Chandrabhan Patel

Report of assignment 1

ES 215: Computer Organisation and Architecture

Question 1:

Function for time measurement:

```
C/C++
long long measureTime(long long (*func)(int), int n) {

    struct timespec start, end;

    clock_gettime(CLOCK_MONOTONIC, &start);

    func(n);

    clock_gettime(CLOCK_MONOTONIC, &end);

    long long duration = (end.tv_sec - start.tv_sec) * 1e9 + (end.tv_nsec -
start.tv_nsec);

    return duration;

}
```

A. Recursion:

Code:

```
C/C++
long long fib_recursive(int n) {
    if (n <= 1)
        return n;
    return fib_recursive(n - 1) + fib_recursive(n - 2);
}
void printFibonacciRecursive(int n) {
    for (int i = 0; i < n; i++) {
        cout << fib_recursive(i) << " ";
  }
    cout << endl;
}
```

Output: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986
Time (recursion): 948109092 ns

B. loop

Code:

```
C/C++
long long fib_loop(int n) {
    long long a = 0, b = 1, c;
    if (n == 0)
        return a;
    for (int i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}

void printFibonacciLoop(int n) {
    for (int i = 0; i < n; i++) {
        cout << fib_loop(i) << " ";
    }
    cout << endl;
}
```

Output: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986
Time (loop): 6833 ns

C. recursion with memoization

Code:

```
C/C++
vector<long long> memo;

long long fib_recursive_memo(int n) {
    if (memo[n] != -1)
        return memo[n];
    if (n <= 1)
        return n;
    memo[n] = fib_recursive_memo(n - 1) + fib_recursive_memo(n - 2);
    return memo[n];
}

void printFibonacciRecursiveMemo(int n) {
    memo.assign(n, -1);
    for (int i = 0; i < n; i++) {
        cout << fib_recursive_memo(i) << " ";
    }
    cout << endl;
}
```

Output: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986
Time (recursion + memoization): 10751 ns

D. loop with memoization

Code:

```C/C++
long long fib_loop_memo(int n) {
    vector<long long> fib(n + 1, 0);
    fib[1] = 1;
    for (int i = 2; i <= n; i++) {
        fib[i] = fib[i - 1] + fib[i - 2];
    }
    return fib[n];
}

void printFibonacciLoopMemo(int n) {
    for (int i = 0; i < n; i++) {
        cout << fib_loop_memo(i) << " ";
    }
    cout << endl;
}
```

Output: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986
Time (loop + memoization): 62669 ns

Speedups:

Code:

```
C/C++

int main() {
    const int n = 40;

    // time for each
    long long timeRec = measureTime(fib_recursive, n);
    long long timeLoop = measureTime(fib_loop, n);
    long long timeRecMemo = measureTime(fib_recursive_memo, n);
    long long timeLoopMemo = measureTime(fib_loop_memo, n);

    // print times
    cout << "Time (recursion): " << timeRec << " ns" << endl;
    cout << "Time (loop): " << timeLoop << " ns" << endl;
    cout << "Time (recursion + memoization): " << timeRecMemo << " ns" << endl;
    cout << "Time (loop + memoization): " << timeLoopMemo << " ns" << endl;

    // Calculate speedups
    cout << "Speedup (loop vs recursion): " << (double)timeRec / timeLoop << endl;
    cout << "Speedup (recursion + memoization vs recursion): " << (double)timeRec /
timeRecMemo << endl;
    cout << "Speedup (loop + memoization vs recursion): " << (double)timeRec /
timeLoopMemo << endl;

    return 0;
}
```

Output:

```
Speedup (loop vs recursion): 138754
Speedup (recursion + memoization vs recursion): 88188
Speedup (loop + memoization vs recursion): 15128.8
```

Observations: Recursion is the slowest method out of all, loops are relatively faster than recursion.
With memoization, speedup increases significantly for both recursion and loop. Loop + memoization is the fastest way to calculate the fibonacci number.

Question 2:

Language 1: c++
Language 2: Python

Language 1: c++

Code for matrix product calculation:

```
C/C++
//matrix product function for int, similar for dtype double
void multiplyMatrices(vector<vector<int>>& A,vector<vector<int>>&B, vector<vector<int>>&C,
int N) {
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            C[i][j] = 0;
            for (int k = 0; k < N; ++k) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
}
```

a) Time function is used in code as well, for finding system time using UNIX command time

```
Unset
┌──(akon㉿Fugaku)-[~]
└─$ time ./a.out
Matrix size: 64x64, Time taken: 2 ms
Matrix size: 128x128, Time taken: 21 ms
Matrix size: 256x256, Time taken: 171 ms
Matrix size: 512x512, Time taken: 1362 ms
Matrix size: 1024x1024, Time taken: 10458 ms

real    0m12.113s
user    0m12.011s
sys     0m0.021s
```

Observation: system time is very less compared to the user time.

Additional information:

1. All the entries of matrices are unit integer.
2. For time calculation in c++, the following segment is used.

C/C++
```cpp
#include <chrono>

auto start = high_resolution_clock::now();
multiplyMatrices(A, B, C, N);
auto end = high_resolution_clock::now();
auto duration = duration_cast<milliseconds>(end - start);
```

Time for float data type:

Unset
```
┌──(akon㉿Fugaku)-[~]
└─$ time ./a.out
Matrix size: 64x64, Time taken: 6 ms
Matrix size: 128x128, Time taken: 49 ms
Matrix size: 256x256, Time taken: 362 ms
Matrix size: 512x512, Time taken: 3035 ms
Matrix size: 1024x1024, Time taken: 25134 ms

real    0m28.709s
user    0m28.604s
sys     0m0.021s
```

Observations: compared to the int matrix multiplication, as expected it took more time. Because it has more cycles per instruction for floating point multiplication.

b) Meat Portion time:

For floating point matrix multiplication times are as follows

```
┌──(akon㉿Fugaku)-[~]
└─$ time ./a.out
Matrix size: 64x64, Time taken: 6 ms
Matrix size: 128x128, Time taken: 49 ms
Matrix size: 256x256, Time taken: 362 ms
Matrix size: 512x512, Time taken: 3035 ms
Matrix size: 1024x1024, Time taken: 25134 ms

real    0m28.709s
user    0m28.604s
sys     0m0.021s
```

for all 5 values of N is 28586(6+49+362+3035+25134) milliseconds, Which is 123 seconds less than the real time.

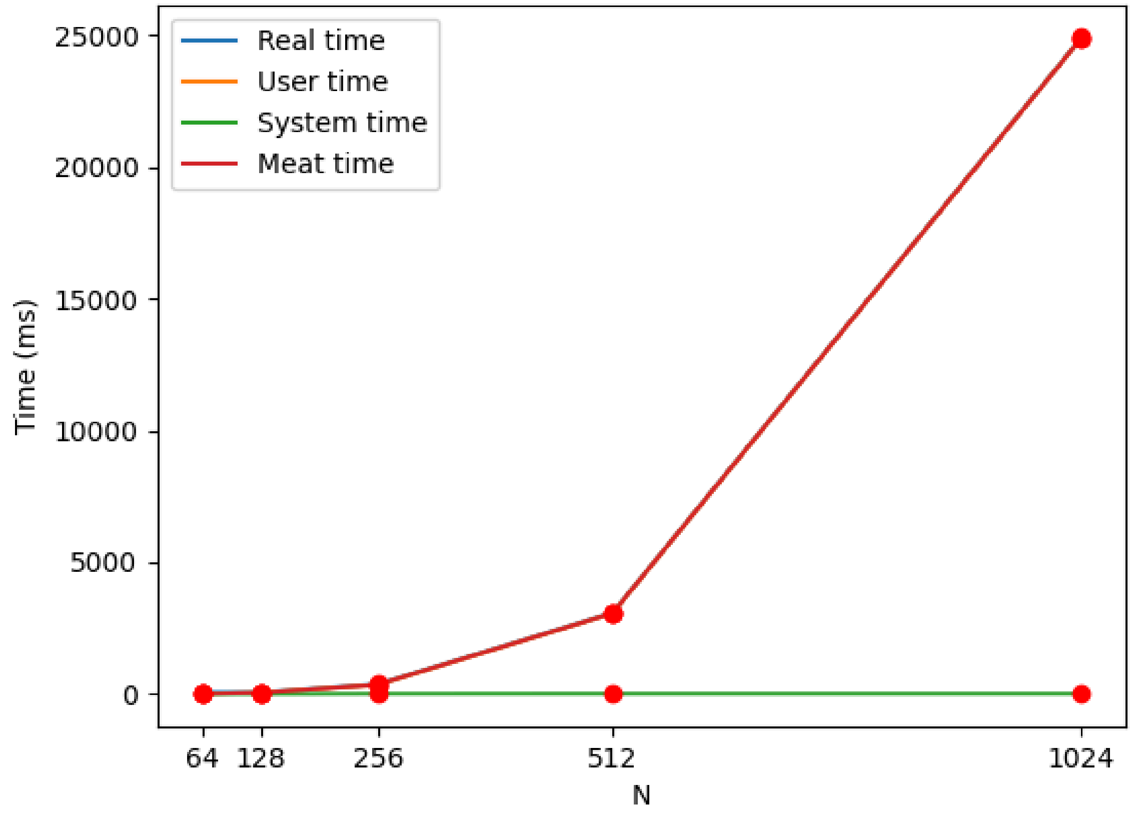
c) Comparison

1. Time data for data type int with different matrix sizes

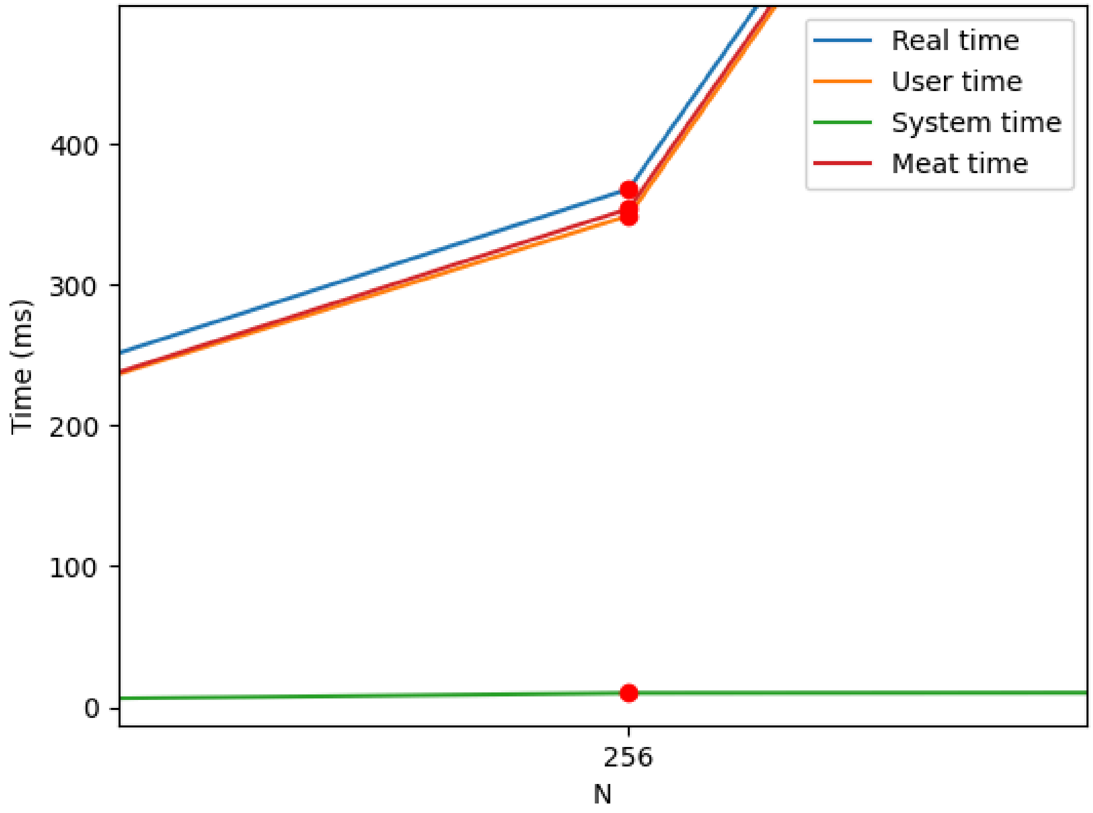
N	Meat time	Real time	User time	System time
64	5	65	12	0
128	51	64	56	0
256	354	368	349	10
512	3057	3070	3053	11
1024	24899	24923	24904	11

*some error in real time measurement by the command for N=64

2. Plot between time and N



3. Observations:
   A. With an increasing number of N, time taken for calculation increases exponentially.
   B. Almost all of the time taken by the program is because of meat time.



C. System time is almost constant with increasing N, thus Real time, User time and System time are almost same

Language 2: Python

Code for matrix product calculation:

```python
def multiply_matrices(A, B, N):
    C = np.zeros((N, N), dtype=int)
    for i in range(N):
        for j in range(N):
            for k in range(N):
                C[i][j] += A[i][k] * B[k][j]
    return C
```

a) Time function is used in code as well, for finding system time using UNIX command time

```
Unset
┌──(akon㉿Fugaku)-[~]
└─$ time python3 '/mnt/d/Random Code/ES 215 COA/q2(int)py.py'
Matrix size: 64x64, Time taken: 132.91 ms
Matrix size: 128x128, Time taken: 1111.57 ms
Matrix size: 256x256, Time taken: 8695.24 ms
Matrix size: 512x512, Time taken: 70387.60 ms
Matrix size: 1024x1024, Time taken: 804697.58 ms

real    14m45.696s
user    14m45.284s
sys     0m1.272s
```

Observation: system time is very less compared to the user time. And Comparing with c++ real time, the speedup is around 75 even after using numpy library.

Additional information:

1. All the entries of matrices are unit integers.
2. For time calculation in python, the time library has been used.

```
C/C++
import time
start_time = time.time()
C = multiply_matrices(A, B, N)
end_time = time.time()
duration = (end_time - start_time) * 1000
```

Time for float data type:

```
Unset
┌──(akon㉿Fugaku)-[~]
└─$ time python3 '/mnt/d/Random Code/ES 215 COA/q2(int)py.py'
Matrix size: 64x64, Time taken: 358.83 ms
Matrix size: 128x128, Time taken: 3000.19 ms
Matrix size: 256x256, Time taken: 22662.08 ms
Matrix size: 512x512, Time taken: 194460.98 ms
Matrix size: 1024x1024, Time taken: 1615980.12 ms

real    30m37.084s
user    30m36.930s
sys     0m1.152s
```

b) Meat Portion time:

For Integer type matrix multiplication in python times are as follows

```
┌──(akon㉿Fugaku)-[~]
└─$ time python3 '/mnt/d/Random Code/ES 215 COA/q2(int)py.py'
Matrix size: 64x64, Time taken: 132.91 ms
Matrix size: 128x128, Time taken: 1111.57 ms
Matrix size: 256x256, Time taken: 8695.24 ms
Matrix size: 512x512, Time taken: 70387.60 ms
Matrix size: 1024x1024, Time taken: 804697.58 ms

real    14m45.696s
user    14m45.284s
sys     0m1.272s
```

for all 5 values of N is 885024.9 (132.91+1111.57+8695.24+70387.60+804697.58) milliseconds,

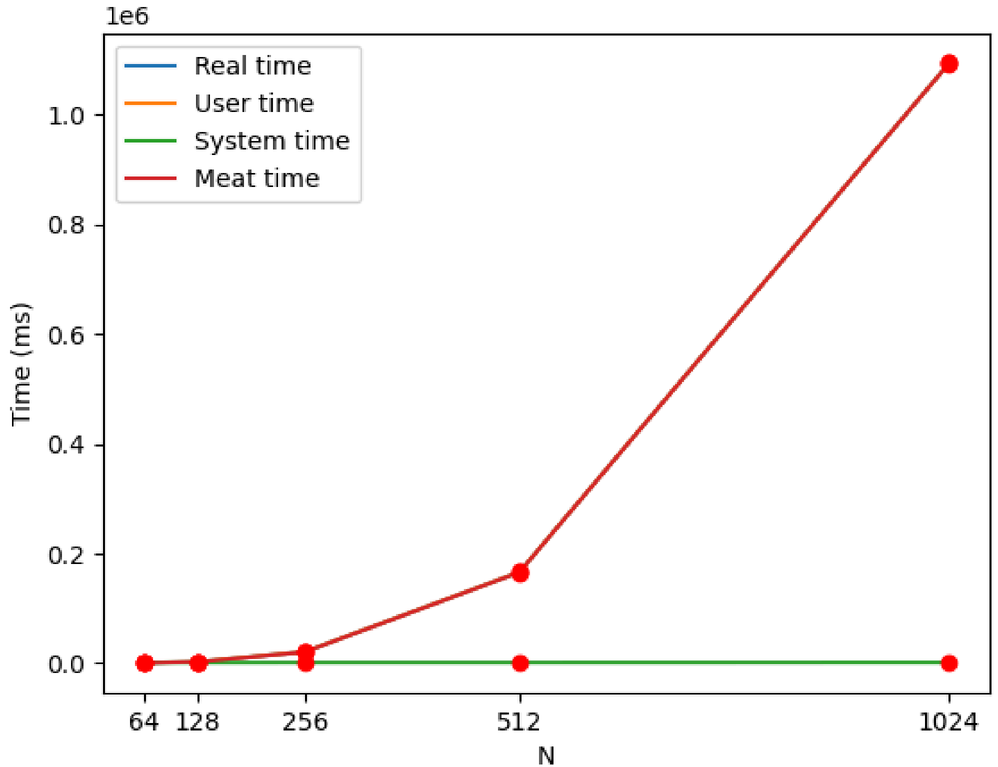
Which is 672 seconds less than the real time.

c) Comparison

1. Time data for data type int with different matrix sizes

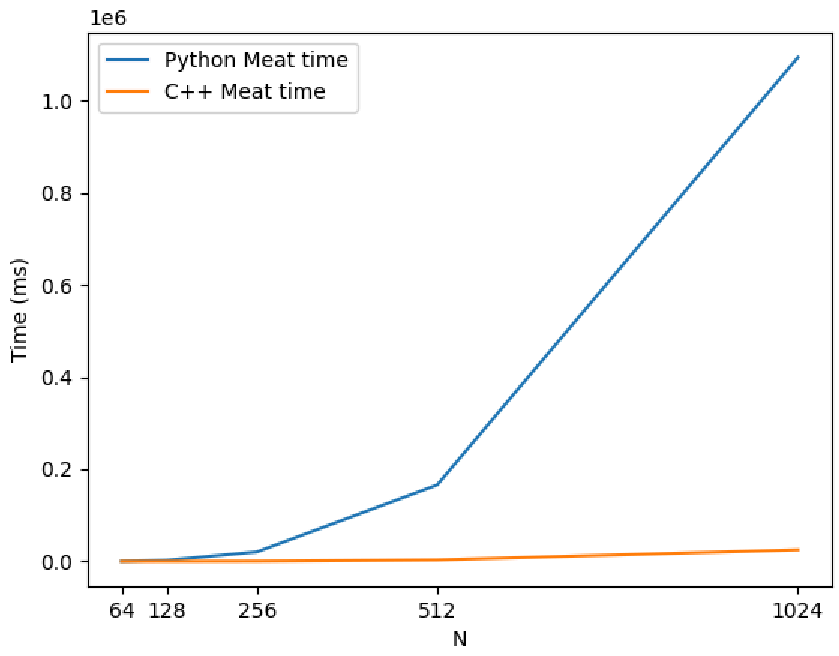
N	Meat time	Real time	User time	System time
64	269	663	747	1027
128	2462	2837	2927	1047
256	20371	20702	20773	1100
512	165850	166234	166288	1091
1024	1093684	1094076	1093999	1221

2. Plot between time and N:

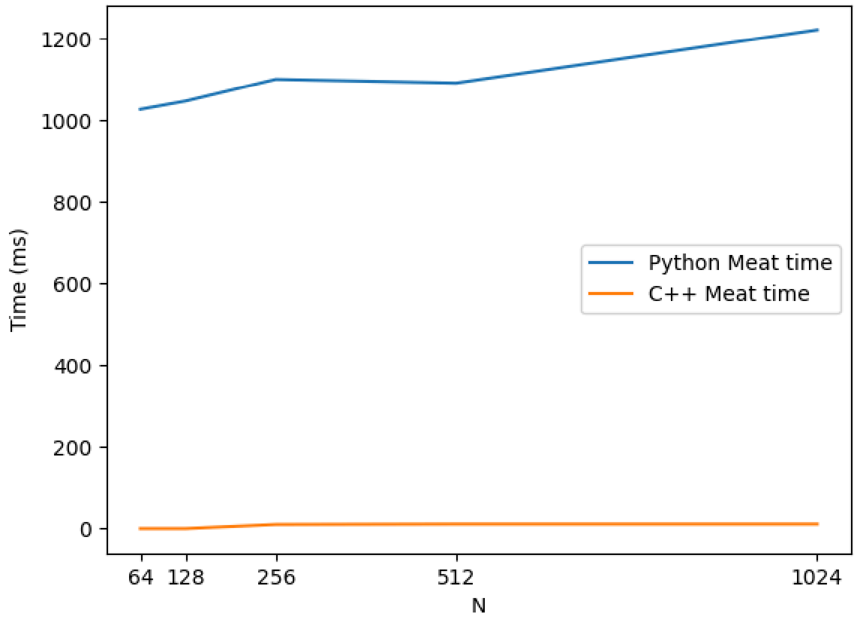


3. **Observations:**
    A. Similar to C++ in python also the time taken to execute the program increases exponentially with increasing number of iterations.
    B. System time is in c++ is lesser compared to python for any value of N.

Comparison of Meal time between python and c++



Comparison of System time between python and c++



Conclusion:

C++ is more efficient in matrix multiplication compared to python because of its low-level optimization. Especially when the matrix size increases c++ becomes more useful.

Code Repository: https://github.com/cpatel321/ES-215-COA

This assignment helped me in bridging the gap between theoretical concepts and practical uses. Also it is developing a sense of time complexity of loops and recursion.