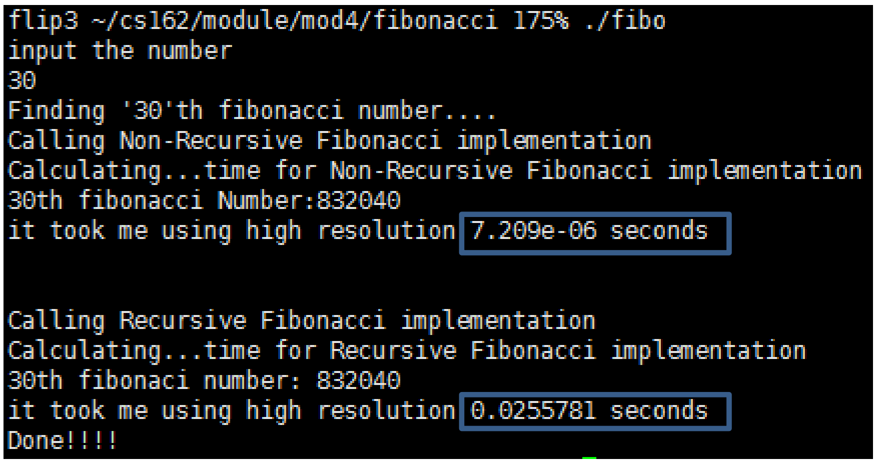
I could get good reference of high_resolution_clock::now() function from

http://www.cplusplus.com/reference/chrono/high_resolution_clock/

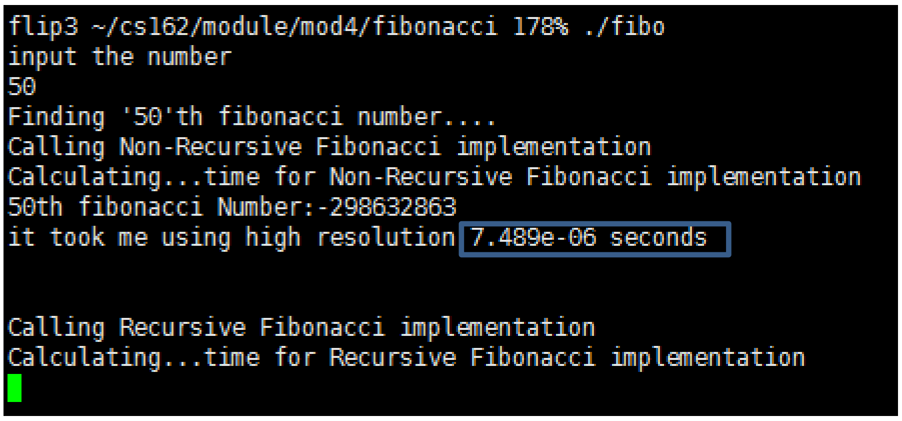
As I test the results,



```
flip3 ~/cs162/module/mod4/fibonacci 174% ./fibo
input the number
10
Finding '10'th fibonacci number....
Calling Non-Recursive Fibonacci implementation
Calculating...time for Non-Recursive Fibonacci implementation
10th fibonacci Number:55
it took me using high resolution 6.943e-06 seconds


Calling Recursive Fibonacci implementation
Calculating...time for Recursive Fibonacci implementation
10th fibonaci number: 55
it took me using high resolution 7.556e-06 seconds
Done!!!!
```



```
flip3 ~/cs162/module/mod4/fibonacci 175% ./fibo
input the number
30
Finding '30'th fibonacci number....
Calling Non-Recursive Fibonacci implementation
Calculating...time for Non-Recursive Fibonacci implementation
30th fibonacci Number:832040
it took me using high resolution 7.209e-06 seconds


Calling Recursive Fibonacci implementation
Calculating...time for Recursive Fibonacci implementation
30th fibonaci number: 832040
it took me using high resolution 0.0255781 seconds
Done!!!!
```



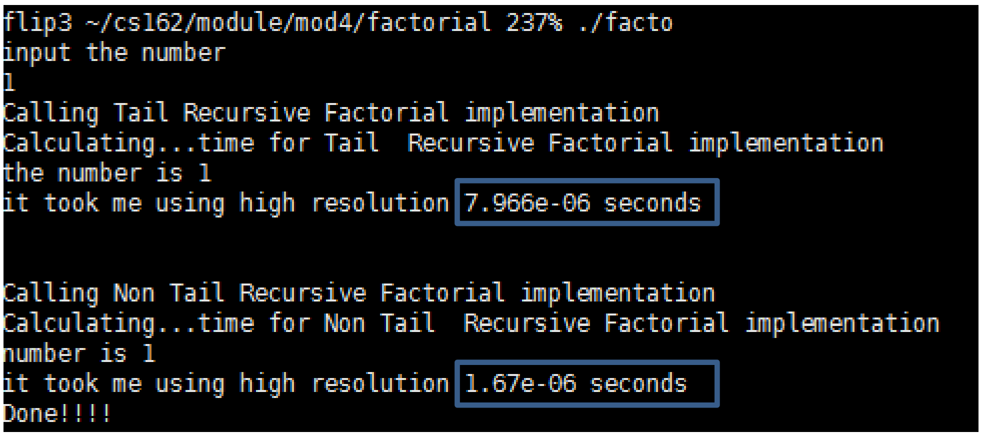
```
flip3 ~/cs162/module/mod4/fibonacci 178% ./fibo
input the number
50
Finding '50'th fibonacci number....
Calling Non-Recursive Fibonacci implementation
Calculating...time for Non-Recursive Fibonacci implementation
50th fibonacci Number:-298632863
it took me using high resolution 7.489e-06 seconds


Calling Recursive Fibonacci implementation
Calculating...time for Recursive Fibonacci implementation
```

I expected Recursive function would take longer than the other one. So I put the iterative one before for the case I input quite large number. But as you can see, I only input 50 but the Recursive function looks having hard time executing the function.

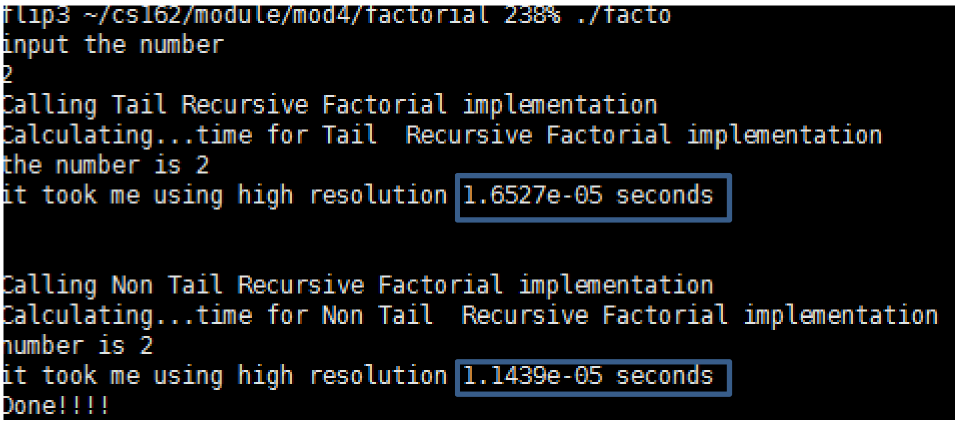
It took more time when I use Recursive way of implementation as you can see from the tests. And what I could know is the time for executing the iterative function does not get to be added as much as the Recursive one does. Iterative function keeps 6.xxxe-06 ~ 7.xxe-06 when the input is 10 to 30 to 50 but you can also see the time measured by recursive function gets added a lot as the input number gets bigger.

2. Not Tail Recursive vs. Tail Recursive implementation of Factorial function

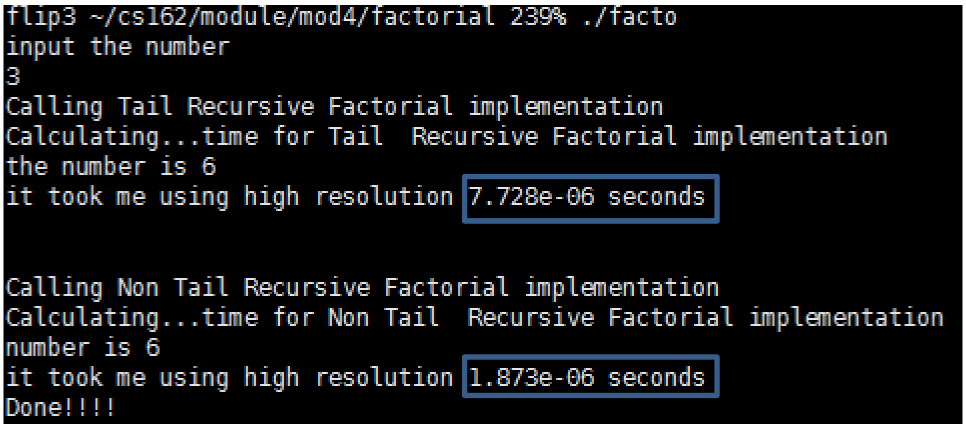
I adopted the code written on the description of mod4 to measure Not Tail recursive and Tail recursive implementation of Factorial function. Also, std::chrono::high_resolution_clcok::now() function has been used to calculate time of execution for each files.



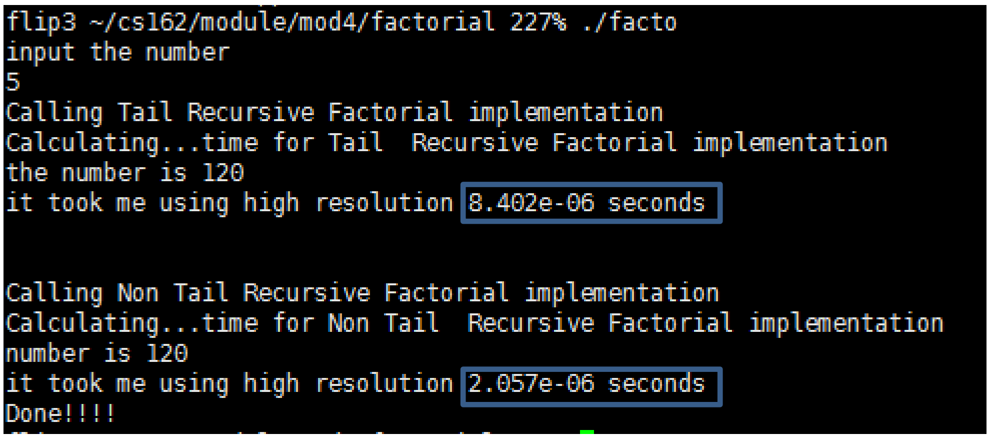
```
flip3 ~/cs162/module/mod4/factorial 237% ./facto
input the number
1
Calling Tail Recursive Factorial implementation
Calculating...time for Tail  Recursive Factorial implementation
the number is 1
it took me using high resolution 7.966e-06 seconds


Calling Non Tail Recursive Factorial implementation
Calculating...time for Non Tail  Recursive Factorial implementation
number is 1
it took me using high resolution 1.67e-06 seconds
Done!!!!
```



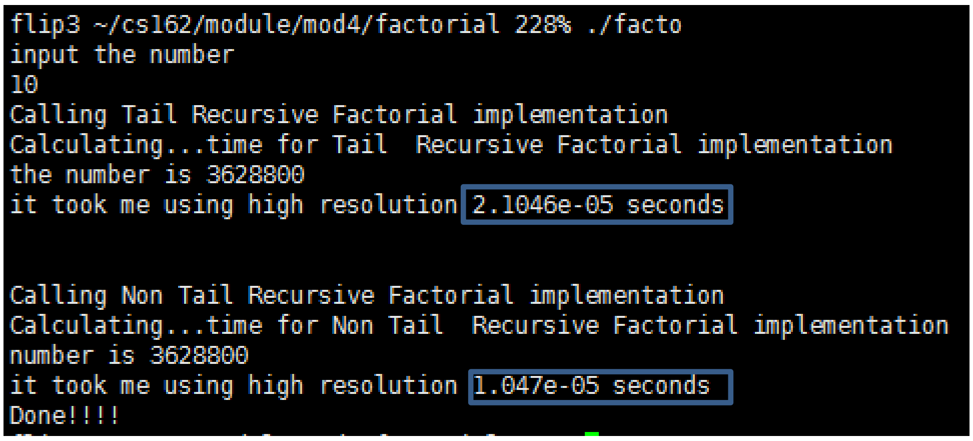
```
flip3 ~/cs162/module/mod4/factorial 238% ./facto
input the number
2
Calling Tail Recursive Factorial implementation
Calculating...time for Tail  Recursive Factorial implementation
the number is 2
it took me using high resolution 1.6527e-05 seconds


Calling Non Tail Recursive Factorial implementation
Calculating...time for Non Tail  Recursive Factorial implementation
number is 2
it took me using high resolution 1.1439e-05 seconds
Done!!!!
```



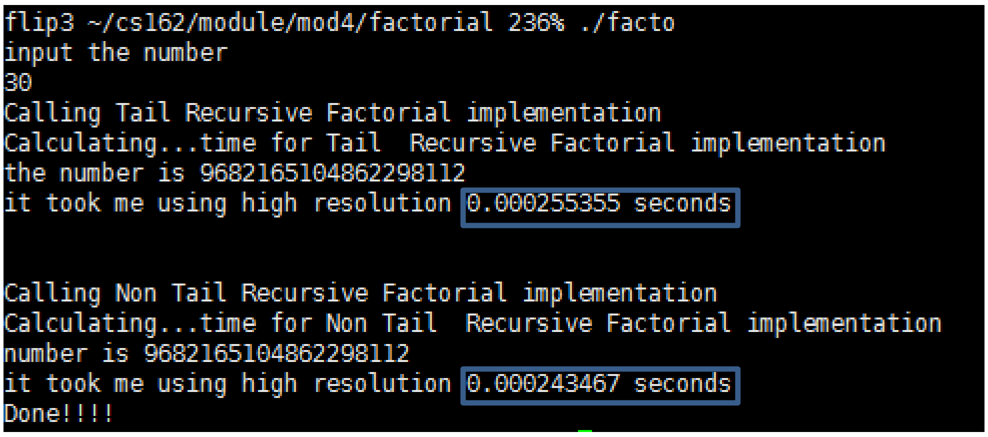
```
flip3 ~/cs162/module/mod4/factorial 239% ./facto
input the number
3
Calling Tail Recursive Factorial implementation
Calculating...time for Tail  Recursive Factorial implementation
the number is 6
it took me using high resolution 7.728e-06 seconds


Calling Non Tail Recursive Factorial implementation
Calculating...time for Non Tail  Recursive Factorial implementation
number is 6
it took me using high resolution 1.873e-06 seconds
Done!!!!
```



```
flip3 ~/cs162/module/mod4/factorial 227% ./facto
input the number
5
Calling Tail Recursive Factorial implementation
Calculating...time for Tail  Recursive Factorial implementation
the number is 120
it took me using high resolution 8.402e-06 seconds


Calling Non Tail Recursive Factorial implementation
Calculating...time for Non Tail  Recursive Factorial implementation
number is 120
it took me using high resolution 2.057e-06 seconds
Done!!!!
```

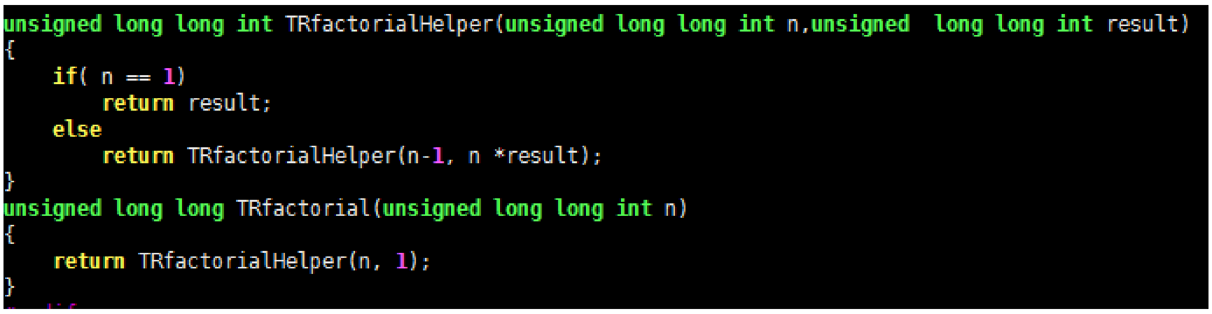


```
flip3 ~/cs162/module/mod4/factorial 228% ./facto
input the number
10
Calling Tail Recursive Factorial implementation
Calculating...time for Tail  Recursive Factorial implementation
the number is 3628800
it took me using high resolution 2.1046e-05 seconds


Calling Non Tail Recursive Factorial implementation
Calculating...time for Non Tail  Recursive Factorial implementation
number is 3628800
it took me using high resolution 1.047e-05 seconds
Done!!!!
```



```
flip3 ~/cs162/module/mod4/factorial 236% ./facto
input the number
30
Calling Tail Recursive Factorial implementation
Calculating...time for Tail  Recursive Factorial implementation
the number is 9682165104862298112
it took me using high resolution 0.000255355 seconds


Calling Non Tail Recursive Factorial implementation
Calculating...time for Non Tail  Recursive Factorial implementation
number is 9682165104862298112
it took me using high resolution 0.000243467 seconds
Done!!!!
```

I did use optimization –O3 to optimize tail recursive function. As you can see on the result, tail recursive function took more time than the other one. I researched that using recursive factorial without any then the stack could be overflow if the function has to deal with the large number(too many quantity of function overhead for stack happens), so tail recursive can solve that problem. But with this experiment I could not witness stack over flow. There were some ways to show the stack over flow happens and I did test that and the stack over flow happened as I expected. Based on my search, Optimizer changes the tail recursive function to be more like a iterative function so there will not be any stack over flow. I assumed that the phenomenon that tail recursive function does take more time than the other one is the tail recursive function is composed of two functions.



```
unsigned long long int TRfactorialHelper(unsigned long long int n,unsigned  long long int result)
{
    if( n == 1)
        return result;
    else
        return TRfactorialHelper(n-1, n *result);
}
unsigned long long TRfactorial(unsigned long long int n)
{
    return TRfactorialHelper(n, 1);
}
```

As you can see from the code, that function is composed of two different functions. I assumed that it takes more time than the other because of function overhead.(Two functions have to be worked together for tail recursion.) I have read that the optimizer does change the tail recursive function to iteration function from some texts. As I compared with iterative function and recursive function before, iterative function took much less than the recursive one. so my assumption is not certain so more inspection and thorough research on the optimizer and function overhead should be done.