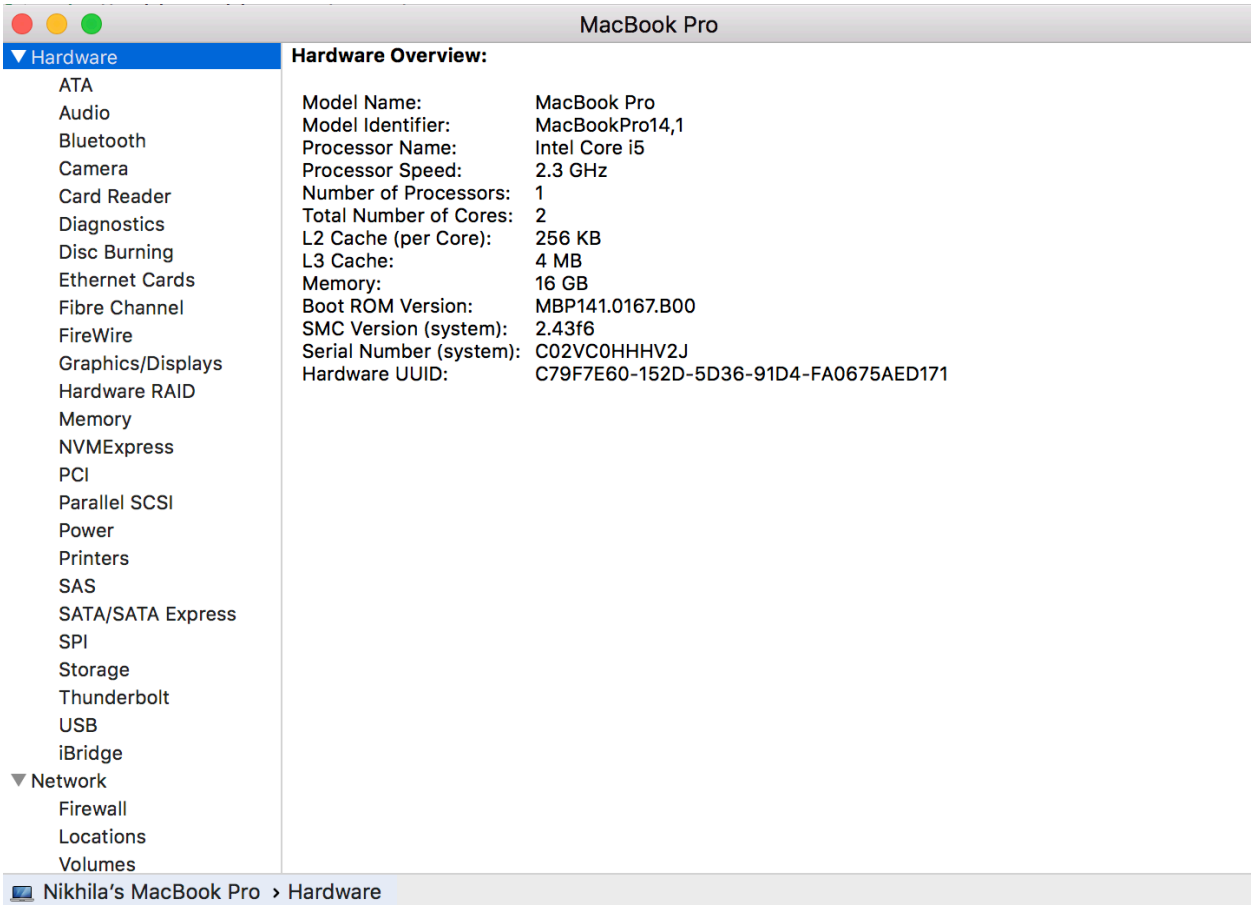
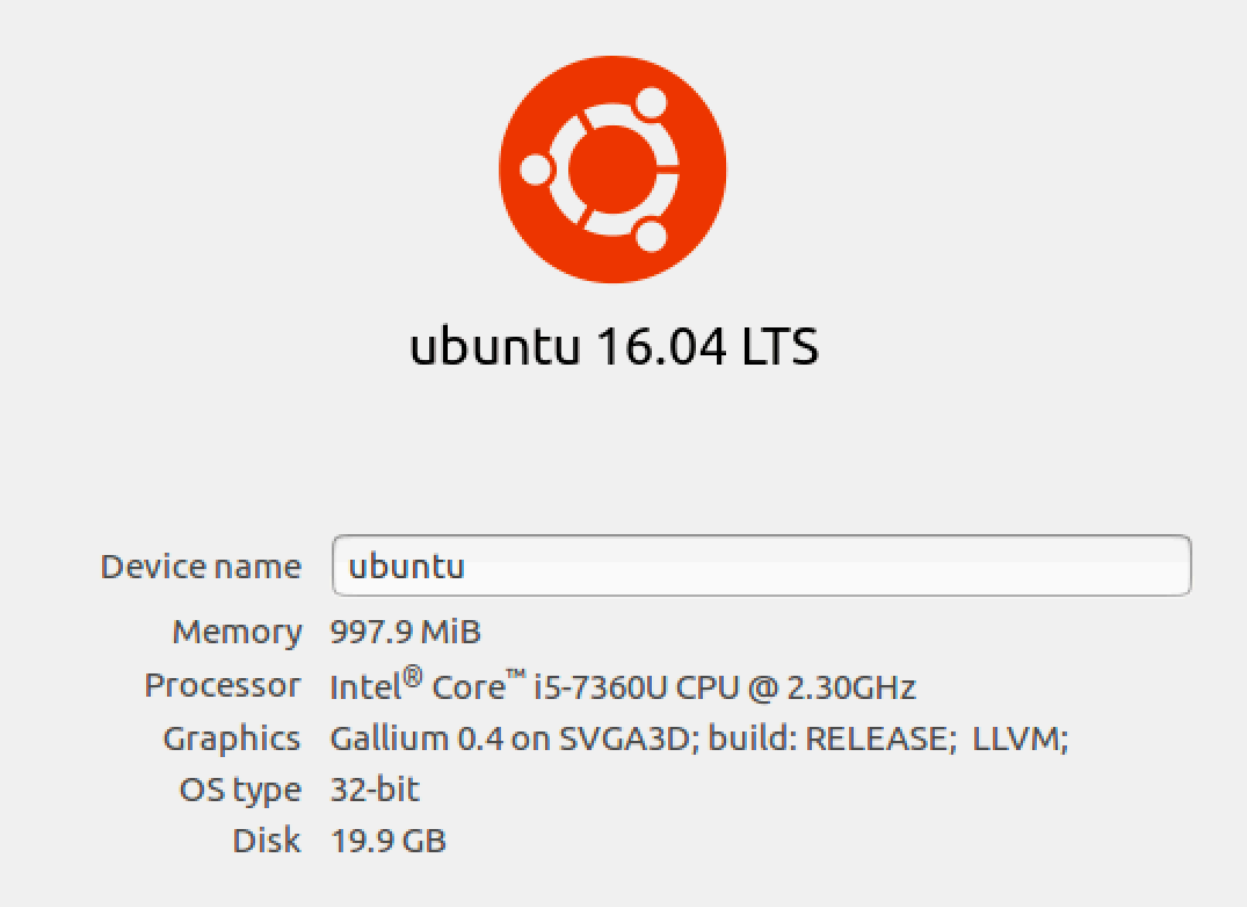
Nikhila Byreddy
SMU ID - 47293718

1)Assignment Requirements :
Install VMWare Fusion.
Install 32 bit Ubuntu as a guest Operating System.
Later install another virtual machine —> 64 bit Ubuntu as a guest Operating System.
Consider the code smuprogram1a which is used in Lab2.

2) Experiment Design :

(a) Machine Configuration :


MacBook Pro
Hardware
ATA
Audio
Bluetooth
Camera
Card Reader
Diagnostics
Disc Burning
Ethernet Cards
Fibre Channel
FireWire
Graphics/Displays
Hardware RAID
Memory
NVMExpress
PCI
Parallel SCSI
Power
Printers
SAS
SATA/SATA Express
SPI
Storage
Thunderbolt
USB
iBridge
Network
Firewall
Locations
Volumes
Hardware Overview:
Model Name: MacBook Pro
Model Identifier: MacBookPro14,1
Processor Name: Intel Core i5
Processor Speed: 2.3 GHz
Number of Processors: 1
Total Number of Cores: 2
L2 Cache (per Core): 256 KB
L3 Cache: 4 MB
Memory: 16 GB
Boot ROM Version: MBP141.0167.B00
SMC Version (system): 2.43f6
Serial Number (system): C02VC0HHHV2J
Hardware UUID: C79F7E60-152D-5D36-91D4-FA0675AED171
Nikhila's MacBook Pro › Hardware



2B) I have used smuprogram1a
It is shown below :

```c
/* John M. Medellin SMU Lyle CSE 7346 Fall 2017, Program 1 a computational exhaustion */

/* Program name is program1a.c compiled using gcc smuprogram1a.c -o smuprogram1a.out */
/* Define Includes */
/* ----------------*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <stdarg.h>
#include <time.h>
#include <signal.h>
#include <limits.h>
#include <pthread.h>
#include <sys/types.h>
#include <stdbool.h>
#include <unistd.h>

/* Define counter…number of modulo multiples to find before the program finishes
randum number, modulus operator to determine if it's a valid number
_____________________________________________________________________ */

        long long int count = 0;
        int maxcount = 3;
        long long int randnum = 1;
        int powerof = 1;
        long long int modoper = 1;
        long long int numtries = 0;
        long long int totnumtries = 0;




int main (void)
{
clock_t begin = clock();
/* Accept input from the key board to drive the number of modular compliant numbers
the modulo operator to be used and the elevation of rand to a power of 10

THE CODE DEFAULTS TO THE FOLLOWING:

100 MULTIPLES OF THE ROUND NUMBER TO FIND
2^34TH MODULO OPERATOR
2^10TH IS USED TO MULTIPLY TIMES ROUND NUMBER TO COMPARE TOT THE 2^34TH MODULO

THE CODE CAN ALSO BE MADE AS INPUT FROM THE KEYBOARD BY MAKING THE PRINTF/SCAN LINES ACTIVE
```

```
//      printf("Enter the number of modulars to find (default is 3) \n");
//      scanf ("%i",&maxcount);
        maxcount = 100;

//      printf("Enter the modulo operator in power of 2 (10 = 1024) \n");
//      scanf ("%lli",&modoper);
        modoper = 34;

        modoper = ldexp(1,modoper);
//      printf("%lli \n",modoper);
//      printf("Enter the power of 2 that we are to elevate the random number \n");
//      scanf ("%i",&powerof);
        powerof = 10;

//      printf("Rand^pwr \t Modulo \t [rand^pwr]/Modulo \t #Iter \n");
//      printf("------------------------------------------------ \n");

// BEGIN LOOP TO FIND THE NUMBER OF OCCURRENCES WHERE THE RANDOM IS A FUNCTION OF THE MODULO

        while (count < maxcount)
{

/* Fetch a random number, elevate it to the power, increase the number of tries counter
_____________________________________________________________________________________ */

                randnum = rand() * ldexp(1,powerof);
                numtries ++;

/* If it is modulo that we want, report it with the number of tries, increase counter
 and zero out the number of tries so we can report the variable again
_____________________________________________________________________________ */

                if (randnum % modoper == 0){

                printf("%lli \t %lli \t %lli \t %lli \n",randnum,modoper,(randnum/modoper),numtries);
                count ++;
                totnumtries = totnumtries + numtries;
                numtries = 0;}
}

// ADD UP THE NUMBER OF TRIES IT TOOK TO GET TO THE NUMBER OF OCCURRENCES AND PRINT IT

printf("Total number of tries %lli \n",totnumtries);

// SYSTEM COMMAND TO ISSUE THE STATISTICS OF THE RUN

system("ps aux");
```

```
system("ps aux");

// PRINT THE PROCESS ID SO IT CAN BE USED TO DETERMINE THE USAGE STATISTICS

printf("process id %d \n",getpid());
clock_t end = clock();
double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
printf("Time taken:%lf",(double)(end-begin)/CLOCKS_PER_SEC);
}
```

2C) I have performed the Optimizations HV-01, HV-02, HV-05

3

(a) Baseline results :

For 32 bit Ubuntu

To find the time taken to execute the smuprogram1a you may need to include begin and end statements as shown below

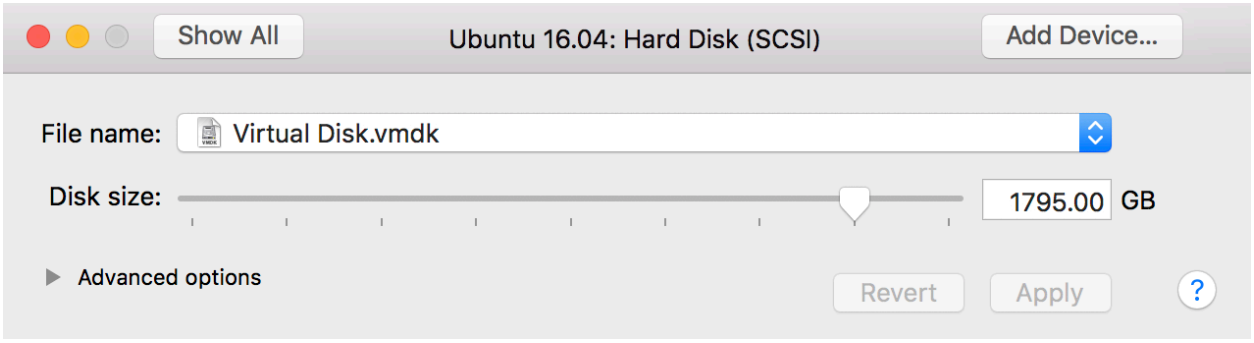
```
int main (void)
{
clock_t begin = clock();
```

```
printf("process id %d \n",getpid());
clock_t end = clock();
double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
printf("Time taken:%lf",(double)(end-begin)/CLOCKS_PER_SEC);
```

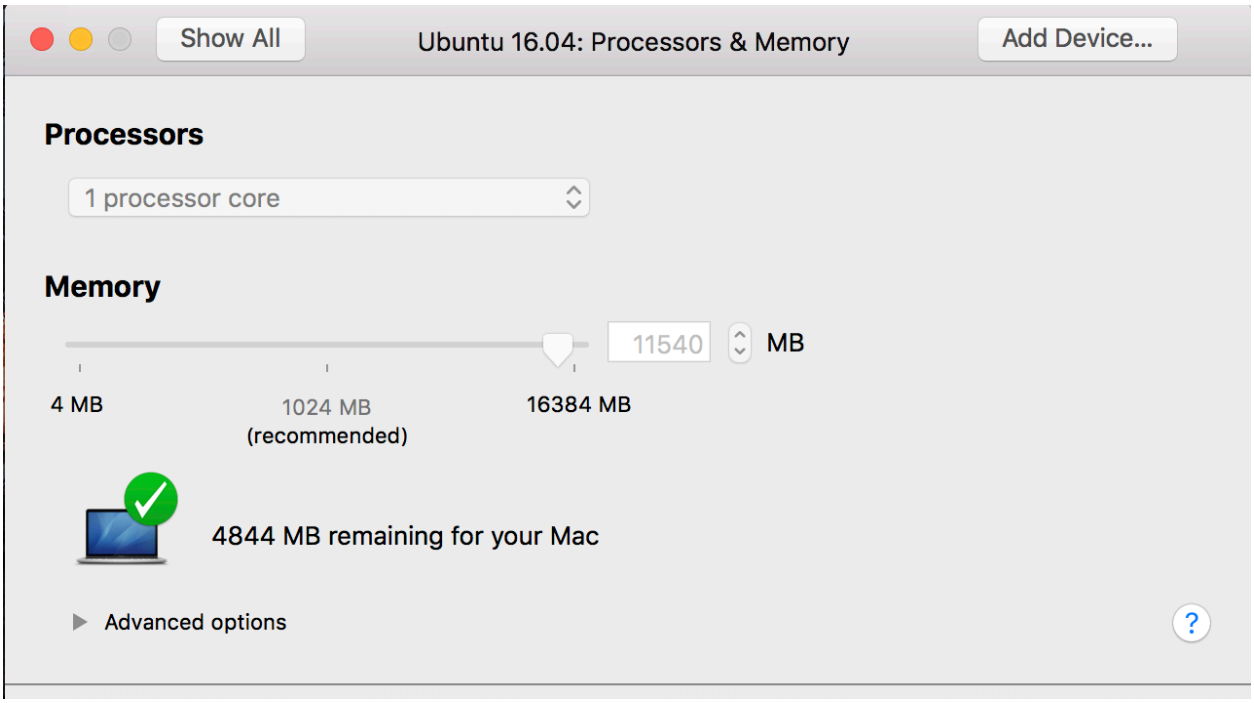
```
process id 5595
Time taken:76.505915nik
```

3b) HV-01

Firstly increase the hard disk memory
This can be done by
Go to Virtual Machine —> Settings —> Hard Disk



Also increase the memory. This can be done by
Go to Settings —> Processors & Memory and then increase the Memory



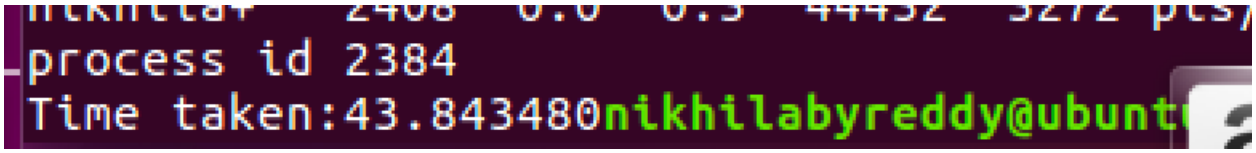
Now when you execute the program you get

```
nikhila+  2174  0.0  0.0   7756  3168 pts/
process id 2170
Time taken:69.876693nikhilabyreddy@ubuntu:
```

HV-02
For 32 bit
Time taken for Smuprogram1a is

```
nikhila+  2174  0.0  0.0   7756  3168 pts/
process id 2170
Time taken:69.876693nikhilabyreddy@ubuntu:
```

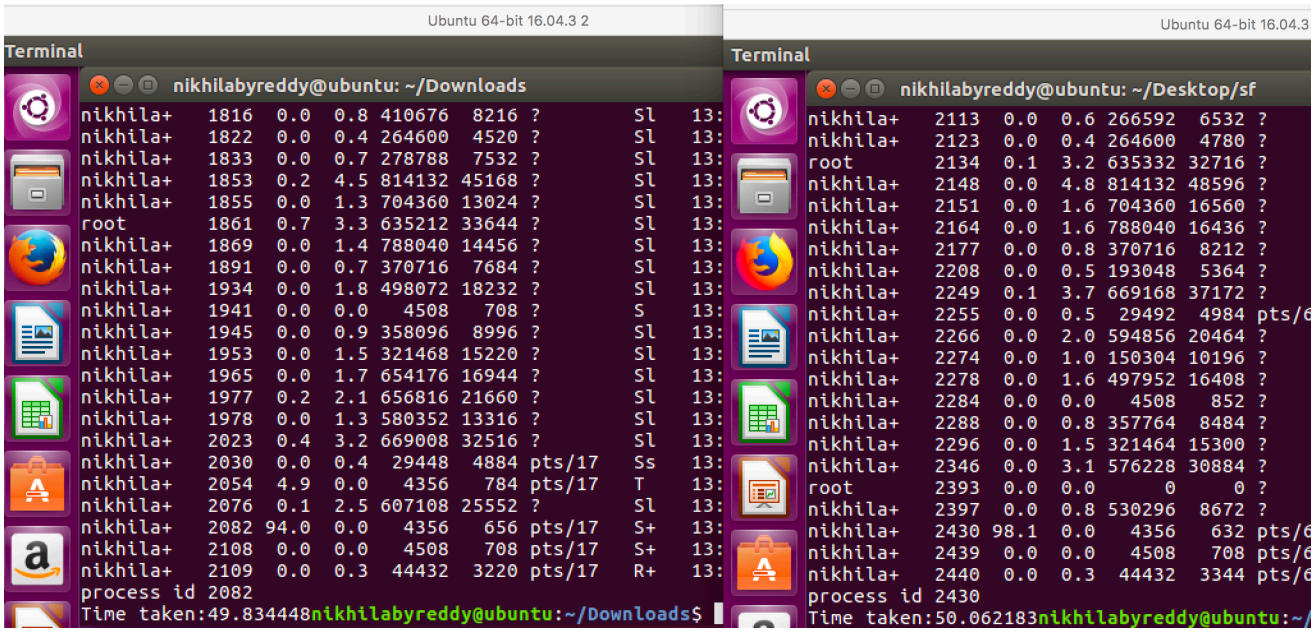
For 64 bit time taken is



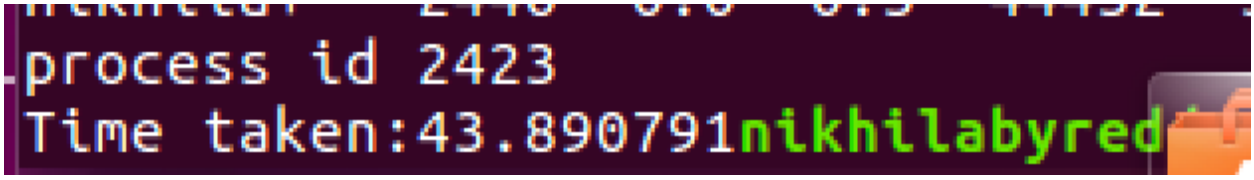
HV-05

When I tried running two concurrent 64 bit virtual machines simultaneously time taken is

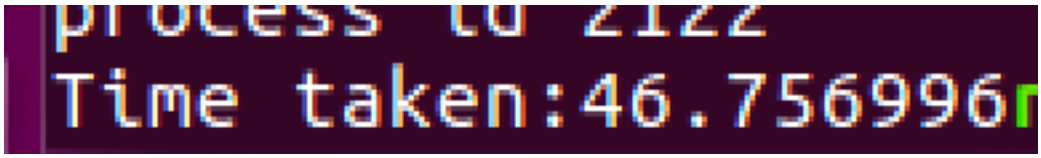
Time taken is



Time taken when I run
Virtual machine 1 alone is

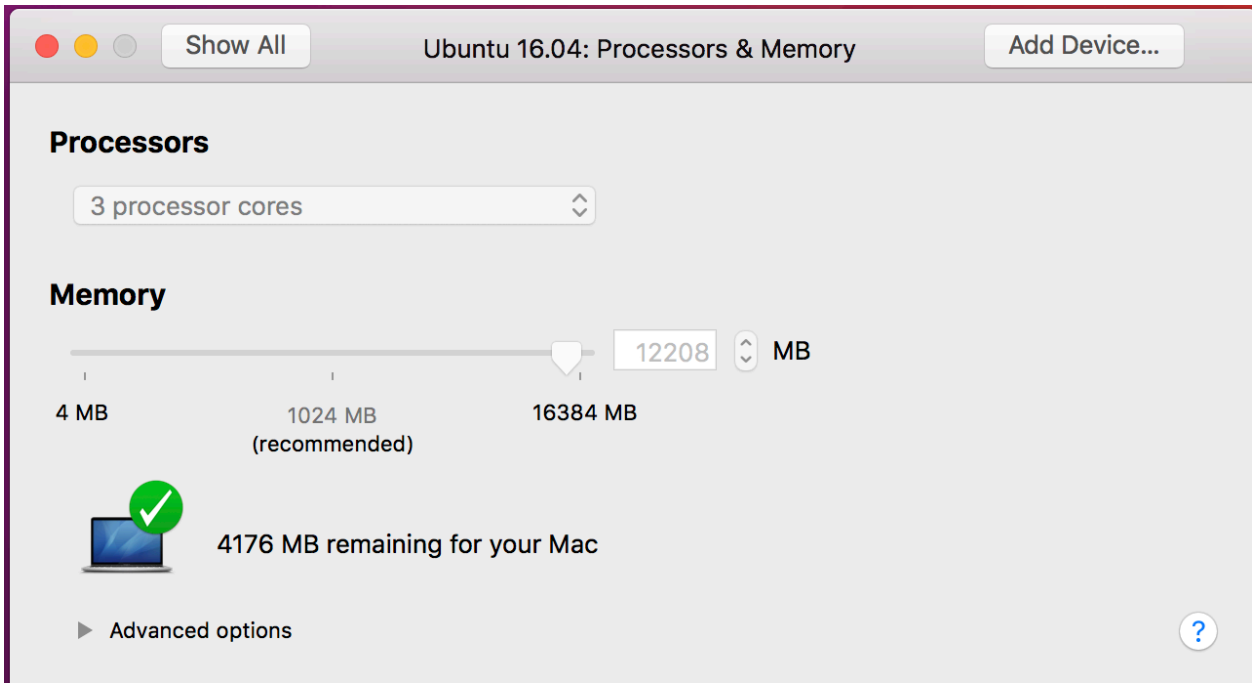


And the time taken when you run Virtual machine 2 alone is



My Own :

Increase the number of core processors.
This can be done by:
Go to settings —> processors & memory



(4A) For Optimization 1

When I ran the code without increasing the memory the time taken was 76.50ms.

Later when I increased the memory space and hard disk space time taken is 69.87ms.

For Optimization 2

when I ran the code for 32 bit Ubuntu Virtual machine the time taken is 76.50ms before increasing the memory and harddisk space and 69.87ms after increasing the space.

when I ran the code for 64 bit Ubuntu Virtual machine the time taken is 43.84ms.

For Optimization 3

When I tried running two 64 bit Ubuntu virtual machines simultaneously time taken for virtual machine 1 is 49.83ms and for virtual machine 2 is 50.06ms.
When I tried running virtual machine 1 alone time taken is 43.89ms and virtual machine2 alone is 50.06ms.

For Optimization 4(MY OWN)

when I tried increasing the number of processor cores from 1 to 3 time taken for the execution of program increased thereby reducing the performance.

(4b) When you increase the memory size as in optimization1, the time taken is reduced. This is because as the memory size increases speed of execution of processes increases thereby decreasing the time of execution.

Similarly when you run 64 bit machine, it has more registers compared to 32 bit registers thereby reducing the time for execution in 64 bit machine.

Similarly when you run two 64 bit machines simultaneously performance degrades much worse than linearly with increased concurrency.