HIGH PERFORMANCE MACHINE LEARNING
ECE-GY-9143
LAB – 1

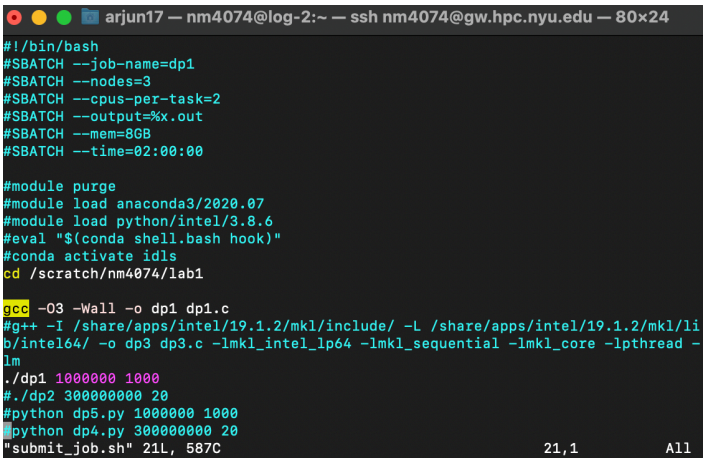
Nagharjun Mathi Mariappan
nm4074

Question C1

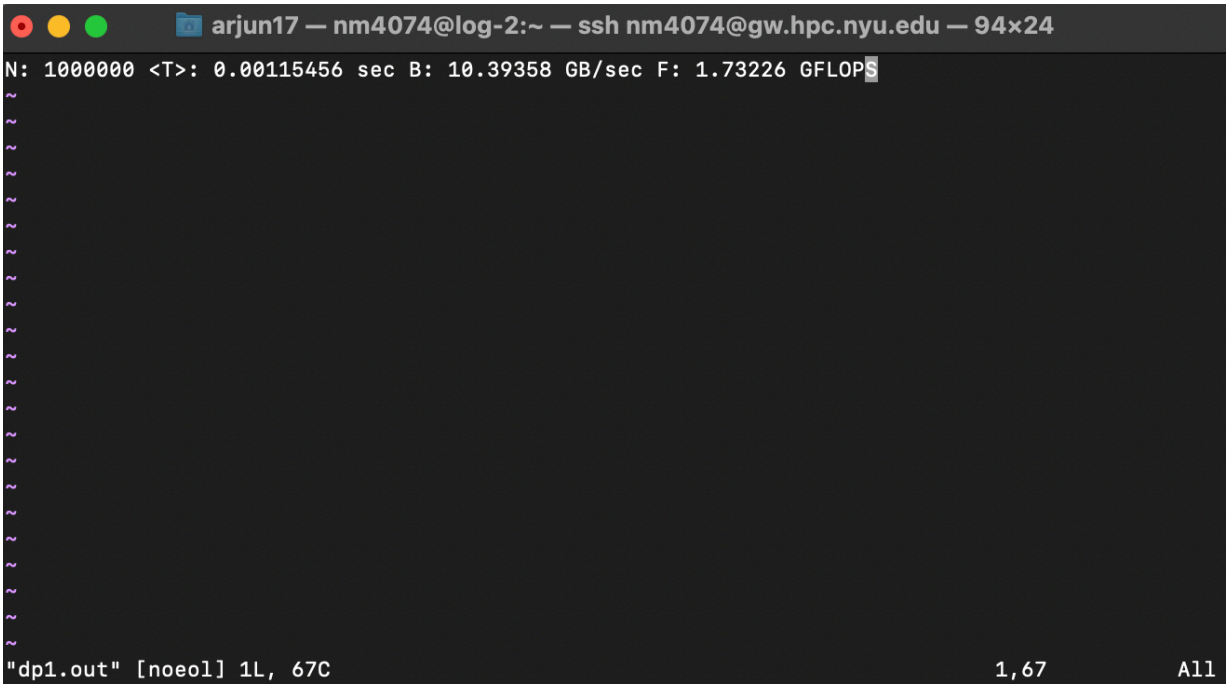
Code:

```c
#include<stdio.h>
#include<stdlib.h>
#include <time.h>

float dp(long N, float *pA, float *pB) {
    float R = 0.0;
    long j;
    for (j = 0; j < N; j++)
    R += pA[j] * pB[j];
    return R;
}
int main(int argc, char *argv[]) {
    struct timespec start, end;
    char *k1 = argv[1];
    char *k2 = argv[2];
    long size_of_array = atoi(k1);
    int iterations = atoi(k2);
    float a[size_of_array];
    float b[size_of_array];

    for(long i = 0; i < size_of_array; i++) {
        a[i] = 1;
        b[i] = 1;
    }

    double average_et = 0.0;
    double average_et_per_iteration_in_sec = 0.0;
    double gflops = 0.0;
    double bandwidth = 0.0;
    float p = 0.0;

    for(long i = 0; i < iterations; i++) {
        clock_gettime(CLOCK_MONOTONIC, &start);
        p = dp(size_of_array, a, b);
        clock_gettime(CLOCK_MONOTONIC, &end);
        double time_usec=(((double)end.tv_sec * 1000000 + (double)end.tv_nsec / 1000) - ((double)start.tv_sec * 1000000 + (double)start.tv_nsec / 1000));
        if(i > (iterations / 2) - 1) {
            average_et = average_et + time_usec;
        }
    }
    average_et_per_iteration_in_sec = average_et / ((float)(iterations / 2) * 1000000);
    bandwidth = 3 * 4 * size_of_array / (average_et_per_iteration_in_sec * 1000000000); // 3 load and store and converstion from bytes to gb
    gflops = 2 * size_of_array / (average_et_per_iteration_in_sec * 1000000000); // 2 floating point ops

    if((long)p < size_of_array) {
            printf("N: %lu <T>: %.08lf sec B: %0.05lf GB/sec F: %.05lf GFLOPS", size_of_array, average_et_per_iteration_in_sec, bandwidth, gflops);
    } else {
            printf("N: %lu <T>: %.10lf sec B: %0.03lf GB/sec F: %.03lf GFLOPS", (long)p, average_et_per_iteration_in_sec, bandwidth, gflops);
    }
    return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dp1.c" 51L, 1792C
```

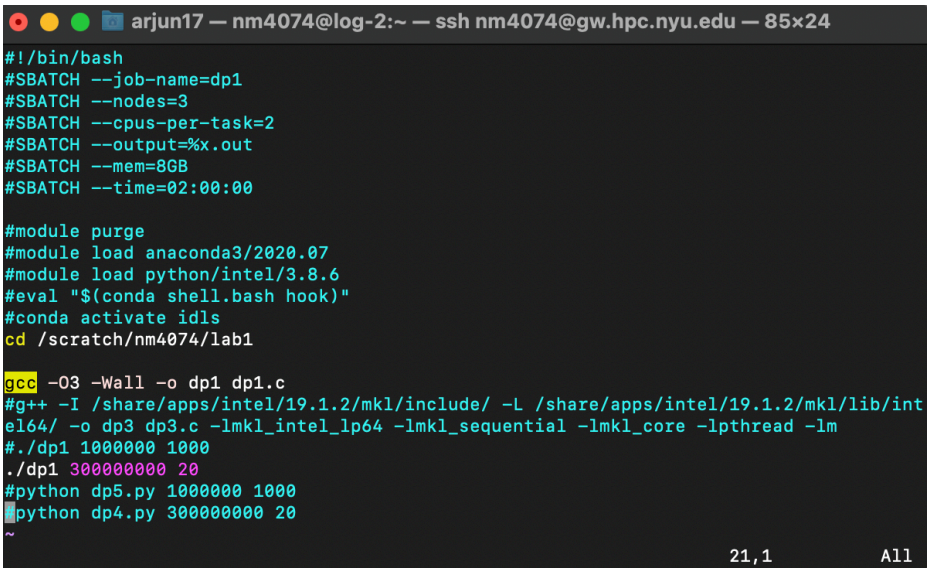
Case 1: N = 1000000 and Repetitions = 1000

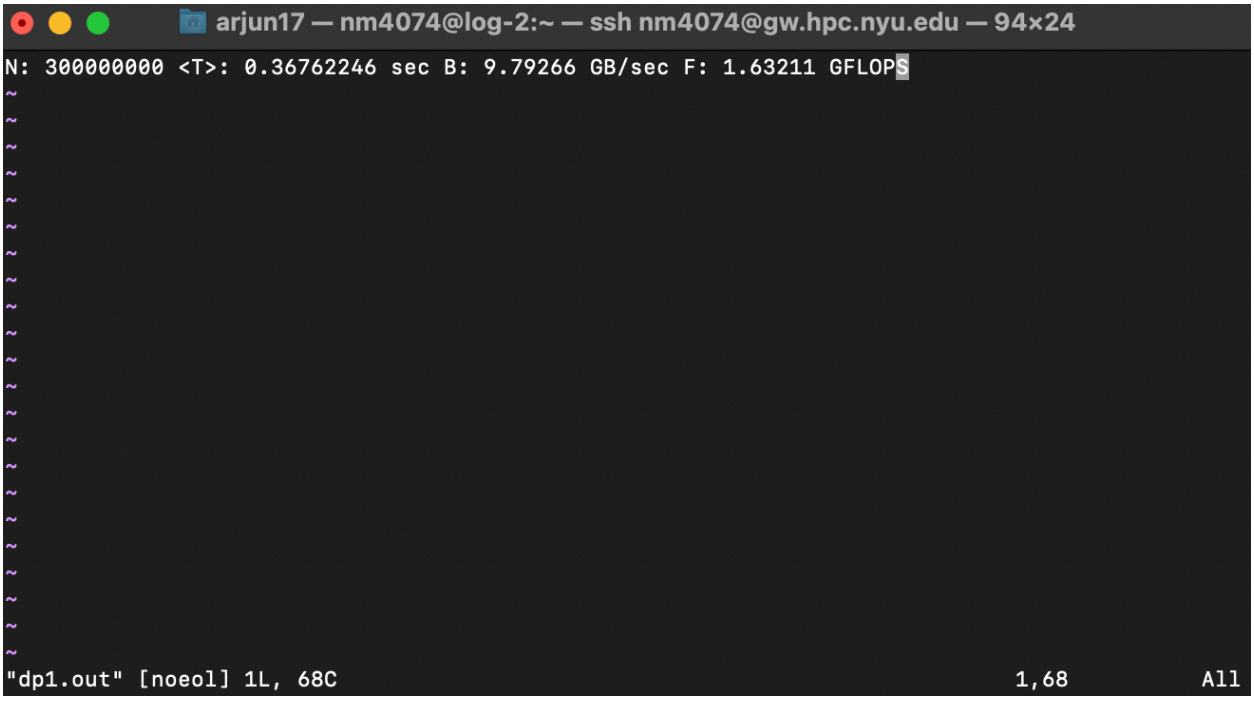



```
(base) [nm4074@log-2 ~]$ vi submit_job.sh
(base) [nm4074@log-2 ~]$ sbatch submit_job.sh
Submitted batch job 25413747
(base) [nm4074@log-2 ~]$ squeue -u nm4074
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
          25413747        cs      dp1   nm4074 PD       0:00      3 (Priority)
```



Case 2: N = 300000000 and Repetitions = 20



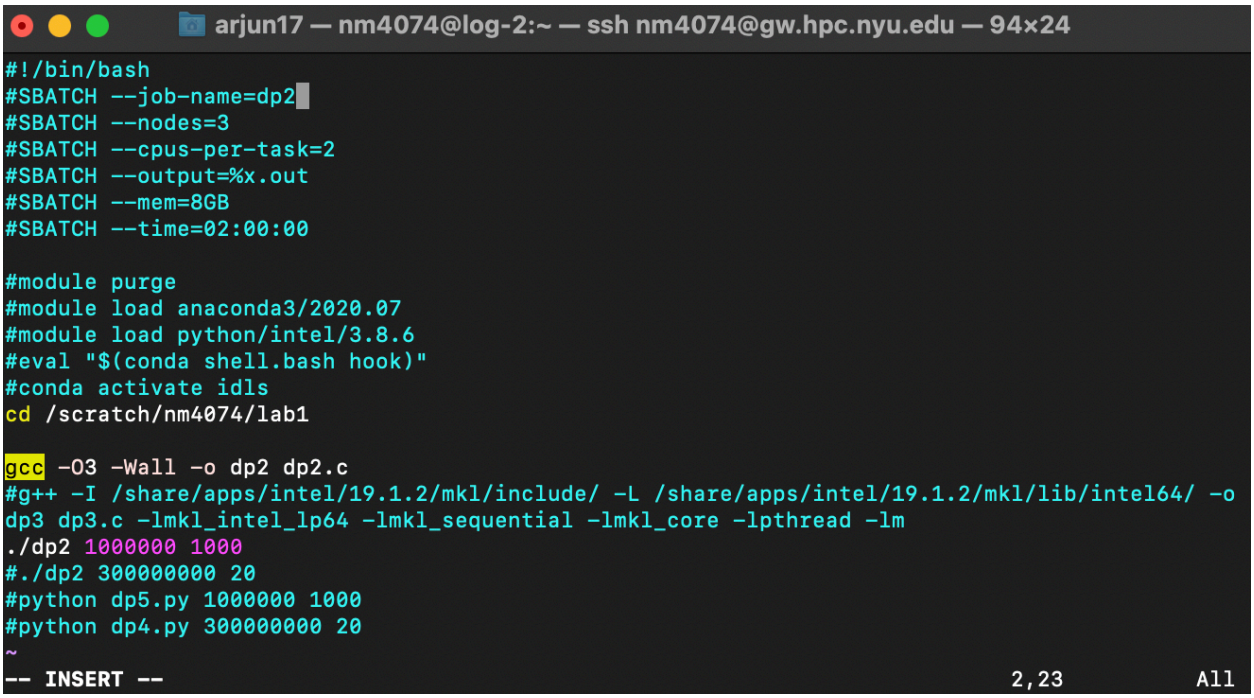


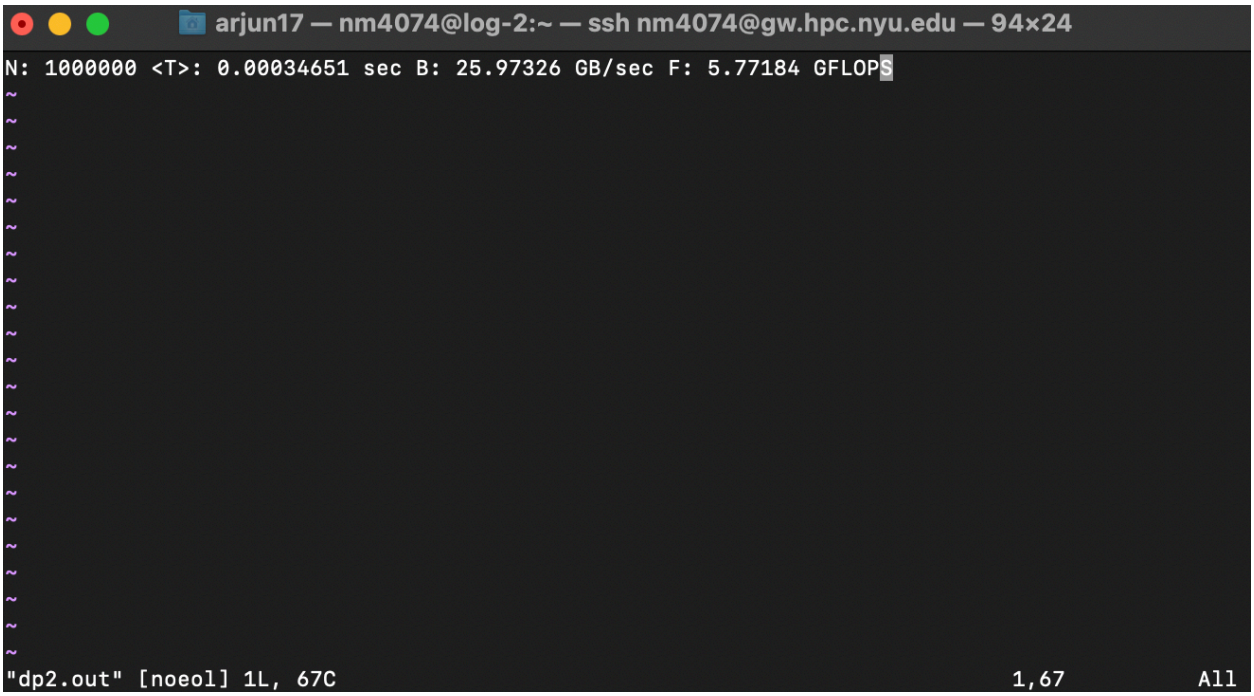
Question C2

Code:

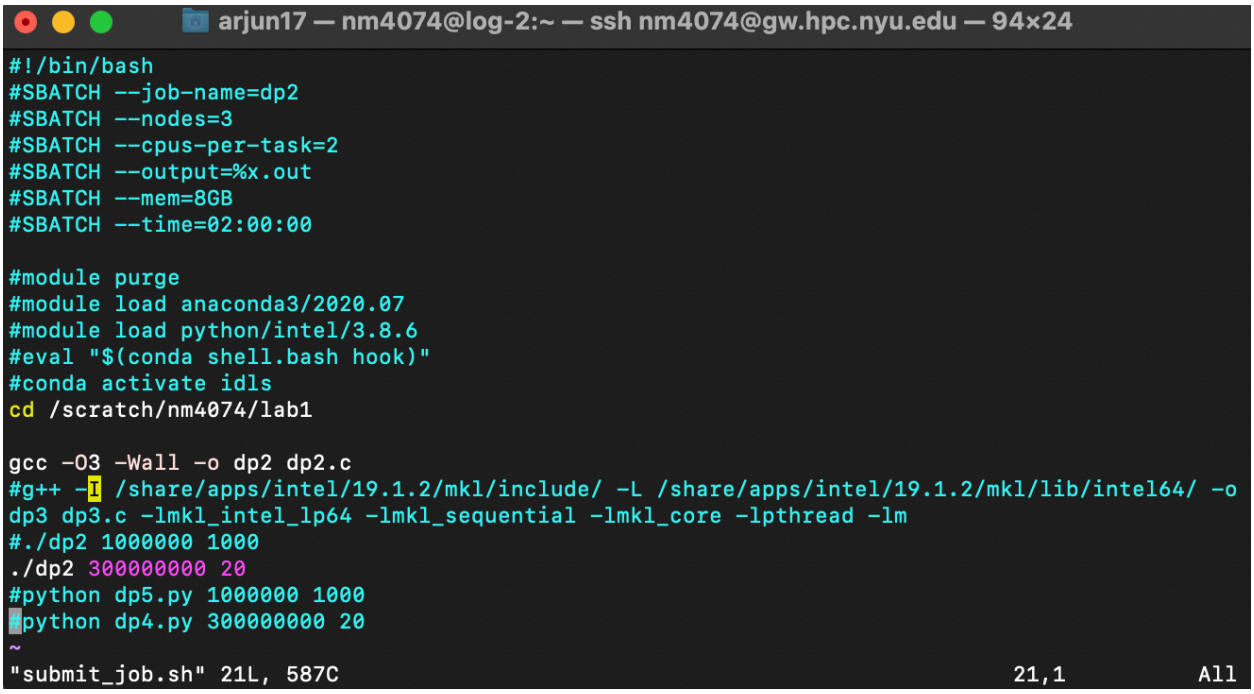
```
#include<stdio.h>
#include<stdlib.h>
#include <time.h>

float dpunroll(long N, float *pA, float *pB) {
float R = 0.0;
int j;
for (j = 0; j < N; j += 4)
R += pA[j] * pB[j] + pA[j + 1] * pB[j + 1] + pA[j + 2] * pB[j + 2] + pA[j + 3] * pB[j + 3];
return R;
}
int main(int argc, char *argv[]) {
    struct timespec start, end;
    char *k1 = argv[1];
    char *k2 = argv[2];
    long size_of_array = atoi(k1);
    int iterations = atoi(k2);
    float a[size_of_array];
    float b[size_of_array];

    for(long i = 0; i < size_of_array; i++) {
        a[i] = 1;
        b[i] = 1;
    }

    double average_et = 0.0;
    double average_et_per_iteration_in_sec = 0.0;
    double gflops = 0.0;
    double bandwidth = 0.0;
    float p = 0.0;

    for(long i = 0; i < iterations; i++) {
        clock_gettime(CLOCK_MONOTONIC, &start);
        p = dpunroll(size_of_array, a, b);
        clock_gettime(CLOCK_MONOTONIC, &end);
        double time_usec=(((double)end.tv_sec * 1000000 + (double)end.tv_nsec / 1000) - ((double)start.tv_sec * 1000000 + (double)start.tv_nsec / 1000));
        if(i > (iterations / 2) - 1) {
            average_et = average_et + time_usec;
        }
    }
    average_et_per_iteration_in_sec = average_et / ((float)(iterations / 2) * 1000000);
    bandwidth = 9 * 4 * (size_of_array / 4) / (average_et_per_iteration_in_sec * 1000000000); // 3 load and store and converstion from bytes to gb
    gflops = 8 * (size_of_array / 4) / (average_et_per_iteration_in_sec * 10000000000); // 2 floating point ops

    if((long)p < size_of_array) {
            printf("N: %lu <T>: %.08lf sec B: %0.05lf GB/sec F: %.05lf GFLOPS", size_of_array, average_et_per_iteration_in_sec, bandwidth, gflops);
    } else {
            printf("N: %lu <T>: %.08lf sec B: %0.05lf GB/sec F: %.05lf GFLOPS", (long)p, average_et_per_iteration_in_sec, bandwidth, gflops);
    }
    return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
"dp2.c" 51L, 1870C
```

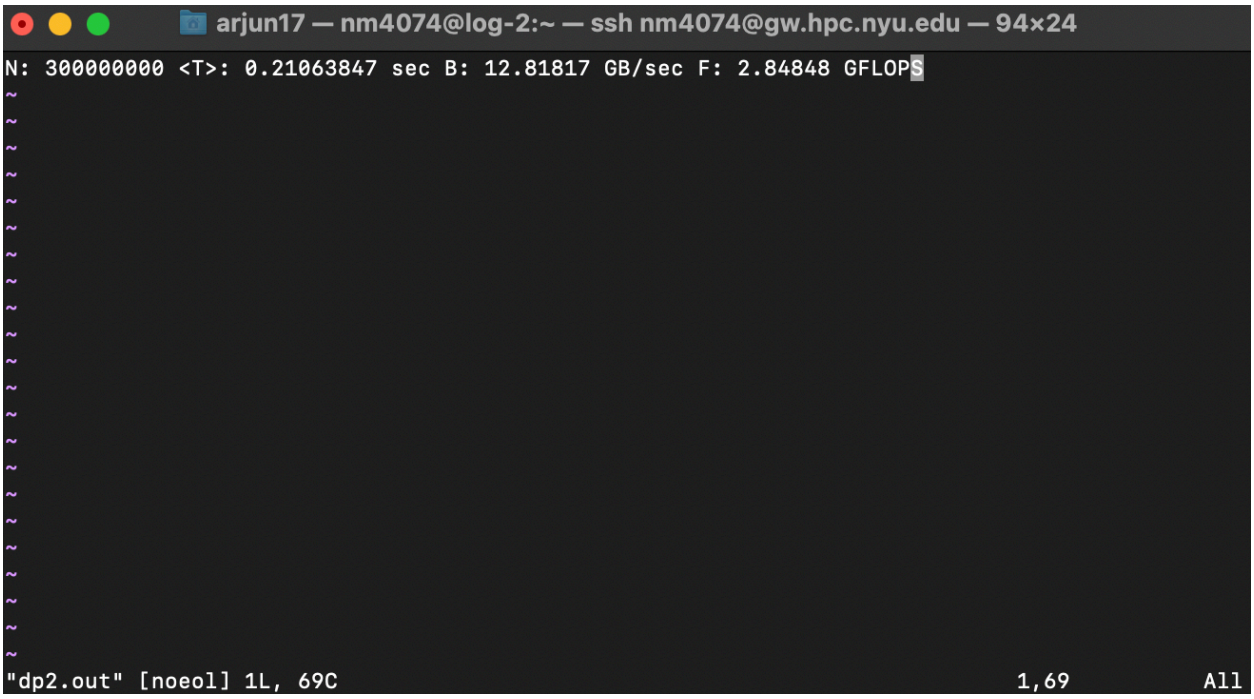
Case 1: N = 1000000 and Repetitions = 1000





Case 2: N = 300000000 and Repetitions = 20



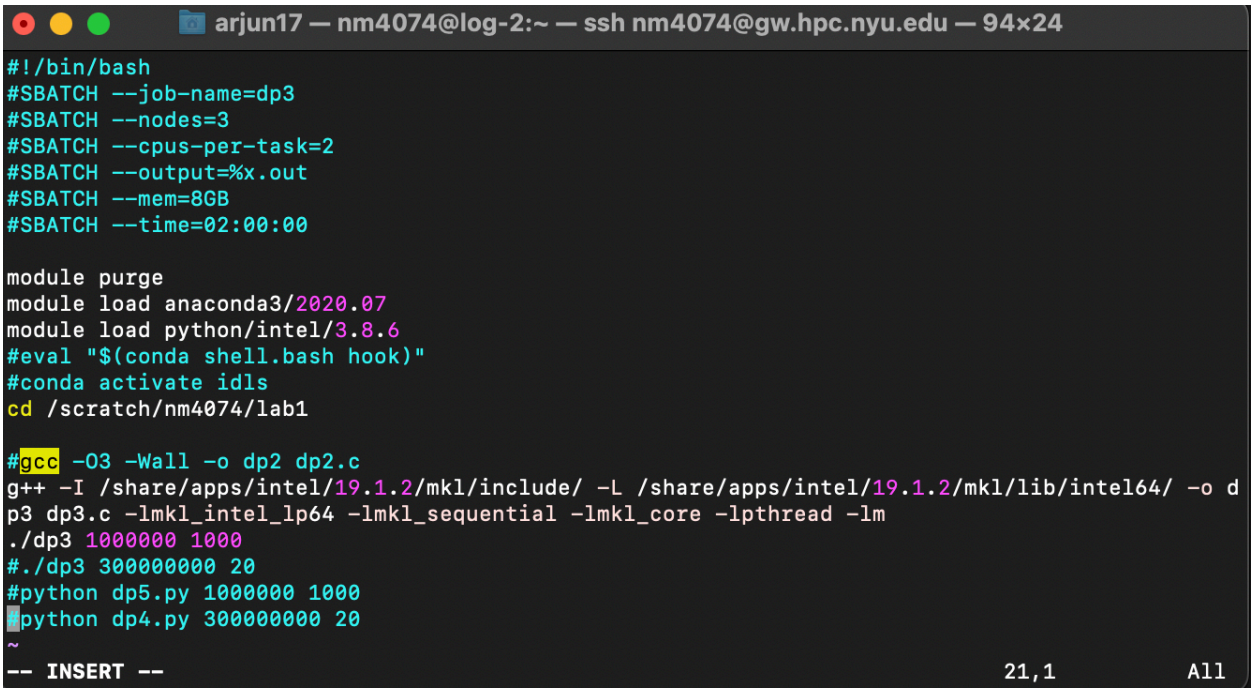


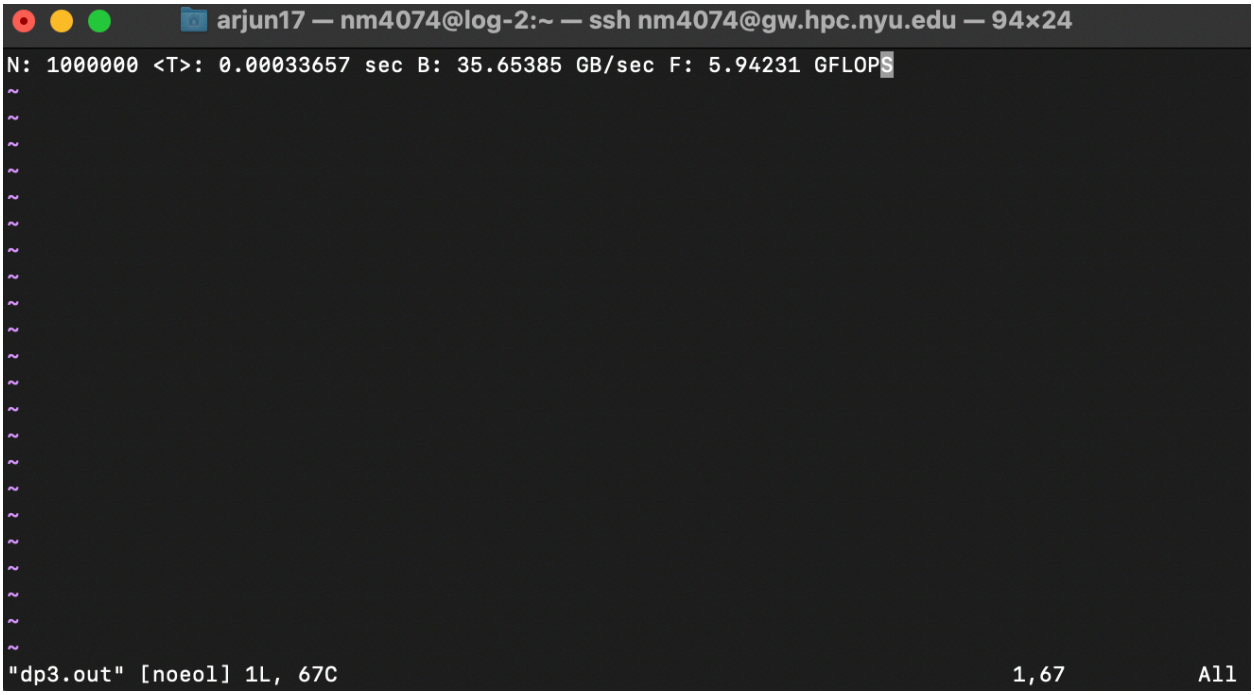
Question C3

Code:

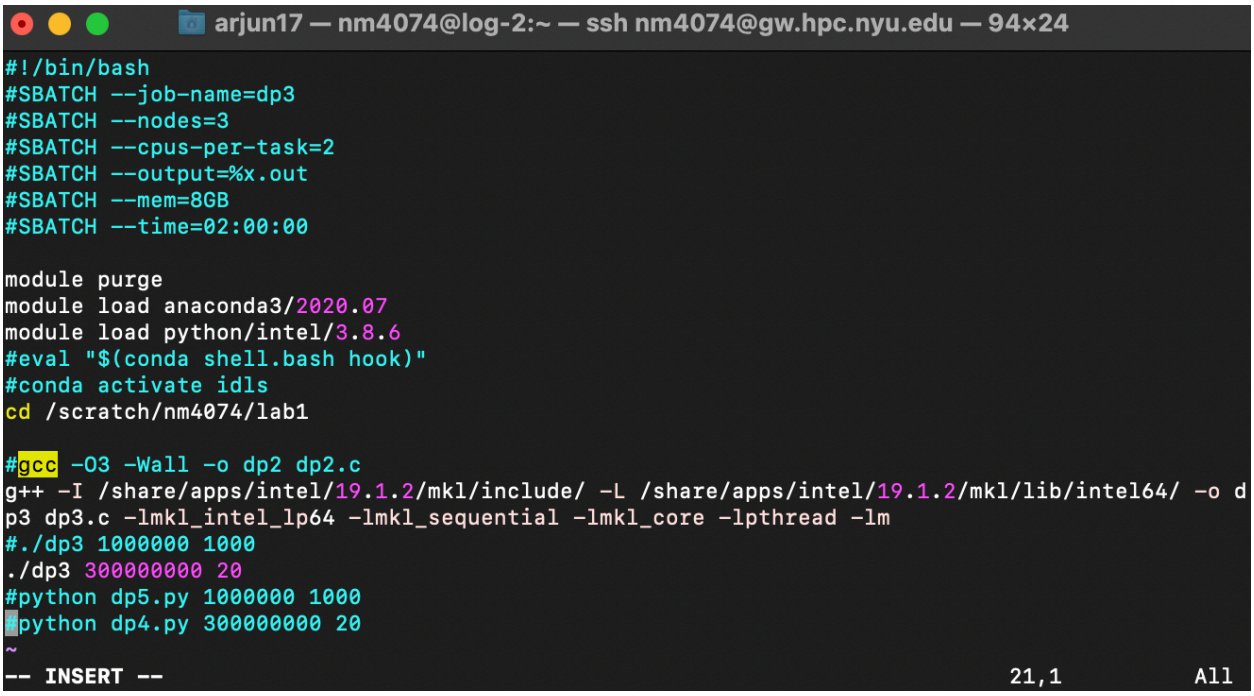
```
#include<stdio.h>
#include<stdlib.h>
#include <time.h>
#include <mkl_cblas.h>

float bdp(long N, float *pA, float *pB) {
    float R = cblas_sdot(N, pA, 1, pB, 1);
    return R;
}
int main(int argc, char *argv[]) {
    struct timespec start, end;
    char *k1 = argv[1];
    char *k2 = argv[2];
    long size_of_array = atoi(k1);
    int iterations = atoi(k2);
    float a[size_of_array];
    float b[size_of_array];

    for(long i = 0; i < size_of_array; i++) {
        a[i] = 1;
        b[i] = 1;
    }

    double average_et = 0.0;
    double average_et_per_iteration_in_sec = 0.0;
    double gflops = 0.0;
    double bandwidth = 0.0;
    float p = 0.0;

    for(long i = 0; i < iterations; i++) {
        clock_gettime(CLOCK_MONOTONIC, &start);
        p = bdp(size_of_array, a, b);
        clock_gettime(CLOCK_MONOTONIC, &end);
        double time_usec=(((double)end.tv_sec * 1000000 + (double)end.tv_nsec / 1000) - ((double)start.tv_sec * 1000000 + (double)start.tv_nsec / 1000));
        if(i > (iterations / 2) - 1) {
            average_et = average_et + time_usec;
        }
    }
    average_et_per_iteration_in_sec = average_et / ((float)(iterations / 2) * 1000000);
    bandwidth = 3 * 4 * size_of_array / (average_et_per_iteration_in_sec * 1000000000); // 3 load and store and converstion from bytes to gb
    gflops = 2 * size_of_array / (average_et_per_iteration_in_sec * 1000000000); // 2 floating point ops

    if((long)p < size_of_array) {
        printf("N: %lu <T>: %.08lf sec B: %0.05lf GB/sec F: %.05lf GFLOPS", size_of_array, average_et_per_iteration_in_sec, bandwidth, gflops);
    } else {
        printf("N: %lu <T>: %.08lf sec B: %0.05lf GB/sec F: %.0lf GFLOPS", (long)p, average_et_per_iteration_in_sec, bandwidth, gflops);
    }
    return 0;
}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dp3.c" 49L, 1782C
```

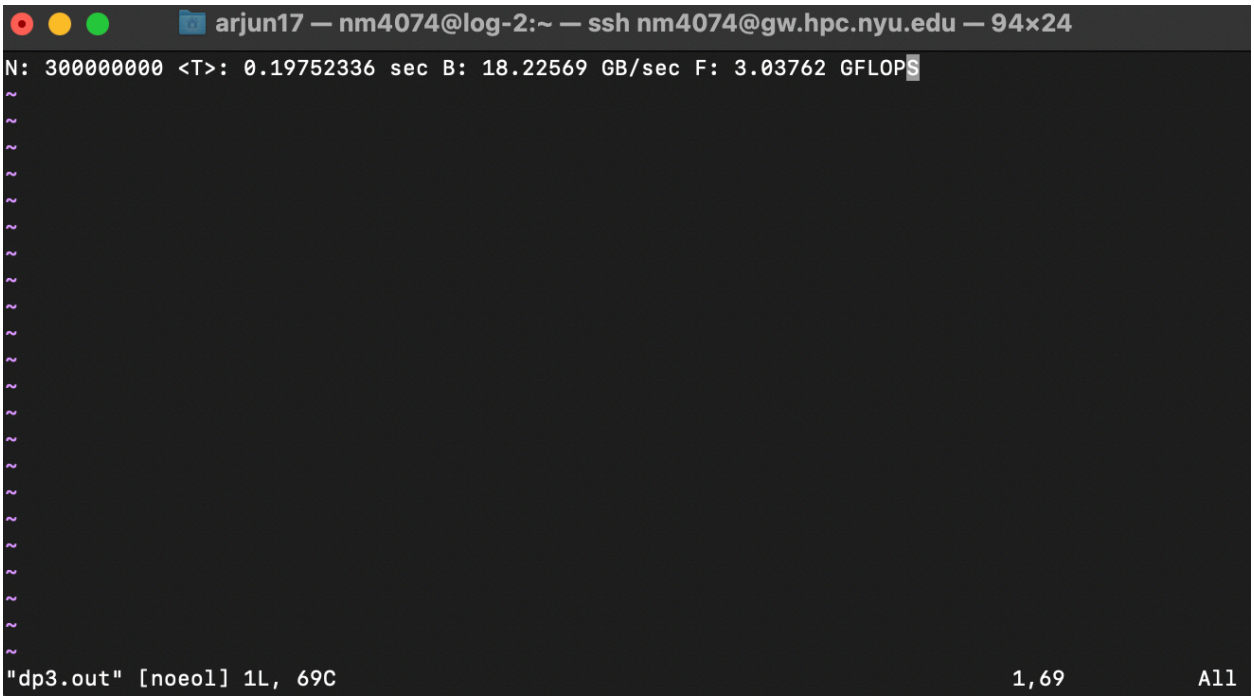
Case 1: N = 1000000 and Repetitions = 1000





Case 2: N = 300000000 and Repetitions = 20



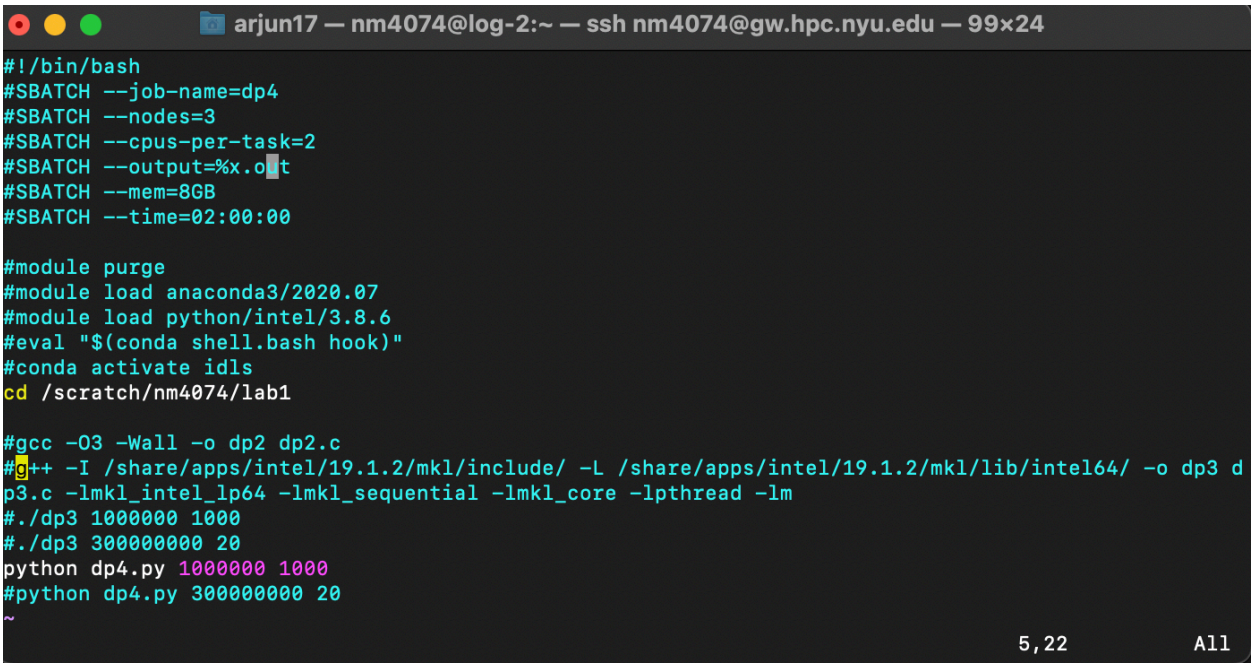


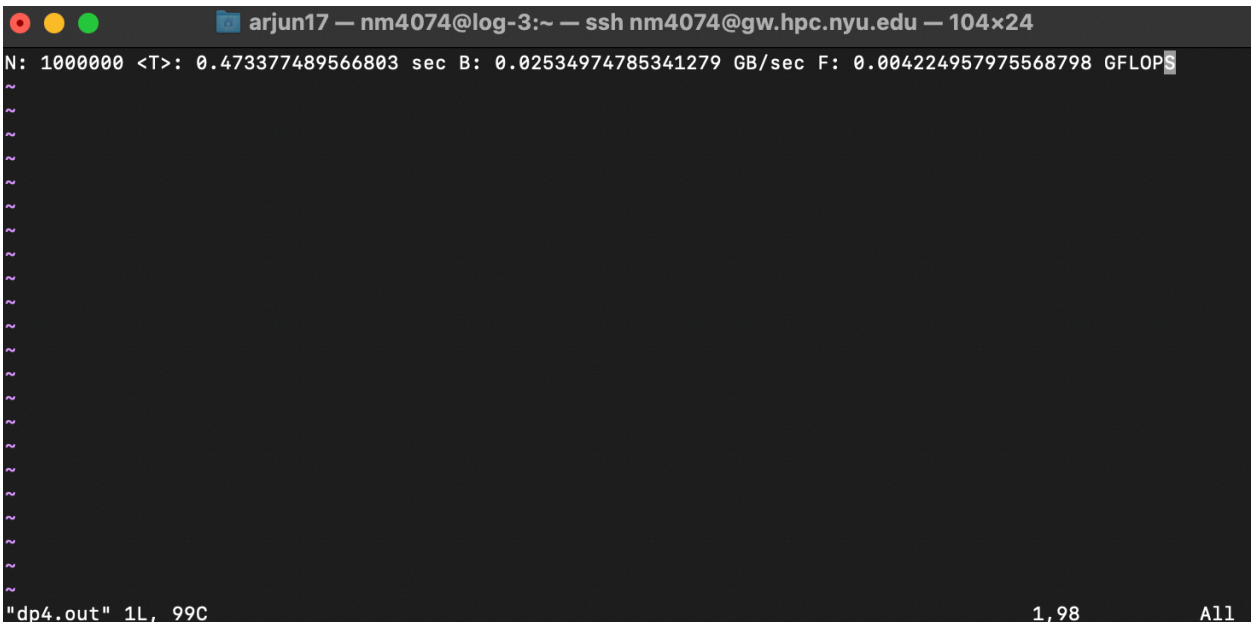
Question C4

Code:

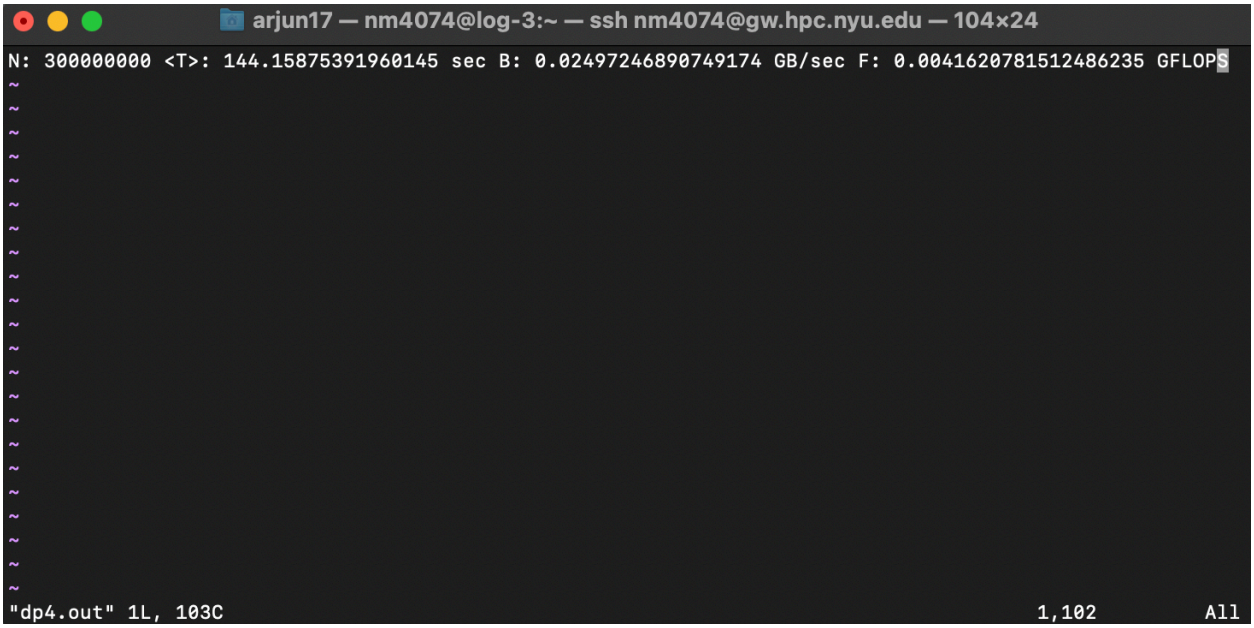
```
import time
import numpy as np
import sys

def dp(N, A, B):
    R = 0.0
    for j in range(0, N):
        R += A[j] * B[j]
    return R

size_of_array = int(sys.argv[1])
iterations = int(sys.argv[2])

A = np.ones(size_of_array, dtype = np.float32)
B = np.ones(size_of_array, dtype = np.float32)

average_et = 0.0
for k in range(0, iterations):
    start = time.time()
    R = dp(size_of_array, A, B)
    end = time.time()
    diff = end - start
    if(k > (iterations / 2) - 1):
        average_et = average_et + diff
average_et = average_et / (float)(iterations / 2)
bandwidth = (3 * 4 * size_of_array / average_et) / 1000000000 # 3 load and store and converstion from bytes to gb
gflops = (2 * size_of_array / average_et) / 100000000 # 2 floating point ops

print("N: " + str(size_of_array) + " <T>: " + str(average_et) + " sec B: " + str(bandwidth) + " GB/sec F: " + str(gflops) + " GFLOPS")
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"dp4.py" 29L, 896C
```

Case 1: N = 1000000 and Repetitions = 1000





Case 2: N = 300000000 and Repetitions = 20

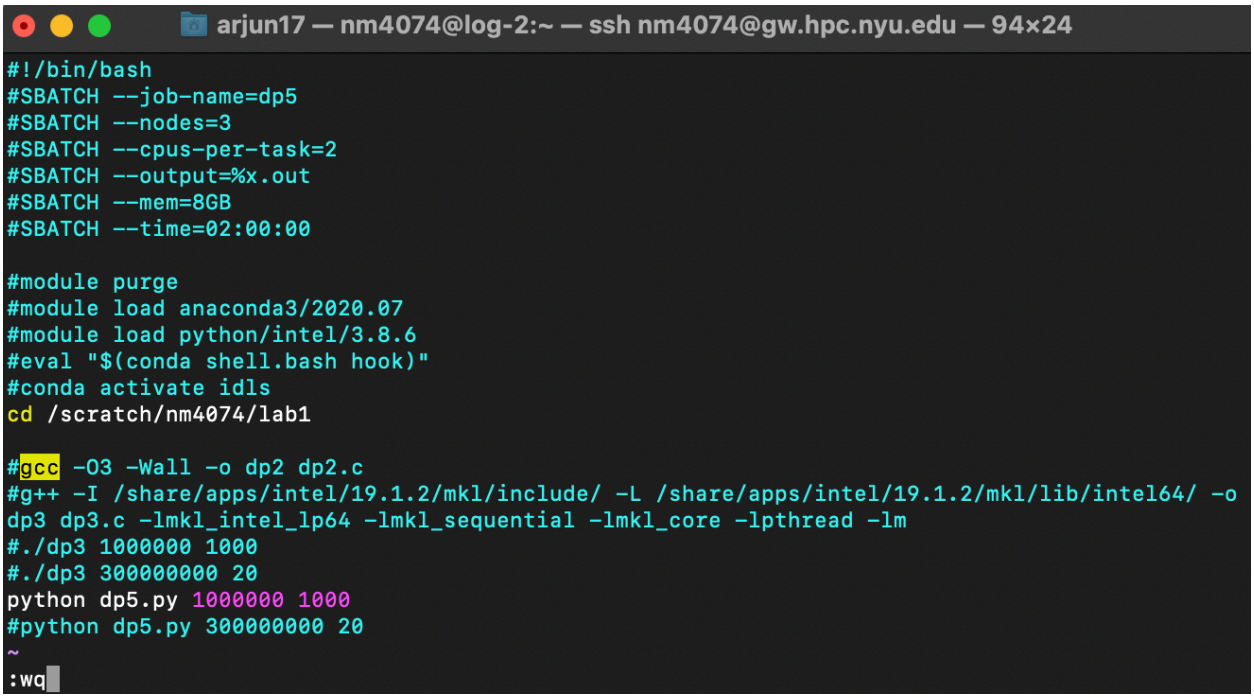


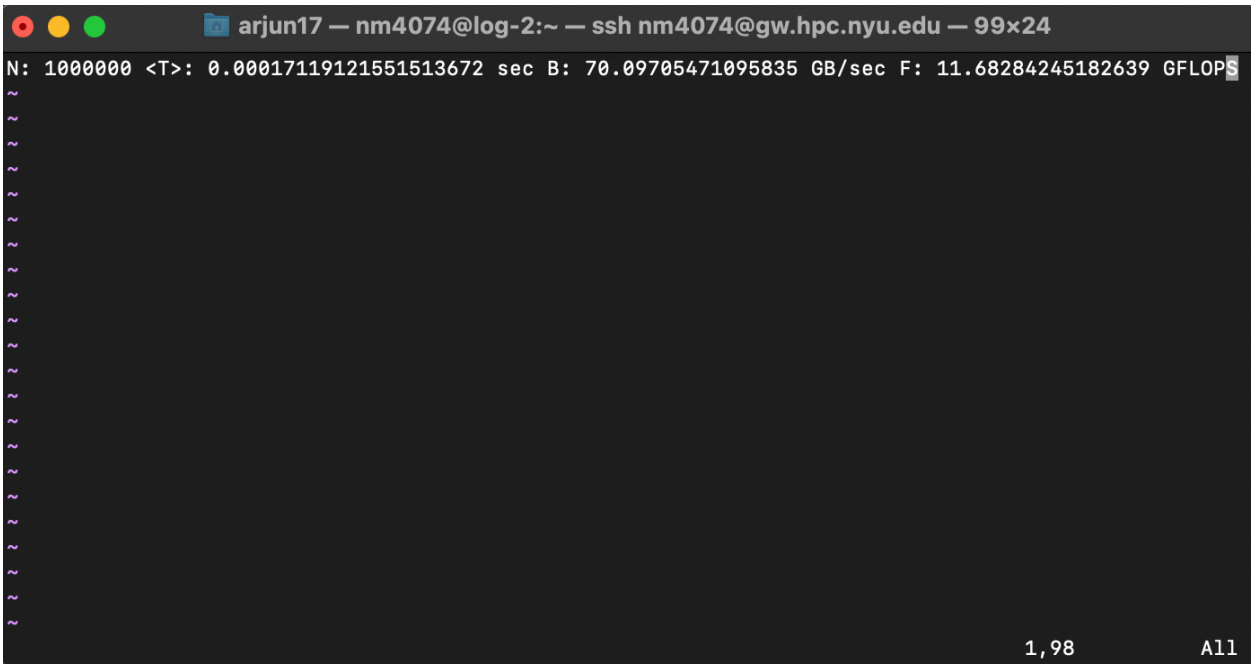
Question 5

Code:

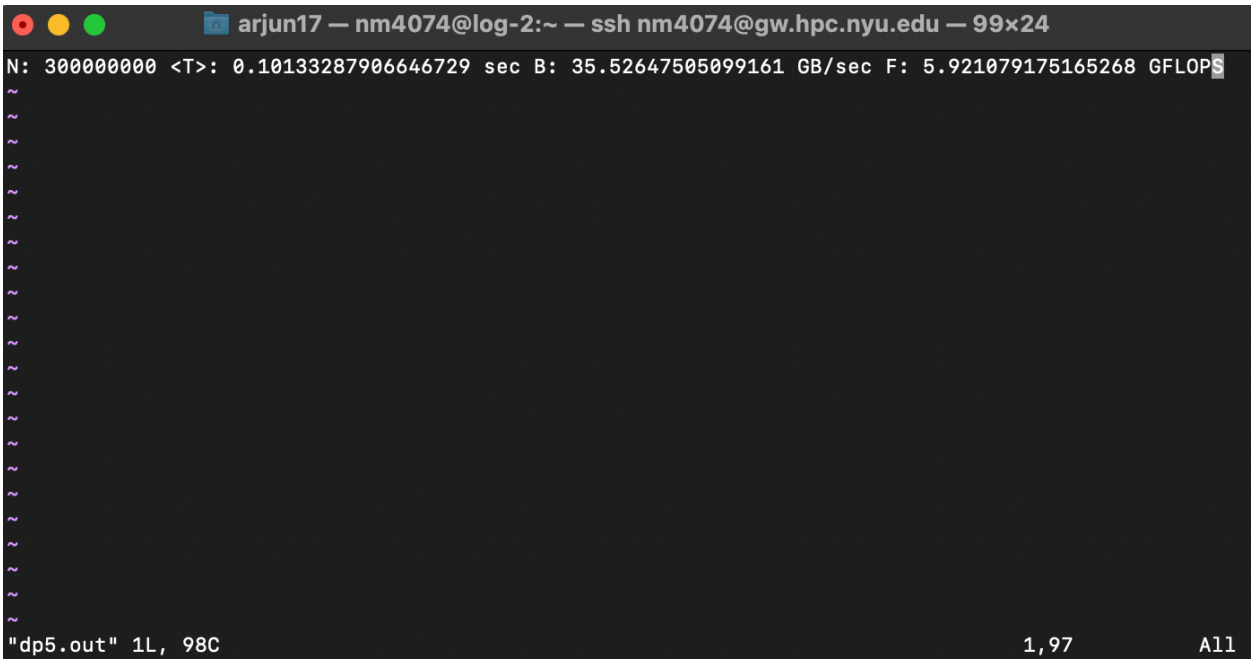
```python
import time
import numpy as np
import sys

size_of_array = int(sys.argv[1])
iterations = int(sys.argv[2])

A = np.ones(size_of_array, dtype = np.float32)
B = np.ones(size_of_array, dtype = np.float32)

average_et = 0.0
for k in range(0, iterations):
    start = time.time()
    R = np.dot(A, B)
    end = time.time()
    diff = end - start
    if(k > (iterations / 2) - 1):
        average_et = average_et + diff
average_et = average_et / (float)(iterations / 2)
bandwidth = (3 * 4 * size_of_array / average_et) / 1000000000 # 3 load and store and converstion from bytes to gb
gflops = (2 * size_of_array / average_et) / 1000000000 # 2 floating point ops

print("N: " + str(size_of_array) + " <T>: " + str(average_et) + " sec B: " + str(bandwidth) + " GB/sec F: " + str(gflops) + " GFLOPS")

"dp5.py" 24L, 792C
```

Case 1: N = 1000000 and Repetitions = 1000





Case 2: N = 300000000 and Repetitions = 20

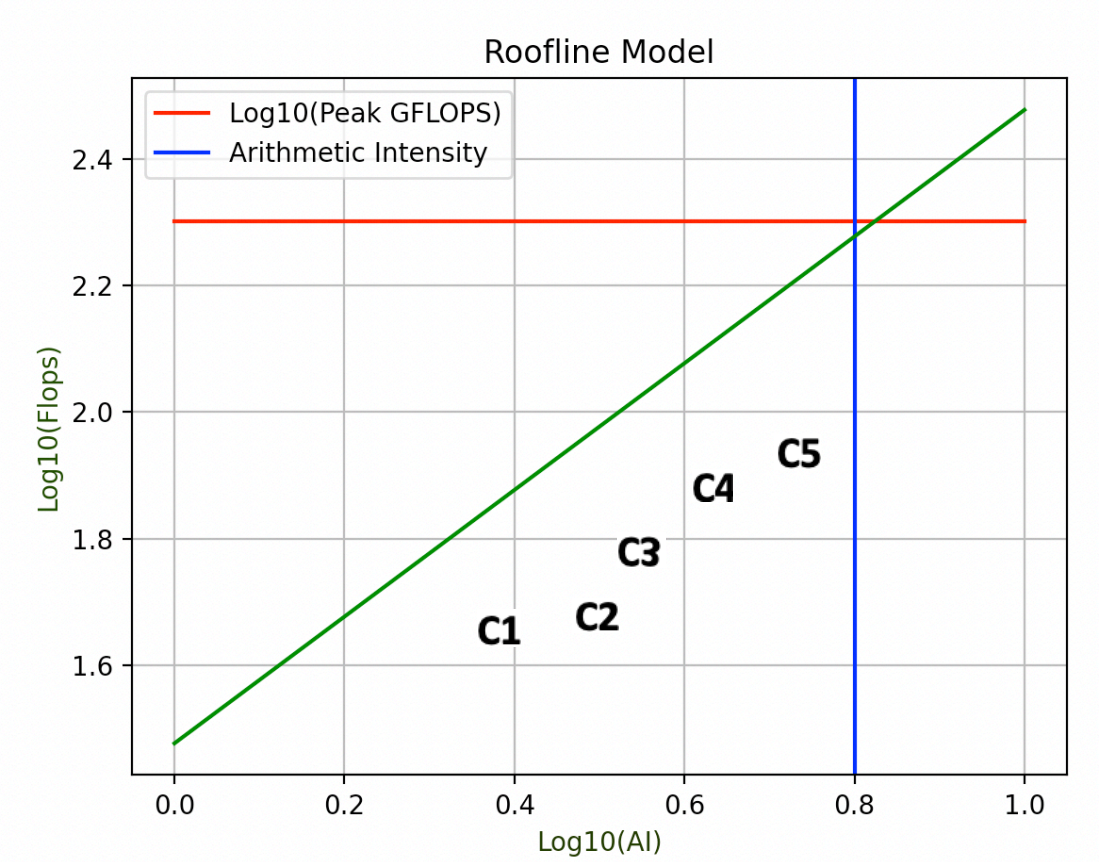


Question Q1

While computing the mean, if we use the second half of the iterations, we will have uniform values as all the caching has been done in the first half. We can see in the image of the example of C1 where the first few iterations have a lot of non-uniform values. If we had taken the first half, we would have calculated a higher mean before caching. Taking the mean in the second half will be precise and accurate.

```
Iteration: 0, Time in uSec: 1224.810547
Iteration: 1, Time in uSec: 1206.732422
Iteration: 2, Time in uSec: 1205.605469
Iteration: 3, Time in uSec: 1201.222656
Iteration: 4, Time in uSec: 1214.189453
Iteration: 5, Time in uSec: 1215.066406
Iteration: 6, Time in uSec: 1191.097656
Iteration: 7, Time in uSec: 1234.021484
Iteration: 8, Time in uSec: 1215.396484
Iteration: 9, Time in uSec: 1187.580078
Iteration: 10, Time in uSec: 1189.050781
Iteration: 11, Time in uSec: 1196.466797
Iteration: 12, Time in uSec: 1190.054688
Iteration: 13, Time in uSec: 1187.777344
Iteration: 14, Time in uSec: 1179.914062
Iteration: 15, Time in uSec: 1186.525391
Iteration: 16, Time in uSec: 1205.105469
Iteration: 17, Time in uSec: 1189.777344
Iteration: 18, Time in uSec: 1199.091797
Iteration: 19, Time in uSec: 1190.730469
Iteration: 20, Time in uSec: 1214.230469
Iteration: 21, Time in uSec: 1199.701172
Iteration: 22, Time in uSec: 1190.013672
Iteration: 23, Time in uSec: 1191.626953
Iteration: 24, Time in uSec: 1192.902344
Iteration: 25, Time in uSec: 1189.742188
Iteration: 26, Time in uSec: 1182.367188
Iteration: 27, Time in uSec: 1175.417969
Iteration: 28, Time in uSec: 1174.888672
Iteration: 29, Time in uSec: 1176.736328
Iteration: 30, Time in uSec: 1255.607422
Iteration: 31, Time in uSec: 1191.691406
Iteration: 32, Time in uSec: 1195.251953
Iteration: 33, Time in uSec: 1189.882812
Iteration: 34, Time in uSec: 1194.591797
Iteration: 35, Time in uSec: 1218.328125
Iteration: 36, Time in uSec: 1200.294922
Iteration: 37, Time in uSec: 1192.693359
Iteration: 38, Time in uSec: 1185.648438
Iteration: 39, Time in uSec: 1179.324219
Iteration: 40, Time in uSec: 1187.224609
Iteration: 41, Time in uSec: 1169.968750
Iteration: 42, Time in uSec: 1177.837891
Iteration: 43, Time in uSec: 1182.578125
Iteration: 44, Time in uSec: 1181.636719
Iteration: 45, Time in uSec: 1185.355469
Iteration: 46, Time in uSec: 1171.484375
Iteration: 47, Time in uSec: 1185.255859
Iteration: 48, Time in uSec: 1190.847656
Iteration: 49, Time in uSec: 1192.361328
Iteration: 50, Time in uSec: 1192.707031
Iteration: 51, Time in uSec: 1195.683594
Iteration: 52, Time in uSec: 1186.031250
Iteration: 53, Time in uSec: 1167.242188
Iteration: 54, Time in uSec: 1172.443359
Iteration: 55, Time in uSec: 1179.966797
Iteration: 56, Time in uSec: 1189.267578
Iteration: 57, Time in uSec: 1179.376953
Iteration: 58, Time in uSec: 1181.082031
Iteration: 59, Time in uSec: 1179.847656
Iteration: 60, Time in uSec: 1182.464844
Iteration: 61, Time in uSec: 1201.361328
Iteration: 62, Time in uSec: 1205.285156
Iteration: 63, Time in uSec: 1189.878906
```

Question Q2:



AI = 200 / 30 = 6.66666
Log(6.66666) = 0.8

Question Q3:

For N = 300000000,
Case C1,
<T>: 0.3676, B: 9.79266, F: 1.63211
Case C2,
<T>: 0.2106, B: 12.81817, F: 2.84848
Case C3,
<T>: 0.19752, B: 18.22569, F: 3.03762

Time taken for C1 > Time taken for C2 > Time taken for C3
This is because C2 runs 4 times lower in loop than C1. And C3 uses a specialized library which is optimized to produce the best output.

Bandwidth for C1 < Bandwidth for C2 < Bandwidth for C3
Throughput for C1 < Throughput for C2 < Throughput for C3
This is because there are lesser number of floating-point operations compared to C1 and the time taken to execute is higher in C1

Question Q4:

For N = 300000000,
Case C1,
<T>: 0.3676, B: 9.79266, F: 1.63211
Case C5,
<T>: 0.101332, B: 35.52647, F: 5.92107

Here, the performance of C5 is very higher than C1.

And in the analytically calculated C1, the floating-point operations like multiplication and addition are not exact. It means that there is a limit in accuracy and precision of the results obtained while calculating analytically.
But in the case of C5, it uses np.dot function. It is a specialized function designed to use the floating-point operations efficiently and effectively. This means that the output we get is accurate and precise as the floating-point operations are exact.